Computing Functions Securely: Theory, Implementation and Cryptanalysis

or, Topics in Insecurity

by

Alexander Davidson

Thesis submitted to the University of London,
for the degree of Doctor of Philosophy in Information Security

Information Security Group
Royal Holloway, University of London
2018

Declaration

These doctoral studies were conducted under the supervision of Professor Carlos Cid.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Department of Information Security as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Alexander Davidson

October, 2018

Abstract

The study of secure computation assesses the ability of one or more entities to securely compute functions over privately chosen inputs. Numerous different methods of computation exist, ranging from protocols for securely computing functions over data from multiple entities; to providing cryptographically obfuscated versions of functions that still evaluate correctly. In the former case, security guarantees that the inputs are not revealed beyond what is revealed naturally by the output of the function. In the latter case, the function itself should remain hidden, even when ran in an adversary-controlled environment.

The vast array of conveyed functionality has led to a wealth of research literature with varying motivations. The theory of secure computation has existed for around three decades, and more recent work has considered the plausibility of developing practical applications based on the underlying concepts of this theory. In particular, demonstrating how efficiently we can run protocols for securely computing general functions. As always, cryptanalysis also remains useful for establishing the actual security guarantees that we can expect for those schemes.

In this thesis we provide a study of the feasibility of various forms of secure computation — through theory, practice and cryptanalysis. Firstly, we give theoretical constructions of general secure computation protocols for performing set operations whilst maintaining the privacy of the input sets. Secondly, we provide a new construction of a constrained pseudorandom function. In comparison to previous work, we achieve stronger security properties from much weaker assumptions. Thirdly, we establish a browser-based implementation of a protocol for anonymous authentication, built upon the secure computation of a verifiable, oblivious pseudorandom function between a client and server. Lastly, we give a cryptanalysis of indistinguishability obfuscation candidates for branching programs, when instantiated under a non-trivial adaptation of an existing graded encoding scheme.

Acknowledgements

Thank you to RHUL, the UK Government and the EPSRC for funding me in this research, as part of the grant EP/K035584/1. Thank you to Carlos Cid for supervising me, and allowing me the creative freedom to seek out my own problems. I appreciate the 3.5 years of cultural introduction to Japan that you imparted. Thank you to Kenny Paterson for all of your advice, for all of the valuable introductions, and the encouragement that you always gave.

Thank you to Martin Albrecht for all of your help and perseverance. In particular, finding the patience to watch me try to describe things (mainly garbage multilinear map attempts) in front of a whiteboard. On a similar note, thank you to Pooya Farshim for crystallising the art of the reduction for me.

Thank you to Nick Sullivan for giving me the opportunity to take (two!) internships at Cloudflare and your help in making Privacy Pass a reality. A special thanks also to everyone at the London office for making feel so welcome during both of my times there. Thanks also to George Tankersley, Filippo Valsorda and Ian Goldberg for being great and critical co-authors.

Thank you to Ryo Nishimaki for supervising my research whilst I was at NTT — it was an honour to work with you. Thank you to Saho Uchida for providing me with so much support while I stayed in Japan. A special mention to Shuichi, Jacob, Takashi, Thomas, Pierre-Alain, Andreas, Charlotte, Mehdi and Junichi for contributing to the wonderful time that I had in Tokyo.

Thank you to all of my friends: those who I've known since school; those who I met in a rural wasteland in the West Midlands; and those that I navigated the horrifying world of mathematics with. You are all part of my fondest memories. Thank you also to the fellow PhD students/post-docs that I've had the pleasure of spending time with at RHUL and while attending conferences, talks, workshops, lattice days, dinners, pubs etc.; especially: Amit, Andreas, Ben, Christian, Eamonn, Ela, Fernando, Gio, Greg, James, Joanne, Pip, Rachel, Rob, Thalia, Thyla, and Torben. Without you guys, I fear that I would have actually have had to do some "networking".

Thank you to Mum, Dad, Sam and Jack. I am grateful for all of your support — I hope you know that.

Thank you to Alex. No brighter light has shone in my life than the one that you, so effortlessly, provide. I cannot spend the rest of these vanishing days without you.

Prologue

The “unofficial” sub-title of this thesis may sound alarming. The word “insecurity” has a pretty concrete meaning in cryptography, though I am not suggesting that any of the results in this work need to be called into question.

The choice of title was to reflect on the passage of time since September 2014, when I started this work, and now (September 2018). While my various states and general health respond to a number of variables, the constant of insecurity has never left. Insecurity that I feel: socially; in existence; in assessing the quality of my own work; in comparisons with other people; in being a silent observer to my own life; in rarely showing appreciation to the people that I love; in not doing enough.

There are opportunities that I feel that I have missed. Occasions where I rushed into making judgements, based on the perception that I had little time. Rushing during a four year PhD now seems absurd to me. But I am not convinced that I would make different decisions, if I had to make them again. Not because I think they were correct, but rather that the pressure can, at times, be all-consuming. Not a physical pressure, just like getting a couple of emails that are slightly awkward to reply to. Or waking up in the middle of the night, convincing myself that I need to be working on something, and then being unable to get back to sleep.

My only regret regarding the collation of this thesis is that some of the ideas that I really enjoyed pursuing did not make the final cut. The ideas that I formed myself, where solutions seemed like they grazed the tips of my fingers whilst I paced darkened rooms. One day, I would hope to return to trying to solve these problems, or others similar. Hopefully I haven’t closed this door for myself.

I’m not sure about the future, but I’m leaving academia, again, for now. I only spent one year away last time. I hope I spend longer away this time. There are still various things that I do not understand about myself. I am still not sure who I will become. These are the things I fear. These are my insecurities. Maybe a change will help.

This prologue is for me. So that after I have stopped obsessing about who I will be, I may realise who I once was.

To Grandad:

I hope the hieroglyphs that I have used now make more sense than they have done in the past.

Contents

List of Figures

List of Tables

List of Publications

This is a list of all the publications that the thesis author contributed to during their studies.

- A. Davidson, G. Fenn, and C. Cid. "A Model for Secure and Mutually Beneficial Software Vulnerability Sharing". In: *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*. WISCS '16. Vienna, Austria: ACM, 2016, pp. 3–14. ISBN: 978-1-4503-4565-1. DOI: 10.1145/2994539.2994547

- A. Davidson and C. Cid. "An Efficient Toolkit for Computing Private Set Operations". In: *ACISP 17, Part II*. ed. by J. Pieprzyk and S. Suriadi. Vol. 10343. LNCS. Springer, Heidelberg, July 2017, pp. 261–278; full version: https://eprint.iacr.org/2016/108

- M. R. Albrecht, A. Davidson, and E. Larraia. "Notes on GGH13 Without the Presence of Ideals". In: *16th IMA International Conference on Cryptography and Coding*. Ed. by M. O'Neill. Vol. 10655. LNCS. Springer, Heidelberg, Dec. 2017, pp. 135–158; full version: https://eprint.iacr.org/2017/906

- A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda. "Privacy Pass: Bypassing Internet Challenges Anonymously". In: *PoPETs* 2018.3 (2018), pp. 164–180. DOI: 10.1515/popets-2018-0026. URL: https://doi.org/10.1515/popets-2018-0026

- M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. W. Postlethwaite, F. Virdia, and T. Wunderer. "Estimate All the {LWE, NTRU} Schemes!" In: *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*. 2018, pp. 351–367. DOI: 10.1007/978-3-319-98113-0_19. URL: https://doi.org/10.1007/978-3-319-98113-0%5C_19; full version: https://eprint.iacr.org/2018/331

The following is a list of works that are under review for conference proceedings, at the time of thesis submission.

- A. Davidson, S. Katsumata, R. Nishimaki, and S. Yamada. "Constrained PRFs for Bit-fixing from OWFs with Constant Collusion Resistance", *under submission*; full version: `https://eprint.iacr.org/2018/982`

I Introduction

the end

The capability to communicate messages that are kept secret from malicious eavesdroppers has been puzzled over for millennia. The scientific pursuit of providing security in such settings is known as the study of *cryptography*. Cryptography is primarily concerned with devising *ciphers* that can be used to convert messages into unreadable text, using a cryptographic *key*. The cipher can then be inverted by anyone in possession of secret components associated with said key.

The usage of cryptography has been an intrinsic part of the formulation of civilisation. The pathway from the earliest known recordings of the Caesar cipher, to the algorithms of the present day has been fraught with intrigue, conflict and — ultimately — death and tragedy. However, it has only been within the last century that cryptographic techniques have been regarded as a meaningful scientific pursuit. The need for widespread cryptographic assurance has only become prominent alongside the rise of provisions for global communication.

The contemporary societal impact of cryptographic research is heavily influenced by the wider implications of widespread encryption of communication. What rights that an individual citizen has to maintain private, online communications has engendered a wide-reaching and, at times, frenzied debate. Usually, this debate is co-opted to advance political agendas and interests related to state security. For instance, encrypted communications are sometimes associated with the planning phases of acts of terrorism. In the wake of such acts, it is common to find the resulting national discourse emphasising the negative aspects of message encryption.[1]

On this topic, there is a well-known fable of the tactics employed by Sir Francis Walsingham to reverse the weak cipher of Mary Queen of Scots. The cipher was used to communicate words of 'treason' from Mary's place of imprisonment in the 17th century; and was eventually used a piece of key evidence in bringing about her execution.[2] We might take this story as an allegory foreshadowing attempts to dismantle cryptographic privacy in the context of the isolationist (and nationalist) security debates of the 21st century. It is only in recent times that such connections have been made.

From out of this long nurturing has come a vibrant research area: moulding aspects of social, mathematical and computational topics into a broader science. The core that remains is that cryptography seeks to provide secure communication channels for individuals to communicate. Since the 20th century, the central question surrounding cryptography is

<u>how</u> can we communicate securely?

A subtle difference that has arisen in more modern studies is that cryptographers have instead begun to also ask:

<u>what</u> can we communicate securely?

[1]https://inews.co.uk/news/politics/whatsapps-end-to-end-encryption-must-end/
[2]http://www.bbc.co.uk/history/british/tudors/spying_01.shtml

The practise of cryptography now takes for granted the existence of wide-spread availability of communication channels ensuring the privacy of communication between two parties.[3] Using these established channels, we can now ask what extended functionality is possible.

In this thesis, we will concern ourselves with a branch of cryptography known as 'secure computation' that was born over three decades ago. The primary goal of secure computation is to provide individuals with the ability to *compute* functions privately, even in the presence of eavesdroppers. In other words, whereas early-stage cryptography was primarily focused on securing *communication*, our work is concerned with securing the *computation* of functions (over private inputs) between individuals.

I;1 Secure computation

The study of secure computation started with the goal of allowing multiple participants to engage in online protocols that allowed them to compute functions over their combined inputs. A vital requirement of the engagement is that the inputs of each participant stay hidden throughout. The work of Yao [296] initiated this study in 1982, alongside the growing shift towards developing provable security in cryptosystems.

Yao's work introduces the concept of protocols that allow two-parties to interactively compute general functions (or binary circuits) securely. This important first step was soon augmented with works by Goldwasser, Micali and Wigderson [165] and Chaum, Damgard and van de Graaf [84] in 1987, for constructing protocols between $N \geq 2$ individuals. These results were developed further by Ben-Or et al. [38] and Chaum et al. [83].

Let $F : (\{0,1\}^*)^N \mapsto (\{0,1\}^*)^N$ be a function that is intended to be computed between N participants. Let $F^{(i)}(x_1, \ldots, x_N)$ denote the i^{th} output of F on some inputs $x_1, \ldots, x_N \in \{0,1\}^*$, for $i \in [N]$. The combination of the above works crystallised the security goals of a protocol computing F as follows.

> *Given participants $(\mathbb{P}_1, \ldots, \mathbb{P}_N)$, construct a protocol ψ_F such that $\mathbb{P}_i$ inputs $x_i \in \{0,1\}^*$, and receives $y_i \in \{0,1\}^*$; for $i \in \{1, \ldots, N\}$. Then ψ_F is secure if, for any subset $S \subset \{1, \ldots, N\}$, then the 'view' of the protocol for any $\mathbb{P}_j$ $(j \in S)$ can be deduced from $\{x_j\}_{j \in [S]}$ and $\{y_j\}_{j \in [S]} = F^{(j)}(\{x_j\}_{j \in [S]})$ alone.*

Intuitively, we ask that the protocol itself does not reveal any extra information, beyond what the known inputs and output reveal. The concept of the *view* is equivalent to all of the information that some participant $\mathbb{P}_j$ witnesses during the execution. The set S refers to participants that

[3] The TLS protocol and end-to-end encrypted messaging applications form the backbone of most communication systems.

are corrupted by some adversary that is looking to subvert the protocol. The capabilities of said adversary help to determine the level of security that is achieved. A weak adversary only has access to the protocol 'transcript' (i.e. the messages that have taken place); such an adversary is known as 'semi-honest', 'honest-but-curious', or 'passive'. A stronger adversary can run arbitrary code and has access to protocol outputs for any input of their choice; this type of adversary is known as 'malicious', or 'active'.

Practical constructions. As the theory of secure computation grew during the 1990s, it became clear that such functionality would have genuine impact in real-world scenarios. For instance, an individual's private data has become a highly valued commodity for many different reasons. This value increases especially, for a variety of different stakeholders, if there is the potential to compute functions over multiple such data sets. In healthcare scenarios, the ability to compute over data allows for much better statistical analysis of the various treatments and diagnoses that are performed. In the world of advertising, computing functions over data associated with an individual's likes and dislikes opens up a world of targeted advertisement that has been realised into a multi-billion dollar industry.[4] As a final example, contact matching in smartphone applications allows users to immediately find acquaintances on new social media platforms. If we can render privacy-preserving variants of these functionalities, then users can be assured that their data is not being used in potentially harmful ways by computing parties.

While privacy-preserving solutions to these problems undoubtedly provide more security for users, the impracticality of such solutions can lead to only sporadic usage in the real-world. Typically, if the computations are very large already, then adding cryptographic overheads can render them unreasonably slow or cumbersome. Early secure computation research focused on creating protocols with little oversight of the overheads that were incurred, typically using slower underlying techniques. As a result, the development of more practically efficient techniques has been vital in devising privacy-preserving tools for real-world computations. In turn, this requires deviating from asymptotic analyses of previous designs and instead considering how to reduce the hidden constants that arise from the methods of computation that are used.

In the recent past, academic literature has met this demand with a growing area of research targeting very fast and high throughput protocols for solving the problem of multi-party computation (to name a few [17, 92, 106, 114, 141, 177, 291]). In fact, there are now Python packages that implement generic multi-party computation operations.[5] Such work helps to illustrate a decreasing trend in the performance overheads associated with using MPC, in comparison with solutions that provide no security guarantees.

[4]In 2016, Facebook alone valued each users data from the United States at $13.54 `https://www.theguardian.com/technology/2016/jan/28/how-much-are-you-worth-to-facebook`.

[5]`https://pypi.org/project/mpyc/`

Specific functionality. A separate line of inquiry within the secure computation framework has been investigating whether protocols for specific and useful functionality can enhance the efficiency of protocols. This has shown to be the case with recent advances into constructing private set intersection protocols [214, 215, 258, 260, 266, 267], and computing over genomic data privately [18]. Such specific protocols have a lot of utility in real scenarios such as those defined above that require private processing of hidden data sets. The hope is that these custom schemes can be embedded into existing workflows for many applications.

I;1.1 Non-interactive computation

The binding philosophy of multi-party computation is that the protocol is computed *online*, in the presence of all of the participants. As computation gets faster and faster, the bottleneck can actually be related to the speed of communication. For instance, the speed of light dictates a natural upper bound on the communication of data between two locations. If those locations are New York and San Francisco, then that data will take $\approx 13ms$ to arrive. Therefore, any protocol that requires communication between two individuals at these locations is going to immediately have a running time that is lower bounded by this time.

To avoid this phenomena, we typically have to replace protocols with *non-interactive* computation. In this setting, an individual renders some computation into a *program*. This program can be distributed to other parties who can then evaluate it as they wish. If the lower bound for communication can be removed then the running time becomes solely dependent on the computational costs. Unfortunately, current multi-party computation techniques are inherently dependent on their interactive nature, in terms of the security that they offer.

Program obfuscation. In 2001, Barak et al. [28] established a new line of research in non-interactive secure computation literature. Their work investigated the possibility for creating programs that *hide* their inner workings, but still allow the programs to be evaluated correctly. Such programs are termed cryptographic *obfuscations*. These programs inherently allow a non-interactive method of securely computing functionalities. The 'obfuscator' can create an obfuscated function with their hidden input embedded. They then send the obfuscated program to users, who can 'evaluate' the program at their own input and learn the corresponding output. The security of the obfuscation states that the evaluator cannot learn anything about the hidden information inside.

Initially, the work of [28] served solely to highlight the impossibility of performing black-box obfuscation for general circuits. By black-box, we mean that the secret information is completely hidden, i.e. the program can be simulated without access to the underlying secrets. However, years later in 2013, Garg et al. [150] showed that the existence of cryptographic multilinear maps [57] may lead to candidate obfuscators in a weaker security model (where obfuscations are only indistinguishable from each other). The utility of IO is not immediately apparent, but circuits satisfying security in this setting can be shown to hide embedded secret information. This initial result prompted the construction of various cryptographic primitives that were previously thought impossible [100, 151, 272].

Unfortunately, constructions of obfuscators have been subject to a painful cryptanalytic process that has seen nearly all current candidates deemed insecure. The current standing candidates are still based on assumptions that are little-understood [2, 13, 224]. Moreover, all constructions are highly inefficient, to the point where implementing any schemes based on these designs is nigh on impossible. As such, these schemes have a long way to go before they can be realised to perform cryptographic tasks in the real-world.

More success has been found recently in achieving obfuscators for very specific functions, both in terms of security reductions and the speed with which they run [43, 70, 294]. Although, there has been less work devising practical scenarios where these constructions are indeed useful.

Constrained primitives. A separate area of non-interactive computation that has seen numerous developments is the design of constrained cryptographic primitives. These primitives are similar to obfuscated programs in that they allow non-interactive evaluation. However they differ in that the 'constrainer' provides a specific key that allows evaluating function on certain types of input, rather than the whole input space.

For instance, puncturable public-key encryption [127, 173, 175] allows the generation of secret keys that be can be used to decrypt all but a set choice of ciphertexts. These ciphertexts are 'punctured' from the secret key and decryption returns $\perp$ on these values. Similarly, constrained pseudorandom functions [52, 55, 58] provide keys that can evaluate subsets of the input space; and puncturable pseudorandom functions [55, 272] provide keys that evaluate the whole space, apart from a set of 'punctured' points.

These primitives fall under the banner of secure computation because they allow computation of certain functions under certain restrictions. In particular, the constrained keys should not reveal the underlying master secret key, and especially cannot be leveraged to gain full access to the scheme. It has been shown that these primitives imply a number of interesting cryptographic primitives that were not previously known (such as 0-round-trip-time and identity-based key exchanges [58, 127, 175]). Moreover, it was shown by [76] that constrained pseudorandom functions imply obfuscation of general circuits, under certain conditions.

I;2 Our contributions

In this thesis we provide a body of work that provides solutions to a variety of different problems within secure computation. Our work demonstrates novel constructions of specific functionalities in both the interactive and non-interactive setting. Additionally, we demonstrate throughout that our constructions can be run for believable real-world scenarios by implementing each of them for widely-used cryptographic parameter sets. Our implementations typically represent the first attempt of practically running the primitives that we are considering. One of our deployments improves anonymity-preserving internet browsing globally, for over 75000 users. Finally, we investigate new avenues for cryptanalysis of program obfuscation.

We now explain each of our contributions, and their motivation, individually.

I;2.1 An Efficient Toolkit for Computing Private Set Operations

Chapter III and Chapter IV are based on the following publications.

- A. Davidson and C. Cid. "An Efficient Toolkit for Computing Private Set Operations". In: *ACISP 17, Part II*. ed. by J. Pieprzyk and S. Suriadi. Vol. 10343. LNCS. Springer, Heidelberg, July 2017, pp. 261–278.
- A. Davidson, G. Fenn, and C. Cid. "A Model for Secure and Mutually Beneficial Software Vulnerability Sharing". In: *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*. WISCS '16. Vienna, Austria: ACM, 2016, pp. 3–14. isbn: 978-1-4503-4565-1. doi: `10.1145/2994539.2994547`.

Chapter III concentrates on new constructions of atomic private set operations, Chapter IV represents an application of these new designs.

Motivation. Set operations represent a particular family of functionalities that are typically computed during data mining operations. If the data mining should be done in a privacy-preserving manner, then this requires protocols that can compute set operations over privately-represented sets. While such computation can be computed using generic protocols for multi-party computation, it has been found that specialised protocols for computing private set operations are typically more efficient [260].

The practicality of such protocols is particularly notable for computing set intersections, as it makes use of largely symmetric-key based primitives for better efficiency. Unfortunately, protocols for computing other operations such as set union, or computing the cardinality of output sets, endure higher overheads due to sub-optimal asymptotic computational complexities. Moreover, far less research has been committed to these operations in general.

Contribution. In Chapter III, we introduce a new technique for computing set union in a privacy-preserving manner between two participants. Establishing private set union protocols allows the computation of a union operation over two sets, without revealing additional information about the intersection of the sets.

Our technique makes use of Bloom filters and homomorphic encryption to construct the first protocol with linear computational complexities in the size of the input sets. We can instantiate our protocol with either somewhat homomorphic encryption, or with additively homomoprhic encryption, with some extra computational overheads. Moreover, we show that our technique can easily be adapted so that it can also compute the set intersection and set cardinality operations.

In summary, we demonstrate a 'toolkit' for performing set union, intersection and cardinality operations — the first of its kind that demonstrates linear computational and communication complexities. Our protocols are proven secure in the semi-honest setting. They can be proven secure in the malicious setting, where at least one input set is authenticated apriori. We give a proof-of-concept implementation of our toolkit in Go, and show that it runs efficiently for widely-used security parameters and set sizes.

In Chapter IV, we further adapt our union protocol to a specific use case based on information sharing. A game-theoretic analysis of [208] showed that information sharing in competitive environments is profitable when there is a trusted third party to negotiate the sharing. We show that an adapted form of our union protocol, that takes into account the *value* of the items being shared, can replace the actions of the mediator. This result enables the competitors to perform a protocol to share their information fairly, rather than locate and outsource their data to a third party.

I;2.2 A Constrained PRF for Bit-fixing Predicates with Constant Collusion-resistance from One-way Functions

Chapter V is based on work co-authored with Shuichi Katsumata, Ryo Nishimaki and Shota Yamada under the name: "Constrained PRFs for Bit-fixing from OWFs with Constant Collusion Resistance". The work is currently under submission and was completed while the thesis author undertook a research internship (Feb-May 2018) at Nippon Telegraph and Telephone Corporation, under the supervision of Ryo Nishimaki. The chpater is based on the pre-print available at [121].

Motivation. Constrained pseudorandom functions (CPRFs) allow users to learn constrained keys that can evaluate the PRF on subsets of the input space (more accurately, where an associated predicate returns true). They have been shown to instantiate a variety of applications including multi-party, identity-based non-interactive key exchange (ID-NIKE), length-optimal broadcast encryption and forms of searchable encryption. Unfortunately, all existing constructions of CPRFs rely on assumptions over multilinear maps (that do not hold for known candidates), or they only allow one constrained key to be learnt. The most prominent applications need to remain secure when multiple constrained keys have been learnt from the CPRF, and thus the utility of these known constructions are reduced significantly.

Contribution. We demonstrate the first construction of a CPRF, for bit-fixing predicates, that allows learning more than one constrained key, from standard assumptions. More precisely, our construction permits $r = O(1)$ keys to be learnt, and is secure under the existence of one-way functions. Prior to our work, the weakest assumptions that were known to imply CPRFs were lattice-based or made in the traditional group setting. Therefore we manage a significant reduction in the strength of assumptions needed to construct CPRFs.

In addition, our construction satisfies 1-key privacy and can be proven secure in the adaptive security model. All previous constructions based on standard assumptions (and within the standard model) can only be proven secure in this model under sub-exponential time reductions.

We show that our CPRF is demonstrably practical for small values of r, via a proof-of-concept implementation of our design in `Go`. This is the first implementation of a dedicated CPRF for general bit-fixing predicates, and previous designs are unlikely to ever reach practical performance due to unfavourable parameter settings. We hope that optimisations of our construction, or implementation, may allow us to increase the value of r. Such results may lead to meaningful, real-world scenarios where we can deploy our CPRF.

I;2.3 Anonymously Authenticating Internet Traffic Using Verifiable Oblivious PRFs

Chapter VI is based on the publication:

- A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda. “Privacy Pass: Bypassing Internet Challenges Anonymously”. In: *PoPETs* 2018.3 (2018), pp. 164–180. doi: `10.1515/popets-2018-0026`. url: `https://doi.org/10.1515/popets-2018-0026`,

that was presented at [PETS18], and won the Andreas Pfitzmann Best Student Paper Award.

Motivation. Content delivery networks (CDNs) are regarded as the gatekeepers to many websites on the internet, particularly those that are visited most frequently. Some of the largest CDNs are Amazon Cloudfront, Akamai, Cloudflare, and Fastly. The *modus operandi* of these corporations is to make the browsing of websites more efficient, by optimising the flow of HTTP requests and responses between clients (the website users) and hosts (the customers of the CDN).

Another reason that CDNs are used is to provide security services to their customers. This includes blocking traffic that appears to be 'malicious', before it reaches the customer's website. As a consequence, some methods of detecting malicious clients lead to more false-negatives (i.e. honest users that are deemed to be malicious) than is truly the case. This, in turn, results in a number of users being blocked from accessing websites; or having to solve challenges that can make browsing difficult and can potentially lead to implementation issues that also block access.

These measures typically affect users of anonymity-preserving tools such as Tor and virtual private networks (VPNs), since various forms of malicious activity is associated with IP addresses used by these tools. Unfortunately, this means that users who must browse anonymously have their accessibility reduced through no fault of their own. Moreover, the anonymity-preserving aspects of these tools means that the use of browser cookies, that can prevent internet challenges from being burdensome, are inadmissible.

Contribution. We devise a browser-based solution to the problem above; allowing users, who are incorrectly deemed to be malicious, to anonymously authenticate with CDN services.

Firstly, we demonstrate a new anonymous authentication protocol, based on a cryptographic protocol for computing a verifiable, oblivious PRF (VOPRF). Our VOPRF design is built upon elliptic curves, similarly to previous constructions such as [194, 195]. Secondly, we design a browser extension (Privacy Pass) and a server that can communicate via HTTP to bypass internet challenges, if a user has been deemed to be honest in the past. The browser extension acquires signed, anonymity-maintaining, tokens from the server when an internet challenge is solved by the client. Such a challenge usually requires some sort of human-interaction, to prevent 'bots' from acquiring such tokens. These tokens are then redeemed in the future when a challenge is required.

We embed the VOPRF server structure into Cloudflare's access mechanism and globally distributed the browser extension for use with Chrome and Firefox (and subsequently Tor browser). We show that the number of internet challenges significantly decreases when clients use Privacy Pass for browsing Cloudflare websites. To date, over 75000 users have downloaded the extension and it continues to make accessing Cloudflare websites an easier experience for honest users. We also show that installing Privacy Pass adds minimal overheads to the browsing experience.

In the future, we hope to integrate Privacy Pass with more providers and explore different use-cases altogether for our VOPRF protocol. We are also working on an IETF draft for standardising our VOPRF protocol.[6]

[6] https://datatracker.ietf.org/doc/draft-sullivan-cfrg-voprf/

I;2.4 A Security Analysis of the GGH13 Graded Encoding Scheme Without Ideals

Chapter VII is based on the publication:

- M. R. Albrecht, A. Davidson, and E. Larraia. "Notes on GGH13 Without the Presence of Ideals". In: *16th IMA International Conference on Cryptography and Coding*. Ed. by M. O'Neill. Vol. 10655. LNCS. Springer, Heidelberg, Dec. 2017, pp. 135–158

This publication appeared at [IMACC2017]. An updated version of the publication is available [9], with Alice Pellet-Mary as an additional co-author.

Motivation. The GGH13 candidate multilinear map (or graded encoding scheme) of Garg, Gentry and Halevi [148] was the first to be used to construct candidate program obfuscation for all circuits [150]. This obfuscator is secure in the indistinguishability paradigm, in an ideal graded encoding model — where all interactions are replaced with oracle queries, and all encodings are uniformly random. Since this initial work, there have been a wide range of obfuscators that are also secure under these conditions [12, 16, 23, 27, 64, 153, 239, 253, 298].

The work of Miles, Sahai and Zhandry [240], along with a variety of subsequent works [15, 87, 90, 256], have shown that security does not hold for some obfuscators, when the ideal model is replaced with a candidate graded encoding scheme. In particular, for the case of GGH13, the program evaluator can reveal hidden information about a principal ideal that is common to all encodings of the value 0.

Once this ideal is revealed, the evaluator can construct evaluations that either belong to the ideal or not, depending on the program that is obfuscated in the indistinguishability game. Consequently, the evaluator can win the security game with non-negligible probability — albeit via a heuristic cryptanalysis.

Contribution. We investigate the possibility of modifying the GGH13 graded encoding scheme to remove the presence of common elements that cause the vulnerability that we mentioned above. We successfully demonstrate a correct graded encoding scheme resembling the GGH13 scheme, except that there are no parameters that are common to all encodings. If we use our candidate scheme to instantiate previously insecure obfuscators, then we immediately avoid all known cryptanalyses.

In contrast, we show instead that a new range of cryptanalytic techniques can be used to algebraically manipulate our modified encodings to break these obfuscators in the indistinguishability game. That is, our modifications introduce some changes to the zero-testing procedure, but these appear to still be exploitable in a different manner (though still via a heuristic argument).

We interpret these results as an indication that the GGH13 scheme is inherently vulnerable in its underlying structure, rather than just via the presence of the ideal that is used by previous attacks. We offer a counterpoint to a conjecture of Halevi [176] by stating that the hardness of finding this ideal is a measure of the security of an application that uses GGH13. We hope that further research may indicate exactly how this vulnerability manifests itself, as a method of devising graded encoding schemes with more knowledge of the vulnerabilities that should be avoided.

We demonstrate our attacks in a new realisation of the indistinguishability game that reduces the number of oracles that are required for working with. This helps to make the notation of the ideal graded encoding model more compact, and illuminates the attack techniques in a clearer manner. We show that our new model is sufficient for demonstrating vulnerabilities by showing that the previous attacks also fit into it.

Finally, we notice that the full extent of attacks against our scheme are slightly more limited than against GGH13. We are unable to demonstrate attacks against some branching program representations of obfuscated circuits, using our modified scheme, that previous attacks are able to exploit using GGH13. In turn, we are unable to transfer this into any meaningful security assumption, and so it is unlikely that this will result in a secure obfuscation candidate. However, we note that this may be a reasonable line of inquiry for future research.

I;3 Other research

We have chosen to omit an additional publication that the thesis author co-authored:

- M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. W. Postlethwaite, F. Virdia, and T. Wunderer. "Estimate All the {LWE, NTRU} Schemes!" In: *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*. 2018, pp. 351–367. DOI: `10.1007/978-3-319-98113-0\_19`. URL: `https://doi.org/10.1007/978-3-319-98113-0%5C_19`.

This was because the publication in question does not fit into the remit of secure computation. However, we will provide a short description of this work here.

Motivation. The National Institute for Standards and Technology (NIST) launched a competition (in December 2016) for standardising algorithms for use in the *post-quantum* setting. In this setting, quantum computation is possible and thus polynomial-time cryptanalysis of various cryptographic assumptions is feasible. For example, using Shor's algorithm [278], the factoring and discrete log assumptions can be solved in polynomial-time.

A large portion of post-quantum cryptography is based on a set of assumptions that are derived from hard problems over high-dimension lattices. These include the learning with errors (LWE) problem [264] and the NTRU problem [186]. For the NIST competition, developers of cryptosystems based on these assumptions have to provide parametrisations that demonstrate security for a variety of different situations.

Unfortunately, the literature on state-of-the-art cryptanalysis of these assumptions is not exactly unified. That is, there are various algorithms and methods for solving underlying problems such as the shortest vector problem, that in turn can be used to find solutions to LWE and NTRU. The 'cost models' used to estimate the hardness of solving these problems in turn determines the size of the parameters that the schemes can tolerate. If a cost model is unnecessarily pessimistic, then this may lead to schemes that are needlessly inefficient. If a cost model is too optimistic, then this may lead to a scheme claiming greater security than is actually the case.

Contribution. In this work, we extract the parameters of each of the proposals of cryptosystems to the NIST competition based on variants of LWE or NTRU. We also extract all the cost models that are used that for estimating the hardness of breaking the given assumptions. We then conduct a unified analysis that calculates the security of each scheme, against each cost model that is used. The estimation of the security of these schemes is carried out using the *LWE estimator* of [11].

Our results help to provide a more accurate summary and comparison of the hardness of breaking each scheme. We leave interpretation of these findings up to future research. Instead, we intend that our findings can be used as a useful point-of-reference for adequately parametrising lattice-based assumptions.

I;4 Layout

In Chapter II and Chapter IV we will provide the necessary cryptographic background for our work. Chapter III will focus on our work regarding new protocols for performing private set operations. Chapter V will demonstrate our new construction of a constrained pseudorandom function. Chapter VI will describe our research and deployment of new methods for browser-based anonymous authentication. Chapter VII will describe our cryptanalysis of our modification to the GGH13 graded encoding scheme. Finally, we will conclude this thesis in Chapter VIII.

II Cryptographic preliminaries

too late

Contents

In this section, we will cover a large portion of the preliminary material that is necessary for consuming this thesis. Some materials that are specific to a single chapter are deferred until the chapter itself.

- We provide a list of acronyms that we commonly use throughout on page 354.
- We provide a summary of the notation used here in Table A, on page 352.

II;1 Standard notation

Sampling. Let S be a set, and let $|S|$ denote its cardinality. If we say that X is a uniformly distributed random variable over S, then we mean that $\Pr[X = x] = 1/|S|$, for all $x \in S$. If Y is said to be the uniform distribution over S, then the result of sampling elements from Y is a uniformly distributed, random variable. We may side-step the actual description of distributions in the uniform case, and write $s \leftarrow_{\$} S$ to indicate that s is sampled via the uniform distribution.

Binary strings and sets. For $n \in \mathbb{N}$, we write $[n]$ to represent the set $\{1, \dots, n\}$; we also write $[n_1, n_2]$ to denote the set $\{n_1, \dots, n_2\}$ for $n_1 < n_2$ and $n_1, n_2 \in \mathbb{N}$. For a string $x = x_1 \dots x_\ell \in \{0,1\}^\ell$, we let $x|_t = x_t \dots x_\ell$, $x|^t = x_1 \dots x_t$, and $x|_{t_1}^{t_2} = x_{t_1} \dots x_{t_2}$, for $t_2 \geq t_1$. Alternatively, let $T \subset [\ell]$ be some subset of indices, we write $x|_T$ to denote the bits $x_i \in \{0,1\}$ such that $i \in T$. We write $x\|y$ to denote the concatenation of binary strings x, y. Let $\mathsf{bitlength}$ be a deterministic algorithm, where $\mathsf{bitlength}(x) = \ell$ for $x \in \{0,1\}^\ell$. We say that $x = \emptyset$ is the 'empty' string.

Unless we state otherwise, we assume that all sets are *ordered*, that is a set $S = \{x_i\}_{i \in [\ell]}$ where the ordering of the x_i is determined via context.[1] For an ordered set S, we define the method $S.\mathsf{pop}()$ to return the element positioned at the first index of the set before it is deleted; then $S[i-1]$ is set to be equal to $S[i]$ — i.e. each element is moved down a position in S (towards the first index). We also define a method $S' \leftarrow S.\mathsf{append}(x)$, where S' is the concatenation of S and $\{x\}$, we can extend this notation to append entire sets $\overline{S}$ also. We will write $W \leftarrow \emptyset$ to indicate that W has been initialised as an 'empty' set.

Let L be some domain, and let $\{x_i\}_{i \in [n]}$ be the set of values where $x_i \in L$, for each $i \in [n]$. Then we may write $\{x_i\}_{i \in [n]} \in L$.

[1]Ordered sets are can additionally be known as vectors or lists.

Equality. We will assume the presence of an equality operator $\overset{?}{=}$ for two objects X and Y of equivalent types, i.e. $X, Y \in \{0,1\}^\ell$. This operator converts the data type into binary form and compares the bits of the objects sequentially, where:

$$1 \leftarrow (X \overset{?}{=} Y) \implies (X_i == Y_i) \, \forall \, i \in [\ell],$$

and

$$0 \leftarrow (X \overset{?}{=} Y) \implies \exists \, i \in [\ell] \text{ s.t. } (X_i \neq Y_i).$$

In this work, we may abuse notation and assume that $(X \overset{?}{=} Y)$ outputs 'true' rather than 1, and 'false' rather than 0.

When writing probability statements, we may write $\Pr\left[X \overset{?}{=} Y\right]$ to be the probability that $X \overset{?}{=} Y$ returns 'true'. Likewise, we will write $\Pr\left[\neg(X \overset{?}{=} Y)\right]$ to be the probability that $X \overset{?}{=} Y$ returns 'false'. This notation is also used in conditional **if** statements, during algorithmic execution.

Algorithms and functions. Let $\mathcal{D}$ be an algorithm. Then we write $a \leftarrow \mathcal{D}(x)$ to indicate that the algorithm outputs a on some input x. If the algorithm is probabilistic then we may write $a \leftarrow\!\!\$\, \mathcal{D}(x)$, or equivalently $a \leftarrow \mathcal{D}(x; r)$ where r are the explicit random coins that are used in running the algorithm. If $b \leftarrow \mathcal{D}$ is output, where $b \in \{0,1\}$, then we may take the returned value to be a boolean expression (i.e. true when $b = 1$, and false otherwise). We write $!\mathcal{D}(x)$ to indicate that the algorithm $\mathcal{D}(x)$ has never been run on input x.

In this work, we say that an algorithm $\mathcal{D}$ is *efficient* if it can be run in polynomial-time, i.e. the running time is bounded by a polynomial in the size of the input. We use the abbreviation PPT to refer to probabilistic polynomial time. For a function $f : \mathcal{X} \mapsto \mathcal{Y}$, we say that it is *efficiently computable* if there exists a deterministic poly-time algorithm $\mathcal{D}$, s.t. for all inputs $x \in \mathcal{X}$, then $f(x) \leftarrow \mathcal{D}(x)$.

We commonly use $\mathsf{negl}(\lambda)$ to denote a *negligible function*. Such a function satisfies the following: for all possible polynomials $\mathsf{poly}(\lambda)$ there exists a parameter λ_0 s.t.

$$\mathsf{negl}(\lambda) < 1/\mathsf{poly}(\lambda)$$

for all $\lambda \geq \lambda_0$. We may omit the mention of the parameter λ_0 in future, since this requirement is implicitly required in all inequalities of this form.

Circuits and predicates. Throughout this work, we may refer to functionalities through their expression as a binary circuit. A binary circuits consists of: ℓ_{in} input wires, ℓ_{out} output wires; n binary logic gates; and depth d. We assume that all gates consist of one or two-input wires (fan-in-one/two) and one output wire. The *size* of the circuit is the total number of gates, the *depth* is the maximum number of gates along a path from an input wire to an output wire. Inputs to the circuit are $x \in \{0,1\}^{\ell_{\mathsf{in}}}$ where each bit of the input is assigned to one of the input wires. The outputs of the circuit are written as $y \in \{0,1\}^{\ell_{\mathsf{out}}}$, where the output bitstring takes each bit from the ℓ_{out} output wires. We may write $C(x) = y$ or $y \leftarrow C(x)$ to associate y with the output of $C(x)$.

We primarily consider circuits taken from the complexity classes $\mathcal{C}_{\mathsf{NC}^1}$ and $\mathcal{C}_{\mathsf{P/poly}}$. Circuits that are part of $\mathcal{C}_{\mathsf{NC}^1}$ have depth that is $O(\log(\ell_{\mathsf{in}}))$. Circuits that are in $\mathcal{C}_{\mathsf{P/poly}}$ have depth $O(\mathsf{poly}(\ell_{\mathsf{in}}))$. We additionally assume that the size of all circuits is $\mathsf{poly}(\ell_{\mathsf{in}})$.

Predicates for a circuit class $\mathcal{C}$ are functionalities of the form $\mathsf{P}_C : \{0,1\}^{\ell_{\mathsf{in}}} \mapsto \{0,1\}$, for some circuit $C \in \mathcal{C}$. We say that $x \in \{0,1\}^{\ell_{\mathsf{in}}}$ satisfies the predicate iff $1 \leftarrow \mathsf{P}_C(x)$; which occurs iff $1 \leftarrow C(x)$.

Provable security

The requirement for establishing provable security of cryptosystems has arisen with the large number of potential constructions that have been developed over the last few decades. Indeed, this increase has rendered it implausible that cryptanalysts can treat every cryptosystem with the rigour that is needed to find attacks and breaks.

A *proof of security* serves to draw together seemingly disparate constructions behind the banner of a single computational assumption. Such a proof then allows cryptanalysts to target the instances of the assumption, rather than individual schemes. As a second advantage, such proofs help to segment the cryptographic landscape into clearer divides — highlighting the power and expressiveness of certain assumptions.

To write a proof of security, one needs to demonstrate a security *reduction* from the computational assumption to the scheme in question. Intuitively, we have a PPT algorithm $\mathcal{B}$ known as an adversary, who is attempting to break the computational assumption ex. Let us assume, for the sake of argument, that this computational assumption is rendered as a pair of decisional experiments, where $\mathcal{B}$ is given a distribution D_b and must output $b \in \{0,1\}$. Now, let $\mathcal{A}$ be a PPT adversary that is attempting to demonstrate that a scheme sch does not satisfy the security property prop. This security property is described by decisional experiments defined by the distributions A_c for $c \in \{0,1\}$.[2]

We say that 'breaking' prop for sch is at least as hard as breaking ex if $\mathcal{B}$ can run $\mathcal{A}$ as a subroutine resulting in a valid solution to ex, with advantage ϵ_{ex}. By advantage, we mean the probability that $b \leftarrow \mathcal{B}$ given that $D_b \rightarrow \mathcal{B}$, minus the probability that $\mathcal{B}$ outputs b when it receives D_{1-b}. In the security proof, $\mathcal{B}$ uses the output $c_{\mathcal{A}} \leftarrow \mathcal{A}$ to construct its own output $b_{\mathcal{B}} \leftarrow \mathcal{B}$. If $b_{\mathcal{B}} \overset{?}{=} b$ with non-negligible probability then $\mathcal{B}$ breaks ex.

[2]We give formalisations of the type of experiments that we consider shortly.

The original assumption, ex, holds iff $\mathcal{B}$ has negligible probability in distinguishing the distributions D_b for $b \leftarrow_\$ \{0,1\}$. Consequently, then $\mathcal{A}$ must have negligible advantage

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{prop}(1^\lambda, \mathsf{sch}))) < \epsilon_{\mathsf{ex}} < \mathsf{negl}(\lambda)$$

in succeeding in distinguishing A_c, for $c \leftarrow_\$ \{0,1\}$, by the terms of the reduction. Otherwise, there is a contradiction wrt the original statement of the assumption ex.

We will write security proofs of this form, when we are required to prove the security of a given construction. We now formalise the ideas behind writing such security proofs, including the experiments we used above, and the capabilities of adversaries in such settings.

Oracles. We write $\mathcal{A}^{\mathcal{O}_{\mathcal{Y}}(f(\cdot))}$ to indicate that a PPT algorithm $\mathcal{A}$ has 'oracle access' to the function f with domain $\mathcal{Y}$. During this access, $\mathcal{A}$ can submit queries $y \in \mathcal{Y}$ to a challenger who returns $f(y)$. If the challenger keeps track of the queries to the oracle using a set φ, then we write $\mathcal{A}^{\mathcal{O}_{\mathcal{Y}}(f(\cdot),\varphi)}$ where for every query $y \in \mathcal{Y}$, then $y \to \varphi$ and φ is known throughout the game. We may also write $\mathcal{O}^{\varphi}_{f(\cdot)}(\mathcal{Y})$ as short-hand, unless stated otherwise. If the algorithm is only permitted to make $m \in \mathbb{Z}$ calls to the oracle, we will write $\mathcal{A}^{\mathcal{O}_{\mathcal{Y}}(f(\cdot);[m])}$.

Security experiments. Let sch be some scheme and let prop be some security property that sch can satisfy. Let λ be a security parameter and, for $b \in \{0,1\}$, denote by $\mathsf{exp}^{\mathsf{prop}}_{b,\mathcal{A}}(1^\lambda, \mathsf{sch})$ a pair of *decisional* experiments. The use of $\mathcal{A}$ is to define an algorithm (known as the *adversary*) that interacts with sch in $\mathsf{exp}^{\mathsf{prop}}_{b,\mathcal{A}}(1^\lambda, \mathsf{sch})$ and attempts to distinguish the cases where $b = 0$ and $b = 1$. We define decisional experiments such that $b_{\mathcal{A}} \leftarrow \mathsf{exp}^{\mathsf{prop}}_{b,\mathcal{A}}(1^\lambda, \mathsf{sch})$ is the final output, where $b_{\mathcal{A}}$ is the 'guess' of $\mathcal{A}$. We adopt the convention that this guess is returned in the final step of the game. Let

$$\mathsf{Adv}(\mathcal{A}, 1^\lambda, \mathsf{prop}(1^\lambda, \mathsf{sch})) = \left|\Pr\left[0 \leftarrow \mathsf{exp}^{\mathsf{prop}}_{0,\mathcal{A}}(1^\lambda, \mathsf{sch})\right] - \Pr\left[0 \leftarrow \mathsf{exp}^{\mathsf{prop}}_{1,\mathcal{A}}(1^\lambda, \mathsf{sch})\right]\right|$$

denote the *advantage* of the adversary $\mathcal{A}$ in distinguishing the two experiments for sch. We say that $\mathsf{exp}^{\mathsf{prop}}_{0,\mathcal{A}}(1^\lambda, \mathsf{sch})$ and $\mathsf{exp}^{\mathsf{prop}}_{1,\mathcal{A}}(1^\lambda, \mathsf{sch})$ are *computationally indistinguishable* if

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{prop}(1^\lambda, \mathsf{sch}))) < \mathsf{negl}(\lambda) \tag{II;1}$$

for some negligible function negl where the maximum is taken over all PPT algorithms $\mathcal{A}$.[3] We may also write $\mathsf{exp}^{\mathsf{prop}}_{0,\mathcal{A}}(1^\lambda, \mathsf{sch}) \approx_c \mathsf{exp}^{\mathsf{prop}}_{1,\mathcal{A}}(1^\lambda, \mathsf{sch})$ to signify computational indistinguishability. We say that they are *statistically indistinguishable* if Equation (II;1) holds for all $\mathcal{A}$, regardless

[3] We sometimes omit explicit mention of the security parameter if the context is obvious.

```
$\mathsf{exp}^{\mathsf{prop}}_{b,\mathcal{A}}(1^\lambda, \mathsf{sch})$
1 :  $\mathsf{pp}, \mathsf{sp} \leftarrow_\$ \mathsf{Setup}(1^\lambda)$;
2 :  $m_0, m_1 \leftarrow \mathcal{A}(1^\lambda)$;
3 :  $c_b \leftarrow f(\mathsf{pp}, \mathsf{sp}, m_b)$;
4 :  $b_\mathcal{A} \leftarrow \mathcal{A}^{\mathcal{O}^{\varphi}_{\mathcal{X}}(f(\mathsf{pp},\mathsf{sp},\cdot))}(1^\lambda, \mathsf{pp}, c_b)$;
5 :  **if** $(m_0 \in \varphi) \vee (m_1 \in \varphi)$ :
6 :    **return** $\perp$;
7 :  **return** $b_\mathcal{A}$;
```

Figure II;1: Example decisional experiment.

of running time. Similarly, *perfectly indistinguishable* if the advantage is equal to 0 for unbounded adversaries. Generally, for two distributions (Z, Y), we write:

$$\{Z \approx_c Y,\ Z \approx_s Y,\ Z \approx_p Y\}$$

if Z and Y are {computationally, statistically, perfectly} indistinguishable.

We provide an example of the notation that we use in decisional experiments in Figure II;1. During the experiment, the 'challenger' computes some function f over one of the adversarial outputs m_b and returns the output to $\mathcal{A}$. In this example, f depends on secret parameters sp that are never given to the adversary, though $\mathcal{A}$ has knowledge of the circuitry of f.

The notation in line 5 indicates that the adversary also has oracle access to the function f (though it cannot ask queries for m_b). The final returned output is the bit $b_\mathcal{A}$ that the adversary computes.

In lines 6 and 7, we prevent trivial wins by preventing the adversary from using either m_b as input to the oracle that they have access to. If we allowed such queries, then $\mathcal{A}$ could win the game trivially.

We can also define *computational* experiments $\mathsf{exp}^{\mathsf{prop}}_{\mathcal{A}}(1^\lambda, \mathsf{sch})$ for determining whether sch satisfies prop. In this case, $\mathcal{A}$ outputs a computational guess z; succeeding if the guess is deemed to be correct by the challenger. We define $b_\mathcal{A}$ to be the final output of the experiment depending on the validity of the adversarial guess, i.e. 1 if correct and 0 if not. The advantage of the adversary is redefined as:

$$\mathsf{Adv}(\mathcal{A}, 1^\lambda, \mathsf{prop}(1^\lambda, \mathsf{sch})) = \left|\Pr\left[b_\mathcal{A} = 1 \,\middle|\, b_\mathcal{A} \leftarrow \mathsf{exp}^{\mathsf{prop}}_{\mathcal{A}}(1^\lambda, \mathsf{sch})\right]\right|$$

and we say that $\mathsf{exp}^{\mathsf{prop}}_{\mathcal{A}}(1^\lambda, \mathsf{sch})$ is *computationally unsolvable*, if

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{prop}(1^\lambda, \mathsf{sch}))) < \mathsf{negl}(\lambda)$$

```
$\mathsf{exp}^{\mathsf{prop}}_{\mathcal{A}}(1^\lambda, \mathsf{sch})$
1 :  $\mathsf{pp}, \mathsf{sp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$;
2 :  $x \leftarrow\!\!\$\ \mathcal{X}$;
3 :  $y \leftarrow\!\!\$\ f(\mathsf{pp}, \mathsf{sp}, x)$;
4 :  $x_{\mathcal{A}} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathcal{X}}(f(\mathsf{pp},\mathsf{sp},\cdot))}(1^\lambda, \mathsf{pp}, y)$;
5 :  **return** $b_{\mathcal{A}} = (x \overset{?}{=} x_{\mathcal{A}})$;
```

Figure II;2: Example computational experiment.

for all PPT adversaries. Again, we say *statistically unsolvable* if the same bound holds for all $\mathcal{A}$, regardless of running time. Similarly, *perfectly unsolvable* if the advantage is equal to 0 for unbounded adversaries.

We provide an example of the notation that we use in computational experiments in Figure II;2. The difference with decisional experiments is that the adversary now guesses a value (e.g. in this example, returning the pre-image of the function f). The experiment outputs 1 if $\mathcal{A}$ is correct and 0 otherwise — this is what the expression in the final line refers to. Note that there are no plausible trivial winning conditions for $\mathcal{A}$ in this example, and so we do not have to prevent any queries as we did in Figure II;1.

Remark II;1.1. *We may abuse notation and omit explicit mention of* sch *if the choice of* sch *is obvious from the choice of* prop *(or, alternatively, if* prop *refers to some hard mathematical problem without an associated instantiation). In the context of writing advantages, this would be:*

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{prop}(1^\lambda))) < \mathsf{negl}(\lambda),$$

instead of:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{prop}(1^\lambda, \mathsf{sch}))) < \mathsf{negl}(\lambda),$$

and in the context of experiments:

$$\mathsf{exp}^{\mathsf{prop}}_{b,\mathcal{A}}(1^\lambda),$$

instead of:

$$\mathsf{exp}^{\mathsf{prop}}_{b,\mathcal{A}}(1^\lambda, \mathsf{sch}).$$

Hybrid arguments. Hybrid arguments are vital techniques required in the writing of security proofs [36, 126, 279]. They help to transition states from those where security cannot be inferred to states that can be shown to have some concrete security guarantees. Building up these transitions allows us to prove security from a number of different invocations of assumptions.

A hybrid argument is structured as a set of intermediate 'games', denoted by $\mathsf{H}_0, \ldots, \mathsf{H}_\ell$. Broadly, we assume that in each H_i, the adversary is interacting with an experiment $\mathsf{exp}^{\mathsf{i}}_{0,\mathcal{A}}(1^\lambda, \mathsf{sch})$, where $\mathsf{exp}^{0}_{0,\mathcal{A}}(1^\lambda, \mathsf{sch}) \approx_p \mathsf{exp}^{\mathsf{prop}}_{0,\mathcal{A}}(1^\lambda, \mathsf{sch})$, and $\mathsf{exp}^{\ell}_{0,\mathcal{A}}(1^\lambda, \mathsf{sch}) \approx_p \mathsf{exp}^{\mathsf{prop}}_{1,\mathcal{A}}(1^\lambda, \mathsf{sch})$. By showing that $\mathsf{exp}^{\mathsf{i}}_{0,\mathcal{A}}(1^\lambda, \mathsf{sch})$ and $\mathsf{exp}^{\mathsf{i+1}}_{0,\mathcal{A}}(1^\lambda, \mathsf{sch})$ are computationally indistinguishable, then we can gradually show that $\mathsf{exp}^{\mathsf{prop}}_{0,\mathcal{A}}(1^\lambda, \mathsf{sch}) \approx_c \mathsf{exp}^{\mathsf{prop}}_{1,\mathcal{A}}(1^\lambda, \mathsf{sch})$, and thus prove security.

Let H_i and H_{i+1} denote consecutive games within a hybrid argument. Define $\mathsf{exp}^{\mathsf{H_i},\mathsf{H_{i+1}}}_{b,\mathcal{D}}(1^\lambda)$ to be a decisional experiment, where the PPT algorithm $\mathcal{D}$ attempts to distinguish between H_i and H_{i+1}. In this situation, $b_\mathcal{D} \leftarrow \mathcal{D}$ is such that $b_\mathcal{D} = 0$ if $\mathcal{D}$ guesses H_i, and $b_\mathcal{D} = 1$ if $\mathcal{D}$ guesses H_{i+1}. We define the advantage of an adversary $\mathcal{A}$ in distinguishing two steps H_i and H_{i+1} by:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{H}_{i,i+1}(1^\lambda, \mathsf{sch}))),$$

and establish the same necessity for bounding the advantage by $\mathsf{negl}(\lambda)$ for the purpose of proving security. Again, we may omit the explicit mention of sch if it is obvious from context.

Protocols. Throughout this work, we will make use of a particular communication structure known as a *protocol*. We will only consider protocols that are carried out between two participants. During a protocol, there are *steps* that are carried out by either one or both of the participants. We provide an example of such a protocol step in Figure II;3. The notation is similar to the cryptographic notation that we described above.

In this example, the communicating parties are C and S. Firstly, $\psi^1(\mathsf{C}, b, x, y, f)$ is known as the 'header'. ψ^1 signifies that this is the first step in the protocol ψ; ψ^2 would signify the second, and so on. The tuple (C, b, x, y, f) is the set of inputs. By convention, the first input is the participant C, this implies that the computation is carried out by C. If the protocol step involved both participants, then we would also include S (see Figure IV;7 for an example). The remaining inputs are b, x, y, f. We include f as input even though it is strictly a function. We may sometimes write $\mathsf{C}[x]$ if C participates in the protocol with input x.

Lines 3 and 6 are noteworthy as they denotes information that is sent to S. Depending on which step is run, this indicates that S receives either $f(x)$ or $f(y)$ as input — potentially to use in a future step. Moreover, line 8 indicates a return statement. This means that C returns the contents of the set φ to itself, to use in the future.

An entire protocol is then composed up of multiple steps, resulting in both parties receiving some sort of output from the computation. We typically refer to the participation of an entity within the protocol as its 'view'. We will describe this concept more formally in Section II;6.

We may abuse notation when referring to the what is returned by the protocol. Considering the protocol step $\psi^1(\mathsf{C}, b, x, y, f)$ in Figure II;3, then we may write

$$(\mathsf{C} : \varphi, \mathsf{S} : z) \leftarrow \psi^1(\mathsf{C}, b, x, y, f),$$

```
$\psi^1(\mathsf{C}, b, x, y, f)$
1 :  $\varphi \leftarrow \emptyset$;
2 :  if $b \overset{?}{=} 1$ :
3 :      $\mathsf{S} \leftarrow f(x)$;
4 :      $\varphi \leftarrow x$;
5 :  else :
6 :      $\mathsf{S} \leftarrow f(y)$;
7 :      $\varphi \leftarrow y$;
8 :  return $\varphi$;
```

Figure II;3: An example protocol step.

where $z \in \{f(x), f(y)\}$. That is, φ is returned but only to C, whereas z is sent to S. We use this notation to highlight what the outputs of that given step are, because both outputs may be used in subsequent protocol steps.

Concrete security parameters. Throughout this chapter we will make use of the security parameter λ as a crucial indicator of how secure a given parametrisation of a cryptosystem is. Moreover, all algorithms run in polynomial time corresponding to their dependency on λ. It is then crucial to discuss potential settings of λ that give adequate security in a variety of situations. When choosing concrete parameters to establish the security of the scheme, the *tightness* of the reduction in the security proof needs to be considered.

For example, suppose that we prove

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{prop}(1^\lambda, \mathsf{sch}))) < \max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{B}, \mathsf{ex}(1^\lambda)))$$

for a scheme sch, security property prop, and some known hard problem ex. Then, this suggests that we can take a concrete setting of λ that ensures security in the case sch. On the other hand, consider the case where security is only guaranteed when

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{prop}(1^\lambda, \mathsf{sch}))) < \mathsf{poly}(\lambda) \cdot \max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{B}, \mathsf{ex}(1^\lambda)))$$

for some polynomial $\mathsf{poly}(\lambda)$. Then we would need to pick a much larger security parameter to ensure that security holds. This distinction should be reflected by the concrete choice of λ when attempting to parametrise the schemes.

When establishing a concrete choice, we discuss the number of bits of security to be the value of $\lambda \in \mathbb{N}$ s.t. $\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{prop}(1^\lambda, \mathsf{sch}))) < 1/2^{\lambda^*}$ for some value λ^*. In this work, we will typically use the guidelines of the National Institute of Standards and Technology for choosing the bitlength of λ^*; longer bitlengths clearly imply greater security [247]. The minimum setting of $\mathsf{bitlength}(\lambda^*)$ that we will consider is 80 bits, we will use 112 bits of security for maintaining the security of applications in the period $2016 - 2030$, and we will use 128 bits as providing security for the present case and beyond. We regard $\lambda^* > 128$ bits as large enough to guarantee security for a significant period of time (potentially into the next century). Based on [247], we assume that for factoring problems (i.e. DCR, Assumption II;3.3) we require 2048 bit moduli to achieve 112 bits of security. We require 256 bit moduli to achieve 128 bit security in the case of assumptions made over elliptic curves (e.g. DL, Assumption II;3.1).

II;2 Mathematical preliminaries

We use $\{\mathbb{R}, \mathbb{N}, \mathbb{Q}, \mathbb{C}\}$ to refer to the {real, natural, rational, complex} numbers. For a complex number $c \in \mathbb{C}$, we write $\Re(c)$ to denote the real component of c, and $\Im(c)$ to denote the imaginary component.

Cyclic groups. A cyclic group is a commutative, finite group generated by a single element. Let $\mathbb{G}(\lambda)$ be a cyclic group of order $p(\lambda)$, where $\mathsf{bitlength}(p) = \mathsf{poly}(\lambda)$ and let $g \in \mathbb{G}(\lambda)$ be a generator. Then we can write $\mathbb{G}(\lambda)$ as $\{g^0, g^1, g^2, \ldots, g^{p-1}\}$, where g^0 is the identity element. We write $\mathsf{ord}(g)$ to denote the order of g. We write $\mathbb{G}(\lambda)$ if the order of the group is not chosen explicitly. If $\mathsf{bitlength}(p) = \mathsf{poly}(\lambda)$ is explicit, then we may write $\mathbb{G}(\lambda)$.

Instantiations of cyclic groups. One of the most common ways of instantiating a cyclic group $\mathbb{G}$ is as a multiplicative subgroup of a finite field. Let p be some prime integer (usually *large*), and s.t. $p = 2p' + 1$ for some other prime p'. Then the subgroup of squares $\mathbb{QR}_p^*$ (elements of the form u^2 for $u \in \mathbb{F}_p$) is a cyclic group of order p'. To pick a generator of $\mathbb{G} = \mathbb{QR}_p^*$ we pick $u \in \mathbb{F}_p$ (resampling if necessary until $u \notin \{0, 1\}$) and let $g = u^2$.

A different method of instantiating $\mathbb{G}$ is to use *elliptic curves*. We consider elliptic curves in Montgomery form: algebraic curves of the form $y = x^3 + ax + b$ for $a, b \in \mathbb{Z}$, on which an additive group operation may be defined. These curves have undergone a very rich study and stand as foundations of a large portion of cryptographic literature and constructions.

Vectors and matrices. We denote matrices by capitalized bold-face (i.e. $\boldsymbol{A}$); we denote vectors in lower-case bold-face (i.e. $\boldsymbol{v}$). We denote the $(i, j)^{\text{th}}$ entry of $\boldsymbol{A}$ by a_{ij}; and write s_i to denote the i^{th} component of the vector $\boldsymbol{s}$. We write $[\boldsymbol{A}_1 \| \boldsymbol{A}_2]$ to denote the horizontal concatenation of matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$; and write $[\boldsymbol{A}_1 \|_{\uparrow} \boldsymbol{A}_2]$ to denote the vertical concatenation (sometimes known as stacking). For a matrix $\boldsymbol{A}$, we write $\boldsymbol{A}^T$ to denote the transpose.

We use the infinity norm $\|\boldsymbol{v}\|_\infty$ of a vector $\boldsymbol{v}$ (resp. $\|\boldsymbol{A}\|_\infty$ for a matrix $\boldsymbol{A}$) to denote the overall *magnitude* of $\boldsymbol{v}$ (resp. $\boldsymbol{A}$). Where appropriate, we will also use the ℓ_2-norm, denoted by $\|\boldsymbol{v}\|_2$. We assume that vectors are written in column notation, unless specified otherwise.

Schwarz-Zippel lemma. The Schwarz-Zippel lemma is an important result that essentially states that for any non-degenerate, n-variate function f; then an evaluation over n random values is unlikely to be 0 whp.

Lemma II;2.1 [Schwarz-Zippel lemma (generalised)]

Let $\mathbb{F}$ be a field. Let f be an n-variate polynomial of degree d where $f \not\equiv 0$. Let S be a finite subset of $\mathbb{F}$, and let $r_1, \ldots, r_n \leftarrow_\$ S$ be sampled uniformly from S. Then:

$$\Pr[f(r_1, \ldots, r_n) = 0] \leq d/|S|$$

II;2.1 Lattices

We give an overview of some key concepts related to the usage of lattice-based cryptography.

An n-dimensional lattice Λ is a discrete, additive subgroup of $\mathbb{R}^n$. Given n linearly independent basis vectors $\boldsymbol{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\} \in \mathbb{R}^{n \times n}$, the lattice generated by $\boldsymbol{B}$ is

$$\Lambda(\boldsymbol{B}) = \{\boldsymbol{v}_i \mid \boldsymbol{v}_i = \sum_{i=1}^{n} x_i \boldsymbol{b}_i;\ x_i \in \mathbb{Z}\}.$$

Let $\Lambda + \boldsymbol{c} = \{\boldsymbol{v} + \boldsymbol{c} \mid \boldsymbol{v} \in \Lambda\}$ denote the $\boldsymbol{c}$ coset of Λ. The *rank* of the lattice is defined to be the rank of the matrix $\boldsymbol{B}$. We will only concern ourselves with lattices Λ s.t. $q\mathbb{Z}^m \subseteq \Lambda \subseteq \mathbb{Z}^m$.

Finally, we define the Gram-Schmidt procedure for recovering a basis of orthogonal vectors from any given basis $\boldsymbol{B}$.

Definition II;2.1 [Gram-Schmidt orthogonalisation]

For a lattice basis $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$, the *Gram-Schmidt orthogonalisation* of $\boldsymbol{B}$ is defined as the vectors of $\boldsymbol{B}^* = (\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^*)$, where:

$$\boldsymbol{b}_1^* = \boldsymbol{b}_1,\ \boldsymbol{b}_i^* = \boldsymbol{b}_i - \sum_{j=1}^{i} \mu_{i,j} \boldsymbol{b}_j^*$$

and $\mu_{i,j} = \frac{\langle \boldsymbol{b}_j^*, \boldsymbol{b}_i \rangle}{\langle \boldsymbol{b}_j^*, \boldsymbol{b}_j^* \rangle}$ are known as the *Gram-Schmidt coefficients*.

Gaussian distributions. For any $s > 0$, define the *Gaussian function* on $\mathbb{R}^n$ centred at $\boldsymbol{c} \in \mathbb{R}^n$ with parameter s to be:

$$\forall\, \boldsymbol{x} \in \mathbb{R}^n,\ \rho_{s,\boldsymbol{c}}(\boldsymbol{x}) = e^{-\pi\|\boldsymbol{x}-\boldsymbol{c}\|_2^2/s^2}.$$

Likewise, for any $s, \boldsymbol{c}$ as above and n-dimensional lattice Λ, define the *discrete Gaussian distribution* over Λ as:

$$\forall\, \boldsymbol{x} \in \Lambda,\ D_{\Lambda+\boldsymbol{c},s}(\boldsymbol{x}) = \frac{\rho_{s,\boldsymbol{c}}(\boldsymbol{x})}{\rho_{s,\boldsymbol{c}}(\Lambda)}.$$

The parameter s is referred to as the *width* of the distribution.

We will state a number of well-known lemmas, the proofs are omitted from this thesis and the reader should refer to the citations for the full details.

In this work, we will make use of *error distributions*, samples from this distribution should have norms bounded below some known value with high probability. We can show such a result for the Gaussian distribution $D_{\Lambda,s}$ over the lattice Λ, with parameter $s > 0$. We also show that it is efficient to sample from such a distribution.

Lemma II;2.2 [Bounded distributions [238, 254]]

Let $\boldsymbol{B}$ be a basis of an n-dimensional lattice Λ and let $\widetilde{\boldsymbol{B}}$ denote the Gram-Schmidt orthogonalisation of $\boldsymbol{B}$. Let $s \geq \|\widetilde{\boldsymbol{B}}\|_\infty \cdot \omega(\log \lambda)$ and $\boldsymbol{x} \leftarrow\!\!\$\, D_{\Lambda,s}$, then:

$$\Pr\big[(\|\boldsymbol{x}\|_\infty \geq s\sqrt{n}) \vee (\boldsymbol{x} = \boldsymbol{0})\big] < \mathsf{negl}(\lambda)$$

Lemma II;2.3 [Efficient Gaussian sampling [63, 161]]

There is a PPT algorithm that, given a basis $\boldsymbol{B}$ of an n-dimensional lattice $\Lambda(\boldsymbol{B})$, $\boldsymbol{c} \in \mathbb{R}^n$, $\sigma \geq \|\widetilde{\boldsymbol{B}}\|_\infty \cdot \sqrt{\ln(2n+4)/\pi}$, outputs a sample from $D_{\Lambda+\boldsymbol{c},\sigma}$.

II;2.2 Rings

In this work, we will be working over rings $\mathcal{R} := \mathbb{Z}[X]/(\phi(X))$ and $\mathcal{R}_q := \mathcal{R}/q\mathcal{R}$ for some degree $n = n(\lambda)$ integer polynomial $\phi(X) \in \mathbb{Z}[X]$ and a prime integer $q > 0$ — notably $\mathcal{R}_q$ is isomorphic to the ring $\mathbb{Z}_q[X]/(\phi(X))$. We let $\mathcal{R}^\times$ denote the ring of invertible elements $r \in \mathcal{R}$.

We perform addition in these rings component-wise in the coefficients of the polynomial elements and multiplication is performed via polynomial multiplication modulo $\phi(X)$ and q. An element in $\mathcal{R}$ (respectively $\mathcal{R}_q$) can be viewed as a degree $(n-1)$ polynomial over $\mathbb{Z}$ (respectively $\mathbb{Z}_q$). We can represent such an element using the vector of its n coefficients (where these will be in the range $\{-\lfloor q/2 \rfloor, \ldots, \lfloor q/2 \rfloor\}$ for elements in $\mathcal{R}_q$). We work with the polynomial $\phi(X) = X^N + 1$ with N a power of two. In particular, $\mathbb{Z}[X]/(\phi(X))$ is isomorphic to the ring of integers of the $2N^{\text{th}}$ cyclotomic field.

Canonical embeddings. Let ζ_m denote a primitive m^{th} root of unity. The m^{th} cyclotomic number field $K = \mathbb{Q}(\zeta_m)$ is the field extension of $\mathbb{Q}$ obtained by adjoining ζ_m. Let n be the degree of K over $\mathbb{Q}$, then there are n embeddings σ_i of $K \to \mathbb{C}$. These n embeddings correspond precisely to evaluation in each of the n distinct roots α_i of $\phi(X)$. In our case, $\psi(X)$ has $2 \cdot s_2 = n$ complex conjugate roots. Order the roots such that $\overline{\alpha_k} = \alpha_{s_2+k}$ for $k = 1, \ldots, s_2$. The *canonical embedding* $\sigma : K \to \mathbb{C}^n$ is defined as

$$a \mapsto (\sigma_1(a), \ldots, \sigma_{s_s}(a), \overline{\sigma_1}(a), \ldots, \overline{\sigma_{s_2}}(a)).$$

The canonical embedding maps into a space $H \subset \mathbb{C}^n$ given by

$$H = \left\{(X_1, \ldots, X_n) \in \mathbb{C}^n : \overline{X_j} = X_{s_2+j}, \forall 1 \le j \le s_2\right\}$$

which is isomorphic to $\mathbb{R}^n$ and we can represent the coordinates of $\sigma(a)$ by a real vector [78]

$$(\widetilde{a}_1, \ldots, \widetilde{a}_n) \propto (\Re(\sigma_1(a)), \ldots, \Re(\sigma_{s_2}(a)), \Im(\sigma_1(a)), \ldots, \Im(\sigma_{s_2}(a))).$$

This naturally induces a geometry on K with ℓ_2-norm $\|\cdot\|_2$ and ℓ_∞-norm $\|\cdot\|_\infty$:

$$\begin{aligned} \|a\|_2 &= \|\sigma(a)\|_2 = \left(\sum_{i=1}^{n} |\widetilde{a}_i|^2\right)^{1/2} \quad \text{and} \\ \|a\|_\infty &= \|\sigma(a)\|_\infty = \max_i |\widetilde{a}_i|. \end{aligned}$$

We use these norms to denote the norm of the ring elements that we are considering.

Sampling of polynomials. We may sample polynomials in $\mathcal{R}$ using arbitrary distributions γ over $\mathbb{Z}$. That is, we denote by $z \leftarrow_{\$} \mathcal{R}(\gamma^n)$ the sampling of an n-dimensional vector $\boldsymbol{v}$ from γ^n and then assigning the value v_i to the i^{th} coefficient of a ring element $z \in \mathbb{Z}[X]$ and output z. Notice that, trivially, $z \in \mathcal{R}$ since the polynomial has degree $n-1$.

Ideals. We denote the norm of an ideal $\mathcal{I} \subseteq \mathcal{R}$ by $\|\mathcal{I}\| = |\mathcal{R}/\mathcal{I}|$. For two ideals $\mathcal{I}, \mathcal{J} \subseteq \mathcal{R}$, let $\mathcal{I} \cdot \mathcal{J}$ be s.t. $Z \subseteq \mathcal{I} \cdot \mathcal{J}$, where $Z = \{z_1 z_2 | z_1 \in \mathcal{I}, z_2 \in \mathcal{J}\}$. In addition, $\|\mathcal{I} \cdot \mathcal{J}\| = \min_{\widetilde{\mathcal{I}}}(\|\widetilde{\mathcal{I}}\|)$ s.t. $Z \subseteq \widetilde{I}$. An ideal is *prime* if $\mathcal{I} \neq \mathcal{R}$ and, if $\mathcal{I} = \mathcal{J}_1 \cdot \mathcal{J}_2$, then this implies that $\mathcal{J}_1 = \mathcal{R}$ or $\mathcal{J}_2 = \mathcal{R}$.

Bounded distributions. We explicitly define (B_1, B_2)-bounded distributions wrt to rings, where all elements sampled from this distribution have an l_∞-norm that is bounded above by B_2, and below by B_1.

Definition II;2.2 [(B_1, B_2)-bounded element)]

An element $p \in \mathcal{R}$ is called (B_1, B_2)-*bounded* if $B_1 \leq \|p\|_\infty \leq B_2$.

We use the convention that $(0, B)$-*bounded* refers to an element of magnitude upper bounded by B, and (B, ∞)-*bounded* if it is lower bounded by B.

Definition II;2.3 [(B_1, B_2)-bounded distribution]

A distribution ensemble $\{\chi_\lambda\}_{\lambda \in \mathbb{N}}$, supported over $\mathcal{R}$, is called (B_1, B_2)-*bounded* (for $B_1, B_2 = \mathsf{poly}(\lambda)$) if for all p in the support of χ_λ, we have $B_1 < \|p\|_\infty < B_2$. In other words, a (B_1, B_2)-bounded distribution over $\mathcal{R}$ outputs a (B_1, B_2)-bounded polynomial.

The magnitude of products of ring elements can now be inferred using the following lemma.

Lemma II;2.4 [Magnitude of ring products [231]]

Let $n \in \mathbb{N}$, let $\phi(X) = X^n + 1$ and let $\mathcal{R} = \mathbb{Z}[X]/\langle \phi(X) \rangle$. For any $s, t \in \mathcal{R}$,

$$\|s \cdot t\|_\infty \leq \sqrt{n} \cdot \|s\|_\infty \cdot \|t\|_\infty \quad \text{and} \quad \|s \cdot t\|_\infty \leq n \cdot \|s\|_\infty \cdot \|t\|_\infty$$

Thus, the corollary below follows.

Corollary II;2.1 [Bounded distribution products [231]]

Take $n, \phi(X), \mathcal{R}$ as before. Let $s_1, \ldots, s_k \leftarrow\!\$\; \chi$ where χ is a (B_1, B_2)-bounded distribution over the ring $\mathcal{R}$. Then

$$s := \prod_{i=1}^{k} s_i$$

is $(n^{k-1} B_1^k, n^{k-1} B_2^k)$-bounded.

Gaussian distribution over rings. By Lemma II;2.2, the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}$ over $\mathbb{Z}^n$ outputs elements that are bounded by $\sigma\sqrt{n}$ with overwhelming probability. As such, we can then take the truncated Gaussian $\overline{D_{\mathbb{Z}^n,\sigma}}$ to be the distribution that outputs in the same way, except that for elements with infinity norm greater than $\sigma\sqrt{n}$ it outputs 0. Since this happens with low probability, the distributions are statistically indistinguishable.

Invertible elements from Gaussians. We finish these preliminaries with a lemma indicating that a ring element in $\mathcal{R}$ sampled from a discrete Gaussian distribution is invertible in $\mathcal{R}$ whp. This lemma was also used by Brakerski et al. [70] and was first stated in [281].

Lemma II;2.5 [Invertibility of Gaussian-sampled ring elements]

Let $n \geq 8$ be a power-of-two s.t. $x^n + 1$ splits into linear factors modulo a prime $5 \leq q \leq 2^n$ (e.g. $q \equiv 1 \mod 2n$). Let $\sigma = \Omega(\sqrt{n \log(q) \log(n)})$, then

$$\Pr\left[s \leftarrow_{\$} D_{\mathbb{Z}^n,\sigma} \middle| s \notin \mathcal{R}_q^{\times}\right] \leq O(n/q),$$

which is clearly negligible in q if $q = n^{O(\lambda)}$.

II;3 Computational assumptions

Computational assumptions form the bedrock of security proofs in provable security arguments. Such assumptions are made with the intention of reducing the security of constructions down to a simple, atomic consideration. Here we list some of the key computational assumptions that we require.

Discrete logarithm. The discrete logarithm (DL) assumption is one of the most fundamental cryptographic assumptions. This assumption is believed to be hard to break using classical algorithms. It was shown by Shor [278] that a polynomial-time quantum algorithm exists for solving the problem, and thus ensuring that the assumption does not hold in this setting.

```
$\mathsf{exp}^{\mathsf{dl}}_{\mathcal{A}}(1^\lambda, \mathbb{G}(\lambda))$
$g \leftarrow_{\$} \mathbb{G}(\lambda)$;
$r \leftarrow_{\$} \mathbb{Z}$;
$r_{\mathcal{A}} \leftarrow \mathcal{A}(\mathbb{G}(\lambda), g, g^r)$;
**return** $b_{\mathcal{A}} = (r \stackrel{?}{=} r_{\mathcal{A}})$;
```

Figure II;4: Experiments for defining the discrete logarithm assumption.

```
$\mathsf{exp}^{\mathsf{ddh}}_{b,\mathcal{A}}(1^\lambda, \mathbb{G}(\lambda))$
1 :  $g \leftarrow\!\!\$\ \mathbb{G}(\lambda);$
2 :  $c, d \leftarrow\!\!\$\ \mathbb{Z};$
3 :  $z = 0;$
4 :  if $b \overset{?}{=} 0$ :
5 :      $z = g^{cd};$
6 :  else :
7 :      $z \leftarrow\!\!\$\ \mathbb{G}(\lambda);$
8 :  $b_{\mathcal{A}} \leftarrow \mathcal{A}(1^\lambda, g^c, g^d, z);$
9 :  return $b_{\mathcal{A}}$;
```

Figure II;5: Experiments for describing the DDH assumption.

Assumption II;3.1 [Discrete logarithm]

Let $\mathcal{A}$ be any PPT algorithm, and let $\mathbb{G}(\lambda)$ be a finite, prime-order group for some prime $p(\lambda)$, where $\mathsf{bitlength}(p) = \mathsf{poly}(\lambda)$. The discrete logarithm assumption (or discrete log, or DL) states that:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{dl}(1^\lambda, \mathbb{G}(\lambda)))) < \mathsf{negl}(\lambda),$$

for $\mathsf{exp}^{\mathsf{dl}}_{\mathcal{A}}(1^\lambda, \mathbb{G}(\lambda))$ defined as in Figure II;4.

Hardness of DL. The hardness of the DL assumption rests upon the group $\mathbb{G}(\lambda)$ that is considered. It is conjectured that, when $\mathbb{G}(\lambda)$ is a multiplicative subgroup of a finite field, the DL assumption holds as long as the prime $p(\lambda)$ is chosen s.t. $\mathsf{bitlength}(p) = \mathsf{poly}(\lambda)$ is large enough. Concretely, this is when p is roughly 2048 bits in length.[4] If we instantiate $\mathbb{G}(\lambda)$ in the elliptic curve setting, then we can settle for a much smaller 256-bit prime.[5] For up-to-date security estimates see [247]. This leads to major efficiency improvements in cryptographic constructions that use these techniques. Some methods of cryptanalysis seem more suited to the elliptic curve setting, but the best known attack is still using Pollard's rho algorithm [41, 212, 241].

Decisional Diffie-Hellman. The decisional Diffie-Hellman (DDH) assumption is a widely-used computational assumption (over a pair of decisional experiments) that is a somewhat natural successor of the DL assumption. The assumption is versatile in the sense that it allows generating uniform elements in traditional group settings (i.e. groups without associated bilinear maps).

[4] As mentioned previously, this roughly translates to about 112 bits of security.
[5] For 128 bits of security.

```
$\mathsf{exp}^{\mathsf{dcr}}_{b,\mathcal{A}}(1^\lambda)$
1 :  $(N, p, q, y) \leftarrow\!\!\$\, \mathcal{C}(1^\lambda)$;
2 :  $z = 0$
3 :  if $b \overset{?}{=} 0$ :
4 :      $z = y^N \mod N^2$;
5 :  else :
6 :      $z \leftarrow\!\!\$\, \mathbb{Z}_{N^2}$;
7 :  $b_\mathcal{A} \leftarrow \mathcal{A}(1^\lambda, N, z)$;
8 :  return $b_\mathcal{A}$;
```

Figure II;6: Experiments for describing the DCR assumption.

Assumption II;3.2 [Decisional Diffie-Hellman]

Let $\mathcal{A}$ be any PPT algorithm, and let $\mathbb{G}(\lambda)$ be a finite, prime-order group. The decisional Diffie-Hellman assumption (or DDH) states that:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{ddh}(1^\lambda, \mathbb{G}(\lambda)))) < \mathsf{negl}(\lambda),$$

for $\mathsf{exp}^{\mathsf{ddh}}_{b,\mathcal{A}}(1^\lambda, \mathbb{G}(\lambda))$ defined as in Figure II;5.

Decisional composite residuocity. Paillier [251] introduced the decisional composite residuocity assumption (DCR) to prove the security of the Paillier cryptosystem (Construction II;5.2).

Assumption II;3.3 [Decisional Composite Residuosity]

Let $\mathcal{C}$ be a PPT challenger that, on input the security parameter λ, outputs (N, p, q, y); such that $N = pq$ for primes $p, q \in \mathbb{Z}$ and $y \leftarrow\!\!\$\, \mathbb{Z}_{N^2}$. Then the *decisional composite residuocity*, or DCR, assumption states that:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{dcr}(1^\lambda))) < \mathsf{negl}(\lambda)$$

holds for all PPT algorithms, $\mathcal{A}$, where $\mathsf{exp}^{\mathsf{dcr}}_{b,\mathcal{A}}(1^\lambda)$ is defined as in Figure II;6.

II;4 Symmetric primitives

We first consider cryptographic primitives that are deemed to be symmetric in their design. We say that a cipher is *symmetric* if enciphering and deciphering utilise the same key. It is *asymmetric* if enciphering uses public components of the key, and deciphering uses secret components.

```
$\mathsf{exp}^{\mathsf{prg}}_{0,\mathcal{A}}(1^\lambda, 1^m)$
1 :  $s \leftarrow_\$ \mathcal{S}$; $\mathsf{prg.Seed}(s)$;
2 :  $\{c_i\}_{i \in [m]} \leftarrow_\$ \mathsf{prg.Eval}(x)$;
3 :  $b_\mathcal{A} \leftarrow \mathcal{A}(1^\lambda, 1^m, \{c_i\}_{i \in [m]})$;
4 :  return $b_\mathcal{A}$;
```

```
$\mathsf{exp}^{\mathsf{prg}}_{1,\mathcal{A}}(1^\lambda, 1^m)$
1 :  $\{c_i\}_{i \in [m]} \leftarrow_\$ \mathcal{Y}^m$;
2 :  $b_\mathcal{A} \leftarrow \mathcal{A}(1^\lambda, 1^m, \{c_i\}_{i \in [m]})$;
3 :  return $b_\mathcal{A}$;
```

Figure II;7: Standard PRG indistinguishability game.

II;4.1 Pseudorandom generator

A pseudorandom generator (PRG) is a cornerstone primitive for generating random sequences of bits from a short, randomly sampled seed.

Definition II;4.1 [Pseudorandom generator]

A *pseudorandom generator* (PRG) is a tuple of stateful algorithms $\mathsf{prg} = (\mathsf{Seed}, \mathsf{Eval})$ that are defined as follows. Let $\mathcal{S} = \{0,1\}^{\ell_{\mathsf{in}}}$ be known as the *seed* space, and $\mathcal{Y} = \{0,1\}^{\ell_{\mathsf{out}}}$ as the *output* space.

- $\mathsf{prg.Seed}(1^\lambda, s)$: Takes as input the security parameter 1^λ, and a *seed* $s \in \mathcal{S}$.
- $\mathsf{prg.Eval}(x)$: Takes as input $x \in \mathbb{N}$; outputs $c_1, \ldots, c_m \in \mathcal{Y}^m$.

Let $\mathsf{exp}^{\mathsf{prg}}_{b,\mathcal{A}}(1^\lambda, 1^m)$ denote the decisional experiments in Figure II;7. We say that prg is a PRG if the properties are true.

- $\ell_{\mathsf{in}} \leq m \cdot \ell_{\mathsf{out}}$: i.e. the prg is *expanding*.
- The inequality:
$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{prg}(1^\lambda))) < \mathsf{negl}(\lambda)$$
holds, for all PPT adversaries $\mathcal{A}$.

II;4.2 Pseudorandom functions

A pseudorandom function (PRF) is a tuple $\mathsf{prf} = (\mathsf{Setup}, \mathsf{Eval})$. The security requirement is that, for $K \leftarrow_\$ \mathcal{K}$, then the outputs of the function $\mathsf{prf.Eval} : \mathcal{K} \times \mathcal{X} \mapsto \mathcal{Y}$ on K and adversarial inputs $\{x\} \in \mathcal{X}$ are computationally indistinguishable from the evaluations of a random function $f : \mathcal{X} \mapsto \mathcal{Y}$ on the same set $\{x\}$. Pseudorandom functions are achievable from one-way functions by the results of Hastad et al. [178] and Goldreich et al. [164].

```
$\mathsf{exp}^{\mathsf{prf}}_{0,\mathcal{A}}(1^\lambda)$
1 : $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{prf}.\mathsf{Setup}(1^\lambda)$;
2 : $x^\dagger \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{prf}}_{\mathcal{X}}(\,\cdot\,,\varphi)}(1^\lambda, \mathsf{pp})$;
3 : $y^\dagger \leftarrow \mathsf{prf}.\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x^\dagger)$;
4 : $b_{\mathcal{A}} \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{prf}}_{\mathcal{X}}(\,\cdot\,,\varphi)}(1^\lambda, \mathsf{pp}, y^\dagger)$;
5 : if $x^\dagger \in \varphi$ :
6 :     return $\bot$;
7 : return $b_{\mathcal{A}}$;
```

```
$\mathsf{exp}^{\mathsf{prf}}_{1,\mathcal{A}}(1^\lambda)$
1 : $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{prf}.\mathsf{Setup}(1^\lambda)$;
2 : $f \leftarrow_{\$} \mathsf{pp}.\mathcal{F}$;
3 : $x^\dagger \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{prf}}_{\mathcal{X}}(\,\cdot\,,\varphi)}(1^\lambda, \mathsf{pp})$;
4 : $y^\dagger \leftarrow f(x^\dagger)$;
5 : $b_{\mathcal{A}} \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{prf}}_{\mathcal{X}}(\,\cdot\,,\varphi)}(1^\lambda, \mathsf{pp}, y^\dagger)$;
6 : if $x^\dagger \in \varphi$ :
7 :     return $\bot$;
8 : return $b_{\mathcal{A}}$;
```

Figure II;8: PRF indistinguishability game.

More formally, let $\mathcal{F} = \{f : f : \mathcal{X} \mapsto \mathcal{Y}\}$, then the PRF indistinguishability game asks a PPT adversary to distinguish the two experiments in Figure II;8. All oracle queries, for $b \in \{0, 1\}$, are handled by $\mathsf{prf}.\mathsf{Eval}$; denote this oracle by $\mathcal{O}_{\mathcal{X}}(\mathsf{prf}.\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, \cdot), \varphi)$, or $\mathcal{O}^{\mathsf{prf}}_{\mathcal{X}}(\,\cdot\,, \varphi)$ for short.[6] At the challenge point, $x^\dagger$, the output is taken from either: the PRF in $\mathsf{exp}^{\mathsf{prf}}_{0,\mathcal{A}}(1^\lambda)$; or a uniform function in $\mathsf{exp}^{\mathsf{prf}}_{1,\mathcal{A}}(1^\lambda)$.

Definition II;4.2 [Pseudorandom function]

Let $\mathsf{prf} = (\mathsf{Setup}, \mathsf{Eval})$ be a tuple of stateful algorithms and let λ be the security parameter. Let $\mathcal{X}$ be the *input space*, and let $\mathcal{Y}$ be the *output space*; and define the algorithms as follows.

- $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{prf}.\mathsf{Setup}(1^\lambda)$: On input λ, outputs a pair $(\mathsf{pp}, \mathsf{msk})$ consisting of public parameters and a master secret key, respectively.
- $y \leftarrow \mathsf{prf}.\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x)$: On input $(\mathsf{pp}, \mathsf{msk})$ and $x \in \mathcal{X}$; outputs a value $y \in \mathcal{Y}$.

We say that prf is a *pseudorandom function*, or a PRF, if

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{prf}(1^\lambda))) < \mathsf{negl}(\lambda)$$

holds, where $\mathcal{A}$ is any PPT algorithm and $\mathsf{exp}^{\mathsf{prf}}_{b,\mathcal{A}}(1^\lambda)$ is defined as in Figure II;8. We may equivalently say that prf satisfies *pseudorandomness*.

Left-right reduction. A more standard framework for analysing PRF security gives the adversary oracle access to the random function in $\mathsf{exp}^{\mathsf{prf}}_{1,\mathcal{A}}(1^\lambda)$, rather than $\mathsf{prf}.\mathsf{Eval}$. This formula-

[6]Recall that φ collates the input queries that have been asked by $\mathcal{A}$.

tion is sometimes known as the 'left-right' definition, in the 'left' case ($b = 0$) the adversary has access to $\mathsf{prf.Eval}$, and in the 'right' case ($b = 1$) access to f.

In the left-right case, there is no specified challenge point $x^\dagger$. Instead, $\mathcal{A}$ outputs their guess $b_\mathcal{A}$ based on the oracle interactions that it makes. Fortunately, there is fairly simple reduction from this case to the version that we give in Figure II;8, see Lemma II;4.1.

Lemma II;4.1 [PRF satisfiability]

Let $\mathsf{exp}^{\mathsf{prf\text{-}lr}}_{c,\mathcal{B}}(1^\lambda)$ denote the decisional experiments for PRF security in the case where the left-right formulation is used, for PPT algorithms $\mathcal{B}$ and $c \in \{0,1\}$. Then the inequality:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{prf}(1^\lambda))) < \max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{B}, \mathsf{prf\text{-}lr}(1^\lambda))) < \mathsf{negl}(\lambda)$$

holds, where $\mathcal{A}$ is any PPT algorithm

Proof. We prove the reduction using a hybrid argument, starting with the execution of $\mathsf{exp}^{\mathsf{prf}}_{0,\mathcal{A}}(1^\lambda)$ and gradually transitioning to a state that is indistinguishable from $\mathsf{exp}^{\mathsf{prf}}_{1,\mathcal{A}}(1^\lambda)$.

- H_0: This is the real execution of $\mathsf{exp}^{\mathsf{prf}}_{0,\mathcal{A}}(1^\lambda)$.
- H_1: This is a modified execution $\overline{\mathsf{exp}^{\mathsf{prf}}_{0,\mathcal{A}}(1^\lambda)}$, where all queries $x \in \mathcal{X}$ are answered with a uniformly sampled function $f \in \{f \mid f : \mathcal{X} \mapsto \mathcal{Y}\}$.
- H_2: This is a modified execution $\widehat{\mathsf{exp}^{\mathsf{prf}}_{0,\mathcal{A}}}(1^\lambda)$, where non-challenge queries $x \in \mathcal{X}$ are answered by real PRF evaluations, i.e. $y \leftarrow \mathsf{prf.Eval}(\widehat{K}, x)$.

Claim II;4.0.1. $\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{H}_{0,1}(1^\lambda))) < \max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{B}, \mathsf{prf\text{-}lr}(1^\lambda)))$

Proof. Note that H_0 and H_1 are distributed exactly as in the two experiments that $\mathcal{B}$ witnesses in $\mathsf{exp}^{\mathsf{prf\text{-}lr}}_{c,\mathcal{B}}(1^\lambda)$ for $c \in \{0,1\}$. Therefore all queries made by $\mathcal{A}$ can be answered using the oracle that $\mathcal{B}$ has access to. This simulates the view of H_c for $\mathcal{A}$ perfectly, and thus $\mathcal{B}$ can simply output $c_\mathcal{B} = b_\mathcal{A}$ and succeed with the same advantage. □

Claim II;4.0.2. $\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{H}_{1,2}(1^\lambda))) < \max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{B}, \mathsf{prf\text{-}lr}(1^\lambda)))$

Proof. $\mathcal{B}$ samples a function $f \in \{f \mid f : \mathcal{X} \mapsto \mathcal{Y}\}$ uniformly. All input queries that are non-challenge queries $x \in \mathcal{X}$ are answered by sending x to their oracle and returning the output y back to $\mathcal{A}$. For the challenge input query $x^\dagger$, $\mathcal{B}$ computes $y^\dagger \leftarrow f(x^\dagger)$ and returns $y^\dagger$ to $\mathcal{A}$.

Then when $c = 1$ this is equivalent to H_1 since all queries are answered by a uniform function. When $c = 0$, this is equivalent to H_2, since all non-challenge queries are answered by real PRF evaluations, and the challenge is answered by a uniform function. Therefore, $\mathcal{B}$ outputs $c_{\mathcal{B}} = 1 - b_{\mathcal{A}}$ and succeeds with the same advantage as $\mathcal{A}$. □

Notice that the two executions $\widehat{\mathsf{exp}^{\mathsf{prf}}_{0,\mathcal{A}}}(1^\lambda)$ and $\mathsf{exp}^{\mathsf{prf}}_{1,\mathcal{A}}(1^\lambda)$ are equivalent in H_2 and thus the advantage of distinguishing the experiments is 0. By the fact that the underlying prf is pseudorandom, then Claims II;4.0.1 and II;4.0.2 show that

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{H}_{0,1}(1^\lambda))) < \mathsf{negl}(\lambda) \text{ and } \max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{H}_{1,2}(1^\lambda))) < \mathsf{negl}(\lambda).$$

Therefore, we can bound $\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{prf}(1^\lambda)))$ by $\mathsf{negl}(\lambda)' = 2\mathsf{negl}(\lambda)$, which is also negligible, and the proof is complete. □

II;4.3 The random oracle model and hash functions

Cryptographic hash functions are a key building block in many cryptographic constructions. The formulation given in Definition II;4.3 is taken from [201].

Definition II;4.3 [Cryptographic hash function]

Let $\mathcal{H}$ be a function family consisting of those functions $\{H \mid H : \{0,1\}^{\ell_{\mathsf{in}}} \mapsto \{0,1\}^{\ell_{\mathsf{out}}}\}$. We say that $\mathcal{H}$ is a *collision-resistant hash family*, if for any function $H \leftarrow_{\$} \mathcal{H}$ it is given that: (1) $\ell_{\mathsf{out}} \leq \ell_{\mathsf{in}}$, i.e. the function is *compressing*; (2) for any PPT algorithm $\mathcal{A}$:

$$\Pr[H(x_1) = H(x_2)|_{x_1,x_2 \leftarrow \mathcal{A}(\mathcal{H})}] < \mathsf{negl}(\lambda).$$

We may refer to a hash function sampled from such a family as a *cryptographic hash function*.

Remark II;4.1. *While cryptographic hash functions typically also satisfy guarantees such as first and second pre-image resistance, we do not need these requirements for our analysis. As such, we will only consider hash functions that satisfy collision-resistance. Ultimately, collision-resistance is the hardest requirement to satisfy, and is usually seen as the barometer of a good hash function in the real-world [282].*

Random oracle model

The seminal work of Bellare and Rogaway [35] showed that replacing specific instantiations of hash functions with the *random oracle model* (ROM) allowed for much more expressive proofs of security. The ROM replaces all hash function evaluations with calls to an oracle that implements a random function. While cryptographic hash functions cannot generically instantiate random oracles, the paradigm allows for much simpler and intuitive security guarantees.

Consider a security game with a challenger and an adversary, and consider a scheme with some specific instantiation of a function H. In the ROM, the oracle has a table and every time an input is submitted it checks the table and returns the output assigned to each input. If the query's input is not already present in the table, then the oracle samples a new random value and returns it; before adding the input/output pair to the table.

No PPT function can instantiate a random function efficiently. Therefore, there is a clear disparity between the model and the real instantiation of a construction. However the paradigm facilitates the writing of security proofs in the ROM. The actual construction is then instantiated using a real function (such as a hash function).

Programmability. There are a number of extra properties that we can consider in the ROM. For this work, we must consider the notion of *programmability*. During a security proof, we can consider that the random oracle in an experiment is being answered by the challenger directly. Then, the challenger can choose to 'program' the random oracle, s.t. for a specific choice of input x, the random oracle always outputs y on x. Providing that y is drawn uniformly from the same domain, then this should go undetected by the adversary.

Extractability. The *extractability* property allows the challenger to extract the queries made by the adversary to the random oracle. This property intuitively follows in the same way as above, by allowing the challenger to answer random oracle queries directly. This models the fact that the adversary cannot learn the value of the random oracle on some input x without explicitly making a query.

Strongly universal hash families

Strongly universal hash families are a sub-class of hash families. They are strictly weaker than random oracles in that they can be instantiated from standard techniques [283].

```
$\mathsf{exp}^{\mathsf{owf}}_{\mathcal{A}}(1^\lambda, 1^r)$
for $i \in [r]$ :
    $x_i \leftarrow \{0,1\}^\lambda$;
    $y_i \leftarrow f(x_i)$;
$x^\dagger \leftarrow \mathcal{A}(1^\lambda, \{y_i\}_{i\in[r]})$;
if $x^\dagger \in \{x_i\}_{i\in[r]}$ :
    return $1$;
else :
    return $0$;
```

Figure II;9: Computational experiment for attempting to invert a one-way function f. Commonly, we choose $r = 1$.

Definition II;4.4 [Strongly universal hash families]

Let $\mathcal{X} = \{0,1\}^\ell$, and $\mathcal{Y} = [m]$ be an interval of length m. We say that $\mathcal{H} = \{h \mid h : \mathcal{X} \mapsto \mathcal{Y}\}$ is a *strongly universal hash family*, if it satisfies:

$$\Pr\left[(y_1 \leftarrow h(x_1)) \wedge (y_2 \leftarrow h(x_2)) \,\middle|\, \substack{x_1,x_2\in\mathcal{X};\; x_1\neq x_2,\\ y_1,y_2\in\mathcal{Y},\\ h \leftarrow\$\, \mathcal{H}}\right] < 1/m^2.$$

This property is sometimes also known as *pairwise independence*.

Corollary II;4.1 [Uniformity]

Let $\mathcal{H}$ be a strongly universal hash family. Then $\mathcal{H}$ satisfies the following:

$$\Pr\left[y \leftarrow h(x) \,\middle|\, \substack{x\in\mathcal{X};\; y\in\mathcal{Y};\\ h \leftarrow\$\, \mathcal{H}}\right] < 1/m.$$

Proof. The corollary follows immediately from the statement of Definition II;4.4. □

II;4.4 One-way functions

One-way functions represent one of the most fundamental primitives in cryptographic literature. Such a function allows efficient computation, but is provably difficult to invert. By inversion we mean that, given an output of the function, then recover the pre-image used to compute it.

Definition II;4.5 [One-way functions]

Let λ be the security parameter. Let $f_\lambda \in \mathcal{F} = \{f_\lambda | f_\lambda : \{0,1\}^\lambda \mapsto \{0,1\}^\lambda\}$ be some function, and let $\mathsf{exp}^{\mathsf{owf}}_{\mathcal{A}}(1^\lambda, 1^r)$ be the computational experiment defined in Figure II;9 for some $r = \mathsf{poly}(\lambda)$. We say that $\mathcal{F}$ is a (strong) *one-way function* (OWF) family if it satisfies the conditions:

- f_λ is efficiently computable;
- the following inequality:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{owf}(1^\lambda))) < \mathsf{negl}(\lambda),$$

  holds, where $\mathcal{A}$ is any PPT algorithm.

II;4.5 Symmetric-key encryption

Symmetric-key encryption provides two parties, in knowledge of some common secret key sk, to communicate without revealing the content of their messages. That is, messages are converted into a garbled representation than can only be inverted using knowledge of sk.[7] We use *semantic security* (also known as IND-CPA security) to elucidate the security guarantee, requiring that an eavesdropper on the communication channel cannot distinguish between encryptions of two plaintexts.

Definition II;4.6 [Symmetric-key encryption scheme]

Let $\mathsf{ske} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ be a tuple of algorithms. We let: $\mathcal{K}$ denote the *key* space; $\mathcal{X}$ denote the *plaintext* space; $\mathcal{Y}$ denote the *ciphertext* space. Define the algorithms in the following way:

- $\mathsf{sk} \leftarrow \mathsf{ske.Setup}(1^\lambda)$: On input the security parameter λ; outputs $\mathsf{sk} \in \mathcal{K}$.
- $c \leftarrow \mathsf{ske.Enc}(\mathsf{sk}, m)$: On input $\mathsf{sk} \in \mathcal{K}$, and $m \in \mathcal{X}$; outputs $c \in \mathcal{Y}$.
- $m \leftarrow \mathsf{ske.Dec}(\mathsf{sk}, c)$: On input $\mathsf{sk} \in \mathcal{K}$, and $c \in \mathcal{Y}$; outputs $m \in \mathcal{X}$.

[7]These differ from OWFs since there is a defined way to invert.

Definition II;4.7 [Correctness]

We say that ske is *correct* if:

$$\Pr\left[m \leftarrow \mathsf{ske.Dec}(\mathsf{sk}, c) \middle| {}^{\mathsf{sk} \leftarrow\$ \mathsf{ske.Setup}(1^\lambda)}_{c \leftarrow \mathsf{ske.Enc}(\mathsf{sk}, m)}\right] > 1 - \mathsf{negl}(\lambda),$$

and perfectly correct if the probability is equal to 1.

The security requirement for symmetric-key encryption is very similar to the case of a *public-key* encryption scheme (that we give later in Definition II;5.1). We use public-key encryption more readily throughout this thesis, therefore we defer the actual experiments that expose the security requirement until later. We will lay out the slight differences in the security models later.

Definition II;4.8 [Security]

Let $\mathsf{exp}^{\mathsf{indcpa}}_{b,\mathcal{A}}(1^\lambda)$ be the pair of experiments defined in Figure II;11, for $b \in \{0, 1\}$. We say that ske satisfies *semantic security*, or IND-CPA security, if:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{indcpa}(1^\lambda, \mathsf{ske}))) < \mathsf{negl}(\lambda)$$

holds, where $\mathcal{A}$ is any PPT algorithm.

II;4.6 Message authentication codes

Message authentication codes are typically used to enshrine the integrity of data in the symmetric setting. In addition, they are often used in authentication scenarios.

Definition II;4.9 [MAC]

Let $\mathsf{mac} = (\mathsf{Setup}, \mathsf{Tag}, \mathsf{Verify})$ be a tuple of algorithms, and let λ be the security parameter. A *message authentication code* (or MAC) scheme is defined in the following way.

- $K \leftarrow_{\$} \mathsf{mac.Setup}(1^\lambda)$: On input the security parameter; outputs a key $K \in \mathcal{K}$.
- $\tau \leftarrow \mathsf{mac.Tag}(K, x)$: On input $K \in \mathcal{K}$ and $x \in \mathcal{X}$; outputs a tag $\tau \in \mathcal{Y}$.
- $b \leftarrow \mathsf{mac.Verify}(K, x, \tau)$: On input $K \in \mathcal{K}$, an input $x \in \mathcal{X}$ and a tag $\tau \in \mathcal{Y}$; outputs a bit $b \in \{0, 1\}$.

We refer to: $\mathsf{mac.Setup} : 1^\lambda \mapsto \mathcal{K}$ as *setup*; $\mathsf{mac.Tag} : \mathcal{K} \times \mathcal{X} \mapsto \mathcal{Y}$ as *tagging*; and $\mathsf{mac.Verify} : \mathcal{K} \times \mathcal{X} \times \mathcal{Y} \mapsto \{0, 1\}$ as *verification*. We refer to: $\mathcal{K}$ as the *key* space, $\mathcal{X}$ as in the *input* space and $\mathcal{Y}$ as the *tag* space.

Definition II;4.10 [Correctness]

Let mac be a MAC scheme. We say that mac is *correct*, if it satisfies the following.

$$\Pr\left[1 \leftarrow \mathsf{mac.Verify}(K, x, \tau) \middle| \substack{K \leftarrow_{\$} \mathsf{mac.Setup}(1^\lambda), \\ \tau \leftarrow \mathsf{mac.Tag}(K, x \in \mathcal{X})}\right] < \mathsf{negl}(\lambda)$$

Definition II;4.11 [Security]

Let mac be a MAC scheme, let $\mathsf{exp}^{\mathsf{mac}}_{\mathcal{A}}(1^\lambda)$ be the computational experiment defined in Figure II;10. We say that a MAC scheme is *secure against existential forgeries* (under adaptively-chosen messages) if

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{mac}(1^\lambda))) < \mathsf{negl}(\lambda)$$

holds, where $\mathcal{A}$ is any PPT algorithm.

The formalisation for the security of MAC schemes is taken from [201, Definition 4.2]. Note that we actually use a stronger definition, where the pair $(x^\dagger, \tau^\dagger)$ is included in φ, rather than just $x^\dagger$.

The hmac algorithm. A popular example of a MAC algorithm is the hmac algorithm, introduced by [33]. The algorithm is standardised by the Federal Information Processing Standards Publication in FIPS PUB 198-1 [140]. It is also written in the form of an IETF RFC (RFC 2104) [216].

```
$\mathsf{exp}^{\mathsf{mac}}_{\mathcal{A}}(1^\lambda)$
1 :  $\varphi \leftarrow \emptyset$;
2 :  $K \leftarrow\!\!\$\ \mathsf{mac.Setup}(1^\lambda)$;
3 :  $(x^\dagger, \tau^\dagger) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathcal{X},\mathcal{Y}}(\mathsf{mac.Tag}(K,\cdot),\varphi)}(1^\lambda)$;
4 :  if $((x^\dagger, \tau^\dagger) \notin \varphi) \wedge (1 \leftarrow \mathsf{mac.Verify}(K, x^\dagger, \tau^\dagger))$ :
5 :    return $1$;
6 :  else :
7 :    return $0$;
```

Figure II;10: Existential forgery computational experiments for MAC schemes. Note that the input messages are chosen adaptively by the adversary. We modify the set φ from the original definition to also include the output to the query x.

This algorithm is very practical to run, requiring only a few hash function evaluations per output. Moreover, it enjoys additional security properties that, in the ROM, allow us to show that hmac is also a viable PRF. As a result, it is common to use hmac with an appropriate choice of hash function, such as SHA-256, as a PRF in practical cryptographic implementations. In Chapter V, we use an implementation of hmac as an atomic PRF as part of an experimental analysis.

Construction II;4.1 [hmac]

Let $H : \{0,1\}^\ell \mapsto \{0,1\}^m$ be a hash function, and let $\mathcal{K} = \{0,1\}^\lambda$, $\mathcal{X} = \{0,1\}^\ell$ and $\mathcal{Y} = \{0,1\}^m$. Additionally, let opad be known as the *outer* padding, and ipad be known as the *inner* padding.
Let hmac be the construction that implements the algorithms of MAC in the following way.

- $K \leftarrow\!\!\$\ \mathsf{hmac.Setup}(1^\lambda)$: Outputs a uniform choice of a key $K \leftarrow\!\!\$\ \{0,1\}^\lambda$.
- $\tau \leftarrow \mathsf{hmac.Tag}(K, m)$: Output

$$\tau = H((K \oplus \mathsf{opad}) \| H((K \oplus \mathsf{ipad}) \| m)).$$

- $b \leftarrow \mathsf{hmac.Verify}(K, m, \tau)$: Compute

$$\tau' = H((K \oplus \mathsf{opad}) \| H((K \oplus \mathsf{ipad}) \| m))$$

  and output $b \leftarrow (\tau \overset{?}{=} \tau')$.

In Construction II;4.1 the outer and inner padding values are deliberately left open to interpretation. In the hmac standard they are set as $\mathsf{opad} = 0x5c \ldots 5c$ and $\mathsf{ipad} = 0x36 \ldots 36$, i.e. hex-encoded strings of equal length to the *block-size* of the hash function H. The block size of H refers to the maximum number of input bits processed by the hash function at once. Moreover, sometimes it is necessary to replace K in $\mathsf{hmac.Tag}$ and $\mathsf{hmac.Verify}$ with K', where K' is an appropriately padded version of K. The padding procedure is important to maintain security in practical scenarios.

That hmac is a secure MAC algorithm is shown in [201, Construction 5.7]. That hmac is a PRF in the ROM is evident from the fact that H will be a random function in this model.[8]

II;5 Asymmetric primitives

We secondly consider cryptographic primitives which are deemed to be asymmetric. That is, algorithms are defined wrt a key pair made up of a public component and a secret component. Some operations can be carried out using knowledge of only the public component of the key. However, at least one algorithm requires the secret component as input for security to hold.

II;5.1 Public-key encryption

A public-key encryption (PKE) scheme allows entities to transform messages into some garbled representation that is plausibly difficult to invert. In fact, it is hard to distinguish the representations of two different encrypted messages. The scheme includes some secret parameters that are only known to one user who can then invert the transformation (this is the difference wrt one-way functions). The public aspect of the scheme allows any entity with access to a set of public parameters to carry out the transformation, without being able to invert.

[8] Length-extension properties of hmac mean that this is not immediately obvious, but the outer hash function evaluation ensures that this is not an issue.

Definition II;5.1 [Public-key encryption scheme]

Let $\mathsf{pke} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a tuple of algorithms, and let λ be the security parameter. A *public-key encryption* (PKE) scheme is defined in the following way.

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{pke.KeyGen}(1^\lambda)$: On input the security parameter; outputs a key pair $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{K}_{\mathsf{pk}} \times \mathcal{K}_{\mathsf{sk}}$. Here, pk is the *public key*, and sk is the *secret key*.
- $c \leftarrow \mathsf{pke.Enc}(\mathsf{pk}, m)$: Let $\mathsf{pk} \in \mathcal{K}_{\mathsf{pk}}, m \in \mathcal{X}$; outputs a *ciphertext* $c \in \mathcal{Y}$, or $\perp$.
- $m \leftarrow \mathsf{pke.Dec}(\mathsf{sk}, c)$: Let $\mathsf{sk} \in \mathcal{K}_{\mathsf{sk}}, c \in \mathcal{Y}$; outputs either $m \in \mathcal{X}$, or $\perp$.

We refer to: $\mathsf{pke.KeyGen} : 1^\lambda \mapsto \mathcal{K}_{\mathsf{pk}} \times \mathcal{K}_{\mathsf{sk}}(\cup\{\perp\}^2)$ as *key generation*; $\mathsf{pke.Enc} : \mathcal{K}_{\mathsf{pk}} \times \mathcal{X} \mapsto \mathcal{Y} \cup \{\perp\}$ as *encryption*; $\mathsf{pke.Dec} : \mathcal{K}_{\mathsf{sk}} \times \mathcal{Y} \mapsto \mathcal{X} \cup \{\perp\}$ as *decryption*. We refer to: $\mathcal{X}$ as the *plaintext* space; $\mathcal{Y}$ as the *ciphertext* space; $\mathcal{K}_{\mathsf{pk}}, \mathcal{K}_{\mathsf{sk}}$ as the *public/secret key* space, respectively.

We consider all pke schemes to be probabilistic, meaning that the $\mathsf{pke.Enc}$ algorithm also implicitly takes random coins as input. These random coins essentially distribute the ciphertext randomly in the codomain of all encryptions of the plaintext value x, we denote this codomain by $\mathcal{Y}_x \subset \mathcal{Y}$. We may sometimes write $\mathsf{pke.Enc}(\mathsf{pk}, x; r)$ if we want to make the choice of random coins explicit in the running of the encryption algorithm.

Definition II;5.2 [Correctness of pke]

Let pke be a PKE scheme. We say that pke is *correct*, if it satisfies the following

$$\Pr\left[\mathsf{pke.Dec}(\mathsf{sk}, c) \neq m \;\middle|\; \substack{(\mathsf{pk},\mathsf{sk})\leftarrow\mathsf{pke.KeyGen}(1^\lambda);\\ c\leftarrow\mathsf{pke.Enc}(\mathsf{pk},m\in\mathcal{X})}\right] < \mathsf{negl}(\lambda).$$

Definition II;5.3 [IND-CPA security]

Let pke be a PKE scheme, let $\mathsf{exp}^{\mathsf{indcpa}}_{b,\mathcal{A}}(1^\lambda)$ be the experiment defined in Figure II;11. We say that pke is *semantically secure*, or that it satisfies IND-CPA security, if

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{indcpa}(1^\lambda, \mathsf{pke}))) < \mathsf{negl}(\lambda)$$

holds, where $\mathcal{A}$ is any PPT algorithm.

Remark II;5.1. *The adversary does not need oracle access to encryption as it possesses the real public key. In the symmetric case, we give the adversary oracle access to* $\mathsf{ske.Enc}$ *under the challenge key.*

```
$\mathsf{exp}^{\mathsf{indcpa}}_{b,\mathcal{A}}(1^\lambda)$
1 :  $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{pke.KeyGen}(1^\lambda);$
2 :  $m_0, m_1 \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk});$
3 :  $c \leftarrow \mathsf{pke.Enc}(\mathsf{pk}, m_b);$
4 :  $b_\mathcal{A} \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}, c);$
5 :  **return** $b_\mathcal{A};$
```

Figure II;11: IND-CPA security model.

We also define IND-CPA security wrt allowing the adversary to specify multiple challenge messages to be encrypted. That is, we allow the adversary to specify a list of plaintexts pairs of length $\rho = \mathsf{poly}(\lambda)$ to be encrypted. The challenger samples $b \leftarrow\$ \{0, 1\}$ and encrypts either the left message in each pair if $b = 0$, or the right message if $b = 1$.

Definition II;5.4 [ρ-IND-CPA security]

Let pke be a PKE scheme, let $\mathsf{exp}^{\rho\text{-}\mathsf{indcpa}}_{b,\mathcal{A}}(1^\lambda)$ be the experiment defined in Figure II;12. We say that pke satisfies ρ-IND-CPA security, if

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \rho\text{-}\mathsf{indcpa}(1^\lambda, \mathsf{pke}))) < \mathsf{negl}(\lambda)$$

holds, where $\mathcal{A}$ is any PPT algorithm.

Lemma II;5.1 shows that ρ-IND-CPA security is implied by IND-CPA security if the encryption scheme is stateless. By stateless, we mean that each invocation is independent of all previous invocations. A proof of this lemma was also given in [201, Theorem 11.6] and thus we do not give an explicit proof in this thesis.

```
$\mathsf{exp}^{\rho\text{-}\mathsf{indcpa}}_{b,\mathcal{A}}(1^\lambda)$
1 :  $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{pke.KeyGen}(1^\lambda);$
2 :  **for** $i \in [\rho]$ :
3 :      $(m^i_0, m^i_1) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk});$
4 :      $c^i \leftarrow \mathsf{pke.Enc}(\mathsf{pk}, m^i_b);$
5 :  $b_\mathcal{A} \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}, \{c^i\}_{i\in[\rho]});$
6 :  **return** $b_\mathcal{A};$
```

Figure II;12: ρ-IND-CPA security model.

Lemma II;5.1 [IND-CPA $\implies$ ρ-IND-CPA [201]]

Let pke be a PKE scheme (Definition II;5.1) satisfying IND-CPA security (Definition II;5.3) and let $\mathsf{pke.Enc}$ be a *stateless* algorithm. Then pke satisfies ρ-IND-CPA security for $\rho = \mathsf{poly}(\lambda)$.

Additional notation. We may sometimes abuse notation and write

$$\{c_j\}_{j\in[\ell]} \leftarrow \mathsf{pke.Enc}(\mathsf{pk}, \{m_j\}_{j\in[\ell]})$$

to indicate that each message m_j is encrypted individually into the ciphertext c_j; and then returned as the ordered set $\{c_j\}_{j\in[\ell]}$.[9]

El Gamal encryption scheme

We give an example of a PKE scheme satisfying IND-CPA security, known as *El Gamal*. This scheme was introduced in [134].

[9]When using this notation, we will always assume that sets have an ordering depending on the order on which elements are added in. This is similar to the notation used by vectors, but we do not use vectors explicitly to avoid confusion with mathematical vectors that come in later sections.

```
$\mathsf{exp}^{\mathsf{omd}}_{\mathcal{A}}(1^\lambda, m)$

1 :   $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{elga.Setup}(1^\lambda)$;
2 :   $\{x_i\}_{i \in [m+1]} \leftarrow_\$ \mathbb{G}$;
3 :   $\{(C_i, D_i) \leftarrow \mathsf{elga.Enc}(\mathsf{pk}, x_i)\}_{i \in [m+1]}$;
4 :   $\{x'_i\}_{i \in [m+1]} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathbb{G}^2}(\mathsf{elga.Dec}(\mathsf{sk}, \cdot); [m])}(1^\lambda, \mathsf{pk}, \{C_i, D_i\}_{i \in [m+1]})$;
5 :   for $i \in [m+1]$ :
6 :     if $\neg(x'_i \overset{?}{=} x_i)$ :
7 :       return $0$;
8 :   return $1$;
```

Figure II;13: Decisional experiments for characterising the one-more-decryption security of El Gamal.

Construction II;5.1 [El Gamal]

Let $\mathbb{G} = \mathbb{G}(\lambda)$ be a cyclic group of prime order $p(\lambda)$, where $\mathsf{bitlength}(p) = \mathsf{poly}(\lambda)$, and let $g \in \mathbb{G}$ be a generator. Let elga denote the El Gamal PKE scheme with plaintext and ciphertext space $\mathbb{G}$, instantiating the required algorithms in the manner shown below.

- $\mathsf{elga.Setup}(1^\lambda, p)$: Sample $x \in \mathbb{Z}_p$ uniformly and compute $h = g^x$. Output $\mathsf{pk} = (\mathbb{G}, p, g, h)$ and $\mathsf{sk} = (x)$.
- $\mathsf{elga.Enc}(\mathsf{pk}, m \in \mathbb{Z}_p)$:
  - Sample $y \in \mathbb{Z}_p$ uniformly and compute $c_1 = g^y$, and $s = h^y$.
  - Calculate $c_2 = m \cdot s$.
  - Return $c = (c_1, c_2)$ as the ciphertext.
- $\mathsf{elga.Dec}(\mathsf{sk}, c)$:
  - Parse c as (c_1, c_2), and calculate $s = c_1^x$
  - Compute $m' \leftarrow c_2 \cdot s^{-1} \in \mathbb{G}$.
  - Return m'.

The fact that elga is correct follows trivially from the fact that $c_1^x = (g^y)^x = (g^x)^y = h^y = s$. The fact that it is IND-CPA secure follows from the DDH assumption. We now show that elga satisfies an extra property known as *one-more-decryption* security.

Lemma II;5.2 [One-more-decryption security [274]]

The inequality:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{omd}(1^\lambda, \mathsf{elga}))) < \mathsf{negl}(\lambda)$$

is satisfied for all PPT algorithms $\mathcal{A}$ running in the generic group model.

In actual fact, Schnorr and Jakobsson [274] prove that no PPT adversary exists in the generic group model that has a non-negligible advantage of breaking a strictly weaker version of Assumption II;5.2.[10] An adversary who can computationally succeed in the experiments in Figure II;13, can easily win the weaker game as well. Thus, Lemma II;5.2 holds in the generic group model. The specification of the generic group model is beyond the scope of this thesis.

II;5.2 Homomorphic encryption

A somewhat homomorphic encryption (SHE) scheme, she, is a standard PKE scheme augmented with two new algorithms $(\mathsf{she.Add}, \mathsf{she.Mult})$. We define such a scheme formally in Definition II;5.5. For the purpose of this section and when we interact with SHE in the future, we will always assume that both the underling plaintext space $\mathcal{X}$ and the ciphertext space $\mathcal{Y}$ are rings. Therefore, there are well-defined operations $+, \cdot$ that act over the ring elements.

A public-key *somewhat homomorphic encryption* scheme is constructed wrt to a levelled structure. That is, the setup algorithm takes an additional parameter 1^ℓ that defines the number of homomorphic multiplications that can be computed over the ciphertext. The levels also designate some extra restrictions on the types of ciphertexts that can be input into the homomorphic algorithms. We formalise the algorithms below, using the notation $c[\iota]$ for an encrypted ciphertext on 'level' ι (i.e. computed via ι invocations of $\mathsf{she.Mult}$). This levelled structure is reflected in the ciphertext space, where $\mathcal{Y}[\iota]$ is the space of all ciphertexts on 'level' ι. Finally, $\mathcal{Y} = \mathcal{Y}[1] \cup \ldots \cup \mathcal{Y}[\ell]$ is the collection of all levelled ciphertext spaces.

[10]In their version, the adversary is given $d > m$ ciphertexts, and the correct plaintexts in a permuted order, and must simply match $m + 1$ plaintexts to their corresponding ciphertexts using at most m queries to the decryption oracle.

Definition II;5.5 [Somewhat homomorphic encryption (SHE)]

Let λ be the security parameter and let $\mathsf{she} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Add}, \mathsf{Mult}, \mathsf{Dec})$ be a PKE scheme, augmented with the algorithms Add, Mult.

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{she.KeyGen}(1^\lambda, 1^\ell)$: On input 1^λ and 1^ℓ for some $\ell = \mathsf{poly}(\lambda)$; outputs a key pair $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{K}_{\mathsf{pk}} \times \mathcal{K}_{\mathsf{sk}}$.
- $c[\iota] \leftarrow \mathsf{she.Enc}(\mathsf{pk}, m, \iota)$: Let $\mathsf{pk} \in \mathcal{K}_{\mathsf{pk}}, m \in \mathcal{X}$; if $\iota > \ell$ outputs $\perp$, else outputs a ciphertext $c[\iota] \in \mathcal{Y}[\iota]$.
- $c_+[\iota_1] \leftarrow \mathsf{she.Add}(\mathsf{pk}, c_1[\iota_1], c_2[\iota_2])$: Let $\mathsf{pk} \in \mathcal{K}_{\mathsf{pk}}, c_j[\iota_j] \in \mathcal{Y}[\iota_j]$ for $j \in \{1, 2\}$; if $\neg(\iota_1 \overset{?}{=} \iota_2)$ then it outputs $\perp$, else it outputs $c_+[\iota_1] \in \mathcal{Y}[\iota_1]$.
- $c_\times[\iota_1 + \iota_2] \leftarrow \mathsf{she.Mult}(\mathsf{pk}, c_1[\iota_1], c_2[\iota_2])$: Let $\mathsf{pk} \in \mathcal{K}_{\mathsf{pk}}$, $c_j[\iota_j] \in \mathcal{Y}[\iota_j]$; if $\iota_1 + \iota_2 > \ell$ then it outputs $\perp$, else it outputs $c_\times[\iota_1 + \iota_2] \in \mathcal{Y}[\iota_1 + \iota_2]$.
- $m \leftarrow \mathsf{she.Dec}(\mathsf{sk}, c[\iota])$: Let $\mathsf{sk} \in \mathcal{K}_{\mathsf{sk}}, c[\iota] \in \mathcal{Y}[\iota]$; if $\iota > \ell$ outputs $\perp$; else outputs $m \in \mathcal{X}$.

We refer to: $\mathsf{she.Add} : \mathcal{K}_{\mathsf{pk}} \times \mathcal{Y} \times \mathcal{Y} \mapsto \mathcal{Y} \cup \{\perp\}$ as *homomorphic addition*; $\mathsf{she.Mult} : \mathcal{K}_{\mathsf{pk}} \times \mathcal{Y} \times \mathcal{Y} \mapsto \mathcal{Y} \cup \{\perp\}$ as *homomorphic multiplication*.

Correctness is the same as that of a PKE scheme, though it is augmented to include requirements on the homomorphic operations algorithms. Recall that we assume that the plaintext space $\mathcal{X}$ is a ring and thus there are natural operations $+, \cdot$ that can be performed over ring elements.

Definition II;5.6 [Correctness of SHE]

Let $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{she.KeyGen}(1^\lambda)$; $c_j[\iota_j] \leftarrow \mathsf{she.Enc}(\mathsf{pk}, m_j, \iota_j)$ for $j \in \{1, 2\}$, $\iota_j \in [\ell]$ and $m_j \in \mathcal{X}$. We say that she is *correct* if it satisfies the standard definition of correctness given in Definition II;5.2 for PKE schemes with the extra requirement that $\iota_j \leq \ell$ for decryption to function correctly. It must also satisfy the following requirements.

$$\Pr\left[\mathsf{she.Dec}(\mathsf{sk}, c_+[\iota]) \neq m_1 + m_2 \;\middle|\; \substack{c_+[\iota] \leftarrow \mathsf{she.Add}(\mathsf{pk}, c_1[\iota_1], c_2[\iota_2]), \\ \iota = \iota_1 = \iota_2 \leq \ell}\right] < \mathsf{negl}(\lambda) \quad \text{(II;2)}$$

$$\Pr\left[\mathsf{she.Dec}(\mathsf{sk}, c_\times[\iota]) \neq m_1 \cdot m_2 \;\middle|\; \substack{c_\times \leftarrow \mathsf{she.Mult}(\mathsf{pk}, c_1[\iota_1], c_2[\iota_2]), \\ \iota_1 + \iota_2 = \iota \leq \ell}\right] < \mathsf{negl}(\lambda) \quad \text{(II;3)}$$

We do not extend the correctness clauses to include the case where the 'level' of the output ciphertext ι is greater than ℓ. The reason that somewhat homomorphic encryption schemes demonstrate such behaviour is because ciphertexts are *noisy*. If this noise rises above a certain threshold then the correctness of decryption is lost. Concretely, the parameters of an SHE scheme are chosen to allow ℓ $\mathsf{she.Mult}$ operations to be carried out on the ciphertexts. Note we also permit some negligible chance of failure in decryption when the levelled structure is abided by. If there was no probability of decryption errors, then these probabilities would be equal to zero instead.

An *additively homomorphic encryption* (AHE) scheme is a homomorphic encryption scheme where only $\mathsf{she.Add}$ is defined, and thus only satisfies correctness for Equation (II;2). A *multiplicatively homomorphic encryption* (MHE) scheme is a homomorphic encryption scheme where only $\mathsf{she.Mult}$ is defined, thus only satisfying correctness for Equation (II;3). We denote such schemes by ahe and mhe instead, respectively.

We define semantic security of she in exactly the same way as a standard public-key cryptosystem (Definition II;5.3).

Remark II;5.2. *If we omit mention of the levelled structure of the scheme, then we assume that the scheme is noiseless. This means that all ciphertexts live in the same space $\mathcal{Y}$.*

Scalar multiplication/exponentiation. It should be noted that we can define a *scalar multiplication* algorithm, $\mathsf{ScMult}(\cdot)$, for any ahe scheme. In particular, for $r \in \mathcal{X}$, then notice that

$$c_r \leftarrow \underbrace{\mathsf{ahe.Add}(\mathsf{pk}, c, \mathsf{ahe.Add}(\ldots \mathsf{ahe.Add}(\mathsf{pk}, c, c)))}_{(r-1)\text{ times}} = \mathsf{ahe.ScMult}(\mathsf{pk}, c, r)$$

for $c \leftarrow \mathsf{ahe.Enc}(\mathsf{pk}, m)$, satisfies $\mathsf{ahe.Dec}(\mathsf{sk}, c_r) \in \{m \cdot r, \perp\}$ by Definition II;5.6.

The method that we have shown here is necessarily naive and it is possible to improve on the efficiency of this operation by utilising the *double-and-add* method instead. This reduces the complexity of the scalar multiplication from $O(|\mathcal{X}|)$ to $O(\log(|\mathcal{X}|))$. Assuming perfect correctness, we can successfully induce scalar multiplication for any ahe scheme, using only additive homomorphisms. An equivalent notion is true for a mhe scheme, allowing scalar exponentiation; we denote this algorithm by ScExp.

Ciphertext rerandomisation

An important property that we require of ciphertexts output by SHE schemes is that they are *rerandomisable*. Informally, for any probabilistic SHE scheme, the ciphertext space $\mathcal{Y}$ is stratified into ranges $\mathcal{Y}_x[\iota]$ for each $x \in \mathcal{X}$ and $\iota \in [\ell]$. These ranges essentially correspond to the codomain of $\mathsf{she.Enc}(\mathsf{pk}, x, \iota; r)$ for all the possible choices of randomness $r \in \{0,1\}^*$. Here the notation $\mathcal{Y}_x[\iota]$ distinguishes the codomain from $\mathcal{Y}_x[\iota']$ where $\iota' \neq \iota$. Therefore, we only apply ciphertext rerandomisation on each disparate level of the SHE scheme.

```
$\mathsf{exp}^{\mathsf{crand}}_{0,\mathcal{A}}(1^\lambda, 1^\ell, \iota)$
1 : $x \leftarrow\!\$\ \mathcal{A}(\lambda, 1^\ell, \mathsf{pk}, \mathsf{sk})$;
2 : $b_\mathcal{A} \leftarrow \mathcal{A}^{\mathcal{O}_\mathcal{Y}(\mathsf{she.CRand}(\mathsf{pk},\cdot))}(1^\lambda, 1^\ell, \mathsf{pk}, \mathsf{sk})$;
3 : **return** $b_\mathcal{A}$;
```

```
$\mathsf{exp}^{\mathsf{crand}}_{1,\mathcal{A}}(1^\lambda, 1^\ell, \iota)$
1 : $x \leftarrow\!\$\ \mathcal{A}(\lambda, 1^\ell, \mathsf{pk}, \mathsf{sk})$;
2 : $b_\mathcal{A} \leftarrow \mathcal{A}^{\mathcal{O}_\mathcal{Y}(f_x(\cdot))}(1^\lambda, 1^\ell, \mathsf{pk}, \mathsf{sk})$;
3 : **return** $b_\mathcal{A}$;
```

Figure II;14: Experiments for establishing whether an SHE scheme she satisfies the crand security property for ciphertext rerandomisation. The function $f_x : \mathcal{K}_{\mathsf{pk}} \times \mathcal{Y}_x[\iota] \mapsto \mathcal{Y}_x(\iota)$ is defined as $f_x(\mathsf{pk}, c[\iota]) = \mathsf{she.Enc}(\mathsf{pk}, x)$

For any given ciphertext $c \in \mathcal{Y}_x[\iota]$ s.t. $x \leftarrow \mathsf{she.Dec}(\mathsf{sk}, c)$, we consider rerandomising c to correspond to choosing a new value $\widetilde{r} \in \{0,1\}^*$ and producing $\widetilde{c} \in \mathcal{Y}_x[\iota]$ s.t. $x \leftarrow \mathsf{she.Dec}(\mathsf{sk}, \widetilde{c})$ holds. This rerandomisation property is broadly used to destroy the dependence of a ciphertext on the operations that have been computed over it. In the following we write $c \leftarrow\!\$\ \mathcal{Y}_x[\iota]$ as shorthand for writing $c \leftarrow \mathsf{she.Enc}(\mathsf{pk}, x, \iota; r)$ for a randomly sampled r.

We define the required functionality as an additional algorithm $\mathsf{she.CRerand} : \mathcal{K}_{\mathsf{pk}} \times \mathcal{Y} \mapsto \mathcal{Y}$ for an SHE scheme she, with ciphertext space $\mathcal{Y}$. Formally the algorithm works as follows.

- $\mathsf{she.CRand}(\mathsf{pk}, c[\iota])$: Takes a public key pk, a ciphertext $c[\iota] \in \mathcal{Y}_x[\iota]$ (where $\iota \leq \ell$ where ℓ defines the upper bound on levels for she). Outputs a new ciphertext $c'[\iota] \in \mathcal{Y}_x[\iota]$.

Correctness of $\mathsf{she.CRerand}$ dictates that $x' \leftarrow \mathsf{she.Dec}(\mathsf{sk}, c'[\iota])$ satisfies $x' \overset{?}{=} x$, where $x \leftarrow \mathsf{she.Dec}(\mathsf{sk}, c[\iota])$. We denote by crand the security property that $\mathsf{she.CRerand}$ has to satisfy. We give experiments $\mathsf{exp}^{\mathsf{crand}}_{b,\mathcal{A}}(1^\lambda, 1^\ell, \iota)$ in Figure II;14 (for $b \in \{0,1\}$) that require a PPT adversary to distinguish between ciphertexts that are output by rerandomisation, and ciphertexts that are new encryptions of $x \in \mathcal{X}$.

We formalise the security requirement for ciphertext rerandomisation in Definition II;5.7. In the experiments, the adversary samples a valid key pair $(\mathsf{pk}, \mathsf{sk})$ and gives pk to the challenger. The adversary must also specify a plaintext value $x \in \mathcal{X}$ and makes ciphertext queries to the oracle $\mathcal{O}$, where each ciphertext must be an encryption of x. While this formulation of the game appears to be contrived, we could get around it by allowing the challenger to have access to the secret key sk.

Definition II;5.7 [Security]

Let she be an SHE scheme, and for a message $x \in \mathcal{X}$, let $\mathcal{Y}_x[\iota]$ define the codomain of the function $c \leftarrow \mathsf{she.Enc}(\mathsf{pk}, m, \iota)$ for $(\mathsf{pk}, \mathsf{sk}) \leftarrow_\$ \mathsf{she.KeyGen}(1^\lambda)$ and all possible choices of randomness. Then $\mathsf{she.CRerand}$ is a *ciphertext rerandomisation* algorithm, if:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{crand}(1^\lambda, \mathsf{she}))) < \mathsf{negl}(\lambda)$$

is satisfied for all PPT algorithms $\mathcal{A}$.

Now for any noiseless[11] SHE scheme she, then we show that $\mathsf{she.CRerand}$ is a ciphertext rerandomisation algorithm if all ciphertexts encrypting $x \in \mathcal{X}$ (at level ι) are distributed uniformly in $\mathcal{Y}_x[\iota]$.

Lemma II;5.3 [Uniform codomain distributions]

Let $c = c[\iota] \leftarrow \mathsf{she.Enc}(\mathsf{pk}, x, \iota)$ for $(\mathsf{pk}, \mathsf{sk}) \leftarrow_\$ \mathsf{she.KeyGen}(1^\lambda)$, any $x \in \mathcal{X}$ and any $\iota \in [\ell]$. Suppose also that all fresh encryptions of any $x \in \mathcal{X}$ are uniformly distributed in the subring $\mathcal{Y}_x[\iota] \subset \mathcal{Y}$. Then $c' \leftarrow \mathsf{she.Add}(\mathsf{pk}, c, c_0[\iota])$ satisfies:

$$(x \leftarrow \mathsf{she.Dec}(\mathsf{sk}, c')) \wedge (c' \approx_c \widetilde{c} \leftarrow_\$ \mathcal{Y}_0[\iota]);$$

where $c_0[\iota]$ is a valid encryption of 0 in $\mathcal{Y}_0[\iota]$. Thus, defining $\mathsf{she.CRand}(\mathsf{pk}, c) = \mathsf{she.Add}(\mathsf{pk}, c, \mathsf{she.Enc}(\mathsf{pk}, 0, \iota))$ satisfies the ciphertext randomisation property in Definition II;5.7

Proof. The proof is trivial, since if c_0 is uniformly distributed in the ring $\mathcal{Y}_0[\iota]$, then $c' \in \mathcal{Y}_x[\iota]$ must inherit the same distribution (providing that $\mathsf{she.Add}$ is implemented as ring operations as we assume). Moreover, the value of the plaintext associated with c_0 means that the homomorphic operation will not change the value of the encrypted plaintext. The result is that the oracles in $\mathsf{exp}^{\mathsf{crand}}_{b,\mathcal{A}}(1^\lambda)$ output computationally indistinguishable distributions and so

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{crand}(1^\lambda, \mathsf{she}))) < \mathsf{negl}(\lambda)$$

as necessary. □

Remark II;5.3. *Note that the experiments* $\mathsf{exp}^{\mathsf{crand}}_{b,\mathcal{A}}(1^\lambda)$ *only consider ciphertexts that successfully decrypt. We do not make security arguments about the ciphertext distribution where correctness errors can be introduced by the adversary. This rules out instantiating ciphertext rerandomisation*

[11] By noiseless we mean that $\ell = 1$.

using the multiplicative operation (which is theoretically possible providing that the level bound is not breached). This prevents the adversary from being able to instigate correctness errors themselves.

Plaintext rerandomisation

Another property that we require is that we can rerandomise the *plaintext* of an encryption. This clearly results in a changed plaintext value and so we do not enforce the same correctness requirement as in the ciphertext rerandomisation case. The aim is to allow a public-key holder to 'wash' ciphertexts s.t. an adversarial secret key holder cannot distinguish the ciphertext from a fresh encryption of a random value.

We define plaintext rerandomisation as an additional algorithm $\mathsf{she.PRerand} : \mathcal{K}_{\mathsf{pk}} \times \mathcal{Y} \mapsto \mathcal{Y}$ for an SHE scheme she with ciphertext space $\mathcal{Y}$. For this algorithm, we explicitly assumes that $\mathcal{X} = \mathbb{Z}_q$ is the ring of integers for some integer $q > 0$ (so that scalar multiplication is intuitively defined). Formally the algorithm works as follows.

- $\mathsf{she.PRand}(\mathsf{pk}, c[\iota], r)$: Takes a public key pk, a ciphertext $c[\iota]$ (where $\iota < \ell$ where ℓ defines the upper bound on levels for she) and a random scalar $r \in \mathbb{Z}$ as input. Outputs a new ciphertext $c'[\iota] \in \mathcal{Y}[\iota]$.

There is no explicit correctness requirement on the output $\mathsf{she.PRerand}$. For security, we use prand to denote the property that $\mathsf{she.PRerand}$ has to satisfy. We give experiments in Figure II;15 that require a PPT adversary to distinguish between ciphertexts that are output by $\mathsf{she.PRerand}$, and ciphertexts that are simply generated as encryptions of random plaintexts. We formalise the security requirement in Definition II;5.7.

We define the property formally in Definition II;5.8, in conjunction with the experiments in Figure II;15.

Definition II;5.8 [Plaintext rerandomisation]

We say that $\mathsf{she.PRerand}$ is a *plaintext rerandomisation* algorithm, if:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{prand}(1^\lambda, \mathsf{she}))) < \mathsf{negl}(\lambda)$$

is satisfied for all PPT algorithms $\mathcal{A}$.

We can instantiate $\mathsf{she.PRerand}$ by computing $\mathsf{she.PRand}(\mathsf{pk}, c, \iota) = \mathsf{she.ScMult}(\mathsf{pk}, c, r)$.

Rerandomisation for noisy SHE schemes. Almost all known SHE schemes (those that are lattice-based) incorporate some sort of noise into the encryption. This noise grows by the

```
$\mathsf{exp}^{\mathsf{prand}}_{0,\mathcal{A}}(1^\lambda, 1^\ell)$
$\mathsf{pk} \leftarrow \mathcal{A}(1^\lambda, 1^\ell, \mathsf{pk}, \mathsf{sk})$;
$c^\dagger[\iota] \leftarrow \mathcal{A}(1^\lambda, 1^\ell, \mathsf{pk}, \mathsf{sk})$;
$\mathbf{if}\ \iota > \ell$ :
    $\mathbf{return}\ \perp$;
$r \leftarrow\!\!\$\ \mathbb{Z}$;
$c'[\iota] \leftarrow \mathsf{she.PRand}(\mathsf{pk}, c^\dagger[\iota], r)$;
$b_\mathcal{A} \leftarrow \mathcal{A}(1^\lambda, 1^\ell, \mathsf{pk}, \mathsf{sk}, c^\dagger[\iota], c'[\iota])$;
$\mathbf{return}\ b_\mathcal{A}$;
```

```
$\mathsf{exp}^{\mathsf{prand}}_{1,\mathcal{A}}(1^\lambda, 1^\ell)$
$\mathsf{pk} \leftarrow \mathcal{A}(1^\lambda, 1^\ell, \mathsf{pk}, \mathsf{sk})$;
$c^\dagger[\iota] \leftarrow \mathcal{A}(1^\lambda, 1^\ell, \mathsf{pk}, \mathsf{sk})$;
$\mathbf{if}\ \iota > \ell$ :
    $\mathbf{return}\ \perp$;
$\widetilde{r} \leftarrow\!\!\$\ \mathcal{X}$;
$c'[\iota] \leftarrow \mathsf{she.Enc}(\mathsf{pk}, \widetilde{r}, \iota)$;
$b_\mathcal{A} \leftarrow \mathcal{A}(1^\lambda, 1^\ell, \mathsf{pk}, \mathsf{sk}, c^\dagger[\iota], c'[\iota])$;
$\mathbf{return}\ b_\mathcal{A}$;
```

Figure II;15: Experiments for establishing whether an SHE scheme she satisfies the crand security property for plaintext rerandomisation. Note that the adversary has access to both pk and sk so it can carry out arbitrary decryptions of its choosing.

homomorphic algorithms, but provided that it doesn't grow too much then decryption should compute correctly. However, this poses an interesting problem when performing rerandomisation because the adversary possesses the secret key. This means that the adversary can observe the manipulated noise on decryption, which reveals the operations that have been performed on the ciphertext and allows distinguishing the experiments in Figure II;14 or Figure II;15.

Indeed, a natural way of implementing $\mathsf{she.PRerand}$ without the additional computational complexity of performing $\mathsf{she.ScMult}$ is to instead compute it in the following way:

- $c' \leftarrow \mathsf{she.PRand}(\mathsf{pk}, c[\iota_c], r)$: output $\mathsf{she.Mult}(\mathsf{pk}, c, \mathsf{she.Enc}(\mathsf{pk}, r, [\iota_r]))$.

However, the ciphertext c' will now be on level $\iota_c + \iota_r$ due to the multiplication algorithm and the adversary with knowledge of sk can now distinguish the experiments in Figure II;15 by observing the noise distributions in the two experiments.

To get around this we can either make a modification to the game itself, or to the SHE scheme. For the game, the adversary could pro-actively compute $\mathsf{she.Enc}(\mathsf{pk}, r, \iota_c + \iota_r)$ so that the ciphertexts in both experiments have roughly the same noise-to-modulus ratio. Otherwise, we can utilise the concept of *bootstrapping* in SHE literature. This concept is usually used to map SHE schemes to fully homomorphic encryption (FHE) schemes. More specifically, it takes ciphertexts at level $\iota > 1$ and outputs a new ciphertext encrypting the same value at level 1. The concept of bootstrapping was first developed by Gentry [156] and has been further developed since [61, 62, 69, 105].

We do not formalise the bootstrapping process in this thesis as it is heavily involved and beyond the scope of the contributions of our work. Instead we simply argue that we can satisfy the rerandomisation notions by applying the bootstrapping algorithm to all rerandomised ciphertexts before they are sent to the adversary in the experiments. In short, when we consider SHE schemes going forward, we assume that they satisfy the security notions for both forms of rerandomisation. In terms of asymptotic efficiency, we consider the bootstrapping operation to be atomic, and only dependent on the security parameter (i.e. independent of other parameters including the size of the plaintext/ciphertext spaces).

Recall that we can instantiate ciphertext rerandomisation using an invocation of $\mathsf{she.Add}$. Since the level ι of the ciphertext does not change, we assume that bootstrapping is not necessary. This is not strictly true in general, since computing enough homomorphic additions can also increase the level of noise in actual SHE schemes. However, this can also be solved using a similar bootstrapping procedure.

Existing SHE schemes

The literature of homomorphic encryption schemes is extensive. Firstly the original definition of the RSA cryptosystem by Rivest et al. [268] is an example of a mhe scheme. Another well-known mhe scheme is the El Gamal cryptosystem [134]. Examples of ahe schemes include the Paillier [251] and Benaloh [39] cryptosystems, we describe the Paillier scheme below. There are also examples of cryptosystems that induce binary operations over plaintext, such as the Goldwasser-Micali encryption scheme [166] and the learning with errors cryptosystems of [233, 264].

Somewhat homomorphic encryption was a highly sought-after paradigm for over three decades after [268], until the seminal work of Gentry [156] using techniques from lattice-based cryptography. Since then there have been a number of steady improvements, mostly using similar techniques and relying on related lattice problems such as NTRU [186] and learning with errors [233, 264]. Examples of such schemes, include [62, 69, 105, 128, 162], but there are also many more.

Finally, in attempts to bridge the gap between $\mathsf{ahe}/\mathsf{mhe}$ and she, a number of works managed to realise a hybrid model of functionality; where a polynomial number of homomorphic additions, and one homomorphic multiplication, are permitted. Examples of such scheme are the BGN cryptosystem [51] and the cryptosystem of Gentry et al. [159].[12]

Paillier encryption scheme

Paillier's encryption scheme was first introduced in [251], with IND-CPA security relying on the hardness of DCR (Assumption II;3.3). The Paillier scheme is additively homomorphic, and is noiseless (i.e. $\ell = 1$). That is, we can compute unbounded numbers of additions over all cipher-

[12]The BGN cryptosystem [51] does not meet our requirements since the decryption algorithm cannot be completed in polynomial time. Concretely, the BGN cryptosystem requires solving the DL problem for exponents taken from the plaintext space, thus decryption is linearly expensive in the size of the plaintext space.

texts generated by $\mathsf{ahe.Enc}$. For this reason we omit the level parameters from the algorithmic inputs and outputs in the description of the construction below.

Construction II;5.2 [Paillier encryption scheme]

Let $\mathcal{X} = \mathbb{Z}_N$ where $N = pq$ for large primes p, q where $\mathsf{bitlength}(p), \mathsf{bitlength}(q) = \mathsf{poly}(\lambda)$. Let $\mathcal{Y} = \mathbb{Z}_{N^2}$, $\mathcal{K}_{\mathsf{pk}} = \mathbb{Z} \times \mathbb{Z}^*_{N^2}$, $\mathcal{K}_{\mathsf{sk}} = \mathbb{Z} \times \mathbb{Z}_N$. Define ahe in the following way.

- $\mathsf{ahe.KeyGen}(1^\lambda)$: Let $\nu = \mathsf{LCM}(p-1, q-1)$ i.e. ν is the Carmichael function of N. Sample $g \in \mathbb{Z}^*_{N^2}$ such that $\exists\, \mu = (L(g^\nu \bmod N^2))^{-1} \bmod N$ where $L(x) = \frac{x-1}{N}$; else output $(\bot, \bot)$. Output $(\mathsf{pk} = (N, g), \mathsf{sk} = (\nu, \mu))$.
- $\mathsf{ahe.Enc}(\mathsf{pk}, m \in \mathcal{X})$: Let $r \leftarrow_\$ \mathbb{Z}^*_N$; output $c = g^m \cdot r^N \bmod N^2$.
- $\mathsf{ahe.Add}(\mathsf{pk}, c_1 \in \mathcal{Y}, c_2 \in \mathcal{Y})$: Output $c_1 \cdot c_2 \bmod N^2$.
- $\mathsf{ahe.Dec}(\mathsf{sk}, c \in \mathcal{Y})$: Output $m = L(c^\nu \bmod N^2) \cdot \mu \bmod N$.
- $\mathsf{ahe.CRand}(\mathsf{pk}, c \in \mathcal{Y})$: Compute the ciphertext $c_0 \leftarrow \mathsf{ahe.Enc}(\mathsf{pk}, 0)$ and output $\mathsf{ahe.Add}(\mathsf{pk}, c, c_0)$.
- $\mathsf{ahe.PRand}(\mathsf{pk}, c \in \mathcal{Y}, r \in \mathbb{Z})$: Compute $\mathsf{ahe.ScMult}(\mathsf{pk}, c, r)$.

Remark II;5.4. *Note that, while the plaintext space is a ring, because all plaintexts live in $\mathbb{Z}_N$, the ciphertext space is actually interpreted as a group, since we only permit additions over the ciphertexts. Fortunately, this does not change the security analysis of the scheme and is merely a result of the scheme only being additively homomorphic.*

Lemma II;5.4 [Correctness]

Construction II;5.2 satisfies correctness wrt Definitions II;5.2 and II;5.6.

Proof. We refer the reader to [251] for the proof that correctness is satisfied, wrt Definition II;5.2. Recall that we write $\mathsf{ahe.Enc}(\mathsf{pk}, m; r)$ to denote that the randomness that is used in the algorithm is made explicit. For Definition II;5.6, notice that:

$$\begin{aligned}
\mathsf{ahe.Add}(\mathsf{pk}, c_1 \in \mathcal{Y}, c_2 \in \mathcal{Y}) &= c_1 \cdot c_2 \bmod N^2 \\
&= g^{m_1}(r_1)^N \cdot g^{m_2}(r_2)^N \bmod N^2 \\
&= g^{m_1+m_2}(r_1 r_2)^N \bmod N^2 \\
&= \mathsf{ahe.Enc}(\mathsf{pk}, m_1 + m_2; r_1 r_2)
\end{aligned}$$

where $c_j \leftarrow \mathsf{ahe.Enc}(\mathsf{pk}, m_j; r_j)$ for $j \in \{1, 2\}$. In the final line, we have a valid encryption of the message $m_1 + m_2$ with randomness $r_1 r_2$. Hence correctness is satisfied. □

The scheme is proven semantically secure under the assumed hardness of DCR.

Lemma II;5.5 [Security]

Construction II;5.2 is semantically secure under the DCR assumption (Assumption II;3.3).

Proof. We defer the reader to the original paper [251] for the security proof. The proof technique is not important wrt the results in this thesis. □

Ciphertext rerandomisation. Notice that $c \leftarrow \mathsf{ahe.Enc}(\mathsf{pk}, 0)$ is uniformly distributed in $\mathcal{Y}_0 \subset \mathbb{Z}_{N^2}$ since $\mathsf{ahe.Add}$ is just a ring multiplication. Therefore computing $\mathsf{ahe.Add}$ over a known ciphertext with some new encryption of zero results in a ciphertext encrypting the same value but with new randomness $(r_1 r_2)^N$. This is uniformly distributed wrt the randomness r_1^N. As a result, we argue that Construction II;5.2 satisfies the requirements for ciphertext rerandomisation via the result in Lemma II;5.3. recall that the Paillier scheme naturally gives secure ways of achieving ciphertext rerandomisation because there is a complete lack of a levelling structure in the encryption scheme. Therefore, all encryptions of x lie in the same ciphertext codomain $\mathcal{Y}_x \subset \mathcal{Y}$.

Plaintext rerandomisation. For a ciphertext $c \leftarrow \mathsf{ahe.Enc}(\mathsf{pk}, x)$, we can compute a scalar multiplication of the underlying plaintext using the $\mathsf{ahe.ScMult}(\mathsf{pk}, c, r)$ algorithm for some $r \in \mathbb{Z}$. This algorithm makes underlying calls to the $\mathsf{ahe.Add}$ algorithm and using the *double-and-add* method for performing scalar multiplications. Notice that $\mathsf{ahe.PRerand}$ uses the $\mathsf{ahe.ScMult}$ algorithm for implementing plaintext rerandomisation. On decryption the result is simply $r \cdot x$. Since this is the case for any encryption of x, then the Paillier scheme unconditionally satisfies the plaintext rerandomisation of Definition II;5.8.

Parameter settings. Essentially, the security parameter in this scheme is determined by the choice of $\mathsf{bitlength}(p)$, $\mathsf{bitlength}(q)$, when sampling the primes p and q. Clearly it is the case that $\mathsf{bitlength}(N) = \mathsf{bitlength}(p) + \mathsf{bitlength}(q)$. The bit-security of the Paillier scheme is decided by the length of N directly, assuming that the DCR assumption is as hard as factoring for the purposes of these choices. That is, for $\lambda = 112$ (i.e. equivalent 112 bits of security), we would roughly need $\mathsf{bitlength}(N) = 2048$; using the recommendations of NIST [247]. Similarly, for $\lambda = 80$, we could use $\mathsf{bitlength}(N) = 1024$. A more detailed outlook on parameter choices for the scheme is given in [200].

II;5.3 Digital signature schemes

Digital signature schemes provide an asymmetric variant of the authentication functionality provided by MACs. We use Definition II;5.9 to define a simple digital signature scheme; we only require the property that the signature scheme is secure against existential forgeries under adaptively-chosen messages, as in Definition II;5.11.[13] This is commonly known as EUF-CMA security [201, Definition 12.2]. The correctness requirement is given in Definition II;5.10.

Definition II;5.9 [Signature scheme]

Let $\mathsf{dss} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a tuple of algorithms, and let λ be a security parameter. A *digital signature scheme* (DSS) is defined in the following way.

- $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{dss.KeyGen}(1^\lambda)$: Outputs a key pair $(\mathsf{vk}, \mathsf{sk}) \in \mathcal{K}_{\mathsf{vk}} \times \mathcal{K}_{\mathsf{sk}}$.
- $c \leftarrow \mathsf{dss.Sign}(\mathsf{sk}, m)$: Let $\mathsf{sk} \in \mathcal{K}_{\mathsf{sk}}$, $m \in \mathcal{X}$; outputs a signature $\omega \in \mathcal{Y}$, or $\perp$.
- $b \leftarrow \mathsf{dss.Verify}(\mathsf{vk}, (m, \omega))$: Let $\mathsf{vk} \in \mathcal{K}_{\mathsf{vk}}$, $m \in \mathcal{X}, \omega \in \mathcal{Y}$; outputs $b \in \{0, 1\}$.

We refer to: $\mathsf{dss.KeyGen} : 1^\lambda \mapsto \mathcal{K}_{\mathsf{vk}} \times \mathcal{K}_{\mathsf{sk}}(\cup\{\perp\}^2)$ as *key generation*; $\mathsf{dss.Sign} : \mathcal{K}_{\mathsf{sk}} \times \mathcal{X} \mapsto \mathcal{Y} \cup \{\perp\}$ as *signing*; $\mathsf{dss.Verify} : \mathcal{K}_{\mathsf{vk}} \times \mathcal{X} \times \mathcal{Y} \mapsto \{0, 1\}$ as *verification*. We refer to: $\mathcal{X}$ as the *message* space; $\mathcal{Y}$ as the *signature* space; $\mathcal{K}_{\mathsf{vk}}, \mathcal{K}_{\mathsf{sk}}$ as the *verification/signing key* space, respectively.

Definition II;5.10 [Correctness]

Let dss be a digital signature scheme. We say that dss is *correct*, if it satisfies the following.

$$\Pr\left[\mathsf{dss.Verify}(\mathsf{vk}, (m, \omega)) \neq 1 \,\middle|\, \substack{(\mathsf{vk},\mathsf{sk}) \leftarrow \mathsf{dss.KeyGen}(1^\lambda);\\ \omega \leftarrow \mathsf{dss.Sign}(\mathsf{sk}, m \in \mathcal{X})}\right] < \mathsf{negl}(\lambda)$$

We define the experiment in Figure II;16 and define the advantage of $\mathcal{A}$ as $\mathsf{Adv}(\mathcal{A}, \mathsf{exp}^{\mathsf{euf}}_{\mathcal{A}}(1^\lambda))$. The formalisation of security that we use in Definition II;5.11 is taken from [201]. Like the definition of security for a MAC, we specify a stronger version of security that φ contains $(m^\dagger, \omega^\dagger)$ (i.e. including the signature $\omega^\dagger$ as well). The weaker definition would only include $m^\dagger$ in φ.

[13] The adversary chooses the messages that are signed adaptively.

```
$\mathsf{exp}^{\mathsf{euf}}_{\mathcal{A}}(1^\lambda)$
1 : $\varphi \leftarrow \emptyset$;
2 : $(\mathsf{vk}, \mathsf{sk}) \leftarrow_\$ \mathsf{dss.KeyGen}(1^\lambda)$;
3 : $(m^\dagger, \omega^\dagger) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathcal{X}}(\mathsf{dss.Sign}(\mathsf{sk},\cdot),\varphi)}(1^\lambda, \mathsf{vk})$;
4 : if $((m^\dagger, \omega^\dagger) \notin \varphi) \wedge (1 \leftarrow \mathsf{dss.Verify}(\mathsf{vk}, m^\dagger, \omega^\dagger))$ :
5 :     return $1$;
6 : else :
7 :     return $0$;
```

Figure II;16: EUF-CMA security model for a digital signature scheme, defined using the computational experiment $\mathsf{exp}^{\mathsf{euf}}_{\mathcal{A}}(1^\lambda)$. We modify the set φ so that it also holds the output that is returned corresponding to a query m

Definition II;5.11 [Existential forgeries]

Let dss be a digital signature scheme, let $\mathsf{exp}^{\mathsf{euf}}_{\mathcal{A}}(1^\lambda)$ be the experiment defined in Figure II;16. We say that dss is *secure against existential forgeries under adaptively-chosen messages*, or that it satisfies EUF-CMA security, if

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{euf}(1^\lambda, \mathsf{dss}))) < \mathsf{negl}(\lambda)$$

holds, where $\mathcal{A}$ is any PPT algorithm.

II;6 Secure computation

In this work, we will be concerned with both interactive and non-interactive methods of computation. By *interactive* secure computation, we are referring to the strand of secure computation better known as multi-party computation and introduced in [84, 165, 296]. Online protocols are constructed for computing functionalities between multiple parties. By *non-interactive* secure computation, we refer to functionalities that can be evaluated in-person by an evaluator, after some initial pre-computation. For example, as laid out in the case of program obfuscation [28].

Indeed, in Chapter III, Chapter IV and Chapter VI, we will consider interactive protocols between two participants for learning the outputs of certain functions. In Chapter V and Chapter VII we will consider non-interactive functionalities.

II;6.1 Interactive computation

An original goal of cryptography was to protect communications between two or more participants. The study of secure computation augments this goal with the objective of protecting the computation between two or more participants. Notice that we can easily protect this computation if we allow for a trusted third party. This party is given all input data and simply computes the required function and returns the relevant output to each party. Unfortunately, creating trust relationships of these kinds between relative strangers can be difficult.

Interactive secure computation seeks to provide privacy-preserving methods of computing functionalities between multiple participants, without the need for additional trust assumptions. That is, only the parties that provide input to the functionality take part in the protocol. In this setting, privacy-preserving means that the inputs of the participants are not revealed to the others. The study of cryptographic constructions and protocols relies on the key tenet that it is advantageous to remove as many trust requirements as is possible.[14] While there may be some disagreements on the potential advantage that is gained; we assume that removal of all third party computation is favourable.

Before we go on to describe the security model for secure computation of this form, we introduce the formal concept of a protocol.

Definition II;6.1 [Protocol]

Let $\{\mathbb{P}_j\}_{j\in[N]}$ be a set of participants, with respective inputs $x_j \in \mathcal{X}_j$. Let aux_j be auxiliary information known by $\mathbb{P}_j$. Let $F : \mathcal{X}_1 \times \ldots \times \mathcal{X}_N \mapsto \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_N$ be some functionality, s.t. $F(x_1, \ldots, x_N) = (y_1, \ldots, y_N)$.
Then a *protocol* for F is a set of algorithms denoted by ψ_F, such that each participant $\mathbb{P}_j$ inputs x_j and learns the view:

$$\mathsf{View}_j = (x_j, \mu_j, \mathsf{msgs}_j, \mathsf{aux}_j),$$

where $\mu_j = y_j$, and msgs_j is the execution of the protocol witnessed by $\mathbb{P}_j$

Remark II;6.1. *We use the convention that each participant $\mathbb{P}_j$ may learn different output information from the associated ranges $\mathcal{Y}_j$.*

We can now define the correctness and security requirements for such a protocol. These formalisations are used widely throughout secure computation literature (for example [226, 243]), but

[14] Or, in so far as reducing them to trust assumptions on underlying computational assumptions that appear to be difficult.

appeared firstly in [165, 296] after generalising similar formalisations of security by Goldreich et al. [167] for interactive proof systems. A more holistic overview can be found in [107].

Definition II;6.2 [Correctness]

Let $F : \mathcal{X}_1 \times \ldots \times \mathcal{X}_N \mapsto \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_N$ be a functionality. Let ψ_F be a protocol that, on given inputs $x_j \in \mathcal{X}_j$ provided by participants $\mathbb{P}_j$, and returns $\mathsf{View}_j \to \mathbb{P}_j$, where $\mathsf{View}_j = (x_j, \mu_j, \mathsf{msgs}_j, \mathsf{aux}_j)$. We say that ψ_F *correctly* computes F if:

$$\Pr\left[y_j = \mu_j \in \mathsf{View}_j \middle| \{y_j\}_{j\in[N]} \leftarrow F(x_1, \ldots, x_N)\right] > 1 - \mathsf{negl}(\lambda),$$

for all inputs $x_j \in \mathcal{X}_j$, where $j \in [N]$.

In the above definition, we implicitly assume that ψ_F is aware of the contents of each aux_j.

Security models. To assess the security of a given protocol ψ_F in computing the function F, we analyse the plausibility of constructing an adversarial algorithm that 'corrupts' one or more participants and attempts to learn the inputs of any 'non-corrupted' participants. A corruption refers to the event that an adversary receives the entire view of the corrupted participant, and can possibly even choose their protocol inputs. There are differing security models give the adversary varying levels of capabilities. The stronger the adversarial assumptions, the weaker the guarantees that ψ_F satisfies. Adversarial restrictions can be categorised into the following:

- whether they can divert from the protocol specification;
- the number of allowed corruptions that they can make;
- whether the corruptions are made selectively or adaptively.

An intuitive security model assumes that security is analysed with respect to the 'ideal functionality' F. In a nutshell, the protocol is deemed to be secure if an adversary, that corrupts one or more participants, learns as much in the execution of ψ_F as they would if the participants sent x_j to a trusted party $\mathbb{T}_F$. For clarity, $\mathbb{T}_F$ computes $\{y_j\}_{j\in[N]} = F(\{x_j\}_{j\in[N]})$ and returns y_j to $\mathbb{P}_j$. Formally, the view of $\mathcal{A}$ in the real execution, is indistinguishable from the view in the ideal setting. A 'corruption' in this context allows the adversary to learn the inputs of the corrupted participants, and further messages that they receive (in other words, the *view* of that participant, View_j in Definition II;6.2). Corruptions are either chosen 'selectively' (specified at the start of the protocol) or 'adaptively' (during the protocol).

The model described above is actually the strongest security model possible, since a proof in this model implies that any attack against the protocol implies an attack in the ideal setting. Thus, the adversary can attempt any arbitrary strategy during the execution of ψ_F — they do not have to obey the protocol specification. In other words, the adversary is *malicious*.

A weaker security model assumes that the adversary *obeys* the structure of the protocol, and attempts to learn more by inspecting the *view* of the protocol that they witness. That is, the adversary receives the views of all the participants that they have corrupted. If the adversary can learn more about the uncorrupted input sets from their view, than is implied by the knowledge of their inputs and outputs alone, then the protocol is deemed to be insecure.

More precisely, if the real execution is indistinguishable from a simulation that uses only the inputs and outputs of $\mathcal{A}$ to simulate the view of $\mathcal{A}$, then ψ_F is secure. The adversary here is deemed to be *semi-honest*. The adversary is 'honest' in the sense that the protocol execution is abided by and 'dishonest' in the sense that they still attempt to subvert the security of other participants. The semi-honest security model is regarded as a plausible adversarial model if the protocol software is attested regularly, or if it is hosted on externally trusted hardware, for example.

In the following definitions, let $F : \mathcal{X}_1 \times \ldots \times \mathcal{X}_N \mapsto \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_N$ be a functionality. In addition, let ψ_F be a protocol that computes $\{y_j\}_{j \in [N]} = F(x_1, \ldots, x_N)$ on given inputs $x_j \in \mathcal{X}_j$ provided by participants $\mathbb{P}_j$; and returns $\mathsf{View}_j \to \mathbb{P}_j$, where $\mathsf{View}_j = (x_j, y_j, \mathsf{msgs}_j, \mathsf{aux}_j)$; msgs_j is the list of messages seen by $\mathbb{P}_j$; aux_j is arbitrary auxiliary data known by $\mathbb{P}_j$. Let J be the set of indices $j \in [N]$ corresponding to participants that have been corrupted, and let $J' = [N] \setminus J$. Let $\mathsf{msgs}_{J' \leftarrow J}$ be the set of messages received by each uncorrupted participant $\mathbb{P}_{j'}$ from each corrupted participant $\mathbb{P}_j$, where $j' \in J'$ and $j \in J$.

Firstly, we give the formal security model for analysing the security of protocols in the presence semi-honest adversaries.

Definition II;6.3 [Semi-honest security]

We say that ψ_F securely computes F, in the presence of *semi-honest* adversaries, if there exists a PPT simulator, Sim, such that:

$$\left\{\mathsf{View}_j \approx_c \mathsf{Sim}(1^\lambda, x_j, y_j, \mathsf{aux}_j, \mathsf{msgs}_{J' \leftarrow J})\right\}_{j \in J}$$

for any PPT adversary $\mathcal{A}$ that selectively corrupts one or more participants.

Intuitively, the security model dictates that the protocol messages msgs_j should be implied by the rest of the knowledge given to Sim by the inputs and outputs of $\mathbb{P}_j$. Notice, that Sim still receives the messages, $\mathsf{msgs}_{J' \leftarrow J}$, that uncorrupted participants $\mathbb{P}_{j'}$ (for $j' \in J'$) would receive during the protocol from corrupted participants $\mathbb{P}_j$, for $j \in J$.

Secondly, we give the formal security model for analysing security against malicious adversaries.

Definition II;6.4 [Malicious security]

We say that ψ_F securely computes F, in the presence of *malicious* adversaries, if there exists a PPT simulator, Sim, such that:

$$\left\{\mathsf{View}_j \approx_c \mathsf{Sim}^{\mathcal{O}_{x_1,\ldots,x_N}(F(\cdot,\ldots,\cdot))}(1^\lambda, \mathsf{aux}_j, \mathsf{msgs}_{j'})\right\}_{j\in[N]}$$

for any PPT adversary $\mathcal{A}$ that selectively corrupts one or more participants.

We assume that F takes fixed inputs $x_{j'}$ for $j' \in J'$. That is, Sim, has access to an oracle that takes fixed inputs for uncorrupted participants and adversarial inputs for corrupted participants (based on what is received in $\mathsf{msgs}_{j'}$).

Remark II;6.2. *We could be more prescriptive with the requirement on the number and timing of corruptions. However, this will not feature prominently in our later analysis — we will assume that all corruptions are made selectively going forwards. By a selective corruption, we assume that an adversary chooses the subset J of corrupted participants before the protocol takes place. This is sometimes known as modelling* static *corruptions. The adversary then learns all the information that is highlighted above. Equally, we could impose fewer restrictions on the adversary by specifying that the distributions in Definitions II;6.3 and II;6.4 are statistically close, or even equivalent.*

Specific notation. Throughout later chapters, we will generically consider two-party protocols denoted by ψ. In ψ, we will denote the two parties by $\mathbb{P}_1$ and $\mathbb{P}_2$, who have inputs defined by sets $\mathbb{S}_1, \mathbb{S}_2 \subset \mathcal{S}$, respectively; where $\mathcal{S}$ is some universe that will be defined by context.[15] In the following, we will sometimes use the notation $\mathbb{P}_j$ and $\mathbb{P}_{2j-3(j-1)}$ to denote the two opposing participants, where $2j - 3(j-1) = 2$ when $j = 1$, and $2j - 3(j-1) = 1$ when $j = 2$. We will adopt shorter notation and write j and $\overline{j} = 2j - 3(j-1)$, instead.

II;6.2 Non-interactive setting

In the non-interactive setting, let $\mathbb{P}_F$ hold a secret input to the two-input function F. Then we investigate ways by which $\mathbb{P}_F$ can encode the partial evaluation $F(\mathsf{sp}, \cdot)$ as some new function $\widetilde{F}(\cdot)$. Then, for correctness we require that:

$$\widetilde{F}(x) = F(\mathsf{sp}, x)$$

[15] We will typically use $\mathcal{S} = \mathbb{Z}_q$ for some integer $q > 0$.

for all valid inputs x. The security requirement of the encoding procedure is that the secret input sp is never revealed. Research into constructions of this form has been termed the study of *program obfuscation* since the work of Barak et al. [28] — because $\widetilde{F}$ represents an obfuscated form of F. The requirement can be made even stronger, stating that $\widetilde{F}$ does not even reveal any information about F (other than that which is revealed by the output distribution of the function). Subsequent to this seminal research, there have been a huge number of works claiming to offer program obfuscation for various functionalities, various definitions of security and under various assumptions [27, 70, 150, 153, 168, 172, 240, 272, 294]. However, the area is still in its infancy, much less investigation has been dedicated to realising potentially practical designs or in-depth cryptanalysis. This setting is considered in Chapter VII, and so we defer the security and correctness requirements until then.

We also consider slightly different forms of non-interactive computation, where $\mathbb{P}_F$ can instead modify the secret input sp to some *constrained* form $\widetilde{\mathsf{sp}}$. These constrained parameters allow computing $F(\widetilde{\mathsf{sp}}, \cdot)$ for some subset of inputs x. Moreover, $\widetilde{\mathsf{sp}}$ does not reveal anything about the original input sp.[16] This setting is considered in Chapter V. Again, we only consider the case of constrained PRFs, and so we leave the specific formalisation until the chapter itself.

[16]The function itself is assumed to be known by the adversary.

III An Efficient Toolkit for Computing Private Set Operations

delusions of grandeur

Preface

In this chapter we will construct two-party protocols that allow performing a range of operations over data sets in a 'secure' manner. The operations that we consider are set union, intersection and union/intersection cardinality. Our proposed protocols all use the same cryptographic 'machinery', and thus our constructions can be seen as a generic 'toolkit' for computing multiple set operations between two participants.

Our constructions can be instantiated with simple black-box cryptographic primitives; IND-CPA secure homomorphic encryption schemes (providing, at least, additive homomorphisms over the ciphertext space) and Bloom filters. The level of homomorphism needed represents a trade-off, where lower asymptotic complexity can be traded for more requirements on the homomorphic capability of the encryption scheme. Under additively homomorphic encryption scheme, our scheme is asymptotically super-linear in the size of input sets; but the primitive is much simpler and efficient to run. The opposite is true for homomorphic schemes that allow multiplications over the plaintext space. Specifically the protocol is asymptotically linear, but the underlying primitive is more complex. Nevertheless, we create the first private set union protocol that scales linearly in the input set sizes; and subsequently for the entire toolkit as a whole.

We give a proof-of-concept implementation of our scheme in the language `Go`, instantiated using the Paillier encryption scheme. Our implementation is remarkably simple — with a distinctly modular design reflecting the reuse of the same primitives for each separate protocol — and is quick to run for varieties of set sizes, including those containing ≥ 100000 distinct elements.

The work displayed in this chapter is based heavily on the results of the following publication [117] [ACISP2017], along with the full version of the manuscript [118]. In this thesis we provide more constructions of our protocols from different primitives — also considering instantiations using somewhat homomorphic encryption schemes where [117] only considered additively homomorphic. In addition, we give more explicit proof arguments that were previously not possible due to space constraints for the publication. We also provide a more rigorous treatment of the primitives that we use. Finally, we correct an erroneous claim that was made in [117] relating to the asymptotic complexities of the protocols, later in Section III;7.2.

Overview of original contributions

- Two-party protocols for computing private set {union (Section III;4), intersection (Section III;5), union-cardinality, intersection-cardinality (Section III;6)} based on additively (or somewhat) homomorphic encryption schemes. Security is proven in the presence of adversaries in the semi-honest security model (Definition II;6.3).

- The first private set union protocol with linear asymptotic computational complexity in the size of the sets, based on somewhat homomorphic encryption (Section III;7).
- A practical implementation (in `Go`) of each of the protocols for real-world parameter sets, based on the Paillier, additively homomorphic, encryption scheme (Section III;8).
- Variants of our protocols that are secure in an 'authenticated' setting, against a malicious adversary that corrupts one of the participants (Section III;9). This requires an extra assumption on the existence of a digital signature scheme satisfying EUF-CMA security.

Contents

III;1 Introduction

III;1.1 Sets and operations

Consider some universe $\mathcal{S}$ of data, and subsets $\mathbb{S} \subset \mathcal{S}$.[1] Naturally, it may be useful to compute functions over multiple subsets to learn some prescribed output from the combined data sets. *Set operations* provide the fundamental mathematical basis for the computation of functionalities over data sets. We provide a (somewhat) formal specification of this functionality in Definition III;1.1.

Definition III;1.1 [Set operations]

Let $\ell \in \mathbb{N}$; and let $\mathbb{S}_1, \ldots, \mathbb{S}_\ell \subseteq \mathcal{S}$, then a function $F : \mathcal{S}^L \mapsto \Delta$ is a *set operation* if it takes $(\mathbb{S}_1, \ldots, \mathbb{S}_\ell)$ and outputs some result $\mu \in \Delta$. Common examples of Δ include: $\Delta = \mathcal{S}$ (i.e. μ is a new subset); $\Delta = \mathbb{N}$ (i.e. μ is a numerical result).

In practice, there are some very common operations that can form the basis of more complicated computations. In the following definitions: let $\ell \in \mathbb{N}$; let $\mathbb{S}_1, \mathbb{S}_2 \subseteq \mathcal{S}$; let $n_j = |\mathbb{S}_j|$ and let:

$$\mathbb{S}_j = \{x_i^{(j)}\}_{i \in [n_j]}$$

for $j \in \{1, 2\}$ and $x_i^{(j)} \in \mathcal{S}$.

Now we define the fundamental operations of 'set union' and 'set intersection'.

Definition III;1.2 [Set Union]

Denote by $\cup$ the *set union* operation, where:

$$F_\cup(\mathbb{S}_1, \mathbb{S}_2) = \mathbb{S}_1 \cup \mathbb{S}_2 = \left\{ \{x_{i_1}^{(1)}\}_{i_1 \in [n_1]}, \{x_{i_2}^{(2)}\}_{\substack{i_2 \in [n_2], \\ x_{i_2}^{(2)} \neq x_{i_1}^{(1)} \forall i_1 \in [n_1]}} \right\}$$

and $F_\cup : \mathcal{S}^2 \mapsto \mathcal{S}$ is the functional realisation of this functionality.

[1] For now we will assume that each element in $\mathcal{S}$ is unique, i.e. there are no elements that appear twice. If we allowed non-distinct elements, then we would refer to subsets $\mathbb{S}$ as multi-sets.

Definition III;1.3 [Set Intersection]

Denote by $\cap$ the *set intersection* operation, where:

$$F_\cap(\mathbb{S}_1, \mathbb{S}_2) = \mathbb{S}_1 \cap \mathbb{S}_2 = \left\{ \{x^{(1)}_{i_1}\}_{\substack{i_1 \in n_1, \\ \exists\, i_2 \in [n_2] \text{ s.t. } x^{(1)}_{i_1} = x^{(2)}_{i_2}}} \right\}$$

and $F_\cap : \mathcal{S}^2 \mapsto \mathcal{S}$ is the functional realisation of this functionality.

Remark III;1.1. *The reader may notice that the definitions are asymmetric in the sense that $\mathbb{S}_1$ is the 'dominant' set in choosing which entries are included. However, the result is identical if $\mathbb{S}_2$ is chosen as the dominant set, and so this is wlog.*

Without delving into an extensive review, the union and intersection operations are the set-based representations of the 'AND' ($\wedge$) and 'OR' ($\vee$) logic gates that, together, satisfy functional completeness. As a result, it is not hard to see that the operations can satisfy many differing requirements when combining and operating over data sets for performing statistical analyses.

We can further define two, more limited, operations that only output the cardinality of the output sets above. In particular, in practical applications that combine data sets, sometimes it is only necessary to learn the size of the resulting set after the operation takes place; if the explicit output set is not required.

Definition III;1.4 [Set Union Cardinality]

Denote by $F_{|\cup|} : \mathcal{S}^2 \mapsto \mathbb{N}$ the *set union cardinality* operation, where:

$$F_{|\cup|}(\mathbb{S}_1, \mathbb{S}_2) = |\mathbb{S}_1 \cup \mathbb{S}_2|.$$

Definition III;1.5 [Set Intersection Cardinality]

Denote by $F_{|\cap|} : \mathcal{S}^2 \mapsto \mathbb{N}$ the *set intersection cardinality* operation, where:

$$F_{|\cap|}(\mathbb{S}_1, \mathbb{S}_2) = |\mathbb{S}_1 \cap \mathbb{S}_2|.$$

It is no coincidence that, in this chapter, we will only be considering set operations taken from the definitions above (or derived from these with additional, as yet unspecified, characteristics).

III;1.2 Private set operations

Let us focus on the case where we have two organisms $\{\mathbb{P}_j\}_{j\in[2]}$ (referred to as the 'participants'), respectively controlling sets $\{\mathbb{S}_j\}_{j\in[2]}$. While we have defined some functions that allow $\mathbb{P}_1$ and $\mathbb{P}_2$ to combine their sets, we have not defined any situations where the proposed functions have any utility. For instance, if the participants want to compute the intersection of their sets, how can they do this without revealing their entire set to the other participant? Once $\mathbb{S}_j$ is revealed to $\mathbb{P}_{\overline{j}}$, then the operation is meaningless since $\mathbb{P}_{\overline{j}}$ can simply compute anything that they want.[2]

One trivial solution involves locating and establishing a trusted third party, $\mathbb{T}$. Then $\mathbb{P}_j$ sends $\mathbb{S}_j$ to $\mathbb{T}$ who computes $\mu_* = F_*(\mathbb{S}_1, \mathbb{S}_2)$ and returns μ_* to both participants, for some set operation $*$. This solves the utility issue, since $\mathbb{P}_{\overline{j}}$ no longer learns the entirety of $\mathbb{S}_j$. However, a more interesting question can now be asked:

How do we establish and implement the functionality of a trusted third party?

For instance, in a real world scenario, it is likely to be difficult to find a trusted party for any two individuals on the planet. Not only that, but we require the additional assumption that the trust dynamic will also not change in the future. In some cases, the 'privacy' of the set operation — the extent to which $\mathbb{S}_j$ should be kept secret from $\mathbb{P}_{\overline{j}}$ — may not be very important. Equally, there are many scenarios where learning the output of $F_*(\mathbb{S}_1, \mathbb{S}_2)$ is valuable, but revealing $\mathbb{S}_j$ to $\mathbb{P}_{\overline{j}}$ is intolerable. It is these situations in the latter case that we explore further in this work. Commonly, the assumption that a viable trusted third party exists is too strong for our participants.

In this void, we see the potential for a cryptographic construction to replace the trusted third party in the computation above. In fact, we will explore the feasibility of constructing a protocol (Definition II;6.1) that $\mathbb{P}_1$ and $\mathbb{P}_2$ can participate in, that upholds the utility of the set operation being considered. Note that we could achieve this with a general secure computation protocol for computing F_*. However, there are overheads that we can overcome if we target the functionality of F_* directly.

We will require that the protocol is secure wrt the security model from Section II;6. In practice, the protocol should enable the participants to learn the outcome of $F_*(\mathbb{S}_1, \mathbb{S}_2)$, while learning minimal information about $\mathbb{S}_{\overline{j}}$ beyond what is revealed by the output itself.

It is important to stress this point regarding security wrt the output, because some operations can reveal almost everything about the unknown set. Consider an extreme case where the two parties know the cardinality, $n_{\overline{j}}$, of the other set $\mathbb{S}_{\overline{j}}$. Then if — after computing the intersection of the sets — the cardinality of the output is $n_\cap = n_{\overline{j}}$, then $\mathbb{P}_j$ knows that it has learnt the entirety of $\mathbb{S}_{\overline{j}}$. Therefore, the security requirement only has utility itself when the output of the function

[2]Recall that $\overline{j} = 2j - 3(j-1)$, so that $\overline{j} = 2$ if $j = 1$, and likewise $\overline{j} = 1$ if $j = 2$.

does not systematically imply the inputs. We give these requirements formally in Section III;1.4 as adaptations of the security model given in Section II;6.1.

Remark III;1.2. *We may generically refer to the PSO for the union operation as private set union (PSU), and likewise private set intersection (PSI) for intersection; private set union cardinality (PSU-CA) for union cardinality; and private set intersection cardinality (PSI-CA) for intersection cardinality. Moreover, we may sometimes combine PSI-CA and PSU-CA into one operation (PSU/I-CA). Notice that*

$$|\mathbb{S}_1 \cup \mathbb{S}_2| = |\mathbb{S}_1| + |\mathbb{S}_2| - |\mathbb{S}_1 \cap \mathbb{S}_2|, \tag{III;1}$$

and thus, as long as the cardinality $|\mathbb{S}_{\bar{j}}|$ is known to $\mathbb{P}_j$ for $j \in \{1, 2\}$, then PSU-CA $\implies$ PSI-CA, and vice-versa.[3]

Usefulness of PSU. One subtle point that we should endeavour to clarify is whether a PSU protocol is useful. For instance, the union of two sets is simply all the elements that appear in either set at least once. Perhaps it would make sense if the participants simply swapped sets rather than computing a specific protocol? We answer this question in the negative, pointing out that there may be many situations where the intersection of the two sets should be hidden — but where the union of the two sets is required. We give an example in Section III;1.3 of such a real-world application, but note that this is a plausible scenario necessitating non-trivial functionality.

Roadmap. The study of protocols for computing private set operations, in general, has been an active area of research for over three decades since the work of Meadows [236]. We summarise both previous and active work in the area in Section III;2.1. Before that, we consider potential applications of PSO protocols and clarify the security models that we consider.

III;1.3 Applications of private set operations

The need to ensure the privacy of secret data from foreign systems is a fundamental goal in both the academic and industrial sectors of cyber security. On the other hand, the utility gained by computing over combined sets of data is undoubtedly non-negligible. For instance, at the state- and corporate-level, data sharing is a vital tool for learning more about the scope of common threat actors and industry competition. The two needs of privacy and sharing are fundamentally opposed: the first asks that data sets be kept private from foreign agencies; the second asks that the information in private data sets be shared to enhance collaboration, i.e. gaining shared usefulness from combined knowledge. However, it may be possible to undertake some limited sharing, while keeping certain parts of data secret.

[3]Learning the cardinality of the opposing set is a widely assumed necessity. Preventing learning of this information is a separate goal for many protocols.

The use of cryptographic PSO protocols (and secure computation protocols, in general) is motivated by the above. The functionality that PSO protocols allow make them vital components of large-scale computations over multiple data sets that are required to enshrine the privacy of the various input sets into formal guarantees [227]. In particular, in the case of data mining over multiple data sources, a computation can usually be segmented into the various set operations that need to take place at each stage. In the end, a final output is derived after each required operation has taken place. This allows a level of 'data sharing' that still keeps elements of data sets private, where they are not part of the output of the operation. In essence, this is the best possible security that we can hope for in such a situation.

The following scenarios provide a brief, non-exhaustive view of some concrete applications that PSO protocols may be useful for.

- Law enforcement: An airline must alert law enforcement authorities if anyone who has been placed on a no-flight list attempts to board a flight. The airline may not want to give away customer data unnecessarily, and the authorities must not give away data on criminals to the airlines. A private set intersection (PSI) protocol allows the two entities to compare the sets of people they are aware of, and learn only the intersection.

- International crime: Two or more states may want to combine the information they have about criminals operating on an international scale. It benefits all parties to know about all the individuals operating at this level, as they could target any of the countries involved. That being said, neither countries want to reveal more information than is required. The entities can use a private set union (PSU) protocol to learn all of their combined data.

- Changes in customer base: An industry sector wants to evaluate whether the customer base has grown or declined over a certain time period. To do this, the major entities in the sector must combine the number of customers they have currently together. The companies can enact a private set union cardinality (PSU-CA) protocol on their set of customers, to learn the combined number of customers in the sector.

- Ad-space usage: An online marketplace selling ad-spaces on their web platform seeks to find out how many new customers current ad-space purchasers gain to give figures out to prospective buyers. In this case, the marketplace and the prospective buyers can perform a private set intersection cardinality (PSI-CA) protocol to learn the number of common customers that they both have.

In terms of the academic literature, PSOs (although mostly PSI) have also been useful in constructing solutions for privacy-preserving variants of:

- Proximity-testing [245];

- Botnet detection [242];
- Detecting genetic relatives using rare variants [188];
- Predictive blacklisting [145].

Again, this list is non-exhaustive, and provides just an idea of the scope of applications that can be considered.

Efficiency. The applications above are only viable if the PSO protocols are efficient to run, in comparison with insecure solutions. To achieve this, it is first common to seek cryptographic solutions with comparable asymptotic efficiency. While asymptotic estimates hide large constants that can be critical to understanding runtimes, they are a good indication of how cryptographic designs will scale with increasing security and input length parameters. These considerations will be fundamental to the contributions that we introduce in this chapter.

III;1.4 Security models for PSOs

We can tailor Definition II;6.3 and Definition II;6.4 for protocol security models so that they are dedicated specifically to the case where F is computing a set operation. The only modifications that we essentially consider are that the protocol inputs x_j are subsets of some general universe $\mathcal{S}$. For completeness, we will give explicit formalisations of the definitions for the PSO case, we will also redefine the notion of correctness given in Definition II;6.2. The formalisation of the security models in this way for PSOs was established by the work of Freedman et al. [143] (adapted from Naor and Pinkas [243]), even though the first construction by Meadows [236] originated in 1986.

In the following definitions, let $F_* : \mathcal{S}^N \mapsto \Delta^N$ be a functionality for some generic output space Δ and a set operation $*$. Let ψ_* be a protocol that computes $\{\mu_j\}_{j\in[N]} \leftarrow F_*(\mathbb{S}_1, \ldots, \mathbb{S}_N)$ for $\mu_j \in \Delta$; input sets $\mathbb{S}_j \in \mathcal{X}_j$ provided by participants $\mathbb{P}_j$; and returns $\mathsf{View}_j \to \mathbb{P}_j$, where $\mathsf{View}_j = (\mathbb{S}_j, \mu_j, \mathsf{msgs}_j, \mathsf{aux}_j)$; msgs_j is the list of messages seen by $\mathbb{P}_j$; aux_j is arbitrary auxiliary data known by $\mathbb{P}_j$. We use the set $J \subset [N]$ to refer to all corrupted participants, and $J' = [N] \setminus J$ for those that are not. We may also write $\mathsf{msgs}_{J' \leftarrow J}$ to denote the messages received by uncorrupted participants in J' from corrupted participants in J.

Firstly, we give an adapted notion of correctness.

Definition III;1.6 [Correctness]

We say that ψ_* *correctly* computes the set operation $*$, if:

$$\Pr\left[y_j = \mu_j \in \mathsf{View}_j \middle| \{y_j\}_{j\in[N]} \leftarrow F(\mathbb{S}_1, \ldots, \mathbb{S}_N)\right] > 1 - \mathsf{negl}(\lambda),$$

for all inputs $\mathbb{S}_j \in \S_j$, where $j \in [N]$.

Secondly, we give adapted notions of achieving security in both security models.

Definition III;1.7 [Semi-honest security (PSO)]

We say that ψ_* securely computes the set operation $*$, in the presence of *semi-honest* adversaries, if there exists a PPT simulator, Sim, such that:

$$\left\{\mathsf{View}_j \approx_c \mathsf{Sim}(1^\lambda, \mathbb{S}_j, \mu_j, \mathsf{aux}_j, \mathsf{msgs}_{J'\leftarrow J})\right\}_{j\in J}$$

for any PPT adversary $\mathcal{A}$, where $|J| \geq 1$.

In our definition Sim receives $\mathsf{msgs}_{J'\leftarrow J}$. This is a slight adaptation of the model used in some previous works [143, 211]. However this extra information is permissible since Sim still only operates with knowledge that is already known to corrupted $\mathbb{P}_j$. We could avoid this convention by just adding more information to $\mathsf{aux}_{j'}$.

Definition III;1.8 [Malicious security (PSO)]

We say that ψ_* securely computes $*$, in the presence of *malicious* adversaries, if there exists a PPT simulator, Sim, such that:

$$\left\{\mathsf{View}_j \approx_c \mathsf{Sim}^{\mathcal{O}_{\mathcal{S}^N}(F_*(\cdot,\ldots,\cdot))}(1^\lambda, \mathsf{aux}_j, \mathsf{msgs}_{J'\leftarrow J})\right\}_{j\in J}$$

for any PPT adversary $\mathcal{A}$, where $|J| \geq 1$.

We assume that F takes fixed inputs $\mathbb{S}_{j'}$ for $j' \in J'$. That is, Sim, has access to an oracle that takes fixed inputs for uncorrupted participants and adversarial inputs for corrupted participants.

We will omit the input 1^λ to Sim in the future, when the security parameter is obvious from context. For PSU and PSI we consider $\Delta = \mathcal{S} \times \{\perp\}$, and for PSU/I-CA we consider $\Delta = \mathbb{N} \cup \{0\} \times \{\perp\}$. That is, only $\mathbb{P}_1$ receives output in the two-party case.

III;2 Related work and contributions

Before we move onto describing our solution, we first analyse related work (Section III;2) completed prior to [117], the publication that this chapter is based on. Secondly, we give a detailed overview of our contributions (Section III;2.2). Lastly, we analyse related work that originated after [117], thus providing an up-to-date analysis (as of mid-2018) of the field as a whole.

III;2.1 Previous work

Deterministic PSI. The work of Meadows initiated the study of private set operations in 1986, as a means of constructing a protocol for establishing the intersection of a set of credentials. The exact problem is known as private matchmaking, but is in fact heavily related to the problem of private set intersection. To elucidate the basic idea behind the protocol of [236], consider $\mathbb{P}_j$ with knowledge of a common cyclic group $\mathbb{G}$ and generator $g \in \mathbb{G}$, for $j \in \{1, 2\}$. Then if the set universe (of credentials) is considered to be $\mathcal{S} = \mathbb{Z}_p$, for some $p > 0$, then $\mathbb{P}_j$ with credential x_j can compute $y_j = g^{x_j}$ and send x_j to $\mathbb{P}_{\bar{j}}$. By the hardness of the discrete log problem, $\mathbb{P}_{\bar{j}}$ is unable to learn x_j, but if $x_j = x_{\bar{j}}$, then $y_j = y_{\bar{j}}$ and thus an intersection of credentials can be established.

While the solution is elegant, it is easy to subvert the protocol if no additional security measures are imposed. For example $\mathbb{P}_{\bar{j}}$ could just replay the message of $\mathbb{P}_j$ to establish a credential match. Moreover, if the likelihood is that credentials form a small subspace of the entire universe, then it could be efficient to check all types of likely credentials against the ones that are received to learn unknown credentials. In addition, until the group $\mathbb{G}$ or generator $g \in \mathbb{G}$ are changed, then all credentials are deterministic — ensuring that the scheme would struggle to meet 'forward secrecy' requirements. In other words, if the value y_j is inverted at some point in the future, an adversary with access to previous exchanges will learn that $\mathbb{P}_j$ possessed x_j corresponding to y_j. The deterministic nature of the encoding mechanism for set elements seems to be the at the centre of the various flaws listed above.

A similar hashing-based solution, highlighted much later by [260], provides similar functionality with an increase in speed (where $y_j = H(x_j)$ for a common hash function H). Again, the deterministic nature of the computation means that it has similar flaws to the solution of [236].

PSOs from oblivious polynomial evaluation. Freedman et al. [143] were the first to demonstrate a solution that uses a probabilistic encoding mechanism for set elements, for computing PSI between two parties. While this does not immediately solve the forward secrecy problem above, an adversary would now have to corrupt the secret 'encoding' key itself in order to abuse security in this manner.[4] For deterministic encodings, only the set element is needed to be found at some future point. Therefore, there is an immediate and concrete improvement in the security guarantees of the scheme.

The method of [143] involves the use of a protocol for *oblivious polynomial evaluation* (OPE) [243]. At a high-level, $\mathbb{P}_1$ constructs an 'encrypted' polynomial, where the coefficients of the polynomial are encrypted using an AHE scheme, to which $\mathbb{P}_1$ knows a key pair $(\mathsf{pk}, \mathsf{sk})$ ($\mathbb{P}_2$ also knows the public key pk). The roots of the polynomial are the elements of the set $\mathbb{S}_1$, where $\mathbb{S}_1 \subseteq \mathbb{Z}_p$.

Let $\widetilde{P} = \mathsf{ahe.Enc}(\mathsf{pk}, \{a_t\}_{t \in [n_1]})$ be the set of encrypted coefficients of the polynomial P. Firstly, using the additive homomorphic property it is possible to evaluate scalar multiplications over ciphertexts $c \in \mathcal{Y}$, for some scalar $z \in \mathbb{Z}_p$, by computing $\mathsf{ahe.ScMult}(\mathsf{pk}, c, z)$. The result is an encryption of the plaintext where $z \cdot m$ where c encrypts m. We can evaluate this scalar multiplication using the double-and-add method that we described earlier, requiring $\log(z)$ operations. Then, 'evaluating' the encrypted polynomial $\widetilde{P}$ at a value x amounts to evaluating

$$\mathsf{ahe.ScMult}(\mathsf{pk}, \widetilde{P}[t], x^t). \tag{III;2}$$

for each $t \in [n_1]$, where $\widetilde{P}[t]$ is the t^{th} coefficient of $\widetilde{P}$ and $x \in \mathbb{Z}_p$.[5]

Taking this on board, the second stage of the protocol involves $\mathbb{P}_1$ sending the encrypted polynomial $\widetilde{P}$ to $\mathbb{P}_2$. Now, $\mathbb{P}_2$ essentially needs to evaluate the encrypted polynomial for each of their set elements, along with computing some masking that prevents $\mathbb{P}_1$ learning too much about $\mathbb{S}_2$. That is, $\mathbb{P}_2$ essentially computes the pair $(r_i^1 \cdot P(x_i^{(2)}), r_i^2 \cdot P(x_i^{(2)} + x_i^{(2)}))$, but in encrypted format and where $r_i^1, r_i^2 \leftarrow\$ \mathbb{Z}_q$ are random scalars for each $i \in [n_2]$. The masking hides the value $x_i^{(2)}$ when it is not in the intersection of the sets since $P(x_i^{(2)}) \neq 0$.

Formally, for $i \in [n_2]$ and $x_i^{(2)} \in \mathbb{S}_2$, $\mathbb{P}_2$ evaluates the pair:

$$(\mathsf{ahe.ScMult}(\mathsf{pk}, \widetilde{P}(x_i^{(2)}), r_i^1),\ \mathsf{ahe.Add}(\mathsf{ahe.ScMult}(\mathsf{pk}, \widetilde{P}(x_i^{(2)}), r_i^2), \mathsf{ahe.Enc}(\mathsf{pk}, x_i^{(2)}))). \tag{III;3}$$

Notice, that the terms

$$\mathsf{ahe.ScMult}(\mathsf{pk}, \widetilde{P}(x_i^{(2)}), r^l)$$

will be randomly distributed (for $l \in [2]$) on decryption, unless $x_i^{(2)} \in \mathbb{S}_2$ is a root of P when it will decrypt to zero. The pairs in Equation (III;3) are returned as $(\widetilde{c_2^{(i)}}, \widetilde{d_2^{(i)}})$ to $\mathbb{P}_1$, who can now compute

$$(\mathsf{ahe.Dec}(\mathsf{sk}, \widetilde{c_2^{(i)}}), \mathsf{ahe.Dec}(\mathsf{sk}, \widetilde{d_2^{(i)}})).$$

[4]Usually a secret key for an encryption scheme.

[5]Strictly speaking, we cannot evaluate as this is a set of encrypted coefficients. We abuse notation to take 'evaluation' to be the implied

The second term only decrypts to a non-uniform value when $x_i^{(2)}$ is a root (due to the masking), which $\mathbb{P}_1$ detects if the first entry decrypts to 0. When this is the case, the second entry decrypts precisely to $x_i^{(2)} \in \mathbb{S}_2$. In summary, $\mathbb{P}_2$ learns the value of those elements $x_i^{(2)} \in \mathbb{S}_2$ iff $x_i^{(2)} \in \mathbb{S}_1$, also. As a consequence, $\mathbb{P}_1$ learns the intersection of the two sets.

Subsequently, the works of Kissner and Song [211] and Frikken [146] expanded the use of OPE techniques to also construct PSO protocols for the PSU and PSU/I-CA operations based on very similar techniques. Both works actually create protocols in the case of multiple parties. They also establish protocols for more complicated operations that can impose external thresholds on the computation (e.g. threshold set intersection that includes elements only if they appear a certain number of times [211]). While the techniques demonstrate subtle differences, we will neglect to explicitly define the set of operations due to their similarity with [143].

In terms of the security satisfied by the schemes, semi-honest security is fairly simple to achieve since $\mathbb{P}_2$ receives no output and only witnesses ciphertexts that are taken from an IND-CPA secure encryption scheme. For $\mathbb{P}_1$, the plaintexts are randomised in the case when $x_i^{(2)} \notin \mathbb{S}_1$ and thus these elements are indistinguishable from encryptions of random elements (which the simulator can generate).

Achieving malicious security without providing extra structure seems difficult. For example in the protocol of [143], if an adversary were to corrupt $\mathbb{P}_1$, then it could simply construct a polynomial such that all elements were roots (i.e. the zero polynomial). This is possible since the IND-CPA security of the encryption scheme means that the encrypted polynomials are indistinguishable to $\mathbb{P}_2$. Such a polynomial would allow $\mathbb{P}_1$ to learn the entire set of $\mathbb{P}_2$. If the adversary corrupts $\mathbb{P}_2$ then, learning anything about $\mathbb{S}_1$ is shown to be difficult under the semantic security of the encryption scheme that is used. However, a corrupt $\mathbb{P}_2$ could still force the computation to output incorrectly by just evaluating incorrect values, instead of their input set.

The first problem can be mitigated if a limited external certifier is introduced that simply signs the set of $\mathbb{P}_1$. This technique was used by later works such as [73, 110, 122, 123, 125, 204] and is known as the *authenticated*-PSO (APSO) setting. Unfortunately, this requires that the set be revealed to some third party but requires less trust assumptions between the two participants because $\mathbb{P}_2$ only has to verify the signature of the certifier.

Alternatively, the works of [146, 181, 211] adapt the OPE technique to allow security to be proven against malicious adversaries without the need for external authentication. In combination, these works provide protocols for private set union and intersection. They do this by adapting the constructions mentioned above to include *zero-knowledge* proof techniques. Intuitively, these techniques ensure that the participants commit to using fair inputs and, as a result, have a negligible probability of successfully deviating from the protocol specification. Unfortunately, while these solutions are elegant and interesting, they dramatically reduce the efficiency of the protocol designs — making them unsuitable for practical applications.

The OPE techniques are related to the techniques that we eventually detail. We focus on these designs with greater clarity throughout this chapter.

Practical constructions of PSI and PSU/I-CA. PSI protocols (and, to a lesser extent, PSU/I-CA) are regarded as cornerstone primitives in many applications. As such, a lot of more recent work targets these operations when considering more practical proposals. In addition, these operations are plausibly symmetric in the sense that each input set contains all of the information that is contained in the output. This has led to the proliferation of a number of protocols that use symmetric cryptographic with greater efficiency in practical scenarios.

Initially, the works of De Cristofaro et al. [109, 122, 123] established different methods for computing PSI and PSI-CA more efficiently by using blinded variants of the RSA cryptosystem [81, 268]. These methods demonstrate very low communicational overheads since the elements that are swapped are essentially just group elements. Secondly, the work of Huang et al. [191] establishes that garbled circuits-based protocols are efficient in settings where high security parameters are required, outperforming the previous protocols that we have discussed.

A common bottleneck in all of the previous constructions is the requirement for computing 'public-key' operations in the computational phases. For instance, the blind-RSA methods require many modular exponentiations which can be expensive for large sets. The same type of operations are required for performing the oblivious transfer (OT) phases of garbled circuit evaluation. However, the 'symmetric' quality of the operations makes them candidates to be implemented using inherently faster, symmetric techniques.

The work of Dong et al. [130] showed that, using a Bloom filter as an underlying data storage mechanism, it was possible to significantly lower the computational overhead of each participant in the context of PSI. Using Bloom filters introduces the possibility of errors occurring in the protocol (we describe Bloom filters in more detail in Section III;3.1). Fortunately, this probability can be reduced to a negligible cost with careful parameter selection. In this case, there is no effect on the correctness and security analysis of the resulting protocol.

The technique of [130] modifies the Bloom filter into a garbled representation, where elements are stored by splitting them into k shares and each share is placed in the Bloom filter at the index indicated by the hash functions.[6] An implementation is provided that demonstrates fast runtimes and low communication, even in the case of billion-element sets. Similarly, it was shown by Egert et al. [133] and Debnath and Dutta [125] independently, that it is possible to construct practical cardinality protocols using an underlying Bloom filter.[7]

[6] Shares are reused if the entry of the Bloom filter is already populated.

[7] Although [125] requires public-key operations and is thus much less efficient.

Pinkas et al. [258, 260] gave constructions of PSI protocols that utilise the technique of oblivious transfer [243, 263], without the need for extra circuit computation as in [191]. Their constructions are practical for sets of huge sizes on standard computational systems, with speeds that are comparable to that of the simple deterministic hashing technique used above. These results help to reinforce the gains that can be made by eliminating the requirement for public-key operations. However, the works of [258, 260] only demonstrate security in the semi-honest setting, while the previous works can all demonstrate in the malicious security model. Fortunately, Rindal and Rosulek [267] show that malicious variants of these protocols can be achieved using a technique known as 'dual execution'.

For PSI/PSU-CA, constructions are less practically efficient than PSI generally; this is largely due to the prevalence of asymmetric techniques. However, the works of [109, 133] give asymptotically optimal designs.

Constructions of dedicated PSU protocols. Focusing on the problem of constructing PSU protocols, the methods are somewhat different. Apart from the original constructions of [146, 181, 211], constructions have largely been based on inherently asymmetric designs that do not necessarily lend themselves to practical cryptographic implementation. It seems hard to build protocols for this functionality from symmetric primitives.

The work of Brickell and Shmatikov [71] establishes a PSU protocol as an application of work that utilises minimum spanning trees for constructing privacy-preserving versions of classic graph algorithms. Otherwise, Seo et al. [275] construct a multi-party PSU protocol where the number of rounds is constant, using reversed Laurent series.

The work of Blanton et al. [47] follows in the footsteps of [211] by constructing protocols for various different set operations, including the main operations that we consider above. Their constructions also include similar threshold designs to those discussed in [211]. Their designs use techniques associated with secret sharing for establishing their protocols, and also accommodate the possibility of using multisets as input.

The work of [71] only considers semi-honest security, while [47, 275] also consider the malicious security model.

Other notable constructions. We acknowledge some alternative constructions of PSI protocols that are related to the constructions that we eventually give later. Firstly, Hazay [180] gives an alternative method of constructing OPE and PSI from 'algebraic PRFs'. The construction differs from those previously since it makes use of a polynomial evaluation procedure in the exponent of group elements. To do this, Hazay shows that it is possible to use a multiplicatively homomorphic encryption scheme in the exponent, rather than an additive scheme as in [143]. Since the OPE techniques of [143] use only additive operations, it is natural to see that such a transformation could exist. The work of Jarecki and Liu [197] establishes a protocol for obliviously evaluating a pseudorandom function, along with the application of describing an efficient PSI protocol. However, the domain of set elements must be of polynomial size only. Therefore, we do not consider this work further.

Secondly, the work of Kerschbaum [204] creates an 'outsourced' PSI protocol using an 'encrypted' Bloom filter — the notion of outsourcing is essentially the model where only $\mathbb{P}_1$ receives the output of the computation. The construction uses a Bloom filter in a similar setting and manner to the way that we use in our constructions in later sections. The key difference is that Kerschbaum requires the usage of the Goldwasser-Micali [166] encryption scheme, that allows homomorphic computation over encrypted bits; whereas we require homomorphic computation that essentially encrypts ring elements. This technique is utilised also by Debnath and Dutta in their construction of protocols for PSI and PSU/I-CA [125], and also by Kerschbaum [205] for protecting supply chain integrity.

Asymptotic analysis. The two main, asymptotic metrics that are used for calculating the efficiency of any given scheme are the communication and computational complexities for each participant. We could also consider the number of rounds needed for the protocol, though this is more of a concern for protocols with $N > 2$ participants (we focus primarily on $N = 2$). In Table III;1 we provide an asymptotic analysis of selected works with respect to the efficiency of the constructions.[8] This does not necessarily take into account the efficiency of the underlying primitives, but provides an accurate measure of the scalability of the designs. We do not consider the deterministic designs as viable methods for private set operations functionality, due to the security flaws that they inherit.

Remark III;2.1. *In this chapter, we will be focusing on making asymptotic improvements compared to already known methods for computing multiple PSOs. The works of [130, 258, 260], amongst others, establish very fast methods that are adapted specifically to the case of computing PSI (using symmetric techniques). Asymptotically, they are also comparable to the most efficient schemes given in Table III;1. However, due to the differing goals that are sought, we choose not to explicitly include these constructions in our complexity analysis.*

Computational complexity of OPE techniques. The computational complexity of the OPE-based techniques in Table III;1 is listed as $O(n \log \log(n))$ rather than $O(n^2)$. That is, each polynomial consists of n encrypted coefficients and $\mathbb{P}_2$ must compute n pairs by operating over each their n set elements once for each coefficient (hence $O(n^2)$). However, it was shown by Freedman et al. [143] that it was possible to reduce this computational complexity considerably by using an external hash function for 'categorising' the set elements into buckets first.

[8] We do not consider the multi-party version of [146] since it involves a transformation that is essentially adapted from the work of [211].

Cite	Communication	Computation	Participants	Operation	Primitive
[143]	$O(n)$	$O(n \log\log(n) \log_2(\|\mathcal{S}\|))$	2	PSI	OPE
[211]	$O(N^2 n \log(\|\mathcal{S}\|))$	$O(n^2)$	N	PSO	OPE
[71]	$O(n \log(\|\mathcal{S}\|))$	$O(n \log n)$	2	PSU	Min. span. tree
[146]	$O(n)$	$O(n \log\log(n) \log_2(\|\mathcal{S}\|))$	2	PSU	OPE
[181]	$O(n)$	$O(n \log\log(n) \log_2(\|\mathcal{S}\|))$	2	PSO	OPE
[47]	$O(N^3 n \log(Nn)$	$O(N^3 n \log(Nn))$	N	PSO	Secret sharing
[275]	$O(N^3 n^2)$	$O(N^4 n^2)$	N	PSU	Laurent series (DL)
[122, 123]	$O(n)$	$O(n)$	2	PSI	RSA
[109]	$O(n)$	$O(n)$	2	PSU/I-CA	RSA
[133]	$O(L)$	$O(L)$	2	PSU/I-CA	Bloom filter
[204]	$O(L)$	$O(L)$	2	PSI	GM
[125]	$O(L)$	$O(L)$	2	{PSI,PSU/I-CA}	GM

Table III;1: Asymptotic analysis of selected previous works from Section III;2.1. We assume that $n = |\mathbb{S}_j|$ for all $j \in [N]$; and $|\mathcal{S}|$ represents the size of the domain of set elements that is being considered, and $c < N$ is the maximum number of corrupted participants (1 for the two participant setting). For our choice of Bloom filter parameters (Section III;3.1), the length L is equivalent to $O(kn)$. We use PSO to denote that the proposal consists of designs for all of PSU, PSI and PSU/I-CA. GM stands for Goldwasser-Micali encryption. We also ignore the multi-party version of [146].

Let $x_{i_1}^{(1)} \in \mathbb{S}_1$, for $i_1 \in [n]$. The degree of the polynomials can be reduced significantly by initially computing $H(x_{i_1}^{(1)}) = t \in [M]$ (for a public hash function H). Thus, we allocate $x_{i_1}^{(1)}$ to the bin B_t for $t \in [M]$. Let $n_t = \max_t(\{B_t\}_{t\in[M]})$, then $\mathbb{P}_1$ creates M polynomials of degree n_t for each bin B_t and sends these to $\mathbb{P}_2$. Then $\mathbb{P}_2$ computes $t = H(x_{i_2}^{(2)})$ for each $x_{i_2}^{(2)} \in \mathbb{S}_2$ and then proceeds to perform the same computations, as in Equation (III;3), except with the protocol corresponding to B_t for $x_{i_2}^{(2)} \in B_t$. This now only requires $n_t \cdot n$ operations to be computed by $\mathbb{P}_2$. Furthermore, it is shown by [143] that $\max_t(n_t) = O(\log\log(n))$ with high probability (associated with the choice of the hash function H).

III;2.2 Overview of contributions

In this chapter, we detail a new technique for constructing PSU, PSI and PSU/I-CA protocols based on the notion of an encrypted Bloom filter, using similar design decisions to those made in [124, 204]. However, the computations made by both participants more closely resemble the techniques used in OPE constructions [143, 146, 180, 181, 211]. Our constructions are proven to be secure in the semi-honest security model, primarily. We can also achieve security in the malicious model in the authenticated PSO scenario [73, 110, 122, 125, 204], see Section III;9 for more details.

Asymptotically optimal PSO toolkit. In short, rather than $\mathbb{P}_1$ constructing an encrypted polynomial with roots set to be the elements of $\mathbb{S}_1$, $\mathbb{P}_1$ creates an encrypted Bloom filter representing the set $\mathbb{S}_1$. We use a homomorphic encryption scheme (allowing at least additions over the plaintext space) that satisfies semantic security to perform the encryption. Now $\mathbb{P}_1$ sends the 'encrypted' Bloom filter to $\mathbb{P}_2$ who makes similar computations to those required in OPE-based techniques and returns the results to $\mathbb{P}_1$.[9] Decryption reveals the correct result, depending on the operations computed by $\mathbb{P}_2$. The actual construction of the protocols are given in Section III;4 for PSU, Section III;5 for PSI and Section III;6 for PSU/I-CA.

Using an underlying Bloom filter instead of the polynomial-based approach yields complexities that are dependent on the size of the Bloom filter, L. The parameter L is in turn dependent on the size of the set n multiplied by the number of hash functions, k, used to evaluate the Bloom filter (i.e. $L = O(kn)$). However, we show in Section III;3.1 that we can make k essentially dependent only on the false-positive parameter that we wish the Bloom filter to have (as first acknowledged in the results of [130]). Treating k as an independent constant in relation to n, our protocols are the first to demonstrate linear communication and computational complexities in the size of the sets being considered. By using a Bloom filter, we sacrifice the perfect correctness of our protocol constructions. As mentioned previously, we can still ensure all-but-negligible numbers of correctness errors with explicit parameter choices that we make in Section III;3.1.

Consequently, we give the first construction of a PSU protocol, and subsequently a generic technique for computing all of the main operations, with linear asymptotic complexities in n. Moreover, we can fix k, for a specific choice of λ, and then the complexities of our protocols grow linearly wrt to the increasing size of n. We view this as an important result in establishing scalable designs of toolkits for computing multiple private set operations. Our techniques also represent a new way of computing complex functionality that was only previously realisable via methods related to OPE. As such, we think our technique may have impact in other situations where OPE protocols have already been used as a cornerstone primitive.

It should be noted that our complexities are inherently dependent on the level of homomorphism acquired from the homomorphic encryption scheme that we use. Indeed, our asymptotic result only holds in the case of somewhat homomorphic encryption schemes that permit at least one multiplication (for example [51, 159]). If we want to use an AHE scheme, then the computational complexity of our PSU protocol naively increases by a factor of $\log(|\mathcal{S}|)$ (due to the requirement to perform scalar multiplications). We should balance this argument that this allows us to more efficient schemes (such as the Paillier encryption scheme [251]) that are practically efficient to run on commodity hardware and systems.[10] We explicitly correct an erroneous claim made in [117], stating that these linear complexities also apply in the case of using an AHE scheme — see Section III;7.2. The asymptotic analysis of our protocols is given in Section III;7. We provide a summary of the efficiency of our contributions in Table III;2 (an update of Table III;1 including the result of our research).

Practical implementation. In terms of practical efficiency, we give a proof-of-concept implementation of our protocols in the widely-used language `Go`, using the Paillier AHE scheme.

[9] We defer the actual specifics of our constructions until Section III;4 and onwards.
[10] This trade-off applies to the OPE-based technique, also.

Cite	Communication	Computation	Participants	Operation	Primitive
[143]	$O(n)$	$O(n \log\log(n) \log_2(\lvert\mathcal{S}\rvert))$	2	PSI	OPE
[211]	$O(N^2 n \log(\lvert\mathcal{S}\rvert))$	$O(n^2)$	N	PSO	OPE
[71]	$O(n \log(\lvert\mathcal{S}\rvert))$	$O(n \log n)$	2	PSU	Min. span. tree
[146]	$O(n)$	$O(n \log\log(n) \log_2(\lvert\mathcal{S}\rvert))$	2	PSU	OPE
[181]	$O(n)$	$O(n \log\log(n) \log_2(\lvert\mathcal{S}\rvert))$	2	PSO	OPE
[47]	$O(N^3 n \log(Nn)$	$O(N^3 n \log(Nn))$	N	PSO	Secret sharing
[275]	$O(N^3 n^2)$	$O(N^4 n^2)$	N	PSU	Laurent series (DL)
[122, 123]	$O(n)$	$O(n)$	2	PSI	RSA
[109]	$O(n)$	$O(n)$	2	PSU/I-CA	RSA
[133]	$O(L)$	$O(L)$	2	PSU/I-CA	Bloom filter
[204]	$O(L)$	$O(L)$	2	PSI	GM
[125]	$O(L)$	$O(L)$	2	PSI	GM
Construction III;4.1	$O(L)$	$O(L)$	2	PSU	SHE
Construction III;5.1	$O(L)$	$O(L)$	2	PSI	SHE
Construction III;6.1	$O(L)$	$O(L)$	2	PSU/I-CA	SHE
Construction III;4.1	$O(L)$	$O(L \log_2(\lvert\mathcal{S}\rvert))$	2	PSU	AHE
Construction III;5.1	$O(L)$	$O(L \log_2(\lvert\mathcal{S}\rvert))$	2	PSI	AHE
Construction III;6.1	$O(L)$	$O(L \log_2(\lvert\mathcal{S}\rvert))$	2	PSU/I-CA	AHE

Table III;2: Updated table to include the complexities in this work.

Our implementation operates with realistic security parameters and shows that our protocol can be run for large set sizes — after allowing for a computationally expensive, offline setup phase for $\mathbb{P}_1$. Concretely, we show that the expensive encryption of the Bloom filter can be handled offline; moreover, it can be used for multiple protocol instantiations and updated to include new set elements. In fact, we actually maintain the entire functionality of the original Bloom filter. As a consequence, this expensive phase can be amortised over multiple protocol instantiations and can be used for all of the different PSOs, since each PSO is completely determined by the single step of $\mathbb{P}_2$. This amortisation is not possible with OPE-based techniques since altering the roots of the polynomial requires modifying all of the encrypted coefficients whp. See Section III;8 for more detailed analysis, along with the discussion of our implementation.

Achieving malicious security. The final part of out work shows that we can achieve security against a malicious corruption of $\mathbb{P}_1$, in the authenticated PSO model [73, 110, 122, 125, 204]. We stop short of proving malicious security in the standard (non-authenticated) model since it would likely involve constructing complicated zero-knowledge mechanisms for proving security. Such mechanisms do not fit with our mantra of constructing simple-to-use toolkits for designing PSO protocols. In addition, the impact on any eventual deployment would be noticeable.

Adapting the toolkit. Our final contribution is to show that our generic 'toolkit' is malleable and that it can be used to construct much more complex functionality. This is afforded by the simplicity of the original protocols that we develop. In particular, in Chapter IV we adapt the PSU protocol given in Section III;4 to settings where the union output is limited by a threshold over associated data for each set element. The motivation that we consider is where set elements have externally (and canonically) allocated values — the threshold allows learning the union until the value of the elements in the output exceeds it. The protocol is an instantiation of an even-handed mediator, who facilitates full-sharing in an information sharing scenario considered in the game-theoretic analyses of [119, 208].

III;2.3 Concurrent and subsequent work

Finally, before detailing the particulars of our construction, we give a brief review of recent constructions that appeared subsequent to our contribution.

Kolesnikov et al. [214] showed that it was possible to construct more practical techniques for set intersection using an application of an oblivious PRF, based on the techniques of [198, 258, 260]. Other practical PSI designs have included [215, 267] where [267] gives a maliciously-secure variant of the OT-based protocols of [258, 260]. Similarly, Pinkas et al. [259] demonstrate a practical solution to circuit-based PSI using similar hashing techniques to those considered in [258, 260].

The work of [131] gives a PSI protocol with logarithmic complexities when the correctness is relaxed so that the output is only approximately equal to the ideal functionality. In a similar vein, it is shown in [86] that PSI can be made fast, even using homomorphic encryption as a primitive, when the size of the inputs is structured so that one player has a much smaller set (the player who undergoes the most computation). This is an interesting observation that also applies to our designs, since most of the computational work depends on the size of the set $\mathbb{S}_1$. See Section III;7 for more details.

Finally, Cerulli et al. [80] give the first construction of a PSI functionality that allows hiding the cardinality of the input sets. This contribution is part of a wider effort to prove security in models that incorporate multiple protocols runs (using the same inputs).

As with previous work, most of this recent research is focused on giving very practical solutions to problem of constructing PSI protocols. There is still a gap in the academic literature for constructing generic toolkits using primarily symmetric techniques, while maintaining asymptotic optimality. Research that filled this gap would undoubtedly lead to a solution that would have demonstrable practical efficiency for all of the main set operations.

III;3 Preliminaries

In this section, we will formalise the primitives that we need for constructing our PSO protocols.

III;3.1 Formalisation of Bloom filters

Bloom filters were first defined by Bloom [49] in 1970. They are a space- and time-efficient data structure expressed as a binary string, allowing for simple and quick mechanism for storing and checking items. We give a formalisation in Definition III;3.1.

Definition III;3.1 [Bloom filter [49]]

Let $L, k, n > 0$ be parameters, and let $\mathbb{S} \subseteq \mathcal{S}$ be a set such that $n = |\mathbb{S}|$. Let $\mathcal{H} = \{h \mid h : \mathcal{S} \mapsto [L]\}$ be a family of hash functions, and sample $\widetilde{h} = (h_1, \dots, h_k) \leftarrow\!\!\$\, \mathcal{H}^k$. In addition, we abuse notation and write $\widetilde{h}(x) = (h_1(x), \dots, h_k(x))$ for $x \in \mathbb{S}$.
Then we define a Bloom filter of length L as the tuple

$$\mathsf{BF} = (\mathsf{arr}, \mathsf{Store}, \mathsf{Query}, \widetilde{h}, \mathbb{S}_{\mathsf{BF}}),$$

where $\mathsf{BF.arr} \in \{0,1\}^L$, $\mathbb{S}_{\mathsf{BF}} \subseteq \mathcal{S}$ (s.t. $\max(|\mathbb{S}_{\mathsf{BF}}|) = n$). Let $\mathsf{BF.pp} = (\mathsf{BF.arr}, \mathsf{BF.Store}, \mathsf{BF.Query}, \widetilde{h})$ be the *public* parameters of BF, and $\mathsf{BF.sp} = \mathbb{S}_{\mathsf{BF}}$ be the *secret* parameters. Define BF.Store and BF.Query in the following way.[a]

- $\mathsf{BF.Store}(\mathsf{BF.pp}, \mathsf{BF.sp}, x \in \mathbb{S})$: Let $(y_1, \dots, y_k) = \widetilde{h}(x) \in [L]^k$, set $\mathsf{BF.arr}[y_j] = 1$ for $j \in [k]$ and add $x \to \mathsf{BF.sp}$.
- $\mathsf{BF.Query}(\mathsf{BF.pp}, x \in \mathbb{S})$: Let $(y_1, \dots, y_k) = \widetilde{h}(x) \in [L]^k$. Output $\bigwedge_{j=1}^{k} \mathsf{BF.arr}[y_j]$.

[a]We use the notation $\mathcal{S}^*$ to indicate that the length of this output is variable.

We will assume that $\mathcal{H}$ is a family of hash functions, such that for $h \leftarrow\!\!\$\, \mathcal{H}, y \leftarrow h(x)$ and $x \in \mathbb{S}$, then y is uniformly distributed in $[L]$. We can achieve this under the assumption that $\mathcal{H}$ is a family of *universal* hash functions over the space $[L]$ (Definition II;4.4 and Corollary II;4.1), or under the much stronger assumption that h is a random oracle.

Optimal parameter settings. A major constraint of Bloom filters is that they tolerate a probability of false-positives that is determined solely by the parameters L, k, n. This means that, with probability ϵ, then $\mathsf{BF.Query}(\mathsf{BF.pp}, x \in \mathbb{S}) = 1$ for $(x \in \mathbb{S}) \wedge (x \notin \mathsf{BF.sp})$. On the other hand, false negatives are never possible, since the hash function evaluations are deterministic.

According to [59, 130], the probability that the bit corresponding to $\mathsf{BF.arr}[i]$, for $i \leftarrow\!\!\$\, [L]$, is set to 1 is $p = 1 - (1 - 1/L)^{kn}$. Therefore, an upper bound of the false-positive probability can be inferred to be

$$\epsilon = p^k \left(1 + O\left(\frac{k}{p}\sqrt{\frac{\ln(L) - k\ln(p)}{L}}\right)\right), \tag{III;4}$$

which is negligible in k.

Again, as noted by [59, 130], and by Equation (III;4), we can impose a lower bound on the size of L and infer the size of the parameter k:

$$L \geq n \log(e) \log(1/\epsilon); \quad k = (L/n) \ln(2). \tag{III;5}$$

Finally, choosing L optimally (wrt the lower bound) renders:

$$k = \log(1/\epsilon) \tag{III;6}$$

as the optimal choice of k. Notice that, in this setting, k is completely independent of the choice of the rest of the parameters, including the size of the set that is stored. In fact, ϵ is now negligible directly in the choice of k (rather than via Equation (III;4)). Therefore, if we have a security parameter λ and set $k = \mathsf{poly}(\lambda)$, then $\epsilon < \mathsf{negl}(\lambda)$. The choice of n is thus made independently of k.

The above is important for establishing the asymptotic complexity of our protocols later, and the favourable comparisons with prior work. In all of the subsequent work, we assume that the optimal parameter choices from Equations (III;5) and (III;6) are always used.

Additional notation. We may sometimes abuse notation and write $\mathsf{BF.Store}(\mathsf{pp}, \mathsf{sp}, \mathbb{S})$ for some subset $\mathbb{S} \subseteq \mathcal{S}$, rather than storing each element individually. We use the notation

$$\mathsf{BF} \leftarrow \mathsf{BF.init}(1^L, 1^n, 1^k)$$

to denote the act of *initialising* an empty Bloom filter: sampling $\widetilde{h} \leftarrow\!\!\$\, \mathcal{H}^k$; $\mathbb{S}_{\mathsf{BF}} \leftarrow \emptyset$; $\mathsf{arr} \leftarrow 0^L$; and setting

$$\mathsf{BF} = (\mathsf{arr}, \mathsf{Store}, \mathsf{Query}, \widetilde{h}, \mathbb{S}_{\mathsf{BF}}).$$

We say that a newly initialised BF is *empty* until $\mathsf{BF.Store}$ is run for the first time.

For ease of exposition in later sections, we define a distribution $\mathcal{BF}[L, n, k]$ s.t. sampling uniformly from $\mathcal{BF}[L, n, k]$ is the same as running the following steps

- initialise $\mathsf{BF} \leftarrow \mathsf{BF.init}(1^L, 1^n, 1^k)$;
- uniformly sample a set $\mathbb{S} \subseteq \mathcal{S}$, s.t. $n = |\mathbb{S}|$;
- run $\mathsf{BF.Store}(\mathsf{pp}, \mathsf{sp}, \mathbb{S})$;
- output BF.

III;3.2 Encrypting Bloom filters

Let $\mathsf{BF} = (\mathsf{arr}, \mathsf{Store}, \mathsf{Query}, \widetilde{h}, \mathbb{S}_{\mathsf{BF}})$ be a Bloom filter as defined in Definition III;3.1. We explicitly define the notion of *inverting* and *encrypting* of a Bloom filter. Similar techniques were also introduced in the works of [125, 204], albeit with different encryption schemes.

A notion of encrypted Bloom filters is also given in [127], for constructing a forward-secure, 0-RTT key exchange. The encryption mechanism that they use is an identity-based encryption scheme and the entries of the Bloom filter are populated with identities. In this work, we encrypt each of the entries directly with standard pke or she schemes.

Definition III;3.2 [Inversion]

We denote an *inverted* Bloom filter for BF by IBF, where IBF is the same as BF, except that $\mathsf{IBF.arr} = 1^L - \mathsf{BF.arr} \in \{0,1\}^L$. For a Bloom filter BF, we denote the inversion procedure by $\mathsf{invert}(\mathsf{BF}) = \mathsf{IBF}$. We modify $\mathsf{IBF.Query}(\cdot)$ to output 1 iff

$$\bigwedge_{i=1}^{k} \mathsf{IBF.arr}[y_j] = 0,$$

for $(y_1, \ldots, y_k) \leftarrow \widetilde{h}(x)$, and $j \in [k]$.

Definition III;3.3 [Encryption]

Let pke be a public-key encryption scheme. We say that EBF is an *encryption* of BF if it is a tuple of the form $\mathsf{EBF} = (\overline{\mathsf{arr}}, \mathsf{Store}, \mathsf{Query}, \widetilde{h}, \mathbb{S}_{\mathsf{BF}})$, where

$$\mathsf{pke.Dec}(\mathsf{sk}, \mathsf{EBF}.\overline{\mathsf{arr}}[i]) = \mathsf{BF.arr}[i]$$

for all $i \in [L]$. Additionally, we redefine $\mathsf{EBF.Store}$ and $\mathsf{EBF.Query}$ as:

- $\mathsf{EBF.Store}(\mathsf{EBF.pp}, \mathsf{EBF.sp}, x \in \mathbb{S})$: Let $\{y_j\}_{j\in[k]} = \widetilde{h}(x)$. Let $c_j \leftarrow \mathsf{pke.Enc}(\mathsf{pk}, 1)$, set $\mathsf{EBF}.\overline{\mathsf{arr}}[y_j] = c_j$ and add $x \to \mathsf{EBF.sp}$.
- $\mathsf{EBF.Query}(\mathsf{EBF.pp}, x \in \mathbb{S})$: Let $\{y_j\}_{j\in[k]} = \widetilde{h}(x)$ and output $\{\mathsf{EBF}.\overline{\mathsf{arr}}[y_j]\}_{j\in[k]}$

for some subset $\mathbb{S} \subseteq \mathcal{S}$. The rest of the tuple components are the same as BF.
For a Bloom filter BF, we define the encryption procedure to be

$$\mathsf{encrypt}(\mathsf{pk}, \mathsf{BF}) = \mathsf{EBF}$$

where $\mathsf{encrypt}$ computes $\overline{\mathsf{arr}}[i] \leftarrow \mathsf{pke.Enc}(\mathsf{pk}, \mathsf{BF.arr}[i])$ for each $i \in [L]$ and setting

$$\mathsf{EBF} = (\overline{\mathsf{arr}}, \mathsf{BF.Store}, \mathsf{BF.Query}, \mathsf{BF}.\widetilde{h}, \mathsf{BF.sp})$$

as the output.

Remark III;3.1. *We typically use* EIBF, *when denoting* $\mathsf{EIBF} \leftarrow \mathsf{encrypt}(\mathsf{pk}, \mathsf{invert}(\mathsf{BF}))$.

Interestingly, the encrypted Bloom filter retains the ability to publicly store items since only the public key is required for running $\mathsf{EBF.Store}$. This is an important consideration in Section III;8 when we consider amortising the encryption of Bloom filters as an optimisation of our implementation. The algorithm $\mathsf{EBF.Query}$ algorithm appears redundant if the evaluator does not hold sk for the encryption scheme. But, if $\mathsf{pke} = \mathsf{she}$, then such an evaluator can still homomorphically compute over the ciphertexts.

Finally, we prove the following lemma — essentially stating that if pke satisfies IND-CPA security, then EBF is indistinguishable from an encryption of the bitstring 0^L.

```
$\mathsf{exp}^{\mathsf{ebf}}_{b,\mathcal{A}}(1^\lambda, 1^L, 1^n, 1^k)$
1 :  $\mathsf{BF} \leftarrow_{\$} \mathcal{BF}[L, n, k]$;
2 :  $(\mathsf{pk}, \mathsf{sk}) \leftarrow_{\$} \mathsf{pke.KeyGen}(1^\lambda)$;
3 :  $W \leftarrow \emptyset$
4 :  if $(b = 0)$ :
5 :    $\mathsf{EBF} \leftarrow \mathsf{encrypt}(\mathsf{pk}, \mathsf{BF})$
6 :    $W = \mathsf{EBF}.\overline{\mathsf{arr}}$
7 :  else :
8 :    $\{c_j\}_{j \in [L]} \leftarrow \mathsf{pke.Enc}(\mathsf{pk}, 0^L)$;
9 :    $W = \{c_j\}_{j \in [L]}$;
10 : $b_{\mathcal{A}} \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}, W)$;
```

Figure III;1: Experiments $\mathsf{exp}^{\mathsf{ebf}}_{b,\mathcal{A}}(1^\lambda, 1^L, 1^n, 1^k)$ for describing the security of an encrypted Bloom filter. We may sometimes shorten the notation and write $\mathsf{exp}^{\mathsf{ebf}}_{b,\mathcal{A}}(1^\lambda)$, when BF has already been specified.

Lemma III;3.1 [Security of EBF]

Let BF be a Bloom filter, and let pke be a public-key encryption scheme satisfying semantic security (Definition II;5.3). For experiments $\mathsf{exp}^{\mathsf{ebf}}_{b,\mathcal{A}}(1^\lambda)$ (Figure III;1):

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{ebf}(1^\lambda, \mathsf{EBF}))) < \mathsf{negl}(\lambda)$$

holds for all PPT adversaries $\mathcal{A}$.

Proof. Firstly, recall the definition of ρ-IND-CPA security from Definition II;5.4. Denote the experiments by $\mathsf{exp}^{\rho\text{-indcpa}}_{b,\mathcal{A}'}(1^\lambda)$ where $\mathcal{A}'$ is a PPT algorithm, and $\rho = \mathsf{poly}(\lambda)$ is a bound on the number of queries that $\mathcal{A}'$ can make.

Secondly, set $L = \rho$ and consider $\mathcal{A}'$ acting against the $\mathsf{exp}^{L\text{-indcpa}}_{b,\mathcal{A}'}(1^\lambda)$ game, we show that the experiments $\mathsf{exp}^{\mathsf{ebf}}_{b,\mathcal{B}}(1^\lambda, 1^L, 1^n, 1^k)$ are computationally indistinguishable for $b \in \{0, 1\}$ and any PPT algorithm $\mathcal{B}$. Our proof takes the form of a standard security reduction, where $\mathcal{A}'$ runs $\mathcal{B}$ as a subroutine.

$\mathcal{A}'$ receives pk from the challenger in $\mathsf{exp}^{L\text{-indcpa}}_{b,\mathcal{A}'}(1^\lambda)$, and assumes the role of the challenger in the experiments $\mathsf{exp}^{\mathsf{ebf}}_{b,\mathcal{A}}(1^\lambda, 1^L, 1^n, 1^k)$. First it samples $\mathsf{BF} \leftarrow_{\$} \mathcal{BF}[L, n, k]$ and a string 0^L. Then, $\mathcal{A}'$ sets $M_0 = \mathsf{BF}.\mathsf{arr}$ and $M_1 = 0^L$ and sends (M_0, M_1) to its challenger in the $\mathsf{exp}^{L\text{-indcpa}}_{b,\mathcal{A}'}(1^\lambda)$ game. The challenger returns $W_b \leftarrow \mathsf{pke.Enc}(\mathsf{pk}, M_b)$ to $\mathcal{A}'$ and $\mathcal{A}'$ returns this result directly to $\mathcal{B}$.

Notice that, $W_0 = \mathsf{pke.Enc}(\mathsf{pk}, \mathsf{BF.arr}) = \mathsf{EBF}.\overline{\mathsf{arr}}$ for $\mathsf{EBF} \leftarrow \mathsf{encrypt}(\mathsf{pk}, \mathsf{BF})$, which is what $\mathcal{B}$ receives in $\mathsf{exp}^{\mathsf{ebf}}_{0,\mathcal{B}}(1^\lambda)$. Also, $W_1 = \mathsf{pke.Enc}(\mathsf{pk}, 0^L)$ which is exactly the form of what $\mathcal{B}$ receives in $\mathsf{exp}^{\mathsf{ebf}}_{1,\mathcal{B}}(1^\lambda)$. Finally, $\mathcal{A}'$ outputs $b_\mathcal{B} \leftarrow \mathcal{B}(1^\lambda, \mathsf{pk}, W_b)$ directly. Notice that, if $\mathcal{B}$ is successful in distinguishing the two cases, then so is $\mathcal{A}'$.

Therefore, if $\max_\mathcal{B}(\mathsf{Adv}(\mathcal{B}, \mathsf{ebf}(1^\lambda, \mathsf{EBF}))) = \epsilon$, then we have that

$$\epsilon \leq \max_{\mathcal{A}'}(\mathsf{Adv}(\mathcal{A}', \mathsf{indcpa}(1^\lambda, \mathsf{pke}))),$$

since each successful output from $\mathcal{B}$ is a successful output for $\mathcal{A}'$. However, by the semantic security of pke, we know that

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{indcpa}(1^\lambda, \mathsf{pke}))) < \mathsf{negl}(\lambda),$$

thus implying that $\epsilon < \mathsf{negl}(\lambda)$; for all PPT adversaries $\mathcal{B}$. □

III;4 A protocol framework for private set union

In this section, we give our first construction of a PSO protocol, specifically targeting the functionality provided by PSU. We give slightly different constructions in the case where the underlying encryption scheme is an she scheme or an ahe scheme. Both protocols are similar, differing slightly in the method of computation.

Let $\mathsf{he} \in \{\mathsf{she}, \mathsf{ahe}\}$ be a homomorphic encryption scheme, satisfying at least additive homomorphism. We require that he satisfies the ciphertext rerandomisation property of Definition II;5.7. The individual steps of our PSU protocol, $\psi_\cup$, are found in Figures III;2, III;3 and III;4. We denote step $i \in [3]$ by $\psi^i_\cup(\mathbb{P}_j, \mathsf{he})$, where the step is computed by $\mathbb{P}_j$ for $j \in \{1, 2\}$. Recall that we use the notation $X \rightarrow \mathbb{P}_j$ in a protocol step to indicate that X is sent to participant $\mathbb{P}_j$. More precisely, $X \rightarrow \mathsf{msgs}_j \in \mathsf{View}_j$.

In the following, let $\mathbb{P}_1$ and $\mathbb{P}_2$ be the two participants in the protocol $\psi_\cup$, with respective input sets $\mathbb{S}_1, \mathbb{S}_2 \subseteq \mathcal{S} = \mathsf{he}.\mathcal{X}$ and auxiliary input data $\mathsf{aux}_1, \mathsf{aux}_2$. Let $F_\cup : \mathcal{S}^2 \mapsto \mathcal{S}^2$ be the functionality that computes:

$$(\mu_1, \mu_2) \leftarrow F_\cup(\mathbb{S}_1, \mathbb{S}_2) \tag{III;7}$$

where $(\mu_1, \mu_2) = (\mathbb{S}_1 \cup \mathbb{S}_2, \emptyset)$ and sends μ_j to $\mathbb{P}_j$ for $j \in \{1, 2\}$.[11] Let $\psi_\cup$ be the protocol that intends to compute $F_\cup$ and returns $\mathsf{View}_j = (\mathbb{S}_j, \mu_j, \mathsf{msgs}_j, \mathsf{aux}_j)$ to $\mathbb{P}_j$. We let $\mathbb{S}_j = \{x^{(j)}_i\}_{i \in [n_j]}$ for $n_j = |\mathbb{S}_j|$.

[11]This scenario is commonly referred to in the literature as the 'client-server' model, where $\mathbb{P}_1$ is the 'client' and $\mathbb{P}_2$ is the 'server' (although we will not use this convention).

```
$\psi^1_\cup(\mathbb{P}_1, \mathsf{he})$
1 : $\mathsf{BF} \leftarrow \mathsf{BF.init}(1^L, 1^n, 1^k)$;
2 : $(\mathsf{BF.arr}, \mathsf{BF}.\mathbb{S}_{\mathsf{BF}}) \leftarrow \mathsf{BF.Store}(\mathsf{BF.pp}, \mathsf{BF.sp}, \mathbb{S}_1)$;
3 : $\mathsf{EIBF} \leftarrow \mathsf{encrypt}(\mathsf{pk}, \mathsf{invert}(\mathsf{BF}))$;
4 : $\mathsf{EIBF.pp} \leftarrow \mathbb{P}_2$;
```

Figure III;2: Step one, computed by $\mathbb{P}_1$ in our PSU protocol. In this step, $\mathbb{P}_1$ constructs an encrypted, inverted Bloom filter from their input set and sends it to $\mathbb{P}_2$. The steps are the same for $\mathsf{he} \in \{\mathsf{she}, \mathsf{ahe}\}$.

We define $\mathsf{aux}_1 = (n_2, (\mathsf{pk}, \mathsf{sk}))$ and $\mathsf{aux}_2 = (n_1, \mathsf{pk})$; where $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{he.KeyGen}(1^\lambda)$. Clearly, $\mathbb{P}_j$ knows the cardinality, $n_{\bar{j}}$, of set $\mathbb{S}_{\bar{j}}$ for $j \in \{1, 2\}$. This is necessary as our protocols give away the size of the sets, which is a standard limitation of most PSO protocols. The only protocols to avoid this limitation are the PSI protocol of [80], and the PSU protocol of [71].

Our protocol is given in Construction III;4.1. In the ahe version of this protocol, we explicitly assume that the set elements $x_i^{(2)}$ can be used as scalar values (i.e. members of $\mathbb{Z}_p$).

Construction III;4.1 [PSU protocol]

Let $\psi_\cup$ be a protocol for the functionality $F_\cup$ (Equation (III;7)) and let $\mathsf{he}, \mathsf{BF}, \mathbb{P}_j, \mathbb{S}_j, \mathcal{S}, \mathsf{aux}_j, \mathsf{msgs}_j, \mathsf{View}_j$ be described as above.
We construct $\psi_\cup$ as a composition of the steps

- $(\mathbb{P}_2 : \mathsf{EIBF.pp}) \leftarrow \psi^1_\cup(\mathbb{P}_1, \mathsf{he})$ [Figure III;2];
- $(\mathbb{P}_1 : W) \leftarrow \psi^2_\cup(\mathbb{P}_2, \mathsf{EIBF.pp}, \mathsf{he})$ [Figure III;3];
- $(\mathbb{P}_1 : \mathbb{S}_\cup) \leftarrow \psi^3_\cup(W, \mathbb{P}_1, \mathsf{he})$ [Figure III;4].

We set $\mu_1 = \mathbb{S}_\cup$ and $\mu_2 = \emptyset$.

Theorem III;4.1 [Correctness]

Let $\psi_\cup$ be defined as in Construction III;4.1; let BF be the Bloom filter used in $\psi_\cup$, with parameters $k = \mathsf{poly}(\lambda)$ and L, n_1 chosen optimally. Then $\psi_\cup$ correctly computes the functionality $F_\cup$ (Equation (III;7)), with probability greater than $1 - \mathsf{negl}(\lambda)$.

Proof. We consider the set $\mathbb{S}_\cup$ returned by $\mathbb{P}_1$ in $\psi^3_\cup(\mathbb{P}_1, W, \mathsf{she})$. We have, $(\widetilde{c_i^+}, \widetilde{d_i^+}) \in W$ received by $\mathbb{P}_1$, for $i \in [n_2]$; and $(z_{i,1}, z_{i,2}) \leftarrow (\mathsf{he.Dec}(\mathsf{sk}, \widetilde{c_i^+}), \mathsf{he.Dec}(\mathsf{sk}, \widetilde{d_i^+}))$. Furthermore,

```
$\psi^2_{\cup}(\mathbb{P}_2, \mathsf{she})$
1 :  $W \leftarrow \emptyset$;
2 :  for $i \in [n_2]$ :
3 :    $\{c_i^l\}_{l \in [k]} \leftarrow \mathsf{EIBF.Query}(\mathsf{EIBF.pp}, x_i^{(2)})$;
4 :    $c_i^+ \leftarrow c_i^1$;
5 :    for $l \in [2, k]$ :
6 :      $c_i^+ \leftarrow \mathsf{she.Add}(\mathsf{pk}, c_i^l, c_i^+)$;
7 :    $d_i^+ = \mathsf{she.Mult}(\mathsf{pk}, c_i^+, \mathsf{she.Enc}(\mathsf{pk}, x_i^{(2)}))$;
8 :    $(\widetilde{c_i^+}, \widetilde{d_i^+}) \leftarrow (\mathsf{she.CRand}(\mathsf{pk}, c_i^+), \mathsf{she.CRand}(\mathsf{pk}, d_i^+))$;
9 :    $(\widetilde{c_i^+}, \widetilde{d_i^+}) \leftarrow W[i]$;
10 : $W \leftarrow \mathbb{P}_1$
```

```
$\psi^2_{\cup}(\mathbb{P}_2, \mathsf{ahe})$
1 :  $W \leftarrow \emptyset$;
2 :  for $i \in [n_2]$ :
3 :    $\{c_i^l\}_{l \in [k]} \leftarrow \mathsf{EIBF.Query}(\mathsf{EIBF.pp}, x_i^{(2)})$;
4 :    $c_i^+ \leftarrow c_i^1$;
5 :    for $l \in [2, k]$ :
6 :      $c_i^+ \leftarrow \mathsf{ahe.Add}(\mathsf{pk}, c_i^l, c_i^+)$;
7 :    $d_i^+ = \mathsf{ahe.ScMult}(\mathsf{pk}, c_i^+, x_i^{(2)})$;
8 :    $(\widetilde{c_i^+}, \widetilde{d_i^+}) \leftarrow (\mathsf{ahe.CRand}(\mathsf{pk}, c_i^+), \mathsf{ahe.CRand}(\mathsf{pk}, d_i^+))$;
9 :    $W[i] \leftarrow (\widetilde{c_i^+}, \widetilde{d_i^+})$;
10 : $\mathbb{P}_1 \leftarrow W$
```

Figure III;3: Step two, computed by $\mathbb{P}_2$ in our PSU protocol. In this step, $\mathbb{P}_2$ combines ciphertexts associated with queries to the encrypted Bloom filter (using the homomorphic operations) and sends the result back to $\mathbb{P}_1$. The set W is shuffled before it is returned to $\mathbb{P}_1$.

```
$\psi^3_{\cup}(\mathbb{P}_1, \mathsf{he})$
1 : $\mathbb{S}_{\cup} \leftarrow \emptyset$;
2 : for $i \in [n_2]$ :
3 :     $(\widetilde{c_i^+}, \widetilde{d_i^+}) \leftarrow W[i]$;
4 :     $(z_{i,1}, z_{i,2}) \leftarrow (\mathsf{he.Dec}(\mathsf{sk}, \widetilde{c_i^+}), \mathsf{he.Dec}(\mathsf{sk}, \widetilde{d_i^+}))$;
5 :     if $z_{i,1} \neq 0$ :
6 :         $\mathbb{S}_{\cup} \leftarrow (z_{i,1})^{-1} \cdot z_{i,2}$;
7 : $\mathbb{S}_{\cup} \leftarrow \mathbb{S}_1$;
8 : return $\mathbb{S}_{\cup}$;
```

Figure III;4: Step three, computed by $\mathbb{P}_1$ in our PSU protocol. Finally, $\mathbb{P}_1$ decrypts the combined ciphertexts and outputs the values that are non-zero. These should correspond to values that are not already contained in $\mathbb{S}_1$. The steps are the same for $\mathsf{he} \in \{\mathsf{she}, \mathsf{ahe}\}$.

$\widetilde{c_i^+} \leftarrow \mathsf{he.CRand}(\mathsf{pk}, c_i^+)$ and likewise for $\widetilde{d_i^+}$. Recall that ciphertext rerandomisation does not change the value of the underlying plaintext by Lemma II;5.3. Therefore

$$z_{i,1} = \sum_{l=1}^{k} \mathsf{BF.arr}[y_{i,l}],$$

where $\{y_{i,l}\}_{l\in[k]} \leftarrow \mathsf{BF}.\widetilde{h}(x_i^{(2)})$, and $z_{i,2} = z_{i,1} \cdot x_i^{(2)}$; since

$$\begin{aligned} c_i^+ &= \mathsf{he.Add}(\mathsf{pk}, c_k^+, \mathsf{he.Add}(\mathsf{pk}, c_{k-1}^+, \mathsf{ahe.Add}(\ldots, \mathsf{ahe.Add}(\mathsf{pk}, c_2^+, c_1^+)))),\\ d_i^+ &= \mathsf{he.Mult}(\mathsf{pk}, c_i^+, \mathsf{he.Enc}(\mathsf{pk}, x_i^{(2)})), \end{aligned}$$

and by the correctness of he (Definition II;5.6). Notice that, with probability $(1-\epsilon)$,

$$\mathsf{BF.Query}(\mathsf{EIBF.pp}, x_i^{(2)}) = \mathsf{he.Enc}(1) \implies x_i^{(2)} \in \mathbb{S}_{\mathsf{BF}}.$$

Thus the same implication holds with $\mathsf{IBF.Query}(\mathsf{EIBF.pp}, x_i^{(2)}) = \mathsf{he.Enc}(0)$, instead. Consequently, with probability $(1-\epsilon)$, $z_{i,1} = 0$ iff $x_i^{(2)} \in \mathbb{S}_{\mathsf{BF}} = \mathbb{S}_1$.

When $z_{i,1} = 0$ then $z_{i,2} = 0$. So $\mathbb{P}_1$ adds $x_i^{(2)}$ to $\mathbb{S}_{\cup}$ when $x_i^{(2)} \notin \mathbb{S}_1$ (with probability $(1-\epsilon)$). Therefore, with the same probability, $\mu_1 = \mathbb{S}_{\cup}$ contains all elements in $\mathbb{S}_1$ and those in $(\mathbb{S}_2 \setminus \mathbb{S}_1)$ — this is precisely the union of the two sets.

Finally, since $k = \mathsf{poly}(\lambda)$ and $\epsilon = \mathsf{negl}(k)$ by Equation (III;6), then $1-\epsilon > 1-\mathsf{negl}(\lambda)$.

The case where $\mathsf{he} = \mathsf{ahe}$ is identical, except that

$$d_i^+ = \mathsf{he.ScMult}(\mathsf{pk}, c_i^+, x_i^{(2)})$$

which does not change the result. Thus, the proof is complete. $\square$

If we tolerated non-negligible ϵ, then $\psi_\cup$ would *approximately* compute $F_\cup$ with probability only $(1-\epsilon) > 1 - \mathsf{poly}(\lambda)$. Such a notion has been used in recent works for constructing more efficient PSI [131, 265] constructions. This would also help to make our scheme more efficient, since L could be smaller. However, for the security proof below, we require that $k = \mathsf{poly}(\lambda)$, as above. The security notion for approximate PSO would have to change to allow incorporation of non-negligible false-positive rates. In particular, larger probabilities will impact the distinguishing probability of the simulations in the proofs that we write.

Theorem III;4.2 [Security]

Let $\psi_\cup$ be defined as in Construction III;4.1. Then $\psi_\cup$ securely computes the functionality $F_\cup$ in the presence of PPT semi-honest adversaries, under the semantic security of he.

Proof. For the proof of security, we define a simulator Sim who simulates View_j using only the inputs and outputs witnessed by $\mathbb{P}_j$. That is, msgs_j can be completely inferred by $\mathbb{S}_j$ and μ_j. To complete the proof, we show that the view of the protocol in the real execution is computationally indistinguishable from both of the simulations for $j \in \{1,2\}$. We first consider the case where $\mathbb{P}_1$ is corrupted by a PPT adversary $\mathcal{A}$ (writing $\mathbb{P}_1^{\mathcal{A}}$), defining the simulator, Sim_1, as follows.

Simulation III;4.1 [$\mathsf{Sim}_1(\mathbb{S}_1, \mathbb{S}_\cup, \mathsf{aux}_1)$]

- Sim_1 computes the set $\mathbb{S}_{\mathsf{Sim}} = \mathbb{S}_\cup \setminus \mathbb{S}_1$ and receives $\mathsf{EIBF.pp}$ from $\mathbb{P}_1^{\mathcal{A}}$.
- For each $x_i^{(\mathsf{Sim})} \in \mathbb{S}_{\mathsf{Sim}}$ where $i \in [|\mathbb{S}_{\mathsf{Sim}}|]$, it computes:

$$c_i^+ \leftarrow \mathsf{he.Add}(\mathsf{pk}, \mathsf{EIBF.Query}(x_i^{(\mathsf{Sim})}, \widetilde{h})).$$

- Let $g_i = \mathsf{he.Enc}(\mathsf{pk}, x_i^{(\mathsf{Sim})})$ for $i \in [|\mathbb{S}_{\mathsf{Sim}}|]$, then Sim_1 stores

$$(\mathsf{he.CRand}(\mathsf{pk}, c_i^+), \mathsf{he.CRand}(\mathsf{pk}, \mathsf{he.Mult}(\mathsf{pk}, c_i^+, g_i))) \rightarrow W_{\mathsf{Sim}}[i].$$

- For the remaining $|\mathbb{S}_\cup \setminus \mathbb{S}_{\mathsf{Sim}}|$ elements, it sets

$$(c_{i'}^+, d_{i'}^+) \leftarrow (\mathsf{he.Enc}(\mathsf{pk}, 0))^2$$

  and adds $(c_{i'}^+, d_{i'}^+) \rightarrow W_{\mathsf{Sim}}[i']$.
- Finally, Sim_1 shuffles W_{Sim} and sends it to $\mathbb{P}_1^{\mathcal{A}}$, i.e. $W_{\mathsf{Sim}} \rightarrow \mathsf{msgs}_{\mathsf{Sim}}$.
- Let $\mathsf{View}_{\mathsf{Sim}} = (\mathbb{S}_1, \mathbb{S}_\cup, \mathsf{msgs}_{\mathsf{Sim}}, \mathsf{aux}_1)$ be the view that $\mathbb{P}_1^{\mathcal{A}}$ witnesses.

Lemma III;4.1 [$\mathsf{View}_1 \approx_p \mathsf{View}_{\mathsf{Sim}}$]

$\mathsf{View}_{\mathsf{Sim}}$ in Simulation III;4.1 is computationally indistinguishable from View_1 in $\psi_\cup$ if he satisfies the ciphertext rerandomisation property of Definition II;5.7.

Proof. The only difference between View_1 in the real world execution and $\mathsf{View}_{\mathsf{Sim}}$ in Simulation III;4.1 is the way that W_{Sim} is constructed in $\mathsf{msgs}_{\mathsf{Sim}}$, from W in msgs_1. Specifically, the protocol messages differ in the way that the ciphertext pairs $(c_{i'}^+, d_{i'}^+)$ are constructed for elements in the intersection of the two sets. The rest of the view is constructed the same in both executions.

Firstly, $\mathsf{he.Dec}(\mathsf{sk}, c_{i'}^+) = \mathsf{he.Dec}(\mathsf{sk}, d_{i'}^+) = 0$ for $i' \in [|\mathbb{S}_{\mathsf{Sim}}|, |\mathbb{S}_\cup|]$, since $x_{i'}^2 \notin \mathbb{S}_1$ (and ϵ is negligible in k). Then, the only difference is the structure of the ciphertexts themselves. In the real execution they are rerandomised ciphertexts from $\mathsf{EBF}.\overline{\mathsf{arr}}$ (after performing homomorphic operations). In the simulation, they are simply new encryptions of 0.

Let $\mathcal{A}$ be an adversary that is attempting to distinguish between View_1 and $\mathsf{View}_{\mathsf{Sim}}$. Let $\mathcal{B}$ attempting to distinguish the experiments $\mathsf{exp}_{\omega,\mathcal{A}}^{\mathsf{crand}}(1^\lambda, 1^\ell, \iota)$ for $\omega \leftarrow\!\!\$\ \{0,1\}$ where ι is the implicit choice of level for encrypting the ciphertexts that we consider.[12] Then $\mathcal{B}$ when constructs W, it uses knowledge of the set $\mathbb{S}_{\mathsf{Sim}}$ to construct the messages:

$$(\mathsf{he.CRand}(\mathsf{pk}, c_i^+), \mathsf{he.CRand}(\mathsf{pk}, \mathsf{he.Mult}(\mathsf{pk}, c_i^+, g_i)))$$

for $g_i \leftarrow \mathsf{he.Enc}(\mathsf{pk}, x_i^{(\mathsf{Sim})})$ and $i \in [\mathbb{S}_{\mathsf{Sim}}]$. For the remaining set elements, $\mathcal{B}$ uses $(m_{i'}, c_{i'})$ as its input in $\mathsf{exp}^{\mathsf{crand}}_{\omega,\mathcal{A}}(1^\lambda, 1^\ell, \iota)$ where $m_{i'} = 0$ for each $i' \in [|\mathbb{S}_{\mathsf{Sim}}| + 1, |\mathbb{S}_\cup|]$. For each oracle output $\widetilde{c_{i'}^+} \leftarrow \mathcal{O}_{\mathcal{Y}}(\widetilde{c_{i'}^+})$ (and likewise for $\widetilde{d_{i'}^+}$) it sets $W[i'] \leftarrow (\widetilde{c_{i'}^+}, \widetilde{d_{i'}^+})$. It finally adds W to the output distribution seen by $\mathcal{A}$.

Note that in the case where $\omega = 0$ then $\mathcal{A}$ receives new encryptions of 0 at level ι. Therefore this is identical to the real execution of the protocol (by the fact that the rest of View_1 is identical in both executions). When $\omega = 1$, then $\mathcal{A}$ receives random elements from $\mathcal{Y}_0[\iota]$ which are identically distributed to fresh encryptions of 0 at level ι. Therefore, this is identically distributed to $\mathsf{View}_{\mathsf{Sim}}$ in the simulation. As a consequence, $\mathcal{B}$ can use the output of $\mathcal{A}$ as ω and distinguish the two ciphertext rerandomisation experiments with the same probability. Transitively, this shows that $\mathcal{A}$ has advantage bounded by $\mathcal{B}$ and, since he satisfies ciphertext rerandomisation, this is negligible.

The output does not change if $\mathsf{he} = \mathsf{ahe}$, the only part that does — the use of the $\mathsf{he.ScMult}$ algorithm — changes functionality only cosmetically. We also do not have to consider the level of the ciphertext encryption, ι. Therefore the proof of Lemma III;4.1 is complete. □

We now define the action of Sim_2, in the case when $\mathbb{P}_2$ is corrupted (denoted $\mathbb{P}_2^{\mathcal{A}}$) by a PPT adversary $\mathcal{A}$.

Simulation III;4.2 [$\mathsf{Sim}_2(\mathbb{S}_2, \mathsf{aux}_2)$]

Sim_2 runs $\mathsf{BF} \leftarrow \mathsf{BF.init}(1^L, 1^n, 1^k)$ and simply computes

$$\mathsf{EBF} \leftarrow \mathsf{encrypt}(\mathsf{pk}, \mathsf{BF})$$

and sends $\mathsf{EBF.pp} \rightarrow \mathbb{P}_2^{\mathcal{A}}$ (adding to $\mathsf{msgs}_{\mathsf{Sim}}$).
Let $\mathsf{View}_{\mathsf{Sim}} = (\mathbb{S}_2, \emptyset, \mathsf{msgs}_{\mathsf{Sim}}, \mathsf{aux}_2)$ be the view that $\mathbb{P}_2^{\mathcal{A}}$ witnesses in the simulation.

Lemma III;4.2 [$\mathsf{View}_2 \approx_c \mathsf{View}_{\mathsf{Sim}}$]

The view $\mathsf{View}_{\mathsf{Sim}}$ in Simulation III;4.2 is computationally indistinguishable from View_2 in $\psi_\cup$, by the semantic security of he.

[12]Since we have one multiplication over the ciphertexts in the SHE case, then $\iota = 1$.

Proof. The output received by $\mathbb{P}_2^{\mathcal{A}}$ is empty, so the only thing required to simulate is the encrypted Bloom filter that it receives in msgs_2, at the end of $\psi_{\cup}^{1}(\mathbb{P}_1, \mathsf{he})$. The difference between the real execution and Simulation III;4.2 is that: in the former $\mathsf{EBF}.\overline{\mathsf{arr}}$ is created by encrypting the real entries of BF; and in the latter $\mathsf{EBF}.\overline{\mathsf{arr}}$ is an encryption of the string 0^L.

These opposing executions are exactly in the form received by the adversary $\mathcal{B}$ in Lemma III;3.1 ($\mathsf{exp}_{b,\mathcal{B}}^{\mathsf{ebf}}(1^{\lambda})$). Therefore, we can use $\mathcal{B}$ as the simulator where it sets $\mathsf{EBF}.\overline{\mathsf{arr}}$ to be the challenge sample that it receives in the game $\mathsf{exp}_{b,\mathcal{B}}^{\mathsf{ebf}}(1^{\lambda})$. When $b = 0$, the encrypted Bloom filter is as in the real execution, and when $b = 1$ it is as in Simulation III;4.2. Therefore, we can bound the advantage of $\mathcal{A}$ by the advantage of $\mathcal{B}$, which is negligible by the statement of Lemma III;3.1 (by the semantic security of he). The proof is the same for $\mathsf{he} \in \{\mathsf{she}, \mathsf{ahe}\}$ and thus the proof of Lemma III;4.2 is complete. □

The proof of Theorem III;4.2 can be concluded by Lemmas III;4.1 and III;4.2. □

III;5 A protocol for private set intersection

Following on from Construction III;4.1, we give an adapted construction that allows $\mathbb{P}_1$ and $\mathbb{P}_2$ to compute the intersection of $\mathbb{S}_1$ and $\mathbb{S}_2$. In other words, we use the protocol $\psi_{\cap}$ to compute the functionality:

$$(\mu_1, \mu_2) \leftarrow F_{\cap}(\mathbb{S}_1, \mathbb{S}_2) \tag{III;8}$$

where $\mu_1 = \mathbb{S}_1 \cap \mathbb{S}_2$ and $\mu_2 = \emptyset$, as before. We adopt the same conventions for

$$\mathsf{he}, \mathsf{BF}, \mathbb{S}_j, \mathcal{S}, \mathsf{aux}_j, \mathsf{msgs}_j, \mathsf{View}_j$$

as we did in Section III;4, and let $n_j = |\mathbb{S}_j|$. Step one of the protocol does not change from Figure III;2, we provide the PSI versions of steps two and three in Figures III;5 and III;6.

```
$\psi^2_\cap(\mathbb{P}_2, \mathsf{he})$
1 :  $W \leftarrow \emptyset$;
2 :  for $i \in [n_2]$ :
3 :      $\{c_i^l\}_{l \in [k]} \leftarrow \mathsf{EIBF.Query}(\mathsf{EIBF.pp}, x_i^{(2)})$;
4 :      $c_i^+ \leftarrow c_i^1$;
5 :      for $l \in [2, k]$ :
6 :          $c_i^+ \leftarrow \mathsf{he.Add}(\mathsf{pk}, c_i^l, c_i^+)$;
7 :      $r_{i,1}, r_{i,2} \leftarrow\$ \mathcal{X}$;
8 :      $\widehat{c_i^{+,1}} \leftarrow \mathsf{he.PRand}(\mathsf{pk}, c_i^+, r_{i,1})$;
9 :      $\widehat{c_i^{+,2}} \leftarrow \mathsf{he.PRand}(\mathsf{pk}, c_i^+, r_{i,2})$;
10 :     $\widehat{d_i^+} = \mathsf{he.Add}(\mathsf{pk}, \widehat{c_i^{+,2}}, \mathsf{he.Enc}(\mathsf{pk}, x_i^{(2)}))$;
11 :     $(\widetilde{c_i^+}, \widetilde{d_i^+}) \leftarrow (\mathsf{he.CRand}(\mathsf{pk}, \widehat{c_i^{+,1}}), \mathsf{he.CRand}(\mathsf{pk}, \widehat{d_i^+}))$;
12 :  $W[i] \leftarrow (\widetilde{c_i^+}, \widetilde{d_i^+})$;
13 :  $\mathbb{P}_1 \leftarrow W$;
```

Figure III;5: Step two, computed by $\mathbb{P}_2$ in our PSI protocol. This step is similar to Figure III;3, we expect that the output $(\widetilde{c_i^+}, \widetilde{d_i^+})$ are encryptions of $(\mathtt{random}, \mathtt{random})$ when $x_i^{(2)}$ is not in the intersection. The steps only differ for $\mathsf{he} \in \{\mathsf{she}, \mathsf{ahe}\}$ depending on the implementation of $\mathsf{he.PRerand}$. We assume that the set W is shuffled randomly before it is sent to $\mathbb{P}_1$.

```
$\psi^3_\cap(\mathbb{P}_1, \mathsf{he})$
1 :  $\mathbb{S}_\cap \leftarrow \emptyset$;
2 :  for $i \in [n_2]$ :
3 :      $(\widetilde{c_i^+}, \widetilde{d_i^+}) \leftarrow W[i]$;
4 :      $(z_{i,1}, z_{i,2}) \leftarrow (\mathsf{he.Dec}(\mathsf{sk}, \widetilde{c_i^+}), \mathsf{he.Dec}(\mathsf{sk}, \widetilde{d_i^+}))$;
5 :      if $z_{i,1} = 0$ :
6 :          $z_{i,2} \rightarrow \mathbb{S}_\cap$;
7 :  return $\mathbb{S}_\cap$;
```

Figure III;6: Step three, computed by $\mathbb{P}_1$ in our PSI protocol. This step sees $\mathbb{P}_1$ output the intersection of the two sets. If an element is not in the intersection then the plaintext rerandomisation procedure prevents it from being learnt. The steps are the same for $\mathsf{he} \in \{\mathsf{she}, \mathsf{ahe}\}$.

Construction III;5.1 [PSI]

Let $\psi_\cap$ be a protocol for the functionality $F_\cap$ (Equation (III;8)) and let $\mathsf{he}, \mathsf{BF}, \mathbb{P}_j, \mathbb{S}_j, \mathcal{S}, \mathsf{aux}_j, \mathsf{msgs}_j, \mathsf{View}_j$ be described as above.
We construct $\psi_\cap$ as a composition of the steps

- $(\mathbb{P}_2 : \mathsf{EIBF.pp}) \leftarrow \psi^1_\cap(\mathbb{P}_1, \mathsf{he})$ [Figure III;2];
- $(\mathbb{P}_1 : W) \leftarrow \psi^2_\cap(\mathbb{P}_2, \mathsf{EIBF.pp}, \mathsf{he})$ [Figure III;5];
- $(\mathbb{P}_1 : \mathbb{S}_\cap) \leftarrow \psi^3_\cap(\mathbb{P}_1, W, \mathsf{he})$ [Figure III;6].

We set $\mu_1 = \mathbb{S}_\cap$ and $\mu_2 = \emptyset$.

Theorem III;5.1 [Correctness]

Let $\psi_\cap$ be defined as in Construction III;5.1; let BF be the Bloom filter used in $\psi_\cap$, with parameters $k = \mathsf{poly}(\lambda)$ and L, n_1 chosen optimally. Then $\psi_\cap$ correctly computes the functionality $F_\cap$ (Equation (III;8)), with probability greater than $1 - \mathsf{negl}(\lambda)$.

Proof. Similarly to the proof of Theorem III;4.1, we consider the set $\mathbb{S}_\cap$ that is output by $\mathbb{P}_1$. Let $\mathsf{IBF.pp}$ be the same as $\mathsf{EIBF.pp}$, except that $\mathsf{IBF.arrhe.Dec}(\mathsf{sk}, \mathsf{EIBF}.\overline{\mathsf{arr}})$. We have that

$$\begin{aligned} z_{i,1} &= \left(\mathsf{IBF.Query}(\mathsf{IBF.pp}, x_i^{(2)}))\right) \cdot r_{i,1}; \\ z_{i,2} &= \left(\left(\mathsf{IBF.Query}(\mathsf{IBF.pp}, x_i^{(2)})\right) \cdot r_{i,2}\right) + x_i^{(2)}; \end{aligned}$$

where $x_i^{(2)} \in \mathbb{S}_2$ and $i \in [n_2]$, by the correctness of he and by Lemma II;5.3 (rerandomisation does not affect the plaintext value). Since $\epsilon = \mathsf{negl}(k)$, and $k = \mathsf{poly}(\lambda)$, then $z_{i,1} = 0$ implies that $x_i^{(2)} \in \mathbb{S}_1$ with all but negligible probability. Moreover, in these cases we have that $x_i^{(2)} = z_{i,2}$ since

$$\left(\mathsf{IBF.Query}(\mathsf{IBF.pp}, x_i^{(2)})\right) \cdot r_{i,2} = 0$$

with the same probability. We add $z_{i,2} \to \mathbb{S}_\cap$ directly for each $i \in [n_2]$, meaning that we add $x_i^{(2)}$ to $\mathbb{S}_\cap$ iff $x_i^{(2)} \in \mathbb{S}_1$ (with all but negligible probability ϵ). This is exactly the intersection of the two sets and thus $\mathbb{S}_\cap = \mathbb{S}_1 \cap \mathbb{S}_2$. This also follows in the same way if $\mathsf{he} = \mathsf{ahe}$ since $z_{i,j}$ is the same in both cases for $j \in \{1, 2\}$ (since $\mathsf{ahe.ScMult}$ induces the same plaintext value). □

Theorem III;5.2 [Security]

Let $\psi_\cap$ be defined as in Construction III;5.1. Then $\psi_\cap$ securely computes the functionality $F_\cap$ in the presence of PPT semi-honest adversaries, under the semantic security of he.

Proof. The simulation in the case of $\mathbb{P}_2$ being corrupted is exactly the same as Simulation III;4.2 in the proof of Theorem III;4.2. For the case when $\mathbb{P}_1$ is corrupted (written $\mathbb{P}_1^{\mathcal{A}}$ for a PPT adversary $\mathcal{A}$) then the manner of Sim_1 changes slightly.

Simulation III;5.1 [$\mathsf{Sim}_1(\mathbb{S}_1, \mathbb{S}_\cap, \mathsf{aux}_1)$]

- Sim_1 uses the input set $\mathbb{S}_{\mathsf{Sim}} = \mathbb{S}_\cap$ and receives $\mathsf{EIBF.pp}$ from $\mathbb{P}_1^{\mathcal{A}}$.
- For each $x_i^{(\mathsf{Sim})} \in \mathbb{S}_{\mathsf{Sim}}$ where $i \in [|\mathbb{S}_{\mathsf{Sim}}|]$; it computes:

$$(\widetilde{c_i^+}, \widetilde{d_i^+}) \leftarrow \\ (\mathsf{he.CRand}(\mathsf{pk}, \mathsf{he.Enc}(\mathsf{pk}, 0)), \mathsf{he.CRand}(\mathsf{pk}, \mathsf{he.Enc}(\mathsf{pk}, x_i^{(\mathsf{Sim})}))),$$

  and adds $(\widetilde{c_i^+}, \widetilde{d_i^+}) \to W_{\mathsf{Sim}}[i]$.
- For the remaining $|\mathbb{S}_2 \setminus \mathbb{S}_{\mathsf{Sim}}|$ elements, it sets

$$(\widetilde{c_{i'}^+}, \widetilde{d_{i'}^+}) \leftarrow \\ (\mathsf{he.CRand}(\mathsf{pk}, \mathsf{he.Enc}(\mathsf{pk}, r_{i',1})), \mathsf{he.CRand}(\mathsf{pk}, \mathsf{he.Enc}(\mathsf{pk}, r_{i',2})))$$

  for $r_{i',1}, r_{i',2} \leftarrow_{\$} \mathcal{X}$, and adds $(c_{i'}^+, d_{i'}^+) \to W_{\mathsf{Sim}}[i']$.
- Finally, Sim_1 shuffles W_{Sim} and sends it to $\mathbb{P}_1^{\mathcal{A}}$ (i.e. $W_{\mathsf{Sim}} \to \mathsf{msgs}_{\mathsf{Sim}}$).
- Let $\mathsf{View}_{\mathsf{Sim}} = (\mathbb{S}_1, \mathbb{S}_\cap, \mathsf{msgs}_{\mathsf{Sim}}, \mathsf{aux}_1)$ be the simulated view for $\mathbb{P}_1^{\mathcal{A}}$.

Lemma III;5.1 [$\mathsf{View}_1 \approx_p \mathsf{View}_{\mathsf{Sim}}$]

$\mathsf{View}_{\mathsf{Sim}}$ in Simulation III;5.1 is computationally indistinguishable from View_1 in $\psi_\cap$, where he satisfies the plaintext rerandomisation property of Definition II;5.8.

Proof. The views View_1 and $\mathsf{View}_{\mathsf{Sim}}$ in the real execution $\psi_\cap$ and Simulation III;5.1 differ only in msgs_1 and $\mathsf{msgs}_{\mathsf{Sim}}$; in the way that the plaintexts are created for each of the encryptions. By a similar argument to the proof of Lemma III;4.1, the ciphertext rerandomisation means that the ciphertexts are indistinguishable from all other ciphertexts in the same codomain inferred by the plaintext value.

For the case of $(\widetilde{c_i^+}, \widetilde{d_i^+})$, corresponding to $x_i^{(\mathsf{Sim})} \in \mathbb{S}_{\mathsf{Sim}} = \mathbb{S}_\cap$. Notice that the plaintexts in $\psi_\cap$ are the same as the plaintexts in Simulation III;5.1 with probability $(1-\epsilon) \approx 1$ by $\epsilon < \mathsf{negl}(\lambda)$. Therefore, we focus on the cases of $(\widetilde{c_{i'}^+}, \widetilde{d_{i'}^+})$ for $x_i^{(2)} \notin \mathbb{S}_1$ (not in the intersection). In the real execution, the plaintexts take the form:

$$(r_{i'',1} \cdot k_i, (r_{i'',2} \cdot k_i) + x_i^{(2)}) = (\mathsf{he.PRand}(\mathsf{pk}, \widehat{c_i^{+,1}}, r_{i'',1}), \mathsf{he.PRand}(\mathsf{pk}, \widehat{c_i^{+,2}}, r_{i'',2})) \tag{III;9}$$

for $r_{i'',j} \leftarrow\!\$\, \mathcal{X}$ $(j \in \{1,2\})$ and where $k_i \in [1,k]$, since they are outputs of the plaintext rerandomisation algorithm $\mathsf{he.PRerand}$. In the simulation the plaintexts are simply the random elements

$$(r_{i',1}, r_{i',2}) \leftarrow\!\$\, \mathcal{X}^2. \tag{III;10}$$

These distributions can be trivially constructed from the outputs of $\mathsf{exp}^{\mathsf{prand}}_{\omega,\mathcal{A}}(1^\lambda, 1^\ell)$ for $\omega \in \{0,1\}$. Therefore, any adversary $\mathcal{A}$ attempting to distinguish between the two views immediately leads to an adversary $\mathcal{B}$ that attempts to break plaintext rerandomisation for he. By the plaintext rerandomisation security of he, we infer that $\mathcal{A}$ has negligible advantage of distinguishing the views and the proof is complete (for $\mathsf{he} \in \{\mathsf{she}, \mathsf{ahe}\}$). □

The proof of Theorem III;5.2 is completed by the proof of Lemma III;5.1. □

III;6 A protocol for private set union/intersection cardinality

We give a further adaptation of our framework that allows $\mathbb{P}_1$ and $\mathbb{P}_2$ to compute the cardinality of either the intersection, or the union, of $\mathbb{S}_1$ and $\mathbb{S}_2$. In other words, we use the protocol $\psi_{|\cap|}$ to compute the functionality:

$$(\mu_1, \mu_2) \leftarrow F_{|\cap|}(\mathbb{S}_1, \mathbb{S}_2) \tag{III;11}$$

where $\mu_1 = |\mathbb{S}_1 \cap \mathbb{S}_2|$ and $\mu_2 = \emptyset$, as before. Using the relation in Equation (III;1), we can compute

$$F_{|\cup|}(\mathbb{S}_1, \mathbb{S}_2) = |\mathbb{S}_1| + |\mathbb{S}_2| - |\mathbb{S}_1 \cap \mathbb{S}_2|$$

using only knowledge of $|\mathbb{S}_j|$ for $j \in \{1,2\}$ (where $|\mathbb{S}_{\bar{j}}|$ is provided in aux_j, and thus both values are known to both players). Therefore, we will only focus on the case of giving a protocol for $F_{|\cap|}$.

We adopt the same conventions for

$$\mathsf{he}, \mathsf{BF}, \mathbb{S}_j, \mathcal{S}, \mathsf{aux}_j, \mathsf{msgs}_j, \mathsf{View}_j$$

```
$\psi^2_{|\cap|}(\mathbb{P}_2, \mathsf{he})$
1 :  $W \leftarrow \emptyset$;
2 :  for $i \in [n_2]$ :
3 :      $\{c_i^l\}_{l\in[k]} \leftarrow \mathsf{EIBF.Query}(\mathsf{EIBF.pp}, x_i^{(2)})$;
4 :      $c_i^+ \leftarrow c_i^1$;
5 :      for $l \in [2, k]$ :
6 :          $c_i^+ \leftarrow \mathsf{he.Add}(\mathsf{pk}, c_i^l, c_i^+)$;
7 :      $r_i \leftarrow_\$ \mathcal{X}$;
8 :      $\widehat{c_i^+} \leftarrow \mathsf{he.PRand}(\mathsf{pk}, c_i^+, r_i)$;
9 :      $\widetilde{c_i^+} \leftarrow \mathsf{he.CRand}(\mathsf{pk}, \widehat{c_i^+})$;
10 :     $\widetilde{c_i^+} \leftarrow W[i]$;
11 : $W \leftarrow \mathbb{P}_1$
```

Figure III;7: Step two, computed by $\mathbb{P}_2$ in our PSU/I-CA protocol and similar to the approach in Figure III;5. We assume that the set W is shuffled randomly before it is sent to $\mathbb{P}_1$.

```
$\psi^3_{|\cap|}(\mathbb{P}_1, \mathsf{he})$
1 :  $n_\cap = 0$;
2 :  for $i \in [n_2]$ :
3 :      $\widetilde{c_i^+} \leftarrow W[i]$;
4 :      $z_i \leftarrow \mathsf{he.Dec}(\mathsf{sk}, \widetilde{c_i^+})$;
5 :      if $z_i = 0$ :
6 :          $n_\cap = n_\cap + 1$;
7 :  return $n_\cap$;
```

Figure III;8: Step three, computed by $\mathbb{P}_1$ in our PSU/I-CA protocol. Here, $\mathbb{P}_1$ decrypts non-zero plaintext values if the element sent in step two was taken from outside of the intersection. The steps are the same for $\mathsf{he} \in \{\mathsf{she}, \mathsf{ahe}\}$. For $\psi_{|\cup|}$, return $n_\cup = n_1 + n_2 - n_\cap$.

as we did in the previous sections, and let $n_j = |\mathbb{S}_j|$. Step one of the protocol does not change from Figure III;2, we provide the PSU/I-CA versions of steps two and three in Figures III;7 and III;8.

Construction III;6.1 [PSU/I-CA]

Let $\psi_{|\cap|}$ be a protocol for the functionality $F_{|\cap|}$ (Equation (III;7)) and let $\mathsf{he}, \mathsf{BF}, \mathbb{P}_j, \mathbb{S}_j, \mathcal{S}, \mathsf{aux}_j, \mathsf{msgs}_j, \mathsf{View}_j$ be described as above.
We construct $\psi_{|\cap|}$ as a composition of the steps

- $(\mathbb{P}_2 : \mathsf{EIBF.pp}) \leftarrow \psi^1_{|\cap|}(\mathbb{P}_1, \mathsf{he})$ [Figure III;2];
- $(\mathbb{P}_1 : W) \leftarrow \psi^2_{|\cap|}(\mathbb{P}_2, \mathsf{EIBF.pp}, \mathsf{he})$ [Figure III;7];
- $(\mathbb{P}_1 : n_\cap) \leftarrow \psi^3_{|\cap|} W, (\mathbb{P}_1, \mathsf{he})$ [Figure III;8].

We write $\mu_1 = n_\cap$ and $\mu_2 = \emptyset$.
We define $\psi_{|\cup|}$ in the same way, except that $\mu_1 = n_\cup = n_1 + n_2 - n_\cap$.

Theorem III;6.1 [Correctness]

Let $\psi_{|\cap|}$ be defined as in Construction III;6.1; let BF be the Bloom filter used in $\psi_{|\cap|}$, with parameters $k = \mathsf{poly}(\lambda)$ and L, n_1 chosen optimally. Then $\psi_{|\cap|}$ correctly computes the functionality $F_{|\cap|}$ (Equation (III;11)) with probability greater than $1 - \mathsf{negl}(\lambda)$. Respectively, $\psi_{|\cup|}$ correctly computes $F_{|\cup|}$, with the same probability.

Proof. By the correctness of he and Lemma II;5.3, then

$$z_i = \left(\mathsf{IBF.Query}(\mathsf{IBF.pp}, x_i^{(2)})\right) \cdot r_i$$

for $i \in [n_2]$. Thus, $z_i = 0$ with probability $(1 - \epsilon) \approx 1$ iff $x_i^{(2)} \in \mathbb{S}_1$ (by $\epsilon = \mathsf{negl}(k)$). When $z_i = 0$ then $n_\cap$ is incremented, thus $n_\cap$ is incremented when $x_i^{(2)} \in \mathbb{S}_1$ which is a count of the intersection of the two sets. The same applies if $\mathsf{he} = \mathsf{ahe}$ by the same arguments as before. Moreover, $n_\cup$ is a count of the union of the two sets by Equation (III;1). Therefore the proof of correctness is complete. □

Theorem III;6.2 [Security]

Let $\psi_{|\cap|}$ be defined as in Construction III;6.1. Then $\psi_{|\cap|}$ securely computes the functionality $F_{|\cap|}$ in the presence of PPT semi-honest adversaries, under the semantic security of he. Respectively, $\psi_{|\cup|}$ securely computes $F_{|\cup|}$ under the same assumption.

Proof. The proof of the case when $\mathbb{P}_2$ is corrupted does not change at all. The proof of the case when $\mathbb{P}_1$ is corrupted changes slightly from Theorem III;4.2. We define the simulator, Sim_1, below.

Simulation III;6.1 [$\mathsf{Sim}_1(\mathbb{S}_1, n_\cap, \mathsf{aux}_1)$]

- Sim_1 receives $\mathsf{EIBF.pp}$ from $\mathbb{P}_1^\mathcal{A}$.
- It computes
$$\widetilde{c_i^+} \leftarrow \mathsf{he.CRand}(\mathsf{pk}, \mathsf{he.Enc}(\mathsf{pk}, 0)),$$
for $i \in [n_\cap]$, and adds $\widetilde{c_i^+} \to W_{\mathsf{Sim}}[i]$.
- For the remaining set elements, it sets
$$c_{i'}^+ \leftarrow \mathsf{he.CRand}(\mathsf{pk}, \mathsf{he.Enc}(\mathsf{pk}, r_{i'}))$$
for $r_{i'} \leftarrow_\$ \mathcal{X}$, and adds $c_{i'}^+ \to W_{\mathsf{Sim}}[i']$.
- Finally, Sim_1 shuffles W_{Sim} and sends it to $\mathbb{P}_1^\mathcal{A}$ ($W_{\mathsf{Sim}} \to \mathsf{msgs}_{\mathsf{Sim}}$).
- Let $\mathsf{View}_{\mathsf{Sim}} = (\mathbb{S}_1, \mathbb{S}_{\mathsf{Sim}}, \mathsf{msgs}_{\mathsf{Sim}}, \mathsf{aux}_1)$ be the view that $\mathbb{P}_1^\mathcal{A}$ witnesses.

Lemma III;6.1 [$\mathsf{View}_1 \approx_p \mathsf{View}_{\mathsf{Sim}}$]

$\mathsf{View}_{\mathsf{Sim}}$ in Simulation III;6.1 is perfectly indistinguishable from View_1 in $\psi_\cap$.

Proof. The proof of this lemma is a subset of the proof of Lemma III;5.1, therefore we refer the reader to that proof for the explicit details. Notice that the ciphertext rerandomisation procedure distributes ciphertexts across the codomain of ciphertexts with the same plaintext value. Moreover, the plaintext rerandomisation hides the information that decryption reveals to $\mathbb{P}_1^\mathcal{A}$ for elements that are not in the intersection. □

The proof of Theorem III;6.2 follows directly from the proof of Lemma III;6.1. The proof in the case of $\psi_{|\cup|}$ is exactly the same. □

III;7 Asymptotic analysis

To finish the analysis of our protocols in the semi-honest security model, we discuss the asymptotic complexity of our designs. For this analysis, we use the optimal parameter settings for a Bloom filter setup, that are given in Equations (III;5) and (III;6); that is $L = \log(e) \cdot k \cdot n_1$, where

$n_j = |\mathbb{S}_j|$ and $k = \log(\epsilon)$ for the false-positive probability $\epsilon = \mathsf{negl}(\lambda)$. Our only requirement is that $k = \mathsf{poly}(\lambda)$, so that $\epsilon = \mathsf{negl}(\lambda)$.

The following lemmas establish the complexities for both communication and computation for all of our protocols. Our complexities are calculated wrt the black-box definition of she given in Definition II;5.5. Hence, this analysis ignores the practical advantage of using ahe over she. We ignore λ factors in our complexity analysis as we focus on the relationship of our overheads to the input sizes, rather than the security parameters that are chosen (which is consistent over all protocols).

Lemma III;7.1 [Communication complexity]

The communication complexity in $\psi_\cup$, $\psi_\cap$, $\psi_{|\cap|}$ and $\psi_{|\cup|}$ is $O(kn_1 + n_2)$ (in terms of the number of ciphertexts that are sent).

Proof. In $\psi^1_\cup(\mathbb{P}_1, \mathsf{he})$, $\mathbb{P}_1$ sends $O(L) = O(kn_1)$ ciphertexts in $\mathsf{EBF}.\overline{\mathsf{arr}}$ to $\mathbb{P}_2$ (the rest of $\mathsf{EBF.pp}$ can be persisted elsewhere). In $\psi^2_\cup(\mathbb{P}_2, \mathsf{EIBF.pp}, \mathsf{he})$, $\mathbb{P}_2$ sends 2 ciphertexts for each $x^{(2)}_i \in \mathbb{S}_2$ to $\mathbb{P}_1$, which is $O(n_2)$. Thus the complexity follows. The asymptotic complexities are the same for $\psi_\cap$, $\psi_{|\cap|}$, $\psi_{|\cup|}$ (the concrete cost for $\mathbb{P}_2$ is lower in $\psi_{|\cap|}$ and $\psi_{|\cup|}$). □

Lemma III;7.2 [$\mathbb{P}_1$ computational complexity]

The asymptotic runtime of $\mathbb{P}_1$ in $\psi_\cup$, $\psi_\cap$, $\psi_{|\cap|}$ and $\psi_{|\cup|}$ is $O(kn_1)$ encryptions for $\mathsf{he} \in \{\mathsf{she}, \mathsf{ahe}\}$.

Proof. In $\psi^1_\cup(\mathbb{P}_1, \mathsf{he})$, $\mathbb{P}_1$ first computes $O(n_1)$ $\mathsf{BF.Store}$ operations (one for each $x^{(1)}_i \in \mathbb{S}_1$). They compute $L = O(kn_1)$ encryptions in the computation $\mathsf{EIBF} = \mathsf{encrypt}(\mathsf{pk}, \mathsf{invert}(\mathsf{BF}))$. The total number of operations for step one is therefore bounded by $O(kn_1 + n_2)$. This is exactly the same in the protocols $\psi_\cap$, $\psi_{|\cap|}$, $\psi_{|\cup|}$, as well.

For $\psi^3_{\cup(\mathbb{P}_1, W, \mathsf{he})}$, $\mathbb{P}_1$ computes $O(n_2)$ decryptions, two for each $x^{(2)}_i \in \mathbb{S}_2$ (one in the case of $\psi_{|\cap|}$ and $\psi_{|\cup|}$). In $\psi_\cup$, $\mathbb{P}_1$ computes an additional $n_2 - |\mathbb{S}_1 \cap \mathbb{S}_2|$ modular inversions and multiplications. Thus the total complexity for this step is $O(n_2)$.

The total complexity for the whole of $\mathbb{P}_1$'s operations is $O(kn_1 + n_2)$, in all of the protocols. □

Lemma III;7.3 [$\mathbb{P}_2$ computational complexity]

The asymptotic runtime of $\mathbb{P}_2$ in $\psi_\cup$, $\psi_\cap$, $\psi_{|\cap|}$ and $\psi_{|\cup|}$ is $O(kn_2)$, where $\mathsf{he} = \mathsf{she}$, and $O(kn_2 \log_2(|\mathcal{X}|))$, where $\mathsf{he} = \mathsf{ahe}$.

Proof. In $\psi_\cup^2(\mathbb{P}_2, \mathsf{she})$, $\mathbb{P}_2$ computes k homomorphic additions for each $x_i^{(2)} \in \mathbb{S}_2$, i.e. $O(kn_2)$. On top of this, $\mathbb{P}_2$ also computes $O(n_2)$ ciphertext rerandomisations, n_2 encryptions of each $x_i^{(2)} \in \mathbb{S}_2$ and n_2 homomorphic multiplications. This is the same in $\psi_\cap^2(\mathbb{P}_2, \mathsf{EIBF.pp}, \mathsf{she})$; and in $\psi_{|\cap|,\mathsf{EIBF.pp}}^2(\mathbb{P}_2, \mathsf{she})$ the number of fresh encryptions of 0 is reduced but is still $O(n_2)$. Thus the total number of operations is $O(kn_2)$.

In $\psi_\cup^2(\mathbb{P}_2, \mathsf{ahe})$, $\mathbb{P}_2$ computes k homomorphic additions for each $x_i^{(2)} \in \mathbb{S}_2$, i.e. $O(kn_2)$. On top of this, $\mathbb{P}_2$ also computes $O(n_2)$ fresh encryptions of 0 (two for each $x_i^{(2)} \in \mathbb{S}_2$ for ciphertext rerandomisation), and n_2 homomorphic scalar multiplications (one for each plaintext rerandomisation invocation). Each scalar multiplication requires sampling a random mask r_i for each $x_i^{(2)}$ and thus computing $\log_2(|\mathcal{X}|)$ homomorphic additions. In total, for the entirety of $\mathbb{S}_2$, this requires $O(n_2 \log_2(|\mathcal{X}|))$ operations. This is the same in $\psi_\cap^2(\mathbb{P}_2, \mathsf{EIBF.pp}, \mathsf{ahe})$, and in $\psi_{|\cap|}^2(\mathbb{P}_2, \mathsf{EIBF.pp}, \mathsf{ahe})$ the number of fresh encryptions of 0 is reduced but is still $O(n_2)$. Thus the total number of operations is $O(kn_2 \log_2(|\mathcal{X}|))$. □

Lemma III;7.4 [Total computational complexity]

The total asymptotic computational complexity for all of our protocols is $O(k(n_1 + n_2))$ when $\mathsf{he} = \mathsf{she}$, and $O(k(n_1 + n_2) \log_2(|\mathcal{X}|))$ when $\mathsf{he} = \mathsf{ahe}$.

Proof. By the results of Lemma III;7.2, $\mathbb{P}_1$ computes $O(kn_1 + n_2)$ operations in both cases of $\mathsf{he} \in \{\mathsf{she}, \mathsf{ahe}\}$. By Lemma III;7.3, $\mathbb{P}_2$ computes $O(kn_2)$ operations when $\mathsf{he} = \mathsf{she}$, and $O(kn_2 \log_2(|\mathcal{X}|))$ when $\mathsf{he} = \mathsf{ahe}$. The total complexity follows immediately by taking the dominant terms in both cases. □

III;7.1 Comparison with previous work

In this comparison, we will naturally be targeting previous work that constructs protocols for all of the three set operations that we consider. As such we will draw comparisons with the works of Kissner and Song [211], Blanton and Aguiar [47] and a combination of Freedman et al., Frikken, and Hazay and Nissim [143, 146, 181] (since the constructions are very similar). We will also assume that we have equal set sizes (i.e. $n_1 = n_2 = n$) for ease of exposition. This is a fair assumption, since the smaller set can be padded to the size of the bigger set generically.

Complexities of previous toolkits. As we mentioned previously (Table III;1), the constructions based on OPE [143, 146, 211] have an inherent reliance on complexities that are super-linear in n. Without optimisations, all of these proposals demonstrate complexities that are $O(n)$ for communication, and $O(n \log \log n)$ for computation (when $\mathsf{he} = \mathsf{she}$).

For [47], they use a much different technique that works for multiple parties and multisets. However, their method (when reduced to the two-party case) requires $O(n \log(n))$ for both communication and computation, which is sub-optimal in comparison with the OPE techniques. Of course, the use of secret-sharing operations (rather than ahe and public-key techniques) means that their method is likely to perform better in a practical scenario, since they use inherently symmetric techniques. However, we are focusing only on the asymptotic scalability of the proposals for now. Furthermore, the proposal of [47] does not currently provide experimental analysis of their constructions.

Our complexities. The idea behind the improvement that we make is that we are able reduce the dependence of the computational complexity on the size of the set to a linear relation, in each of our protocols. A Bloom filter query requires k hash function evaluations and this determines that only k homomorphic additions have to be made by $\mathbb{P}_2$. By Equation (III;6), k is only chosen so that ϵ is negligible and thus has no dependence on n. Thus our complexities can be expressed as $O(kn) = O(L)$, where L is the length of the Bloom filter. We reiterate that we have linearity in the number of ciphertexts that are sent.

The costs of our findings is that we introduce the possibility (with probability $\epsilon = \mathsf{negl}(\lambda)$) of false-positives, and our complexities are now effectively 'quadratic' in two variables (n and k) rather than just n. For the former consideration, it is simply a matter of choosing $k = \mathsf{poly}(\lambda)$ appropriately. For the latter point, we note that k can effectively be fixed for some setting of λ. As such, increasing the set size will still only have a linear effect on the runtime of the protocols that we have constructed.

With this in mind, and practically speaking, the effect of k on our complexities is essentially as a constant, where typical parameter settings have that $k \ll n$. Therefore, fixing k for a particular choice of λ, we obtain complexities that are $O(n)$ for both communication and computation; in the case where we use an SHE scheme for encryption.

In the case where we use an AHE scheme, the complexities increase by a factor of $\log_2(|\mathcal{X}|)$, since we require sampling a random element from $\mathcal{X}$ and scalar multiplying by this value. Typically, we would expect there to be a dependence between n and $|\mathcal{X}|$. Additionally, we require the invariant that $n < |\mathcal{X}|$ (even if $\mathcal{X}$ is fixed). Therefore, this change affects the scalability considerably. This change also affects the complexities of the OPE-based protocols. The advantage of using AHE schemes is that they are much simpler to implement, and are typically far more efficient than an SHE scheme. In our proof-of-concept implementation in Section III;8, we choose to implement our scheme using the Paillier ahe scheme due to its simplicity. Future work, would establish whether the improved scalability of protocol in the she case translates to a more efficient scheme (especially for larger sets).

Ineffectiveness of bucket-hashing. Using the bucket-hashing optimisation of [143], for reducing computational OPE costs, does not result in the same reduction for our protocols. In particular, while hashing the set elements into buckets is likely to result in Bloom filters that are much shorter ($L_t = O(k \log\log(n))$ for each bucket B_t), the overall effect is that $\mathbb{P}_2$ would still have to compute $O(kn_2)$ operations, since k is not dependent on the set size n.

III;7.2 Correction of claims in [117]

In [117], it was erroneously claimed that the complexities for the scheme in the case where $\mathsf{he} = \mathsf{ahe}$ are $O(kn)$. We would like to make explicit that the complexities are *only* $O(kn)$ when $\mathsf{he} = \mathsf{she}$. As noted above, the complexities are $O(kn \log_2(|\mathcal{X}|))$ in the case where $\mathsf{he} = \mathsf{ahe}$ due to the usage of scalar multiplication. This does not affect the experimental results from [117]; we only correct the claims of the asymptotic efficiency of the protocols.

III;8 Toolkit implementation

While the asymptotic efficiency of our protocols differ quite starkly depending on the choice of he scheme, it is not clear how the relative efficiency of ahe schemes (against more complex she alternatives) could potentially lead to comparably efficient schemes. To this end, we demonstrate an implementation of our scheme using the Paillier AHE scheme (Definition II;5.2).

The aim with this implementation is to demonstrate that our scheme performs favourably against the following criteria.

1. easy to implement;
2. minimal differences in implementations of different protocols;
3. practical to run for large set sizes and believable security parameters.

We believe that satisfying these criteria makes a protocol framework *developer-friendly*, in that it should facilitate the writing of efficient implementations with limited scope for errors. Currently, the available implementations of PSO protocols: only satisfy PSI and PSU/I-CA functionality;

they are written in highly optimised formats that are difficult to understand for non-experts [260]. To the best of our knowledge, this represents the first attempt at formally implementing a toolkit that computes PSU, PSI and PSU/I-CA using a unified design technique.

Our implementation helps us to answer in the affirmative for our criteria. Firstly for (1), we show that our scheme can be implemented easily providing knowledge of a suitable AHE scheme implementation. To expose this advantage, we give an implementation of our protocol in the language `Go` [285]. `Go` is a popular language with performance that is not too far away from older languages like C. As such, it provides a valuable middle-ground between the use of performance-focused programming languages, and the ease-of-use of languages like Python. We use an implementation of the Paillier AHE scheme for implementing our protocols, the performance of which has been benchmarked favourably against other high-performance implementations [40]. The code is open-source.[13] The mathematical operations that we require are written in the `Go` native `math/big` library. Another reason that we chose `Go` is that it demonstrates features that make implementing concurrency simple (using `goroutines`). This is beneficial for our design as there are a lot of operations that can be run in parallel (such as the encryption and homomorphic operations).

For (2), since the protocol only deviates from step 2, we can reuse much of the same computation methods (with minor mathematical differences that are illustrated in the previous sections). Therefore, the toolkit that we propose results in a clean code-base of only 281 lines for the PSO computation. We include a separate implementation of the API between the protocol and the encrypted Bloom filter that is used. The number of lines used to implement the Bloom filter framework is 432, and thus still very concise.

For (3), our experimental data supports the assertion that our toolkit can be run efficiently, even for large set sizes. While our design is much slower than dedicated protocols for computing PSI, our techniques provides much greater PSO functionality. Moreover, much of the runtime ($\approx 90\%$) of our design is contained in the initial encryption of the Bloom filter. We show later that this encryption can be precomputed offline, at any time. Moreover, as we showed in Definition III;3.3, the functionality of the Bloom filter is preserved after encryption. This allows further precomputation that allows the holder to keep updating the Bloom filter after encryption has already taken place. For instance, one can precompute encryptions of 0 for rerandomising the Bloom filter between protocol instances. Once this initial cost is removed, our protocols can be run very efficiently.

All the code for our toolkit is open-source including code for the PSO toolkit;[14] and the implementation of the encrypted Bloom filter API.[15] The implementation is a proof-of-concept in the sense that the experiments merely test the functionality and do not implement fully operational

[13] `https://github.com/snipsco/paillier-libraries-benchmarks`
[14] `https://github.com/alxdavids/pso-toolkit`
[15] `https://github.com/alxdavids/yabf`

clients for performing the PSO protocol. Our main focus is on exploring the efficiency of the primitive that we built. Additionally, mapping our implementation into a fully-operational design is simply a matter of writing trivial servers and clients, that communicate using the same messages in our proof-of-concept proposal.

III;8.1 Parameter choices

All experiments[16] have been run on hardware with 256GB RAM with an Intel(R) Xeon(R) CPU E5-2667 v2 @ 3.30GHz and utilising a maximum of 8 cores (when parallel computation is required). We examine running times for sets sizes ranging from 2^8 to 2^{18} elements; this includes sets of the same size as shown in the comparison of [260] (the largest scale comparison of PSI protocols, to date).

We choose a false positive probability of $\epsilon = 2^{-30}$ (i.e. $k = 30$ by Equation (III;6)) alongside the choice of optimal parameters for our Bloom filter. For instance, for $n = 256$, this implies that $L \approx 11000$ by Definition III;5.

For the Paillier encryption scheme we experiment with moduli N with bit-lengths 1024 and 2048 roughly equivalent to 80 and 112 bit security [247]. We chose the domain of possible elements to be $5n$ where n is the set size and we choose the sets at random from this domain. This choice was made merely to guarantee that the size of the intersection is not trivially close to 0, ensuring a realistic simulation. If we chose the domain to be too large, then choosing random sets would leave the intersection close to 0 with high probability.

During our experimentation we make use of concurrency features in `Go` to make significant savings via parallel execution of operations. Times were $\sim 3\times$ quicker using parallel execution (via `goroutines`) and thus we do not present our single-threaded results.

III;8.2 Experimental results and discussion

In Table III;3, we give the total runtimes for our protocols, using the parameter settings from above. Table III;4 analyses the portion of the runtimes that are allocated to encrypting the Bloom filter that is used in the PSO protocol. Finally, Table III;5 analyses the concrete, maximum communication overheads for the protocols (in MB).

Comparable implementations from previous work. In Table III;6 and Table III;7 we provide details from experiments performed by Pinkas et al. [260] that compared contemporary, state-of-the-art PSI protocols. We do not include the full experimental result set, only analysing

[16]Using the version of code corresponding to commit `426b6d4fbbd8a6f960b6e75bf03a15d6e9b6392a`.

Set size	Timings	PSU	PSI	CA
	Hom. ops	0.49	0.5	0.5
2^8	Out time	0.56	0.54	0.55
	Full time	**11.78**	**11.76**	**11.75**
	Hom. ops	1.94	1.96	1.95
2^{10}	Out time	2.21	2.2	2.22
	Full time	**44.73**	**44.68**	**44.7**
	Hom. ops	7.82	7.82	7.87
2^{12}	Out time	8.61	8.74	8.86
	Full time	**175.7**	**175.79**	**175.96**
	Hom. ops	31.37	31.32	31.59
2^{14}	Out time	35.78	34.9	35.48
	Full time	**702.4**	**702.39**	**703.24**
	Hom. ops	126.16	127.43	127.01
2^{16}	Out time	141.72	138.82	141.76
	Full time	**2836.5**	**2834.68**	**2837.19**
	Hom. ops	510.19	503.95	508.53
2^{18}	Out time	536.48	556.72	556.05
	Full time	**11341.2**	**11327.78**	**11331.67**

Set size	Timings	PSU	PSI	CA
	Hom. ops	3.33	3.36	3.33
2^8	Out time	3.66	3.55	3.58
	Full time	**78.02**	**77.76**	**77.76**
	Hom. ops	13.45	13.33	13.44
2^{10}	Out time	14.77	14.26	14.31
	Full time	**312.44**	**311.61**	**311.76**
	Hom. ops	52.97	53.41	53.15
2^{12}	Out time	55.59	57.98	56.44
	Full time	**1233.59**	**1235.69**	**1233.84**
	Hom. ops	212.33	212	212.55
2^{14}	Out time	228.13	223.31	225.11
	Full time	**4952.94**	**4947.32**	**4949.66**
	Hom. ops	856.27	859.67	857.9
2^{16}	Out time	902.81	906.9	907.27
	Full time	**19881.51**	**19888.79**	**19887.17**
	Hom. ops	3411.87	3416.9	3419.2
2^{18}	Out time	3580.25	3595	3575.94
	Full time	**79272.48**	**79290.82**	**79274.15**

Table III;3: Runtimes (s) for increasing set sizes, left = 1024-bit moduli, right = 2048-bit. 'Hom. ops' refers to time taken for homomorphic operations; 'Out time' refers to time taken to compute output; 'Full time' includes time for encryption from Table III;4.

	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}
1024 bits	10.7	40.53	159.23	636.17	2568.41	10267.03
2048 bits	70.85	284.02	1124.3	4512	18122	72278.95

Table III;4: Bloom filter encryption times (s)

Set sizes	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}
Comms (MB)	2.83	11.32	45.28	181.12	724.49	2897.97

Table III;5: Maximum communication costs (MB) for our protocols for 1024 bit security.

λ	80					112				
Set sizes	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}
De Cristofaro et al. [123]	0.5	2.0	7.9	31.3	124.9	7.7	31.0	124.3	497.2	1982.1
Huang et al. [191]	1.2	5.1	21.2	100.3	462.7	1.9	7.8	36.5	168.9	762.4
Dong et al. [130]	0.15	0.5	2.0	8.1	34.3	0.27	1.0	4.1	16.7	67.6

Table III;6: Runtimes (s) taken from [260]

λ	80					112				
Set sizes	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}
De Cristofaro et al. [123]	0.3	1.1	4.3	17.3	69.0	0.8	3.1	12.5	50.0	200.0
Huang et al. [191]	18.8	90.0	420.0	1920.0	8640.0	30.0	144.0	672.0	3072.0	13824.0
Dong et al. [130]	1.1	4.5	18.1	72.6	290.4	2.9	11.6	46.2	184.9	739.7

Table III;7: Communication costs (MB) taken from [260]

protocols necessitating public-key operations [123, 191], or that are built using similar primitives [130]. We use these tables to offer some non-rigorous material for comparing the efficiency of our design against previous work. Since these results were acquired using different configurations of hardware and software, it is impossible to offer an accurate comparison of the protocols from these results alone. Moreover, the runtimes were taken from optimised implementations that were not released as open-source configurations, meaning that repeating the results of [260] would be difficult.

It should be noted that protocols built on symmetric primitives such as [214, 258, 260, 267] are much more efficient than our design, but arguably also achieve different goals (i.e. much less functionality). Therefore, we do not include them in the tables.

Discussion. Clearly, there is a large gap in computational efficiency between our protocols and the previous work in Tables III;6 and III;7. On the other hand, the communication cost of our protocol is comparable with the other designs, improving on circuit-based approach of Huang et al. [191]. We reinforce that the protocols that we are comparing with only compute the PSI functionality, while our design computes much more expressive functionality (PSU, PSI and PSU/I-CA).

Additionally, observe that the majority of our running times are spent on encrypting the initial Bloom filter that is sent to $\mathbb{P}_2$. In fact, the homomorphic operations and output computation typically take $< 10\%$ of all operating runtime for all set sizes. Subsequently, we can see that that the actual online phase of our protocol could be regarded as practical. As a consequence, the main bottleneck of our design appears to be the encryption phase and thus any optimisation in the underlying encryption scheme would drastically improve the practicality of our construction.

It should finally be noted that the runtimes grow approximately linearly with the increase in the set size. In fact, the growth is typically directly proportional to this growth. This provides further evidence of the linear asymptotic relationship between the efficiency of our protocols and the size of the sets that we consider.

III;8.3 Amortising Bloom filter encryption

Importantly, we can think of the Bloom filter encryption phase as an offline cost. By encrypting with an additively homomorphic scheme, we are able to retain functionality of the Bloom filter even after encryption has taken place. Notice that the encrypting party is only required to store new elements, and recall that it is impossible to remove elements — even from a standard Bloom filter. After a Bloom filter has been encrypted, elements can still be added to the set by adding encryptions of '1' to any specified ciphertext that currently encrypts '0'. Alternatively, the holder of the Bloom filter can simply encrypt k new encryptions of 1 and replace $\mathsf{EIBF}.\overline{\mathsf{arr}}[y_l]$ for $\{y_l\}_{l\in[k]} \leftarrow \widetilde{h}(x)$ for each $x \in \mathbb{S}$.

Now, suppose that EIBF has been used in a protocol instantiation with $\mathbb{P}_2$, and that $\mathbb{P}_1$ wishes to use EIBF again in another instantiation with a new participant $\mathbb{P}_2'$. Then, as long as $\mathbb{P}_2$ does not collude with $\mathbb{P}_2'$, $\mathbb{P}_1$ can update EIBF to include any new set items and re-use this Bloom filter without giving away information about the underlying set. If we assume that $\mathbb{P}_2$ and $\mathbb{P}_2'$ can collude, then $\mathbb{P}_1$ needs to rerandomise the ciphertext entries. Fortunately, $\mathbb{P}_1$ can also pre-compute the necessary items needed for rerandomisation (essentially corresponding to approximately L encryptions of 0). Notice that $\mathbb{P}_1$ can just use the $\mathsf{she.CRerand}$ algorithm to rerandomise any entry that does not change between protocol instantiations. Consequently, this means that EIBF is computationally indistinguishable from a newly constructed encrypted Bloom filter.

With this in mind, precomputing the initial encryption, and necessary encryptions of 0 for rerandomisation, can be thought of as a one-time cost. The 'online' phase of our protocol is very efficient to run and so it is an advantageous feature of our design that the main cost can be amortised across several instantiations. Concretely, this would see an approximate 90% decrease in subsequent runtimes. Extrapolation suggests that our protocol would come close to being as computationally efficient as the design of [123]. Coupled with the comparable communication costs, our protocols can be seen as providing a viable trade-off that allows much more expressive PSO functionality. Our designs demonstrate online phases that are comparably efficient with well-known dedicated PSI protocols, with the caveat of a necessary and expensive offline phase. We re-emphasise the previous OPE-based techniques are not eligible for this kind of amortisation, since the addition of one new root would require changing all of the encrypted coefficients in the polynomial whp.

III;9 Achieving malicious security

The final contribution of this chapter is dedicated to showing that we can alter the security model that our protocols are situated within, and realise a variant of our protocols where security is proven in the event of a malicious adversary corrupting $\mathbb{P}_1$. We still require that $\mathbb{P}_2$ is only corrupted by an adversary within the semi-honest model.[17] Specifically, the proof is based on previous works such as [73, 110, 122, 125, 204], where the set of $\mathbb{P}_1$ is first authenticated by a separate certification authority, $\mathbb{C}$. This security model is referred to as the APSI setting (thus APSO for general private set operations).

It is likely that we could achieve malicious security using complex instantiations of zero-knowledge proofs in the standard PSO security model. For instance, we could leverage ideas from [122, 146, 181, 211] since they compute very similar functionality using similar methods. However, the aim of this work is to demonstrate a simple, efficient toolkit of PSO protocols. Constructing complicated zero-knowledge mechanisms does not fit with this approach.

APSO security model. Similarly to [125, 204], $\mathbb{P}_1$ submits $\mathbb{S}_1$ and BF to $\mathbb{C}$; which returns a signed, encrypted Bloom filter representing $\mathbb{S}_1$. The idea behind this proof strategy is that the simulator Sim plays the role of both $\mathbb{C}$ and the simulator for malicious security (Definition III;1.8), when $\mathbb{P}_1$ is corrupted. Informally, since Sim receives $\mathbb{S}_1$ and signs it, this means that Sim essentially has access to the same information as in the semi-honest security model. Therefore, using exactly the same proof strategy as before, it is possible to simulate the interaction that $\mathbb{P}_1$ has in the real world execution. Notice that if $\mathbb{P}_1$ could use an inconsistent input set that it submits to $\mathbb{C}$, then this would constitute a valid forgery on a signature scheme. Using a signature scheme that satisfies EUF-CMA security ensures that the probability of such an event occurring is negligible.

In this model we ultimately prove the malicious security of our scheme, without the major protocol changes that would be necessary in the standard model for instantiation ZK proof techniques. However, it is valid to question the model itself since it enforces that the set of $\mathbb{P}_1$ is revealed to a third party. We believe that while APSO generally may have little utility, it is useful to elucidate exactly the circumstance in which our simple protocol design provides malicious security. The implied outcome is that if $\mathbb{P}_1$ is forced to commit to an honestly-generated input set, then our protocol can be adapted to demonstrate malicious security guarantees.

Concrete adaptation. Steps one and two of our protocols are required to change, to enable the action of $\mathbb{C}$ to be included in our protocols. We provide details of the concrete changes to step one in Figure III;9; note that this step is the same for all three protocols so it is enough to only provide the adaptations in the case of $\psi_\cup$. For step two, $\mathbb{P}_2$ now receives $(\mathsf{EIBF}.\mathsf{pp}, \omega)$, where ω

[17] Achieving security for malicious $\mathbb{P}_2$ is less important as $\mu_2 = \emptyset$.

```
$\psi_{\cup}^{1}(\mathbb{P}_1, \mathsf{he})$
1 : $\mathsf{BF} \leftarrow \mathsf{BF.init}(1^L, 1^n, 1^k)$;
2 : $\mathsf{BF.Store}(\mathbb{S}_1, \mathsf{BF}.\widetilde{h})$;
3 : $(\mathbb{S}_1, \mathsf{invert}(\mathsf{BF})) \rightarrow \mathbb{C}(\mathsf{he}, \mathsf{dss})$;
4 : $(\mathsf{EIBF}, \omega) \leftarrow \mathbb{C}(\mathsf{he}, \mathsf{dss}, \mathsf{IBF}, \mathbb{S}_1)$;
5 : $(\mathsf{EIBF.pp}, \omega) \rightarrow \mathbb{P}_2$;
```

```
$\mathbb{C}(\mathsf{he}, \mathsf{dss}, \mathsf{IBF}, \mathbb{S})$
1 : for $x \in |\mathbb{S}|$ :
2 :     if $\neg(\mathsf{IBF.Query}(x) \stackrel{?}{=} 0)$;
3 :         return $\perp$;
4 : $\mathsf{EIBF} \leftarrow \mathsf{encrypt}(\mathsf{pk}, \mathsf{IBF})$;
5 : $\omega \leftarrow \mathsf{dss.Sign}(\mathsf{sk}_{\mathsf{dss}}, \mathsf{EIBF}.\overline{\mathsf{arr}})$;
6 : return $(\mathsf{EIBF}, \omega)$;
```

Figure III;9: Left: Step one, computed by $\mathbb{P}_1$ in our APSU protocol. The steps are the same for $\mathsf{he} \in \{\mathsf{she}, \mathsf{ahe}\}$. Right: Action of the certifier $\mathbb{C}$, if $\perp \leftarrow \mathbb{C}$ then the protocol aborts.

is a signature for $\mathsf{EIBF}.\overline{\mathsf{arr}}$. Step two changes to incorporate the verification of the signature pair (Equation (III;12)) in the first line:

$$b \leftarrow \mathsf{dss.Verify}(\mathsf{vk}_{\mathsf{dss}}, (\mathsf{EIBF}.\overline{\mathsf{arr}}, \omega)) \qquad \text{(III;12)}$$

before proceeding with the original definitions for each of the individual protocols. Since this is the only change, we neglect to write out new versions of step two in its entirety.

We provide a new proof of security in Theorem III;9.1 for the PSU case, satisfying the case where $\mathbb{P}_1^{\mathcal{A}}$ acts maliciously. Proofs for the other protocols follow identically using this argument and the rest of the simulation from the original proofs in the semi-honest security model. In Figure III;9, we assume that $\mathbb{C}$ has key pairs

$$(\mathsf{pk}_{\mathsf{he}}, \mathsf{sk}_{\mathsf{he}}) \leftarrow\!\!\$\ \mathsf{he.KeyGen}(1^\lambda),$$
$$(\mathsf{vk}_{\mathsf{dss}}, \mathsf{sk}_{\mathsf{dss}}) \leftarrow\!\!\$\ \mathsf{dss.KeyGen}(1^\lambda),$$

where dss is a digital signature scheme secure against existential forgeries under adaptively-chosen queries (Definition II;5.9). We require that the message space $\mathcal{X}_{\mathsf{dss}}$ for dss is the ciphertext space $\mathcal{Y}_{\mathsf{he}}$ for he. Moreover, we modify the auxiliary information in the protocol to be

$$\mathsf{aux}_j = (|\mathbb{S}_{\bar{j}}|, \mathsf{pk}_{\mathsf{he}}, \mathsf{vk}_{\mathsf{dss}}),$$

for $j \in \{1, 2\}$; noting that $\mathbb{P}_1$ no longer has access to $\mathsf{sk}_{\mathsf{he}}$.

We restate the construction of the PSU protocol here, for completeness. The only changes are that step one is defined as in Figure III;9, and step two requires the signature verification check from Equation (III;12) to be made in line 1.

Construction III;9.1 [Malicious server]

Let $\psi_\cup$ be a protocol for the functionality $F_\cup$ (Equation (III;7)) and let $\mathsf{he}, \mathsf{BF}, \mathbb{P}_j, \mathbb{S}_j, \mathcal{S}, \mathsf{aux}_j, \mathsf{msgs}_j, \mathsf{View}_j$ be described as above.
We construct $\psi_\cup$ as a composition of the steps:

1. $(\mathbb{P}_2 : \mathsf{EIBF.pp}) \leftarrow \psi^1_\cup(\mathbb{P}_1, \mathsf{he})$ [Figure III;9];
2. $(\mathbb{P}_1 : W) \leftarrow \psi^2_\cup(\mathbb{P}_2, \mathsf{EIBF.pp}, \mathsf{he})$ [Figure III;3] (with the modification made in Equation (III;12));
3. $(\mathbb{P}_1 : \mathbb{S}_\cup) \leftarrow \psi^3_\cup(\mathbb{P}_1, W, \mathsf{he})$ [Figure III;4].

We set $\mu_1 = \mathbb{S}_\cup$ and $\mu_2 = \emptyset$.

The correctness of Construction III;9.1 is exactly the same as in Theorem III;4.1, so we do not restate it. We give the new proof of security for the case when $\mathbb{P}_1^{\mathcal{A}}$ is corrupted by a PPT algorithm, $\mathcal{A}$, in the malicious security model (Definition III;1.8).

Theorem III;9.1 [Security]

Let $\psi_\cup$ be defined as in Construction III;9.1. Then $\psi_\cup$ securely computes the functionality $F_\cup$, under the IND-CPA security of he and the EUF-CMA security of dss, in the presence of PPT adversaries maliciously corrupting $\mathbb{P}_1$, and semi-honestly corrupting $\mathbb{P}_2$.

Proof. The proof of the case when $\mathbb{P}_2$ is corrupted is the same as in the proof of Theorem III;4.2. We define the simulator, Sim_1, for the case when $\mathbb{P}_1$ is corrupted.

Simulation III;9.1 [$\mathsf{Sim}_1(\mathsf{aux}_1)$]

- Sim_1 plays the role of $\mathbb{C}$ and receives $(\mathsf{IBF}, \mathbb{S}_1)$ from $\mathbb{P}_1$, and returns (EIBF, ω) to $\mathbb{P}_1$ (or aborts).
- Sim_1 simulates the rest of the protocol using oracle access to the ideal functionality $F_\cup(\cdot, \mathbb{S}_2)$, the set $\mathbb{S}_1$, and computes the simulation of $\mathbb{P}_2$ in exactly the same way as Simulation III;4.1.

Lemma III;9.1 [$\mathsf{View}_1 \approx_c \mathsf{View}_{\mathsf{Sim}}$]

$\mathsf{View}_{\mathsf{Sim}}$ in Simulation III;9.1 is computationally indistinguishable from View_1 in $\psi_{\cup}$, under the existential unforgeability of dss.

Proof. The security proof can be split into two cases, the first case involves $\mathcal{A}$ forging a new signature for a pair $(\mathsf{EIBF}'.\overline{\mathsf{arr}}, \omega')$ and the second case is where $\mathcal{A}$ plays the rest of the game using the original pair $(\mathsf{EIBF}.\overline{\mathsf{arr}}, \omega)$. The probability that $\mathcal{A}$ succeeds in the first case can be bounded by an adversary $\mathcal{B}$ that attempts to break the existential unforgeability of dss. For example, if $\mathcal{A}$ forges a signature with probability δ, then $\mathcal{B}$ acts as Sim_1 and submits the new pair $(\mathsf{EIBF}'.\overline{\mathsf{arr}}, \omega')$ to the $\mathsf{exp}^{\mathsf{euf}}_{\mathcal{B}}(1^{\lambda})$ challenger and wins with probability δ. As a result, we have that $\delta < \mathsf{negl}(\lambda)$ if dss computationally satisfies EUF-CMA security.

In the second case, notice that Sim_1 essentially has the same information that Simulation III;4.1 has (i.e. the set $\mathbb{S}_1$, access to the output $(\mathbb{S}_1 \cup \mathbb{S}_2)$ via the ideal functionality, and the set cardinalities). As such, it can simulate the rest of the game in exactly the same way as before. Consequently, the proof of Lemma III;9.1 is complete. □

The proof of Theorem III;9.1 follows from the proof of Lemma III;9.1. □

While we do not state explicit proofs for the malicious security variants of our PSI and PSU/I-CA protocols, they follow under exactly the same assumptions as above. Therefore, our toolkit can be proven secure in the APSO model, provided that dss satisfies computational EUF-CMA security.

Full malicious security. Achieving security against a malicious $\mathbb{P}_2$ is much more difficult because the adversary can always refuse to compute the correct functionality, and $\mathbb{P}_1$ cannot detect this. However, it is also less meaningful — since the semantic security of the encryption scheme means that $\mathbb{P}_2$ can still not learn anything about $\mathbb{S}_1$, so confidentiality is preserved.

III;10 Conclusion

In this chapter, we gave new two-round protocols for computing private set union, private set intersection and private set cardinality operations between two participants, and in the presence of semi-honest adversaries. Our techniques are similar in composition to the OPE-based techniques of [143, 146, 181, 211] but instead use an underlying encrypted Bloom filter, rather than an encrypted polynomial. We require the usage of an additively homomorphic encryption scheme, or a somewhat homomorphic encryption scheme, that satisfies semantic security. We can improve the security guarantees in the setting where the set of $\mathbb{P}_1$ is authenticated by an external certifier.

In this case, we show that security against malicious corruptions of $\mathbb{P}_1$ under the same primitives and using a digital signature scheme that satisfies EUF-CMA security.

In the case of the somewhat homomorphic encryption scheme, the complexities are linear in the size of the sets, as such our protocol for private set union is the first of its kind to provide such asymptotic efficiency. As a corollary, we also give the first toolkit for computing PSU, PSI and PSU/I-CA with linear computational and communication complexities wrt the input set sizes. When using an additive homomorphic encryption scheme, the complexities are slightly larger since there is an additional number of operations that is logarithmic in the size of the universe that the sets are sampled from. The usage of the Bloom filter is the reason that we can make the improvements, since a Bloom filter query requires a number of operations, k, that is independent of the set size. In the OPE case, a query requires a linear number of operations in the set size.

We also provide a working implementation of our scheme written in `Go` [285]. The results of our implementation show that our protocol is comparable to recent efforts to construct PSO toolkits for multiple operations from public-key based techniques. In fact, most of the cost is in establishing the encrypted Bloom filter, and we show that this can be amortised over multiple protocol runs, helping to lower the running times by approximately 90%.

III;11 Future work

In this section, we detail possible avenues for improving on the results in this chapter.

Linear PSU from symmetric primitives. We noted that PSU seems hard to achieve from symmetric primitives since the elements that are transferred are unknown to the receiver. Recent fast protocols for PSI and cardinality variants make use of the fact that the elements that are transferred are known to the receiver. However, we have no actual evidence that such constructions are impossible. Consequently, any work that was able to achieve linear computational complexities while making use of inherently symmetric cryptographic techniques would be highly valuable. The closest work to achieving this was [47], though that construction requires super-linear numbers of operations and is thus not very scalable. Expanding on this idea, any unified symmetric technique for computing PSU may result in a fast toolkit for all of the main PSOs under the same primitive.

Expansion of malicious security guarantee. Whilst we prove security against a malicious corruption of $\mathbb{P}_1$ in the authenticated setting, it would be beneficial to consider expansion of the guarantees that we can provide. Firstly, establishing security against a malicious corruption of $\mathbb{P}_2$ in the same model would help to unify the guarantee. Secondly, proving malicious security in the standard model, where sets are not certified would be a welcome addition. When exploring this avenue, a key consideration should be that the asymptotic complexities need to be kept at least linear to ensure that scalability is possible. Also, using techniques that do not require zero-knowledge proofs would be useful step in preserving the simplicity of our designs.

Threshold designs for PSOs. The works of [146, 191, 211] give a number of adaptations to their protocols that allow thresholds to be embedded into the computation. These thresholds are not over values, as we provide in Chapter IV, but instead implement lower and upper thresholds that apply to multisets (e.g. intersection with a lower threshold implies that an element is learnt if it appears more times than the lower threshold). Since [146, 211] use OPE-based techniques that are similar to our design, it may be fairly simple to adapt their designs to work over the encrypted Bloom filter that we use.

Cardinality-hiding protocols. The work of [80] shows that it is possible to construct PSI protocols that hide the cardinality of the sets that are used. Moreover, the protocol of [71] gives a design of PSU that achieves the same goal. An adaptation to our protocol that provided the same guarantee would enable us to remove the cardinality information that is provided to both players in the form of the auxiliary information that they control. This would be advantageous and would help to provide the simplest possible proof of security.

Multi-party protocols. It would also be interesting to adapt our protocols to the multi-party setting. This would provide a more generic toolkit and would allow collaborative information sharing between more than two participants. The works of Frikken [146] and Kissner and Song [211] give realisations of multi-party protocols in the OPE setting (though they are fairly efficient). Therefore, it seems plausible that a multi-party design can be adapted from our current constructions, using similar techniques.

IV Applications of PSU for information sharing

necessities

Preface

This chapter is based on elements of the publication [119] [WISCS@CCS2016]. Our contribution represents an adaptation of the PSU protocol that we constructed in Chapter III to a novel use-case.

We only focus on the parts of the original paper [119] that were explicitly contributed by the author of this thesis. As such, we only briefly describe the elements of the paper (relating to the game-theoretic analysis that is presented there) that were mainly the contributions of the other authors. We provide a rigorous treatment of the security proofs that are required for the construction taken from this work. These were not present previously due to space constraints. We also make some small modifications to the protocol and security model to aid in making this treatment more precise.

Overview of original contributions

- A formalisation of an even-handed mediator for fair vulnerability sharing between two individuals.
- Adaptation of mediator to the scenario where vulnerabilities are assigned explicit values and information is shared based on these values.
- A novel protocol that instantiates the mediator functionality without the need for trusted third parties. Our protocol is constructed from the PSU design of Section III;4. The adapted protocol is secure in the semi-honest security model.

Contents

Having given efficient scalable protocols for constructing a toolkit for private set operations, we turn to potential applications of such protocols. In this section, we show that the simplicity of our protocols from the previous sections affords the ability to construct much more specific and technical functionality.

While generic data mining procedures that utilise the individual protocols are obviously intended targets of PSO protocols, we consider the specific case of a particular type of procedure known as *information sharing*. Information sharing is a collaborative exercise that is used within competitive industries, or during state-level diplomacy, to enhance the knowledge of the entities taking part. The sharing refers to the action of entities revealing private data to other entities, in exchange for learning data that they do not already know.

IV;1 Motivation

In recent decades, with the vast increase of data that is stored and hidden, information sharing has taken on a new importance in the 'cyber security' space. The need for all data-holding entities to securely maintain and store such data requires accurate knowledge of the threat model that each of the entities face. In most industries, the threat models for industry competitors can be very similar, and likewise for the methods of storage.

Rather than spending resources on establishing the required knowledge of the threat model themselves, it may be cheaper to exchange information with competitors; subsequently helping each other to collaboratively describe the plausible threats. However, there are a number of parameters that have to be considered before it can be stated whether such a trade can be beneficial for the participants. For instance, if the resultant gain made via collaboration cannot be quantified effectively, then disparate entities are unlikely to engage in a trade.

It is widely believed that sharing such information between entities (or nation states) in this way is a mutually beneficial operation; i.e. providing shared utility. As a result, there have been a number of joint initiatives (between industry and governments) that have attempted to advance the process of data sharing and reduce the number and impact of exploits on states and businesses.

For example, in the United Kingdom, the Cyber-security Information Sharing Partnership [111], launched in 2016, is:

> "... *a joint industry and government initiative set up to exchange cyber threat information in real time, in a secure, confidential and dynamic environment, increasing situational awareness and reducing the impact on UK business*".

Additionally, in the United States, the Cybersecurity Information Sharing Act [270] was signed in December 2015 with the goal of improving

"... cyber-security in the United States through enhanced sharing of information about cybersecurity threats, and for other purposes".

Such partnerships have grown alongside the development of platforms and standards facilitating the transferral of information in numerous ways.

Vulnerability sharing. In this section we will be focusing on the particular case where entities share knowledge of vulnerabilities in commonly used software platforms. Such situations arise often in industries where numerous competitors use similar software backbones to store and manipulate data that is relevant to their operation. Therefore, any vulnerabilities that occur are likely to affect industries as a whole rather than just any one entity. This is similar at the state-level, with systems that are used for providing key parts of infrastructure. A good example is the usage of programming logic controllers via commonly used operating systems and platforms — for providing key parts of national infrastructure, on a global scale. These can be easy targets for 'blind' computer worms and viruses that simply search for known configurations, before releasing their payload (as was in the case of the `Stuxnet` worm [250]).

Sharing vulnerability information in these cases, could potentially lead to the prevention of such attacks in the future. A common incentive for doing this is to prevent growing levels of distrust in an entire industry as a whole due to an exposed vulnerability, even if an attack only affected one particular entity. For example, recent research suggests that exploited vulnerabilities can lead to customer distrust in consumers and investors alike [79, 163]. This ultimately results in large losses of market value and reputation for the firms and institutions in question.

On the other hand, there are some problems that make vulnerability sharing especially difficult. One problem is that sharing sensitive vulnerability data in systems in a non-secure manner, could easily lead to threat actors exploiting such vulnerabilities before all necessary systems have been updated. Collaborative platforms such as `ThreatExchange` [286] (developed by Facebook) and `ThreatStream` [287], aim to provide secure online platforms for allowing entities to post knowledge of such vulnerabilities, without revealing such data publicly.

Another problem is that there is no real method of structuring and canonicalising vulnerability data. Therefore, the data that one entity receives may not be written in a format that it can parse. To this end, the OASIS standards `STIX` and `TAXII` [249], provide frameworks within which vulnerability information can be exchanged in standardised manners, so that it can be understood by all of the exchanging participants.

Unfortunately, while these systems may fulfil the tasks they set out to achieve, they do not provide cryptographic guarantees to the participants. This means that the systems that are used for sharing are vulnerable to the possibility of data extraction. Additionally, the standards only provide the frameworks for describing the sharing of data. They do not give secure implementations of such sharing mechanisms.

IV;2 Achieving cryptographic hardness via PSOs

In the interests of developing cryptographic mechanisms for allowing information sharing, notice that the sharing of data is similar to the case of a private set union protocol. That is, we have two entities with hidden data sets — containing vulnerability information of a mutual interest.[1] The only difference is that information sharing requires a 'fair trade', in the sense that both participants receive *new* data. This establishes the shared utility of the interaction. In PSU, if one participant controls a set that is the entirety of the other set, then the union operation will be empty. Such an eventuality is only learnt after the protocol is carried out. To prevent this occurring, one possibility is for the parties to impose a pre-agreed upper limit on the value of the trade; this prevents one participant from trading a high-value set of information, in return for a set of data that is relatively low-valued.

With this in mind, in this section we show how the PSU protocol from Section III;4 (Construction III;4.1) can be extended to allow exchange of items $(x, \nu) \in \mathcal{S} \times \mathbb{N}$, where ν is the *value* of x. Specifically, we set $\mathbb{V} \in \mathbb{N}^{|\mathbb{S}|}$ and then provide an extension to the protocol allowing implementation of a threshold over the values in $\mathbb{V}$, corresponding to the set $\mathbb{S} \subseteq \mathcal{S}$. The implication is that $\mathbb{P}_1$ learns the union of the two sets, up to the maximum limit implied by the threshold on the values $\nu \in \mathbb{V}$. Such a protocol can be used for constructing methods for information sharing where the 'information' (elements in $\mathbb{S} \subseteq \mathcal{S}$) is separated from the value (elements in $\mathbb{V}$). It should be noted that protocols for computing private set intersection over set elements with associated data have already been considered by De Cristofaro et al. [123]. However, the associated data has a much more generic structure in [123], and additionally union-type operations are not considered.

The construction that we use inherits the scalable techniques that are allowed by the underlying encrypted Bloom filter. However, the design is significantly more complex and requires a linear number of communication rounds in the size of the set of $\mathbb{P}_2$. The original PSU protocol only required two rounds of communication. Moreover, we make generic use of protocols for performing (1-out-of-n) oblivious transfers, this concept is defined formally in Section IV;5. The actual design of the protocol is instantiable using the OPE-based PSU protocols as well [146, 181, 211], since the mechanics of the value assertion is suitably external to the PSO design. However, we only describe the adaptation in the case of the PSU construction given in Section III;4.

[1]We make the assumption that vulnerability information can be canonicalised.

In particular, this section is based on parts of the work published in [119]. The work of [119], as a whole, gives a game-theoretic model for the sharing of information when each set element has a unique value ν assigned. It is shown there, that if a *trusted mediator* exists that can guarantee that $\mathbb{P}_1$ and $\mathbb{P}_2$ can swap equal values of information, then both players will share as much as possible. This operation is clearly similar to a union protocol, but where the amount of information is limited by the necessary value restriction. The protocol that we develop allows us to replace the mediator with a PSO protocol. We provide a more detailed background in Section IV;3.

This chapter will focus entirely on the construction of the bespoke protocol; thus reflecting the contribution of the thesis author in the original work of [119]. In the rest of this section, we will commonly refer to the collaboratively-chosen threshold as τ. We assume that τ is computed prior to the enacting of the protocol.

IV;3 Incentives for vulnerability sharing

The game-theoretic analysis of [119] essentially establishes the parameters that describe the dynamics of the trading. While the tools and initiatives that we described previously can provide useful and efficient *platforms* for exchange of cyber security information, the role of *incentives* must not be ignored. If they are not described sufficiently then the natural course of action is to not share information, since it is not clear how revealing sensitive data can benefit an entity. For instance, sharing can often lead to leaked disclosure of security breach incidents that can harm consumer and investor confidence, and lead to significant decreases in the market value of firms [79, 163].

The description and analysis of these vulnerability sharing scenarios was first given a game-theoretic analysis by Khouzani et al. [208]. In summary, their work establishes a two-participant model analysing the situations under which these two entities will share vulnerability information. Their analysis uses three parameters e_1, e_2, e_3, where:

- e_1 is the loss of utility for $\mathbb{P}_j$ if a vulnerability isn't known and is then exploited;
- e_2 is the competitive gain that $\mathbb{P}_j$ receives from a vulnerability exploited in the system of $\mathbb{P}_{\bar{j}}$;
- e_3 is the loss of utility for both players, modelling the negative effect of a vulnerability exploitation on the market as a whole.

Several other assumptions are necessary, including that vulnerabilities all carry the same value. In this model, they show that the existence of a mediator that can guarantee fair trades of information — e.g. that $\mathbb{P}_1$ and $\mathbb{P}_2$ both receive τ new elements from the trade — incentivises both participants to share as much information as is possible. If no such mediator exists, then the parameters suggest

```
$\mathcal{M}(\mathbb{S}_1, \mathbb{S}_2, \tau)$
1 :  for $j \in [2]$ :
2 :      $W \leftarrow \emptyset$;
3 :      while $\tau > 0$ :
4 :          if $\mathbb{S}_j[i] \notin \mathbb{S}_{\bar{j}}$ :
5 :              $W \leftarrow \mathbb{S}_j[i]$;
6 :              $\tau = \tau - 1$;
7 :      $W \leftarrow \mathbb{P}_{\bar{j}}$;
```

Figure IV;1: Informal mediator specification from [208].

that sharing no information is the most profitable scenario. Given the real-world examples of incentives for performing information sharing, it is beneficial to explore the necessary structure to motivate the two participants to share freely. The mediator given by [208], denoted by $\mathcal{M}$, is specified informally in Figure IV;1.

In this trade, it is assumed that τ is computed by $\mathbb{P}_1, \mathbb{P}_2$ in tandem, prior to the mediator being activated. In the economic model of [208], the participants would propose τ_1, τ_2 and choose $\tau = \min(\tau_1, \tau_2)$. Since the existence of a mediator implies that the highest utility is gained by sharing as much as possible, this implies that $\tau = \min(|\mathbb{S}_1| - |\mathbb{S}_1 \cap \mathbb{S}_2|, |\mathbb{S}_2| - |\mathbb{S}_1 \cap \mathbb{S}_2|)$.

Davidson et al. [119] (in the work that this chapter is based on) improve the model to allow vulnerabilities to have differing values assigned from a set $\mathcal{V}$. That is, there is a canonical function $\mathsf{value} : \mathcal{S}^* \mapsto \mathcal{V}$ such that, for any $x \in \mathcal{S}$, then $\nu_x = \mathsf{value}(x)$ is a deterministic output and ν_x is the *value* of x. In this widened model, the dynamics do not change and full sharing is still the optimal strategy, when a mediator exists. This mediator now computes fair trades based on the maximum threshold $\tau \in \mathbb{Z}$, indicating the maximum value of trades that can be tolerated by both of $\mathbb{P}_j$ for $j \in \{1, 2\}$. This value of τ can be computed apriori using a general two-party secure computation protocol for the function $\min(\cdot, \cdot)$, that outputs the minimum of two values (many fast generic constructions exist, e.g. [37, 113, 290]).

Using the values of the elements instead, we update the informal functionality of the mediator in Figure IV;2. Describing the action of the mediator in this way is necessary for helping us to give a cryptographic construction that achieves the same functionality.

Note, that Figure IV;2 represents a necessary relaxation from the ideal mediator laid out above. In other words, $\mathcal{M}$ does not seek to maximise the value of the trade. For the mediator to maximise $\mathsf{value}(\mathbb{S}_{\tau,j})$, it would essentially require solving an instance of the subset sum problem; a variant of the knapsack problem (where the weights are equal to the values of the items). We detail the knapsack problem below.

```
$\mathcal{M}(\mathbb{S}_1, \mathbb{S}_2, \tau)$
1 :  for $j \in [2]$ :
2 :    $W \leftarrow \emptyset$;
3 :    for $i \in [n_j]$ :
4 :      $\tau_i = \tau - \mathsf{value}(\mathbb{S}_j[i])$;
5 :      if $(\mathbb{S}_j[i] \notin \mathbb{S}_{\bar{j}}) \wedge (\tau_i > 0)$ :
6 :        $W[i] \leftarrow \mathbb{S}_j[i]$;
7 :        $\tau = \tau_i$;
8 :    $W \leftarrow \mathbb{P}_{\bar{j}}$;
```

Figure IV;2: Informal (greedy) mediator specification for vulnerability sharing with associated values.

(Knapsack problem): *Given a bound $B \in \mathbb{N}$, a set $S = \{a_1, \ldots, a_n\}$ with corresponding weights $\{w(a_1), \ldots, w(a_n)\}$ and values $\{v(a_1), \ldots, v(a_n)\}$. Then the problem is to find a subset $I \subseteq [n]$, s.t. $\sum_{i \in I} w(a_i) \leq B$ and $\sum_{i \in I} v(a_i) = \max_{I'}(\sum_{i \in I'} v(a_i))$ for all other subsets $I' \subseteq [n]$.*

While polynomial-time approximations of the knapsack problem exist (e.g. see [218]), it is preferable to consider the above relaxation to stay as general as possible.

Underlying trade assumptions. While the canonical value assignment function, value, seems far-fetched, it should be noted that there have been attempts to practically construct systems for performing this exact requirement. Firstly, the OASIS standards [249] provide methods for describing vulnerability data in strict frameworks. Secondly, platforms such as the NIST national vulnerability database [248], provide open-access to huge databases of known vulnerabilities — along with metrics that assert the severity of any given vulnerability. Vulnerabilities are assigned a score from (0-5) assessing the severity of an exploit using it. Such efforts can be construed as equivalent to the function value that we consider.

Definition IV;3.1 [Value assignment]

Let $\mathcal{S}$ be a universe. Then, there is a deterministic function, $\mathsf{value} : \mathcal{S} \mapsto \mathbb{N}$, s.t. $\mathsf{value}(x) = \nu_x \in \mathbb{N}$ for all $x \in \mathcal{S}$. For any $\mathbb{S} \in \mathcal{S}$, let $n = |\mathbb{S}|$. Then, we write $\mathbb{V}$ to be the set, s.t. $\mathbb{V}[i] = \mathsf{value}(\mathbb{S}[i])$, $\forall$ elements in $\mathbb{S}$. We say that $\mathbb{V}$ is the *associated value set* for $\mathbb{S}$. We abuse notation and let $\mathsf{value}(\mathbb{S}) = \sum_{i=1}^{n} \mathsf{value}(\mathbb{S}[i])$ be the value of the entire set.

Another assumption that we make is that vulnerabilities can be deterministically (and efficiently) assigned a tag in $\mathcal{S}$. This is necessary for describing the exchange in the language of PSO protocols.

IV;4 Related work

Before proceeding any further, we give a brief summary of previous work that is related to the themes discussed in this section.

PSOs for information sharing. The protocol that we develop in this section is highly specialised to the task of vulnerability sharing in the scenarios considered by [119, 208]. However, Freudiger et al. [145] consider similar settings where data sharing is necessary for collaboratively improving the knowledge of participants wrt implementing blacklists. As we mentioned previously, the design of private set operations protocols that implement thresholds (without the associated data) has been explored thoroughly [47, 146, 191, 211]. These thresholds are implemented over the set elements themselves, rather than any associated data.

Information sharing economics. The need for studying the economics of sharing information derives from the conflicting incentives to maintain an advantage over rivals, while supporting allies. Typically information has a high cost to generate, for example in R&D, but is cheap to distribute, such as over a secure internet connection. Sometimes information is given away for free, such as in a health awareness campaign. But usually, information is traded at a price, paid for with cash or other information in return. Traditional forms of information of economic interest include market data, newspaper and magazine access, or consumer data for advertising purposes. For a guide on information economics, see [276]. For a more a theoretical survey on information sharing games, see [280].

Information sharing in the context of cybersecurity is investigated in [170, 179, 228, 295]. For example, Gordon *et al.* present a model for evaluating the effects of US policy on computer systems security, showing that when economic or legal mechanisms incentivise information sharing between private firms, firms invest less in security but the overall network is made more secure [170]. Further research into the effects of US policy, specifically the Sarbanes-Oxley Act, on security information sharing can be found in [179]. Phillips, Ting and Demurjian take a geopolitical approach to the information economics of extreme security situations, such as international disasters or military action [257].

More recently, Laube and Böhme [221] investigate how legally mandatory sharing of security breaches affects the balance of private security investment from detective to preventative measures. They show that when law enforcement for reporting of security breaches is too strong, firms may over-invest in detection and under-invest in prevention. This socially sub-optimal situation may also arise when information sharing is too effective.

Security investment games. Cyber security, such as network, application, web or hardware security, is a private good. This means that a user derives benefits from using secure products, and conversely firms that offer more secure products can command a higher price. To improve a product or tool, firms invest in security goods such as penetration testing, formal analysis, network redundancy and anti-virus software. Security games examine various economic scenarios relevant to cybersecurity, with each game analysing a different behavioural dynamic [174].

For instance, when devices or applications are connected with each other, the security of the whole network depends on the security of each of the individual objects. This means that security is also a public good, as improved security of one firm's products and systems can indirectly improve the security of other firms. The trade-off between investing in private security, and 'free-riding' off the security of others gives rise to interdependent security investment games [199]. For example, Kunreuther and Heal examine how the network effects on security can deincentivise security investment, as players (nodes) can decide to accept the risk from not investing on the hope of their neighbours being secure enough to protect them for free [217]. For a detailed survey of classic research into interdependent security investment games, see [220]. For a discussion on network games in general, see [193].

IV;5 Oblivious transfer

Let $\mathbb{P}_1$ hold k indices, and let $\mathbb{P}_2$ hold a set $\mathbb{S}$ with cardinality n. Then a (k-out-of-n) oblivious transfer protocol allows $\mathbb{P}_1$ to learn k elements from $\mathbb{S}$ without: (1) revealing the indices to $\mathbb{P}_2$; (2) $\mathbb{P}_1$ learning any of the other $(n-k)$ elements left in $\mathbb{S}$. We will denote such protocols by OT^k_n.

Oblivious transfer protocols were introduced by the works of [135, 263] and have been made very simple and efficient in works such as [19, 93, 192, 213, 243]. In such protocols, $\mathbb{P}_2$ is referred to as the 'sender' and $\mathbb{P}_1$ the 'receiver'.

Definition IV;5.1 [Oblivious transfer]

Let $\mathbb{P}_1$ hold a set $\mathbb{K} \subset [n]$ of k indices, let $\mathbb{P}_2$ hold a set $\mathbb{S} \subseteq \mathcal{S}$ where $|\mathbb{S}| = n$. Then OT^k_n is a (k*-out-of-*n) *oblivious transfer* protocol if:

$$(\mu_1, \mu_2) \leftarrow \mathsf{OT}^k_n(\mathbb{P}_1, \mathbb{P}_2, \mathbb{K}, \mathbb{S});$$

where $\mu_1 = \{\mathbb{S}[i]\}_{i \in \mathbb{K}}$ and $\mu_2 = \emptyset$. We will define the views of the protocol to be $\mathsf{View}_1 = (\mathbb{K}, \mu_1, \mathsf{msgs}^{\mathsf{ot}}_1, \mathsf{aux}_1)$ and $\mathsf{View}_2 = (\mathbb{S}, \mu_2, \mathsf{msgs}^{\mathsf{ot}}_2, \mathsf{aux}_2)$ — where $\mathsf{msgs}^{\mathsf{ot}}_j$ and aux_j abide by the conventions given in Definition II;6.1.

Remark IV;5.1. *We may occasionally remove the explicitly mentioned inputs to the protocol* OT^k_n*, if they are implied by context.*

We will only assume the existence of protocols OT_n^1 that are secure in the presence of semi-honest adversaries. The security guarantee is equivalent to the notion given in Definition II;6.3. Such a protocol was constructed in [93]. Our design is completely agnostic to the OT that we require and so we do not give any specific instantiation. Oblivious transfers usually require a large number of public-key operations, but it was shown by [32] that a large number of oblivious transfers can be created from a small constant number. This technique is known as *OT extension* and there are many such instantiations [19, 192, 203, 213].

Advantage. We will denote the advantage of distinguishing the real and simulated executions of OT_n^k, in the presence of PPT semi-honest adversaries $\mathcal{A}$, by:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{ot}(1^\lambda))) < \mathsf{negl}(\lambda)$$

where $\mathcal{A}$ outputs 0 if it guesses that it is witnessing the real execution and 1 for the simulated execution.

Layout. For the rest of this chapter, we only briefly refer back to the economic model and focus more on the formalisation of the mediator functionality in Figure IV;2. That is, the aim of this section is to formalise the ideal functionality needed in the vulnerability sharing scenario that we examine; and then construct a protocol that can instantiate this functionality securely. Subsequently, we give a two-party adaptation of the PSU protocol in Construction III;4.1; securely instantiating this mediator in the presence of semi-honest adversaries and allowing the participants to engage in full information sharing without the need for additional trust assumptions. This is heavily based on what is written in [119][Section 4 onwards], with some modifications to the structure of the protocol that was described.

IV;6 Formalisation of mediator functionality

Before we can construct a secure protocol, we should first crystallise the ideal 'mediator' functionality that we stated in Figure IV;2. This is the functionality that allows us to establish fair trades that result in optimal information sharing, as laid out in [119, 208]. The value assignment function of Definition IV;3.1 allows $\mathbb{P}_1$ and $\mathbb{P}_2$ to assign consistent values to any element $x \in \mathcal{S}$, in the joint universe $\mathcal{S}$.

Threshold subset-finders. Describing the ideal mediator functionality requires access to an algorithm $\Pi(\mathbb{S}, \tau)$ that locates some subset $\mathbb{S}_I$ of $\mathbb{S}$ where $I \subseteq [n]$, under the condition that $\mathsf{value}(\mathbb{S}_I) \leq \tau$ for $\tau > 0$. The algorithm only takes $\mathbb{S}$ as input, where $\mathbb{S}$ has associated value set $\mathbb{V}$, and is faced with a challenger that verifies the value of the final output set. We define the

```
$\mathsf{exp}^{\mathsf{tsf}}_{\Pi}(1^\lambda, \mathbb{S}, \tau)$
1 :  $W = []$;
2 :  for $i \in Q = \mathsf{poly}(\lambda)$ :
3 :    $x_i \leftarrow \Pi(\mathbb{S}, \tau)$;
4 :    if $x_i \leq \tau$ :
5 :      $\tau = \tau - \mathsf{value}(x_i)$;
6 :    $W \leftarrow x_i$;
7 :  if $(W \subseteq \mathbb{S}) \wedge (\tau \geq 0)$ :
8 :    return $1$;
9 :  else :
10 :    return $0$;
```

Figure IV;3: Threshold subset finder experiment for evaluating whether an algorithm Π can find a subset $\mathbb{S}_I$ of $\mathbb{S}$ such that $\sum_{i \in I} \nu_i \leq \tau$.

interaction in Figure IV;3, where λ is a parameter describing the running time of Π. The algorithm succeeds if it is PPT and returns 1, otherwise it returns 0.

This is not a cryptographic game. In actual fact, we require the existence of PPT algorithms Π that satisfy $\mathsf{exp}^{\mathsf{tsf}}_{\Pi}(1^\lambda)$ with probability 1. We can illustrate a (naive) concrete example of such a Π, running in time $O(n)$ for where n is polynomially-bounded by the number of queries Q. The algorithm simply sets $Q = n$ and simply queries each set element in the order that they are placed in. The output is clearly a subset of $\mathbb{S}$ and is thus viable in $\mathsf{exp}^{\mathsf{tsf}}_{\Pi}(1^\lambda, \mathbb{S}, \tau)$. We will always assume that n is only polynomial in size.

Definition IV;6.1 [Threshold subset-finder]

We say that an algorithm Π, for inputs $(\mathbb{S}, \tau) \in \mathcal{S}^n \times \mathbb{Z}$, successfully finds subsets $\mathbb{S}'$ of $\mathbb{S}$ such that $\mathsf{value}(\mathbb{S}') \leq \tau$, if it succeeds in $\mathsf{exp}^{\mathsf{tsf}}_{Q,\Pi}(1^\lambda, \mathbb{S}, \tau)$ with probability equal to 1. Let $\mathbb{C}$ denote the challenger in $\mathsf{exp}^{\mathsf{tsf}}_{\Pi}(1^\lambda, \mathbb{S}, \tau)$, we can make the challenger explicit in the definition of Π by writing $\Pi^{\mathbb{C}}$.

We further require an additional property that Π is *agnostic* to the universe of set elements that are being considered.

Definition IV;6.2 [Set agnostic]

Let Π satisfy Definition IV;6.1 for sets $\mathbb{S} \in \mathcal{S}$ and $\mathbb{S}' \in \mathcal{S}$. Let $\mathbb{S}, \mathbb{S}'$ be s.t. $|\mathbb{S}| = |\mathbb{S}'| = n$. Let $\widehat{\mathbb{V}}$ be a value set s.t. $\mathsf{value}(\mathbb{S}[i]) = \mathsf{value}(\mathbb{S}'[i]) = \widehat{\mathbb{V}}[m]$ for all $i \in [n]$ (i.e. they share the same value set).
We say that Π is *set agnostic* if: $\mathbb{S}_M \leftarrow \Pi(\mathbb{S}, \tau), \mathbb{S}'_M \leftarrow \Pi(\mathbb{S}', \tau)$ and M is an index set s.t., for all $j \in M$, then $\mathbb{S}[m] \in \mathbb{S}_M$ and $\mathbb{S}'[m] \in \mathbb{S}'_M$. That is, Π effectively picks the same index set, regardless of the input $\mathbb{S}$. If Π is set agnostic, then we say that it is a threshold subset finder for all universes $\mathcal{S}$.

The naive algorithm satisfying Definition IV;6.1, that we described above, is set agnostic.

The ideal functionality that we use in Definition IV;6.3 is defined wrt some choice of the algorithm Π. It is necessary to define the ideal functionality in this way to prevent trivial distinguishing attacks in the security proof of the protocol that we eventually build. This definition summarises the functionality of the mediator that is required in [119, 208] for establishing fair trades.

Definition IV;6.3 [Mediator idealisation]

Let $\mathbb{S}_j \subseteq \mathcal{S}$ be the input set with associated value set $\mathbb{V}_j$, for $j \in \{1, 2\}$. Let $\tau > 0$ be the maximum information sharing value threshold. Let Π be a threshold subset-finder algorithm (Definition IV;6.1).
Then, let

$$F^{\Pi}_{j,\tau} : (\mathcal{S} \times \mathbb{N})^{n_1} \times (\mathcal{S} \times \mathbb{N})^{n_2} \times \mathbb{N} \mapsto \mathcal{S}^*,$$

be a functionality of the form:

$$F^{\Pi}_{j,\tau}((\mathbb{S}_1, \mathbb{V}_1), (\mathbb{S}_2, \mathbb{V}_2), \tau) = \mathbb{S}_j \cup \Pi(\mathbb{S}_{\bar{j}} \setminus (\mathbb{S}_1 \cap \mathbb{S}_2), \tau).$$

Importantly, our definition of the ideal mediator functionality depends entirely on the output of the algorithm Π. We are required to make this explicit, since the participants in our protocol have to run an algorithm of the form of Π in $\mathsf{exp}^{\mathsf{tsf}}_{\Pi}(1^\lambda, \mathbb{S}, \tau)$ to garner their output. If Π changes, so does the output distribution. Thus, for providing correctness, we need to use a common algorithm Π for both the ideal functionality and the protocol.

Remark IV;6.1. *As we mentioned above, in the economic scenario of [119] the mediator chooses the maximum value subset — but such a mediator is unlikely to run in polynomial-time, and so this relaxation is necessary. As a final note on this topic, it is unclear whether the approximation algorithms of [218] would establish a viable algorithm in* $\mathsf{exp}^{\mathsf{tsf}}_{\Pi}(1^\lambda, \mathbb{S}, \tau)$ *since the algorithm* Π *has no access to the values in this setting. The naive* $O(n)$ *algorithm that we detailed above is sufficient for our needs.*

IV;7 Protocol for information sharing with value threshold

We will denote the protocol in this section by ψ_τ^Π taking as input $(\mathbb{S}_j, \mathbb{V}_j) \in (\mathcal{S} \times \mathbb{Z})^{n_j}$ for $n_j = |\mathbb{S}_j|$, for $j \in \{1, 2\}$. The aim of the protocol is to securely instantiate the ideal functionality given in Definition IV;6.3. The key idea behind our protocol is that $\mathbb{P}_2$ randomises the messages from step two of the PSU protocol in Chapter III so that $\mathbb{P}_1$ cannot immediately decrypt. Then the two participants engage in an oblivious transfer protocol that reveals the random masks for elements that are permitted to be learnt by the value threshold τ that is imposed over the set elements that we consider.

Let he be a homomorphic encryption scheme, BF be a Bloom filter and let

$$\mathsf{aux}_j = (n_{\bar{j}}, (\mathsf{pk}_j, \mathsf{sk}_j), \mathsf{pk}_{\bar{j}}, \tau, \mathsf{value});$$

where $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow\!\!\$\ \mathsf{he.KeyGen}(1^\lambda)$, $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow\!\!\$\ \mathsf{he.KeyGen}(1^\lambda)$. We abuse the notation of he and allow encryptions of elements taken from $\mathcal{X} = \mathbb{Z}_q$.[2] As with our previous protocols, only $\mathbb{P}_1$ will learn the explicit outcome of the set operation. That is, let

$$\mathbb{S}_M = \Pi(\mathbb{S}_2 \setminus (\mathbb{S}_1 \cap \mathbb{S}_2), \tau) \subseteq \mathbb{S}_2,$$

and let $\mathbb{V}_M \subseteq \mathbb{V}_2$ be the associated values for the indices $i \in M$. Then,

$$\mu_1 = (\mathbb{S}_1 \cup \mathbb{S}_M, \mathbb{V}_M); \;\; \mu_2 = (\mathbb{V}_M);$$

are the outputs of the protocol and $\mathsf{View}_j = (\mathbb{S}_j, \mu_j, \mathsf{aux}_j, \mathsf{msgs}_j)$ for $j \in \{1, 2\}$. We will only be proving security in the case of semi-honest adversaries, therefore it will be required that we can simulate msgs_j the protocol using only $(\mathbb{S}_j, \mu_j, \mathsf{aux}_j, \mathsf{msgs}_{\bar{j}})$.

The protocol is split into the steps given in Figures IV;4, IV;5, IV;6 and IV;7. It should be noted that step four is unlike the rest in that it is engaged by both participants simultaneously. Firstly, $\mathbb{P}_1$ engages the algorithm $\Pi^{\mathbb{P}_2}$ where $\mathbb{P}_2$ plays the challenger in the experiment $\mathsf{exp}_\Pi^{\mathsf{tsf}}(1^\lambda)$. Secondly, the participants engage in numerous invocations of a 1-out-of-n_2 oblivious transfer protocol ($\mathsf{OT}_{n_2}^1$), with $\mathbb{P}_1$ the receiver and $\mathbb{P}_2$ the sender.

The entire protocol is given in Construction IV;7.1, and we prove correctness in Theorem IV;7.1 and security in the presence of semi-honest adversaries in IV;7.2.

[2]This is not necessarily a stretch from the truth where most ahe, she schemes permit encryptions of elements in $\mathbb{Z}_q$ for some $q > 0$ (thus satisfying our needs).

```
$\psi_\tau^{\Pi,1}(\mathbb{P}_1, \mathsf{he})$
1 :  $\mathsf{BF} \leftarrow \mathsf{BF.init}(1^L, 1^n, 1^k)$;
2 :  $\mathsf{BF.Store}(\mathsf{BF.sp}, \mathbb{S}_1)$;
3 :  $\mathsf{EIBF} \leftarrow \mathsf{encrypt}(\mathsf{pk}, \mathsf{invert}(\mathsf{BF}))$;
4 :  $\mathsf{EIBF.pp} \leftarrow \mathbb{P}_2$;
```

Figure IV;4: Step one of our information sharing protocol. The steps are the same for $\mathsf{he} \in \{\mathsf{she}, \mathsf{ahe}\}$ and are also the same as in step one of our PSU protocol. In other words, $\mathbb{P}_1$ encodes their set into an encrypted, inverted Bloom filter and sends it to $\mathbb{P}_2$

Construction IV;7.1 [Information sharing with value threshold]

Let ψ_τ^Π be a protocol for the functionality F_τ^Π (Definition IV;6.3) and let $\mathsf{he}, \mathsf{BF}, \mathbb{P}_j, \mathbb{S}_j, \mathcal{S}, \mathsf{aux}_j, \mathsf{msgs}_j, \mathsf{View}_j, \Pi, \mathsf{OT}_n^1$ be described as above. Let τ be the maximum value threshold that is calculated prior to the start of the protocol by $\mathbb{P}_1$ and $\mathbb{P}_2$, and added to aux_j for $j \in \{1, 2\}$.
We construct ψ_τ^Π as a composition of the steps

1. $(\mathbb{P}_2 : \mathsf{EIBF.pp}) \leftarrow \psi_\tau^{\Pi,1}(\mathbb{P}_1, \mathsf{he})$ [Figure IV;4],
2. $(\mathbb{P}_1 : W, \mathbb{P}_2 : R) \leftarrow \psi_\tau^{\Pi,2}(\mathbb{P}_2, \mathsf{he})$ [Figure IV;5],
3. $(\mathbb{P}_1 : (E, Z)) \leftarrow \psi_\tau^{\Pi,3}(\mathbb{P}_1, \mathsf{he}, W)$ [Figure IV;6],
4. $(\mathbb{P}_1 : \mathbb{S}_\tau^\Pi, \mathbb{P}_2 : \mathbb{V}_M) \leftarrow \psi_\tau^{\Pi,4}(\mathbb{P}_1, \mathbb{P}_2, \mathsf{he}, \Pi, E, Z, R, \tau)$ [Figure IV;7];

retaining this ordering. We set $\mu_1 = \mathbb{S}_\tau^\Pi$ and $\mu_2 = \mathbb{V}_M$

It may be helpful to further explain the intuition behind step four. In essence, $\mathbb{P}_1$ initiates the algorithm Π with $\mathbb{P}_2$ as the challenger. We make a simple adaptation to the steps in $\mathsf{exp}_\Pi^{\mathsf{tsf}}(1^\lambda, \mathbb{S}, \tau)$ to allow the input set to be encrypted, and thus to allow $\mathbb{P}_2$ to decrypt the encrypted values before applying the threshold checks. This is equivalent to the original experiment under these cosmetic changes. Since there exists a PPT ($O(n_2)$) algorithm that can find a subset in these conditions, then the set E_M is a subset that satisfies the threshold τ.

Secondly, using the elements of E_M, $\mathbb{P}_1$ and $\mathbb{P}_2$ engage in $|E_M|$ instances of the $\mathsf{OT}_{n_2}^1$ protocol, so that $\mathbb{P}_1$ can learn the masking values for these elements. Once these oblivious transfers are concluded, then $\mathbb{P}_1$ can remove the masks and learn the underlying values from $\mathbb{S}_2$. In these steps, it is necessary that $\mathbb{P}_1$ knows the original indices i for the ordering of the set in step three. Providing that $\mathbb{P}_2$ does not shuffle $\mathbb{S}_2$ in between, then $\mathbb{P}_1$ learns the correct element.[3] We assume that E is shuffled before being sent to $\mathbb{P}_2$ (during $\Pi^{\mathbb{P}_2}$) to prevent trivial learning of which elements corre-

[3]This is why it is necessary to add i to Z in step two: as the input index for $\mathbb{P}_1$ during the oblivious transfer step.

```
$\psi_\tau^{\Pi,2}(\mathbb{P}_2, \mathsf{EIBF.pp}, \mathsf{she})$
1 :  $W, R \leftarrow \emptyset$;
2 :  for $i \in [n_2]$ :
3 :      $\{c_i^l\}_{l \in [k]} \leftarrow \mathsf{EIBF.Query}(\mathsf{EIBF.pp}, x_i^{(2)})$;
4 :      $c_i^+ \leftarrow c_i^1$;
5 :      for $l \in [2, k]$ :
6 :          $c_i^+ \leftarrow \mathsf{she.Add}(\mathsf{pk}_1, c_i^l, c_i^+)$;
7 :      $r_{i,1}, r_{i,2} \leftarrow\!\$\ \mathcal{X}$;
8 :      $\widehat{c_i^+} \leftarrow \mathsf{she.Mult}(\mathsf{pk}_1, c_i^+, \mathsf{she.Enc}(\mathsf{pk}_1, r_{i,1}))$;
9 :      $\widehat{d_i^+} = \mathsf{she.Add}(\mathsf{pk}_1, \mathsf{she.Mult}(\mathsf{pk}_1, c_i^+, \mathsf{she.Enc}(\mathsf{pk}_1, x_i^{(2)})), \mathsf{she.Enc}(\mathsf{pk}_1, r_{i,2}))$;
10 :     $(\widetilde{c_i^+}, \widetilde{d_i^+}) \leftarrow (\mathsf{she.CRand}(\mathsf{pk}_1, \widehat{c_i^+}), \mathsf{she.CRand}(\mathsf{pk}_1, \widehat{d_i^+}))$;
11 :     $\widehat{e_i} \leftarrow \mathsf{he.Enc}(\mathsf{pk}_2, \mathsf{value}(x_i^{(2)}))$;
12 :     $(\widetilde{c_i^+}, \widetilde{d_i^+}, \widehat{e_i}) \leftarrow W[i]$;
13 :     $(r_{i,1}, r_{i,2}) \leftarrow R$;
14 : $W \leftarrow \mathbb{P}_1$;
15 : return $R$;
```

```
$\psi_\tau^{\Pi,2}(\mathbb{P}_2, \mathsf{EIBF.pp}, \mathsf{ahe})$
1 :  $W, R \leftarrow \emptyset$;
2 :  for $i \in [n_2]$ :
3 :      $\{c_i^l\}_{l \in [k]} \leftarrow \mathsf{EIBF.Query}(\mathsf{EIBF.pp}, x_i^{(2)})$;
4 :      $c_i^+ \leftarrow c_i^1$;
5 :      for $l \in [2, k]$ :
6 :          $c_i^+ \leftarrow \mathsf{ahe.Add}(\mathsf{pk}_1, c_i^l, c_i^+)$;
7 :      $r_{i,1}, r_{i,2} \leftarrow\!\$\ \mathcal{X}$;
8 :      $\widehat{c_i^+} \leftarrow \mathsf{ahe.ScMult}(\mathsf{pk}_1, c_i^+, r_{i,1})$;
9 :      $\widehat{d_i^+} = \mathsf{ahe.Add}(\mathsf{pk}_1, \mathsf{ahe.ScMult}(\mathsf{pk}_1, c_i^+, \mathsf{ahe.Enc}(\mathsf{pk}_1, x_i^{(2)})), r_{i,2})$;
10 :     $(\widetilde{c_i^+}, \widetilde{d_i^+}) \leftarrow (\mathsf{ahe.CRand}(\mathsf{pk}_1, \widehat{c_i^+}), \mathsf{ahe.CRand}(\mathsf{pk}_1, \widehat{d_i^+}))$;
11 :     $\widehat{e_i} \leftarrow \mathsf{he.Enc}(\mathsf{pk}_2, \mathsf{value}(x_i^{(2)}))$;
12 :     $(\widetilde{c_i^+}, \widetilde{d_i^+}, \widehat{e_i}) \leftarrow W[i]$;
13 :     $(r_{i,1}, r_{i,2}) \leftarrow R$;
14 : $W \leftarrow \mathbb{P}_1$;
15 : return $R$;
```

Figure IV;5: Step two, computed by $\mathbb{P}_2$ in our information sharing protocol. In this step, $\mathbb{P}_2$ computes a randomised pair of encryptions for each element in their set. The plaintexts corresponding to $(\widetilde{c_i^+}, \widetilde{d_i^+})$ are essentially $(r_{i,1} \cdot k_i, x_i^{(2)} \cdot k_i + r_{i,2})$ and thus randomly distributed for all $i \in [n_2]$. The ciphertext $\widehat{e_i}$ contains the value of the element $x_i^{(2)}$ and is encrypted under pk_2. We assume that the sets W, R are shuffled randomly (but identically wrt each other) before W is sent to $\mathbb{P}_1$.

```
$\psi_\tau^{\Pi,3}(\mathbb{P}_1, \mathsf{he}, W)$
1 :  $E, Z \leftarrow \emptyset$;
2 :  for $i \in [n_2]$ :
3 :      $(\widetilde{c_i^+}, \widetilde{d_i^+}, \widehat{e_i}) \leftarrow W[i]$;
4 :      $(z_{i,1}, z_{i,2}) \leftarrow (\mathsf{he.Dec}(\mathsf{sk}_1, \widetilde{c_i^+}), \mathsf{he.Dec}(\mathsf{sk}_1, \widetilde{d_i^+}))$;
5 :      if $z_{i,1} \neq 0$ :
6 :          $Z \leftarrow (z_{i,1}, z_{i,2}, i)$;
7 :          $E \leftarrow \mathsf{he.CRand}(\mathsf{pk}_2, \widehat{e_i})$;
8 :  return $E, Z$;
```

Figure IV;6: Step three, computed by $\mathbb{P}_1$ decrypts all ciphertexts sent by $\mathbb{P}_2$ and stores the encrypted values $\widehat{e_i}$. The steps are the same for $\mathsf{he} \in \{\mathsf{she}, \mathsf{ahe}\}$.

```
$\psi_\tau^{\Pi,4}(\mathbb{P}_1, \mathbb{P}_2, \mathsf{he}, \Pi, E, Z, R, \tau)$
1 :  $\mathbb{S}_\tau^\Pi \leftarrow \emptyset$;
2 :  $(\mathbb{P}_1 : E_M, \mathbb{P}_2 : \mathbb{V}_M) \leftarrow \Pi^{\mathbb{P}_2}(E, \tau)$;
3 :  for $j \in J$ :
4 :      $(z_{i,1}, z_{i,2}, i) \leftarrow Z[m]$;
5 :      $((r_{i,1}, r_{i,2}), \emptyset) \leftarrow \mathsf{OT}_{n_2}^1(\mathbb{P}_1, \mathbb{P}_2, i, R)$;
6 :      $z \leftarrow z_{i,1}^{-1} \cdot r_{i,1}^{-1} \cdot (z_{i,2} - r_{i,2})$;
7 :      $\mathbb{S}_\tau^\Pi \leftarrow z$;
8 :  $\mathbb{S}_\tau^\Pi \leftarrow \mathbb{S}_1$;
9 :  return $\mathbb{S}_\tau^\Pi$;
```

```
$\Pi^{\mathbb{P}_2}(E, \tau)$
1 :  $E_M, \mathbb{V}_M \leftarrow \emptyset$;
2 :  for $\widetilde{e_m} \in E$ :
3 :      $\nu_m \leftarrow \mathsf{he.Dec}(\mathsf{sk}_2, \widetilde{e_m})$;
4 :      $\tau_m = \tau - \nu_m$;
5 :      if $\tau_m \geq 0$ :
6 :          $E_M \leftarrow \widetilde{e_m}$;
7 :          $\mathbb{V}_M \leftarrow \nu_m$;
8 :          $\tau = \tau_m$;
9 :  return $E_M, \mathbb{V}_M$;
```

Figure IV;7: Left: Step four, computed by both $\mathbb{P}_1$ and $\mathbb{P}_2$, sees the two participants engage in an oblivious transfer that allows $\mathbb{P}_1$ to unmask some of the randomised ciphertexts up to the value threshold τ. In the oblivious transfer, $\mathbb{P}_1$ reveals the random masks used in step two. The steps are the same for $\mathsf{he} \in \{\mathsf{she}, \mathsf{ahe}\}$. We assume that the set E is shuffled before it is sent to $\mathbb{P}_2$ in line 2; and then E_M is unshuffled following when it is received back.
Right: Threshold subset finder algorithm, computed by $\mathbb{P}_2$, used to enforce the threshold τ.

spond to the entries of E_M. Likewise, this requires $\mathbb{P}_1$ to 'unshuffle' E_M before it can proceed to initiate the oblivious transfer protocols.

We can now move onto proving the correctness of the scheme wrt F_τ^Π in Definition IV;6.3.

Theorem IV;7.1 [Correctness]

Let ψ_τ^Π be defined as in Construction III;4.1; let BF be the Bloom filter used in $\psi_\cup$, with parameters $k = \mathsf{poly}(\lambda)$ and L, n_1 chosen optimally. Let τ be the pre-agreed value threshold and let Π be a set agnostic (Definition IV;6.2) algorithm that satisfies Definition IV;6.1, for all universes $\mathcal{S}$.
Then ψ_τ^Π correctly computes the functionality F_τ^Π (Definition IV;6.3) with probability greater than $1 - \mathsf{negl}(\lambda)$.

Proof. By the correctness of he, in either case where he $\in \{\mathsf{she}, \mathsf{ahe}\}$, then in step three: for each $i \in [n_2]$, $\mathbb{P}_1$ receives elements of the form:

- $z_{i,1} = l_i \cdot r_{i,1} = \mathsf{he.Dec}(\mathsf{sk}_1, \widetilde{c_i^+})$;
- $z_{i,2} = (l_i \cdot x_i^{(2)}) + r_{i,2} = \mathsf{he.Dec}(\mathsf{sk}_2, \widetilde{d_i^+})$;
- $\widehat{e_i} = \mathsf{he.Enc}(\mathsf{pk}_2, \nu_i)$, where $\nu_i = \mathsf{value}(x_i^{(2)})$;

and where $l_i = \sum_{i=1}^k \mathsf{IBF.arr}[y_{i,l}]$, where $\{y_{i,l}\}_{l\in[k]} = \mathsf{IBF}.\widetilde{h}(x_i^{(2)})$. Then, if $x_i^{(2)} \in \mathbb{S}_1$: $z_{i,1} = 0$ and $z_{i,2} = r_{i,2}$; else: $z_{i,1} = l_i \cdot r_{i,2} \neq 0$ and $z_{i,2} = (l_i \cdot x_i^{(2)}) + r_{i,2} \neq 0$ (with all but negligible probability in $k = \mathsf{poly}(\lambda)$). Consequently, only when $x_i^{(2)} \notin \mathbb{S}_1$ are $(z_{i,1}, z_{i,2}, i)$ then added to Z, and likewise for $\widetilde{e_i} = \mathsf{he.CRand}(\mathsf{pk}_2, \widehat{e_i}) \to E$.

By the correctness of Π, then $\Pi(E, \tau)$ finds a subset E_M s.t. $\mathsf{value}(E_M) \le \tau$. We have to make sure that E_M is the same in the case of $\Pi^{\mathbb{P}_2}(E, \tau)$ in the protocol and $\Pi(\mathbb{S}_2 \setminus (\mathbb{S}_1 \cap \mathbb{S}_2), \tau)$ in the ideal functionality. Fortunately, since Π is set agnostic (as in Definition IV;6.2), then Π picks the same index set in both cases.

By the correctness of $\mathsf{OT}^1_{n_2}(\mathbb{P}_1, \mathbb{P}_2, i, R)$, $\mathbb{P}_1$ learns $(r_{i,1}, r_{i,2})$, where $(z_{i,1}, z_{i,2}, i) \leftarrow Z[m]$ for each $m \in M$. It is simple to verify that

$$x_i^{(2)} \leftarrow r_{i,1} \cdot z_{i,1}^{-1} \cdot (z_{i,2} - r_{i,2})$$

and this is added to $\mathbb{S}_\tau^\Pi$ for each $m \in M$. At the end of the protocol, $\mathbb{P}_1$ learns $\mathbb{S}_1 \cup \mathbb{S}_\tau^\Pi$ which is the same as the set that is learnt in F_τ^Π by the set agnosticism of Π, and the fact that $x_i^{(2)} \notin \mathbb{S}_1$ with all but negligible probability by the choice of parameters L, k, n_1. □

Theorem IV;7.2 [Security]

Let ψ_τ^Π be defined as in Construction III;9.1. Let Π be a PPT algorithm satisfying Definitions IV;6.1 and IV;6.2. Let he be a homomorphic encryption scheme satisfying semantic security (Definition II;5.1). Let OT_n^1 be a 1-out-of-n oblivious transfer protocol with security against semi-honest adversaries (Definition IV;5.1). Then ψ_τ^Π securely computes the functionality F_τ^Π, under the IND-CPA security of he, in the presence of PPT, semi-honest adversaries corrupting $\mathbb{P}_1$ and $\mathbb{P}_2$.

Proof. We split the proof of this theorem up into two lemmas, that are proved via a series of claims constructing a hybrid argument from the real execution to the simulations. We will use Sim_j to simulate the execution for corrupted $\mathbb{P}_j^\mathcal{A}$, where $\mathcal{A}$ is any PPT adversary.

The proof of this theorem is completed by the proofs of Lemmas IV;7.1 and IV;7.2, below.

Simulation IV;7.1 [$\mathsf{Sim}_1(\mathbb{S}_1, \mathbb{S}_\tau^\Pi, \mathsf{aux}_1)$]

Let $\mathbb{S}_{\mathsf{Sim}} = (\mathbb{S}_\tau^\Pi \setminus \mathbb{S}_1)$. Sim_1 receives $\mathsf{EIBF.pp}$ from $\mathbb{P}_1^\mathcal{A}$ and does the following.

- Computes $c_i^+ \leftarrow \mathsf{EIBF.Query}(\mathsf{EIBF.pp}, x_i^{\mathsf{Sim}})$, for each $x_i^{\mathsf{Sim}} \in \mathbb{S}_{\mathsf{Sim}}$ where $i \in [|\mathbb{S}_{\mathsf{Sim}}|]$.
- Samples $(r_{i,1}, r_{i,2}) \leftarrow\$ \mathcal{X}^2$
- Computes
$$\widehat{c_i^+} \leftarrow \mathsf{he.Mult}(\mathsf{pk}_1, c_i^+, \mathsf{he.Enc}(\mathsf{pk}_1, r_{i,1}));$$
$$\widehat{d_i^+} \leftarrow \mathsf{he.Add}(\mathsf{pk}_1, \mathsf{he.Mult}(\mathsf{pk}_1, c_i^+, \mathsf{he.Enc}(\mathsf{pk}_1, x_i^{\mathsf{Sim}})), \mathsf{he.Enc}(\mathsf{pk}_1, r_{i,2}));$$
$$\widehat{e_i} \leftarrow \mathsf{he.Enc}(\mathsf{pk}_2, \mathsf{value}(x_{\mathsf{Sim}})).$$
- Let $(\widetilde{c_i^+}, \widetilde{d_i^+}) \leftarrow (\mathsf{he.CRand}(\mathsf{pk}_1, \widehat{c_i^+}), \mathsf{he.CRand}(\mathsf{pk}_1, \widehat{d_i^+}))$ and add $(\widetilde{c_i^+}, \widetilde{d_i^+}, \widehat{e_i}) \rightarrow W_{\mathsf{Sim}}[i]$.
- For $i' \in [|\mathbb{S}_1 \cup \mathbb{S}_2| - |\mathbb{S}_{\mathsf{Sim}}|]$: lets $\{g_{i',u} = \mathsf{he.Enc}(\mathsf{pk}_1, r_{i',u})\}_{u \in \{1,2\}}$, and sets
$$(\widetilde{c_{i'}^+}, \widetilde{d_{i'}^+}) \leftarrow (\mathsf{he.CRand}(\mathsf{pk}_1, g_{i',1}), \mathsf{he.CRand}(\mathsf{pk}_1, g_{i',2}));$$
$$\widehat{e_{i'}} \leftarrow \mathsf{he.Enc}(\mathsf{pk}_2, \tau + 1).$$
It adds $(\widetilde{c_{i'}^+}, \widetilde{d_{i'}^+}, \widehat{e_{i'}}) \rightarrow W_{\mathsf{Sim}}[|\mathbb{S}_{\mathsf{Sim}}| + i']$.
- For $i'' \in [|\mathbb{S}_1 \cap \mathbb{S}_2|]$, it sets
$$(c_{i''}^+, d_{i''}^+) \leftarrow (\mathsf{he.Enc}(\mathsf{pk}_1, 0), \mathsf{he.Enc}(\mathsf{pk}_1, r_{i''}));$$
$$\widehat{e_{i''}} \leftarrow \mathsf{he.Enc}(\mathsf{pk}_1, \tau + 1);$$
for $r_{i''} \leftarrow\$ \mathcal{X}$, and adds $(c_{i''}^+, d_{i''}^+, \widehat{e_{i''}}) \rightarrow W_{\mathsf{Sim}}[|\mathbb{S}_1 \cup \mathbb{S}_2| + i'']$.
- Shuffle W_{Sim} and let M be the set of all indices i above, and likewise M' and M'' for indices i' and i'', respectively. Add $W_{\mathsf{Sim}} \rightarrow \mathsf{msgs}_{\mathsf{Sim}}$ (i.e. send it to $\mathbb{P}_1^\mathcal{A}$).
- When $E_M \leftarrow \Pi(E, \tau)$ is activated, then Sim_1 plays the challenger in $\mathsf{exp}_\Pi^{\mathsf{tsf}}(1^\lambda, E, \tau)$. Adds the output to $\mathsf{msgs}_{\mathsf{Sim}}$.
- For each of the $|\mathbb{S}_{\mathsf{Sim}}|$ invocations of $\mathsf{OT}_{n_2}^1$, Sim_1 invokes $\mathsf{Sim}_{\mathsf{ot}}$ to simulate the OT for $\mathbb{P}_1$ playing as the receiver; Sim_1 uses the set $W_M = W[m]$, for $m \in M$. Adds the simulated view $\mathsf{View}_{\mathsf{Sim}}^{\mathsf{ot}}$ to $\mathsf{msgs}_{\mathsf{Sim}}$.
- Outputs the view $\mathsf{View}_{\mathsf{Sim}} = (\mathbb{S}_1, \mathbb{S}_\tau^\Pi, \mathsf{msgs}_{\mathsf{Sim}}, \mathsf{aux}_1)$ that $\mathbb{P}_1^\mathcal{A}$ witnesses.

We show that msgs_1 in the real execution and $\mathsf{msgs}_{\mathsf{Sim}}$ in the simulated paradigm are computationally indistinguishable, using the following lemma.

Lemma IV;7.1 [$\mathsf{View}_1 \approx_c \mathsf{View}_{\mathsf{Sim}}$]

Let $\mathbb{P}_1^{\mathcal{A}}$ be the corrupted participant $\mathbb{P}_1$ for some PPT adversary $\mathcal{A}$. Let $\mathsf{OT}^1_{n_2}$ be a (1-out-of-n_2) oblivious transfer protocol, secure in the presence of semi-honest adversaries. Let he be an IND-CPA secure encryption scheme, s.t. he is either an SHE or AHE scheme that satisfies ciphertext and plaintext rerandomisation security. Then View_1 in ψ^{Π}_{τ} is computationally indistinguishable from $\mathsf{View}_{\mathsf{Sim}}$ created by $\mathsf{Sim}_1(\mathbb{S}_1, \mathbb{S}^{\Pi}_{\tau}, \mathsf{aux}_1)$.

Proof. We prove this lemma with a sequence of hybrid arguments. In each of the hybrid arguments, up until H_3, it is assumed that there is a simulator for ψ^{Π}_{τ} that plays the role of $\mathbb{P}_2$ using the input set $\mathbb{S}_2$, and receiving the inputs of Sim_1 in Simulation IV;7.1. By the last hybrid, the simulator no longer has access to $\mathbb{S}_2$ and only to the inputs of Sim_1.

- H_0: This hybrid is equivalent to the real execution of the protocol.
- H_1: This hybrid is equivalent to the real execution of the protocol, except that

  $$\widehat{e_\iota} \leftarrow \mathsf{he}.\mathsf{Enc}(\mathsf{pk}_2, \tau + 1),$$

  for $\iota \in (M' \cup M'')$.
- H_2: This hybrid is equivalent to H_1, except that $(\widetilde{c_\iota^+}, \widetilde{d_\iota^+})$, for all $\iota \in [n_2]$, are constructed as in Simulation IV;7.1.
- H_3: This hybrid is equivalent to H_2, except that the simulator is only given the set $\mathbb{S}_\tau \subseteq \mathbb{S}_2$ that $\mathbb{P}_1$ receives as output; along with $\mathbb{S}_1$. In other words, the same inputs as Sim_1.
- H_4: This hybrid is equivalent to H_3, except that $\mathbb{P}_1^{\mathcal{A}}$ receives simulated messages $\mathsf{msgs}^{\mathsf{ot}}_{\mathsf{Sim}} \leftarrow \mathsf{Sim}_{\mathsf{ot}}$, where $\mathsf{Sim}_{\mathsf{ot}}$ simulates OT^1_n in the presence of the semi-honest receiver $\mathbb{P}_1^{\mathcal{A}}$.

Let $\rho = n_2 - |\mathbb{S}_{\mathsf{Sim}}|$, then the proof of Lemma IV;7.1 can be proven using the set of claims below.

Claim IV;7.2.1. $\max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{B}, \mathsf{H}_{0,1}(1^\lambda))) \leq \max_{\mathcal{A}'}(\mathsf{Adv}(\mathcal{A}', \rho\text{−}\mathsf{indcpa}(1^\lambda, \mathsf{he})))$ *under the IND-CPA security of the encryption scheme* he.

Proof. The ciphertexts $\widehat{e_\iota}$ that are received are just encryptions under pk_2 of the same values for $\iota \in J$. However, for $\iota \in (M' \cup M'')$ defined above in Simulation IV;7.1, the encryptions are of the plaintext value $\tau + 1$ which is not equivalent to the original execution. Let $\mathcal{A}'$ be any adversary attempting to win $\mathsf{exp}^{\rho\text{-}\mathsf{indcpa}}_{b,\mathcal{A}'}(1^\lambda)$ where $\rho = n_2 - n_{\mathsf{Sim}}$, then $\mathcal{A}'$ plays the role of $\mathbb{P}_2$ in the protocol using $\mathbb{S}_2$. Then $\mathcal{A}'$ creates message pairs of the form $(\mathsf{value}(x_i^{(2)}), \tau + 1)$ for each $\iota \in (M' \cup M'')$ and concatenates these pairs into a list of $2(n_2 - |\mathbb{S}_{\mathsf{Sim}}|)$ messages $(\{\mathsf{value}(x_i^{(2)})\}_{\iota \in (M' \cup M'')}, \{\tau + 1\}_{\iota \in M' \cup M''})$. These messages are then submitted to the challenger in the $\mathsf{exp}^{\rho-\mathsf{indcpa}}_{b,\mathcal{A}'}(1^\lambda)$ experiment, and $\mathcal{A}'$ receives encryptions (under pk_2) of either the left set of messages or the right set. These encryptions are sent as the ciphertexts $e_\iota \in W$ for $\iota \in (M' \cup M'')$. The rest of the simulation is constructed by $\mathcal{A}'$ playing the role of $\mathbb{P}_2$ in H_0.

In the case where the left messages are encrypted ($b = 0$), then the view for $\mathcal{B}$ is equivalent to H_0. Otherwise (when $b = 1$), the view is equivalent to H_1. Now, consider δ as the maximum advantage that a distinguishing adversary $\mathcal{B}$ distinguishes in the experiments $\mathsf{Adv}(\mathcal{A}, \mathsf{exp}^{\mathsf{H}_0;\mathsf{H}_1}_{b,\mathcal{B}}(1^\lambda))$. Then when $\mathcal{B}$ guesses $b_\mathcal{B} \in \{0, 1\}$, $\mathcal{A}'$ also guesses $b_{\mathcal{A}'} = b_\mathcal{B}$ and wins with the same probability δ. Therefore, we have

$$\delta = \max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{B}, \mathsf{H}_{0,1}(1^\lambda))) \leq \max_{\mathcal{A}'}(\mathsf{Adv}(\mathcal{A}', \rho\text{−}\mathsf{indcpa}(1^\lambda, \mathsf{he}))))$$

for all adversaries $\mathcal{A}'$. Recall that proving security under $\mathsf{exp}^{\rho-\mathsf{indcpa}}_{b,\mathcal{A}'}(1^\lambda)$ is equivalent to proving security in the experiments $\mathsf{exp}^{\mathsf{indcpa}}_{b,\mathcal{A}''}(1^\lambda)$ for any PPT adversary $\mathcal{A}''$ [201]. The statement in the claim follows. □

Claim IV;7.2.2. $\max_\mathcal{B}(\mathsf{Adv}(\mathcal{B}, \mathsf{H}_{1,2}(1^\lambda))) = 0$

Proof. Firstly the distribution of the ciphertexts received in W_{Sim} are computationally indistinguishable to the distribution of the ciphertexts received in the set W in $\psi_\tau^{\Pi,3}$ (by the ciphertext rerandomisation property). Secondly, the distributions of the plaintext elements are also perfectly indistinguishable. To see this, note that the distributions are either: both uniform (for $\iota \in (M \cup M')$) due to the usage of the randomising plaintext elements $r_{\iota,1}, r_{\iota,2} \in \mathcal{X}$; or 0 and uniform respectively, for $\iota \in M''$ corresponding to the remaining elements that are unknown to the simulator (i.e. the intersection of the set). Note that we don't actually require usage of the plaintext rerandomisation property here because computing additions over ciphertexts does not increase the encryption level. The resulting ciphertext is still at the same level and so the fact that $r_{\iota,1}, r_{\iota,2}$ are uniformly distributed values is enough.

This is the same in both H_1 and H_2 due to a similar argument as the one made in the proof of Theorem III;5.2, by the uniform sampling of $r_{\iota,1}, r_{\iota,2}$ from $\mathcal{X}$ (and where $\mathcal{X}$ is assumed to be a ring of the form $\mathbb{Z}_q$, for prime integer $q > 0$).

Therefore the distributions witnessed by $\mathcal{B}$ in either of the hybrids are identical and thus the claim follows immediately. □

Claim IV;7.2.3. $\max_\mathcal{B}(\mathsf{Adv}(\mathcal{B}, \mathsf{H}_{2,3}(1^\lambda))) = 0$

Proof. In the real execution, the set $\mathbb{S}_2$ is only used in constructing the messages that $\mathbb{P}_1$ receives in step three. These messages are now constructed as in the simulation of $\mathsf{Sim}_1(\mathbb{S}_1, \mathbb{S}_\tau, \mathsf{aux}_1)$, where $\mathbb{S}_2$ is not known. Therefore the output distribution in both of these hybrids (when viewed by $\mathcal{B}$) are identical regardless of whether $\mathbb{S}_2$ is known, or not. □

Claim IV;7.2.4. $\max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{B}, \mathsf{H}_{3,4}(1^\lambda))) \leq \max_{\mathcal{A}'}(\mathsf{Adv}(\mathcal{A}', \mathsf{ot}(1^\lambda))))$

Proof. The proof of this switch is not complicated since the simulator $\mathsf{Sim}_{\mathsf{ot}}$ can be invoked by the main simulator during the proof. This is because there are no messages sent externally to any invocation of $\mathsf{OT}^1_{n_2}$ and thus the execution is composable with the rest of the protocol. Since we know that $\max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{A}, \mathsf{exp}^{\mathsf{H}_3,\mathsf{H}_4}_{b,\mathcal{B}}(1^\lambda))) = \delta$, then $\mathcal{A}'$ simply acts as the simulator in H_3 and replaces any execution of $\mathsf{OT}^1_{n_2}$ with the execution that it witnesses in $\mathsf{exp}^{\mathsf{ot}}_{b,\mathcal{A}'}(1^\lambda, 1, n_2)$. In the case when $\mathsf{OT}^1_{n_2}$ is the real execution, then this is the same as the simulation in H_3, and otherwise it is the same as in H_4. Thus, $\mathcal{A}'$ guesses $b_{\mathcal{A}'} = b_{\mathcal{B}}$ and wins with the same advantage δ, since this is the only difference in the view of $\mathcal{B}$. The proof of the claim is now complete. □

Claim IV;7.2.5. *The views in* H_4 *and Simulation IV;7.1 are perfectly indistinguishable.*

Proof. Notice that the simulations are computed in exactly the same way. Furthermore, since Π is set agnostic the output of $\Pi(E_M, \tau)$ is the same in all of the hybrids and in Sim_1. The change in H_1 means that only the elements in the output set $\mathbb{S}_{\mathsf{Sim}}$ can be included by Π since the other elements have values that are greater than the threshold τ. Therefore the views are identical, and the claim follows naturally. □

By applying a union bound over all of the claims, we can bound the advantage of an adversary distinguishing the real execution and Simulation IV;7.1 by

$$\max_{\mathcal{A}'}(\mathsf{Adv}(\mathcal{A}', \rho\text{−}\mathsf{indcpa}(\mathcal{A}', \mathsf{he}))) + 0 + 0 + \max_{\mathcal{A}'}(\mathsf{Adv}(\mathcal{A}', \mathsf{ot}(1^\lambda))) = \delta_1 + \delta_2$$

where δ_1 is the advantage in breaking the IND-CPA security of he, and δ_2 is the advantage in breaking the security of the protocol $\mathsf{OT}^1_{n_2}$. Since we assume that performing either of these breaks is computationally difficult (for PPT adversaries $\mathcal{A}, \mathcal{A}'$) with all but negligible probability, then we have $\delta_1 + \delta_2 < \mathsf{negl}(\lambda)$. The proof of Lemma IV;7.1 follows immediately. □

Simulation IV;7.2 [$\mathsf{Sim}_2(\mathbb{S}_2, \mathbb{V}_{\mathsf{Sim}}, \mathsf{aux}_2)$]

Sim_2 receives the output $\mathbb{V}_{\mathsf{Sim}} = \mathsf{value}(\mathbb{S}_1 \setminus \mathbb{S}_\tau^\Pi)$, the associated value set for the output $\mathbb{S}_\tau^\Pi \setminus \mathbb{S}_2$ received by $\mathbb{P}_1^\mathcal{A}$, that $\mathbb{P}_2$ receives. It does the following.

- Runs $\mathsf{BF} \leftarrow \mathsf{BF.init}(1^L, 1^n, 1^k)$ and simply computes

$$\mathsf{EBF} \leftarrow \mathsf{encrypt}(\mathsf{pk}, \mathsf{BF})$$

  and adds $\mathsf{EBF.pp} \rightarrow \mathsf{msgs}_2$.
- For each $i \in [n_2]$: compute $\widetilde{e_i} \leftarrow \mathsf{he.Enc}(\mathsf{pk}_2, \nu_i)$, where $\nu_i = \mathsf{value}(\mathbb{S}_2[i])$.
- Lets $E = \{\widetilde{e_i}\}_{i \in [n_2]}$.
- Runs $E_M \leftarrow \Pi^{\mathsf{Sim}}(E, \tau)$ where it plays the role of the challenger, shuffles $\mathbb{V}_2$ and adds $\mathbb{V}_2 \rightarrow \mathsf{msgs}_{\mathsf{Sim}}$.
- Invokes $\mathsf{Sim}_{\mathsf{ot}}$ for each invocation of $\mathsf{OT}_{n_2}^1$, with $\mathbb{P}_2^\mathcal{A}$ playing the role of the 'sender', and adds the simulated view $\mathsf{View}_{\mathsf{Sim}}^{\mathsf{ot}}$ to $\mathsf{msgs}_{\mathsf{Sim}}$.
- Outputs the view $\mathsf{View}_{\mathsf{Sim}} = (\mathbb{S}_2, \mathbb{V}_{\mathsf{Sim}}, \mathsf{msgs}_{\mathsf{Sim}}, \mathsf{aux}_2)$ that $\mathbb{P}_2^\mathcal{A}$ witnesses.

Lemma IV;7.2 [$\mathsf{View}_2 \approx_c \mathsf{View}_{\mathsf{Sim}}$]

Let $\mathbb{P}_2^\mathcal{A}$ be the corrupted participant $\mathbb{P}_2$ for some PPT adversary $\mathcal{A}$. Let $\mathsf{OT}_{n_2}^1$ be a (1-out-of-n_2) oblivious transfer protocol, secure in the presence of semi-honest adversaries. Let he be an IND-CPA secure encryption scheme, s.t. he is either an SHE or AHE scheme.
Then the view of $\mathbb{P}_2^\mathcal{A}$ in ψ_τ^Π is computationally indistinguishable from $\mathsf{View}_{\mathsf{Sim}} \leftarrow \mathsf{Sim}_1(\mathbb{S}_1, \mathbb{S}_\tau^\Pi, \mathsf{aux}_1)$.

Proof. We prove this lemma with a sequence of hybrid arguments. In each of the hybrid arguments, for H_1 onwards it is assumed that there is a simulator for ψ_τ^Π that simulates the role of $\mathbb{P}_2$ using only the inputs and outputs of Sim_2. This proof is much simpler than in the case of Sim_1, since $\mathbb{P}_2^\mathcal{A}$ receives much less expressive output.

- H_0: This hybrid is equivalent to the real execution of the protocol.
- H_1: This hybrid is equivalent to the real execution of the protocol, except that

$$\mathsf{EIBF.arr} = \mathsf{he.Enc}(\mathsf{pk}_1, 0^L).$$

and the simulation only receives the inputs that Sim_2 receives.

- H_2: This hybrid is equivalent to H_1, except that the protocol $\mathsf{OT}^1_{n_2}$ is simulated by $\mathsf{Sim}_{\mathsf{ot}}$, where $\mathbb{P}_2^{\mathcal{A}}$ is a semi-honest 'sender'.

- H_3: This hybrid is equivalent to H_1, except that the set E is constructed as in Simulation IV;7.2.

We prove that changes displayed in the above hybrid can only be detected with negligible probability, under plausible computational assumptions.

Claim IV;7.2.6. $\max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{B}, \mathsf{H}_{0,1}(1^\lambda))) \leq \max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, L{-}\mathsf{indcpa}(1^\lambda, \mathsf{he})))$ *under the IND-CPA security of the encryption scheme* he.

Proof. The statement of this claim is almost identical to the proof of Theorem III;4.2. Due to the similarity we do not give the full simulation of the hybrids but it can be shown that a distinguishing advantage between H_0 and H_1 can be used as a distinguishing adversary against the security of the encrypted Bloom filter EIBF (similarly to the proof of Theorem III;4.2). Now, the advantage of an adversary in the encrypted Bloom filter case is bounded by the advantage of an adversary $\mathcal{A}'$ in the experiments $\mathsf{exp}^{L\text{-}\mathsf{indcpa}}_{b,\mathcal{A}'}(1^\lambda)$ (Lemma III;3.1). The claim follows by noting that the advantage of $\mathcal{A}'$ can be bounded by the advantage of $\mathcal{A}$ in $\mathsf{exp}^{L\text{-}\mathsf{indcpa}}_{b,\mathcal{A}}(1^\lambda)$ by Lemma II;5.1. $\square$

Claim IV;7.2.7. $\max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{B}, \mathsf{H}_{1,2}(1^\lambda))) \leq \max_{\mathcal{A}'}(\mathsf{Adv}(\mathcal{A}', \mathsf{ot}(1^\lambda)))$

Proof. The advantage of $\mathcal{B}$ can be bounded by the advantage of the $\mathcal{A}'$ attempting to distinguish simulations of $\mathsf{OT}^1_{n_2}$ as the 'sender' by an identical argument to Claim IV;7.2.4. As before, the invocations of $\mathsf{OT}^1_{n_2}$ are standalone protocols engaged by $\mathbb{P}_1$ and $\mathbb{P}_2$. Therefore, the simulator $\mathsf{Sim}_{\mathsf{ot}}$ can be invoked without modification. $\square$

Claim IV;7.2.8. $\max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{B}, \mathsf{H}_{2,3}(1^\lambda))) = 0$

Proof. The plaintexts for the encryptions $\widehat{e_i}$ are identical to the ones created by $\mathbb{P}_2^{\mathcal{A}}$ in the real-world execution. The only difference is that the encryptions in the simulation that are sent back to $\mathbb{P}_2^{\mathcal{A}}$ are new encryptions. However, the ciphertext rerandomisation property (Definition II;5.7) means that the ciphertexts are distributed identically and the claim follows. $\square$

The advantage of an adversary $\mathcal{A}$ in distinguishing H_3 and Simulation IV;7.2 is 0 since the two distributions are identical. Therefore, our proof of security for Lemma IV;7.2 follows if we can show that $\max_{\mathcal{A}'}(\mathsf{Adv}(\mathcal{A}', \mathsf{ot}(1^\lambda))) + \max_{\mathcal{A}'}(\mathsf{Adv}(\mathcal{A}', L{-}\mathsf{indcpa}(1^\lambda, \mathsf{he}))) < \mathsf{negl}(\lambda)$. This follows under the assumptions that he is an IND-CPA secure encryption scheme, and that $\mathsf{OT}^1_{n_2}$ is a secure protocol for (1-out-of-n_2) oblivious transfers (under the presence of semi-honest adversaries). $\square$

The proof of Theorem IV;7.2 follows by the proofs of Lemma IV;7.1 and Lemma IV;7.2. □

IV;8 Asymptotic analysis

We provide a brief asymptotic analysis of the design of our protocol.

Lemma IV;8.1 [Number of rounds]

The number of rounds required for running Construction IV;7.1 is $O(n_2)$.

Proof. There are two rounds needed to compute steps $1-3$. For step four, the number of rounds is proportional to the number of rounds needed to collaboratively compute Π (with $\mathbb{P}_2$ as the challenger) and the number of rounds needed to compute $\mathsf{OT}^1_{n_2}$. For Π, the naive method that we gave previously requires $O(n_2)$ rounds. For $\mathsf{OT}^1_{n_2}$, constructions such as [93, 243] require only a constant number of rounds. Therefore, the total complexity is $O(1) + O(n_2) + O(n_2)$, which is $O(n_2)$. □

Lemma IV;8.2 [Communication complexity]

Let $\mathsf{comms}(\mathsf{OT}^1_{n_2})$ refer to the asymptotic communication complexity of the protocol $\mathsf{OT}^1_{n_2}$. Then the total communication complexity for Construction IV;7.1 is $O(kn_1+n_2)+\mathsf{comms}(\mathsf{OT}^1_{n_2})$.

Proof. The communication for steps $1-3$ is asymptotically the same as in Lemma III;7.1, that is $O(kn_1+n_2)$. For step four, the communication complexity of Π is $O(n_2)$ since there is one ciphertext sent from $\mathbb{P}_1$ to $\mathbb{P}_2$ for each of the n_2 elements in $\mathbb{S}_2$. The claim follows after adding the communication complexity that is required for instantiating the $\mathsf{OT}^1_{n_2}$ protocol. We leave this cost generic as we are agnostic to the OT protocol. □

Lemma IV;8.3 [Computational complexity]

Let $\mathsf{comp}(\mathsf{OT}^1_{n_2})$ refer to the total computational complexity of the protocol $\mathsf{OT}^1_{n_2}$. Then the total computational complexity of Construction IV;7.1 is given by $O(k(n_1 + n_2)) + \mathsf{comp}(\mathsf{OT}^1_{n_2})$ when $\mathsf{he} = \mathsf{she}$, and $O(k(n_1 + n_2)\log_2(|\mathcal{X}|)) + \mathsf{comp}(\mathsf{OT}^1_{n_2})$ when $\mathsf{he} = \mathsf{ahe}$.

Proof. This lemma follows almost immediately from the fact that the computation for the protocol Π is $O(n_1+n_2)$ (not changing from the overall asymptotic complexity given in Lemma III;7.4), and the fact that the oblivious transfer computational complexity is left generic. The complexity for Π is clear since $\mathbb{P}_1$ computes $O(n_1)$ ciphertext rerandomisations and $\mathbb{P}_2$ computes $O(n_2)$ decryptions. We ignore the additional computations needed to compute the output since these only induce a linear number of ring operations; as such they are already included in the asymptotic notation that we use. The case of $\mathsf{he} = \mathsf{ahe}$ follows naturally. □

Discussion. We do not give a concrete experimental analysis of this protocol since the extra primitives and protocol steps that we require are likely to make the base design fairly inefficient. However, as long as the protocol $\mathsf{OT}^1_{n_2}$ has linear computational and communicational complexities then the protocol maintains the linear scalability in the set size $n = n_1 = n_2$. That is, the design remains scalable as set sizes increase, even if the initial costs are large enough to make the protocol considerably less efficient than the toolkit implementation in Section III;8. Moreover, the Bloom filter amortisation improvements still apply and using such improvements may still see a dramatic reduction in the runtimes.

Malicious security. Achieving malicious security for ψ^{Π}_{τ} is a considerably difficult task, even in the authenticated PSO model. This is because $\mathbb{P}_1$ can now choose to ignore the evaluation of Π and try to compute E_M in any arbitrary manner. The proof of Theorem IV;7.2 no longer applies, since the encryptions of $\tau+1$ mean that learning the subset E_M in the simulation will take noticeably longer than in the real execution with high probability. As a result, we do not consider a formal proof of security in this model. It should be noted that the motivating trading dynamics from [119] require some sort of pre-arranged trust relationship between the entities taking place anyway. The semi-honest setting is a plausible security model, if the participants have some degree of trust already.

IV;9 Conclusion

We showed that we can adapt our PSU protocol from Chapter III to a situation where set elements have associated data in the form of values that are canonically assigned. Our new protocol allows implementing an upper threshold on the value of the elements that are learnt in this protocol, preventing the whole union being learnt after this threshold is surpassed.

The purpose of this design is to meet the needs of a particular idealised mediator that enables two entities to share vulnerability information, whilst guaranteeing a fair trade between the two participants. The adapted protocol is significantly more complex and requires a linear number of rounds in the set sizes (for instantiating a proportional number of oblivious transfer protocols). However, the computation and communication complexities remain asymptotically similar, depending on the complexities of the inferred OT protocol.

We view the results of this chapter as verification that it is possible to achieve complex functionality from the simplicity of our original PSO protocols. It should be noted that we have to modify our PSU protocol in a non-black-box manner to achieve this, and so we do not claim that we can achieve such functionality in a generic way. Nevertheless, this still shows that our protocols are malleable and can be used to construct non-trivial adaptations of the original PSO functionality.

Future research follows the same aims as Chapter III, i.e. the investigation of more efficient protocols, and the pursuing of multi-party variants for information sharing scenarios.

V A Constrained PRF for Bit-fixing Predicates with Constant Collusion-resistance from One-way Functions

a lack of understanding

Preface

In this chapter, we devise a new construction of a constrained pseudorandom function (CPRF) satisfying security guarantees that were not previously achievable from standard cryptographic assumptions. A CPRF, in short, has the same functionality as a standard PRF; except that the adversary has access to a 'constrained' key that can evaluate the function on subsets of the input space. The security goal specifies that input points not in those subsets still result in pseudorandom evaluations.

Our CPRF construction allows constrained keys for the bit-fixing predicate. It satisfies collusion-resistance for $r = O(1)$ constraint queries, i.e. the adversary can receive up to $r = O(1)$ keys. Previous constructions from standard assumptions (such as LWE) only achieved security for 1 query. In addition, our construction requires only the assumption that one-way functions exist. This is comparatively weaker than all known previous constructions, for any predicates covering the bit-fixing functionality. Our security proof is valid in a model where the adversary makes all queries adaptively. This is the only construction to achieve adaptivity from standard assumptions and in the standard model; it also achieves 1-key privacy.

Finally, we give an implementation of our scheme. Preliminary results suggest that it is very practical for small values of r (e.g. ≤ 5) and various input lengths. No previous constrained PRFs came equipped with implementations (partly due to the fact that they were likely to be largely impractical to run). In this sense, we make a large jump in providing a practical implementation of such a construction. We hope that this may lead to interesting applications of CPRFs in real-world scenarios

This chapter is based on work that is currently under submission. The work was jointly authored with Shuichi Katsumata, Ryo Nishimaki and Shota Yamada and was completed while the thesis author undertook a research visit at NTT Secure Platform Laboratories (under the supervision of Ryo Nishimaki). The submission is itself derived from the pre-print here [121].

Overview of contributions

- A new CPRF construction for bit-fixing predicates from one-way functions.
- Collusion-resistance for $O(1)$ constraint queries.
- Proof of adaptive security with only polynomial security loss.
- Proof of perfect 1-key privacy.

- An implementation (written in `Go`) of the CPRF, demonstrating practicality for small numbers r of collusions. See Appendix A for the source code.

Contents

V;1 Introduction

Historically, pseudorandom functions (PRFs) provide the basis of a huge swathe of cryptography. Intuitively, such a function takes a uniform key and some binary string x as input, and outputs (deterministically) some value y. The pseudorandomness of the function dictates that y is indistinguishable from the output of a uniformly sampled function operating solely on x. PRFs typically provide useful sources of randomness in cryptographic constructions that take adversarially-chosen inputs.

Simple constructions of PRFs exist based on well-known standard assumptions: Goldreich, Goldwasser and Micali give a construction based on the existence of pseudorandom generators [164]; Naor and Reingold [244] give a simple construction from assumptions related to the discrete log problem.

There have been numerous expansions of the definitional framework surrounding PRFs. In this work, we focus on a strand of PRFs that are known as *constrained* PRFs or CPRFs. CPRFs were first introduced by Boneh and Waters [58] alongside the concurrent works of Kiayias et al. [209] and Boyle et al. [60]. They differ from standard PRFs in that they allow users to learn 'constrained' keys that can evaluate the function on a subset of the input space. That is, let $\mathcal{X}$ denote the input space, and let $S \subseteq \mathcal{X}$. Then a constrained key CK_S allows evaluating the CPRF if and only if $x \in S$.

In the security game, the adversary is permitted to make queries for learning PRF evaluations as with standard PRFs. The adversary is also permitted to learn constrained keys for any subsets $S \subseteq \mathcal{X}$ that it wants. Security now dictates that the CPRF remains pseudorandom on an input point that lies outside of the queried subsets. If an adversary can ask more than one constrained key query, then we say the CPRF is *collusion-resistant*.

In this work our main question is:

Can we construct collusion-resistant constrained PRFs from standard assumptions?

Up until now, this question has been unanswered in either the affirmative or the negative regardless of the predicate that is considered for constraining keys.

Predicates. While constrained keys can be defined with respect to subsets, a more natural definition defines functionality with respect to predicates. That is, the constrained key allows evaluation of the function on x, if and only if the associated predicate is equal to 1 on x. We denote such a predicate by $\mathsf{P}(\cdot)$ and $1 \leftarrow \mathsf{P}(x)$ indicates that the input satisfies the constraint. Otherwise, we write $0 \leftarrow \mathsf{P}(x)$.

Many suitable predicates for CPRFs have been proposed in the literature, such as:

- puncturing [55, 58, 60, 209];
- prefixes [24, 58];
- left-right (LR) [58];
- bit-fixing (BitFix) [20, 55, 58, 76];
- general circuits in $\mathcal{C}_{\mathsf{NC}^1}$ [20, 76, 88];
- general circuits in $\mathcal{C}_{\mathsf{P/poly}}$ [55, 58, 67, 68, 187, 255].

We highlight the two predicates that are the most interesting due to the expressibility and flexibility of the predicate class that they support.

- Bit-fixing predicates are associated with a string $v \in \{0, 1, *\}^\ell$ as input; where $v_i = *$ indicates a *wildcard* entry. Denote the predicate by $\mathsf{P}_v(x)$ for some $x \in \{0, 1\}^\ell$. Then we say that $1 \leftarrow \mathsf{P}_v(x)$ iff $(x_i = v_i) \vee (v_i = *)$ for each $i \in [\ell]$. The class of such predicates will be denoted by BitFix.
- General circuit predicates are associated with some representative circuit $C \in \mathcal{C}$. We say that $1 \leftarrow \mathsf{P}_C(x)$ for $x \in \{0, 1\}^\ell$ if and only if $C(x) = 1$.

V;1.1 CPRF security notions

The original work of [58] conceived CPRFs as pseudorandom functions (Definition II;4.2) with additional algorithms $\mathsf{Constrain}$ and CEval. The $\mathsf{Constrain}$ algorithm outputs "constrained" keys that corresponding to predicates that exist over the input space. The CEval algorithm can be run using a constrained key and a valid PRF input and evaluates the CPRF correctly if x satisfies the associated the predicate. In other words, security can be informally categorised into the following properties for a CPRF denoted by cprf.

- Correctness: Let $\mathsf{P}_C(\cdot)$ be a predicate for some $C \in \mathcal{C}$, and let $x \in \mathcal{X}$ be an input s.t. $1 \leftarrow \mathsf{P}_C(x)$, where $\mathcal{X}$ is the input space of $\mathcal{C}$. Let msk be the master secret key of cprf, and let CK_C be a constrained key for the predicate $\mathsf{P}_C(\cdot)$. Then evaluating cprf at x using msk should give an equivalent output to evaluating cprf at x using the constrained key CK_C.
- Pseudorandomness: Let $\mathsf{P}_C(\cdot)$, msk and CK_C be as above, but consider $x \in \mathcal{X}$ s.t. $0 \leftarrow \mathsf{P}_C(x)$. Then $\mathsf{cprf}.\mathsf{Eval}(x)$ should appear pseudorandom.[1]

[1]This should still hold even given multiple constrained keys, as long as the input is 'constrained' wrt all of them.

If these two properties are satisfied, then we say that cprf is a constrained PRF (or CPRF). The later work of Boneh et al. [55] introduced an extra security property relating to the *privacy* of the CPRF.[2]

- 1-KEY PRIVACY: Let $C_0, C_1 \in \mathcal{C}$ and let CK_{C_b} be a constrained key for C_b for $b \leftarrow_{\$} \{0, 1\}$. Then there is no PPT adversary that can distinguish whether b is equal to 0 and 1 with anything other than negligible advantage.

We say that a CPRF is a *private* CPRF (or PCPRF) if it satisfies this extra security property. Note that we could adapt the definition to m-key privacy for $m \geq 1$, by allowing the adversary to query m pairs of constraint circuits.

Remark V;1.1. *The security notion above is known as* weak *key privacy in [55], a stronger security notion allows the adversary to make evaluation queries as well. Fortunately, it has been shown that* weak *key privacy is typically enough for most applications of PCPRFs [55, 76]. In this thesis, we will focus only on the case of weak key privacy.*

We give formalisations of all of the above security properties in Section V;2.2 . In these formalisations, the adversary is able to query for constrained keys, and then receives a challenge equal to the CPRF evaluated at an input $x^\dagger$ (chosen by the adversary) that cannot be evaluated by any of the known constrained keys. This requires that $0 \leftarrow \mathsf{P}_C(x^\dagger)$, for any C that has been queried. The security game asks the adversary to distinguish the evaluation of the CPRF at $x^\dagger$ from an evaluation made at the same point by a randomly sampled function. If the adversary makes all queries adaptively, then we say that the CPRF achieves adaptive security; otherwise it only achieves selective security.

V;1.2 Existing constructions of CPRFs

Since the original work of [58], numerous constructions of CPRFs have been given, relying on different primitives and providing a range of functionality. The work of [58] gave constructions of CPRFs for LR, BitFix and $\mathcal{C}_{\mathsf{NC}^1}$ circuit predicates. The LR predicate CPRF was derived from the bilinear decisional Diffie-Hellman (BDDH) assumption and the existence of random oracles. The other constructions were derived from multilinear maps that satisfy the multilinear DDH (MDDH) assumption. These original constructions satisfy collusion-resistance for any polynomial number of constrained keys. That is, the adversary in the CPRF security game can learn multiple constrained keys.

The CPRFs of [60, 209] construct CPRFs for puncturing from GGM-based techniques (and thus OWFs). More accurately, they show that they can achieve prefix- and range-based predicates. The works of Bitansky [45] and Goyal et al. [171] construct CPRF-like constructions for substring

[2]This property was also indirectly considered in [209].

predicates. However we ignore these constructions further since the adversary is not permitted access to the evaluation oracle. The work of Banerjee et al. [24] constructs a CPRF for prefix-based predicates from lattices.

Hofheinz et al. [187] develop a construction with stronger security guarantees in that all queries can be answered adaptively, this is the only construction that satisfies adaptive security without incurring a sub-exponential loss in security. Unfortunately their construction is based on very strong assumptions — namely a combination of indistinguishability obfuscation (IO) and random oracles. All other works require sub-exponential time reductions to achieve adaptive security.

More recent constructions have constructed CPRFs from much weaker assumptions, at the expense of providing weaker guarantees. The works of [67, 68, 76, 88, 255] construct CPRFs from learning with errors (LWE), and other lattice-based assumptions. Unfortunately, none of these constructions satisfy collusion-resistance or adaptive security (using polynomial-time reductions). However, each of these constructions can create a constrained key for circuits taken from the class $\mathcal{C}_{\mathsf{NC}^1}$. The works of [67, 68, 88, 255] actually provide constrained keys for $\mathcal{C}_{\mathsf{P/poly}}$. The work of [20] provides a construction of CPRFs, from assumptions in traditional groups, for circuit predicates in $\mathcal{C}_{\mathsf{NC}^1}$ — again security only holds for one constrained key query.

In Table V;1, we provide a summary of the known CPRF constructions. We categorise [60, 209] as puncturing CPRFs since they use GGM-based techniques. We do not consider the CPRFs of [45, 171] since they do not permit evaluation queries. We also ignore the construction of [24] due to the weakness of prefix-based predicates.

V;1.3 Our contributions

In this work we develop a new CPRF construction for the bit-fixing predicate. While the BitFix class is less expressive than predicates for circuits, our construction is derived only from the existence of one-way functions; which is a remarkably weak assumption compared to any of those that have been used in past constructions. Moreover, our construction is the first to satisfy collusion-resistance for BitFix from an assumption that is not related to the existence of indistinguishability obfuscation or multilinear maps.

Specifically, our construction is secure against PPT adversaries who learn $r = O(1)$ constrained keys (i.e. a constant number with respect to the security parameter). Our security proof holds in the setting where queries are made adaptively with a security loss of only $1/\mathsf{poly}(\lambda)$. Specifically, for an adversary that succeeds in the adaptive model with advantage ϵ, we essentially construct an adversary that succeeds in the selective security model with advantage $(1/\mathsf{poly}(\lambda))\epsilon$. Finally, our construction satisfies perfect weak 1-key privacy [55]. Our contribution is summarised alongside the existing state-of-the-art at the bottom of Table V;1.

	Collusion-resistance	Privacy	Adaptive	Predicate	Assumption
[58]	1	0	◊	Puncturing	OWF
	poly(λ)	poly(λ)	♦	LR	BDDH & ROM
	poly(λ)	0	◊	BitFix	MDDH
	poly(λ)	0	◊	$\mathcal{C}_{\mathsf{P/poly}}$	MDDH
[209]	1	1	◊	Puncturing	OWF
[60]	1	0	◊	Puncturing	OWF
[187]	poly(λ)	0	♦	$\mathcal{C}_{\mathsf{P/poly}}$	IO & ROM
[24]	poly(λ)	0	◊	Prefix	LWE
[68]	1	0	◊	$\mathcal{C}_{\mathsf{P/poly}}$	LWE
[55]	poly(λ)	1	◊	Puncturing	MDDH
	poly(λ)	1	◊	BitFix	MDDH
	poly(λ)	poly(λ)	◊	$\mathcal{C}_{\mathsf{P/poly}}$	IO
[67]	1	1	◊	$\mathcal{C}_{\mathsf{P/poly}}$	LWE
[76]	1	1	◊	BitFix	LWE
	1	1	◊	$\mathcal{C}_{\mathsf{NC}^1}$	LWE
[20]	1	0	◊	$\mathcal{C}_{\mathsf{NC}^1}$	L-DDHI
	1	1	◊	BitFix	DDH
	1	0	♦	$\mathcal{C}_{\mathsf{NC}^1}$	L-DDHI & ROM
	1	1	♦	BitFix	ROM
[88]	1	1	◊	$\mathcal{C}_{\mathsf{NC}^1}$	LWE
[255]	1	1	◊	$\mathcal{C}_{\mathsf{P/poly}}$	LWE
This work	$O(1)$	1	♦	BitFix	OWF

Table V;1: List of existing constructions of CPRFs along with their functionality and the assumptions required. BDDH and MDDH assumptions refer to the *bilinear* and *multilinear* DDH assumptions, we cover the case of MDDH in Chapter VII. The L-DDHI assumption is the L-decisional Diffie-Hellman Inversion problem. The adaptive column only refers to works that achieve adaptive security with polynomial security losses.

Finally, we demonstrate a proof-of-concept implementation that suggests that our construction is practical for small values of r. Our implementation represents the first practical instantiation of a CPRF. The construction is written in Go and demonstrates that our scheme is particularly efficient during evaluation and constraining. This is mainly due to the simple evaluation procedure relying only on PRF evaluations.

In comparison, the key generation procedure is expensive in terms of computation and memory usage — these costs are exponential in the growth of r. For example, the size of the master secret key is (by Stirling's approximation):

$$\sum_{j=1}^{r} 2^j \cdot \binom{\ell}{j} \geq \sum_{j=1}^{r} 2^j \cdot \ell^j / j!,$$

and thus taking $\ell = \lambda = 128$ and $r = 4$, we get the size to be greater than $2^4 \cdot 128^4/4! = 2^4 \cdot 2^{28}/24 > 2^{27}$. Thus for large input lengths, generating the required number of keys is a cumbersome process. We can alleviate the burden slightly by taking in smaller input lengths, but this does not capture all scenarios. The polynomial dependence of ℓ on λ for achieving meaningful security means that we can only achieve constant size r in relation to λ, i.e. only a small increase in r increases the overheads significantly. If we could achieve $r = O(p(\lambda))$ for some (potentially bounded) polynomial $p(\lambda)$, then we should see a notable improvement in functionality and efficiency. See Section V;3.2 for a technical overview of our construction.

V;2 Preliminaries

V;2.1 Additional notation

Let $\mathbb{T}$ be a set such that

$$\mathbb{T} = \{(t_1, \ldots, t_r) \in [\ell]^r \,|\, t_1 \leq t_2 \leq \ldots \leq t_r\};$$

In other words, $\mathbb{T}$ is the set of all ordered vectors with entries taken from $[\ell]^r$.

For $\boldsymbol{b} \in \{0,1\}^\ell$ and $t \in \mathbb{T}$, we will write $\boldsymbol{b_t} \leftarrow \mathsf{reindex}(\boldsymbol{b}, \boldsymbol{t})$ to denote the vector that is reindexed with respect to the unique entries t_i in $\boldsymbol{t} \in \mathbb{T}$ (entries where $(i = 1) \vee (t_{i-1} < t_i)$). In other words, let z be the number of such unique entries; for shorthand, we write $z \leftarrow \mathsf{unique}(\boldsymbol{t})$.[3] Then we have $\boldsymbol{b_t} \in \{0,1\}^z$, with $\boldsymbol{b_t}$ including only those components $b_{t_i} \in \boldsymbol{b_t}$ where $t_i \in [\ell]$ is a unique entry of $\boldsymbol{t}$. We may abuse notation and write $x_{\boldsymbol{t}} \leftarrow \mathsf{reindex}(x, \boldsymbol{t})$ similarly, where $x \in \{0,1\}^\ell$ is explicitly said to be a bitstring. Therefore, if the unique entries t_i of $\boldsymbol{t}$ are (t_2, t_5, t_7, t_9), then $\boldsymbol{b_t} = (b_{t_2}, b_{t_5}, b_{t_7}, b_{t_9})$.

Finally, we define the procedure $\mathbb{T} \leftarrow \mathsf{dedup}(\mathbb{T})$ that deduplicates the vectors in $\mathbb{T}$ that have the same unique entries. For example, the vectors $(1,3,3,3)$, $(1,1,3,3)$ and $(1,1,1,3)$ are all equivalent under this consideration. We use this procedure to remove vectors that would otherwise result in duplicated evaluations of PRF in the evaluation of our CPRF. We will discuss this more later. After running $\mathsf{dedup}(\mathbb{T})$, only the vector with the highest lexicographic ordering is retained for each possible vector of unique entries.

V;2.2 Definitions

Recall the formalisation of a PRF given in Definition II;4.2. Then the concept of a CPRF is enshrined in Definition V;2.1.

[3]As an example, if $\boldsymbol{t} = (1,1,2,5,7,7)$, then $z = 4$. The unique entries are (t_1, t_2, t_5, t_7).

Definition V;2.1 [Constrained PRF]

A constrained PRF is a tuple cprf consisting of four algorithms:

$$(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Constrain}, \mathsf{CEval}),$$

and satisfying the following functionality:

- $\mathsf{cprf}.\mathsf{Setup}(1^\lambda, 1^r, \mathcal{C})$: On input the security parameter λ, a parameter r, and a class of predicates $\mathcal{C}$: outputs public parameters pp and master secret key msk.
- $\mathsf{cprf}.\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x \in \mathcal{X})$: On input $x \in \mathcal{X}$, outputs some value $y \in \mathcal{Y}$.
- $\mathsf{cprf}.\mathsf{Constrain}(\mathsf{pp}, \mathsf{msk}, C \in \mathcal{C})$: On input $C \in \mathcal{C}$, outputs a *constrained* key CK_C.
- $\mathsf{cprf}.\mathsf{CEval}(\mathsf{pp}, \mathsf{CK}_C, x)$: On input a constrained key CK_C for $C \in \mathcal{C}$, if $1 \leftarrow \mathsf{P}_C(x)$: then outputs $y \in \mathcal{Y}$, else: outputs $\perp$.

We may omit the class $\mathcal{C}$ from the inputs to $\mathsf{cprf}.\mathsf{Setup}$, if it is obvious from context.

A constrained key $\mathsf{CK} \leftarrow \mathsf{cprf}.\mathsf{Constrain}(\mathsf{pp}, \mathsf{msk}, C)$ can evaluate the original pseudorandom function at inputs $x \in \mathcal{X}$ where $1 \leftarrow \mathsf{P}_C(x)$, using $\mathsf{cprf}.\mathsf{CEval}(\mathsf{CK}_C, x)$. Such inputs x are termed *unconstrained*. Inputs x' that cannot be evaluated (i.e. $0 \leftarrow \mathsf{P}_C(x')$) are termed *constrained*, since they are *constrained* wrt to the constrained key.

The parameter r that is input to the setup algorithm is used as a bound on the number of queries to $\mathsf{cprf}.\mathsf{Constrain}$ that can be made. If this parameter is omitted, we assume that the number of constrained keys that can be learnt is unbounded.

V;2.3 Correctness

Recall that the correctness property ensures that unconstrained inputs (in relation to some $C \in \mathcal{C}$) can be evaluated using either: the master secret key msk in conjunction with the real evaluation algorithm $\mathsf{cprf}.\mathsf{Eval}$; or the constrained key CK_C in conjunction with the constrained evaluation algorithm $\mathsf{cprf}.\mathsf{CEval}$. We formalise this property in Definition V;2.2.

Definition V;2.2 [Correctness]

Let:

$$P = \Pr\left[\mathsf{cprf.CEval}(\mathsf{pp}, \mathsf{CK}_C, x) \neq \mathsf{cprf.Eval}(\mathsf{pp}, \mathsf{msk}, x) \,\middle|\, \substack{(\mathsf{pp},\mathsf{msk}) \leftarrow \mathsf{cprf.Setup}(1^\lambda, 1^r, \mathcal{C}) \\ \mathsf{CK}_C \leftarrow \mathsf{cprf.Constrain}(\mathsf{pp},\mathsf{msk},C) \\ 1 \leftarrow \mathsf{P}_C(x)}\right]$$

then we say that cprf is *correct* if:

1. all algorithms run in time polynomial in λ;
2. we have that $P < \mathsf{negl}(\lambda)$ for all $C \in \mathcal{C}$.

We say that cprf is *perfectly correct* if $P = 0$.

V;2.4 Security

For the constrained PRF indistinguishability security game [58] (experiments $\mathsf{exp}^{\mathsf{cprf}}_{b,\mathcal{A}}(1^\lambda, 1^r)$), we modify the standard PRF adversary $\mathcal{A}$ so that it also has access to an oracle

$$\mathcal{O}^\theta_\mathcal{C}(\cdot) = \mathcal{O}_\mathcal{C}(\mathsf{cprf.Constrain}(\mathsf{msk}, \mathsf{pp}, \cdot), \theta)$$

for learning constrained keys, along with an oracle

$$\mathcal{O}^\varphi_\mathcal{X}(\cdot) = \mathcal{O}_\mathcal{X}(\mathsf{cprf.Eval}(\mathsf{msk}, \mathsf{pp}, \cdot), \varphi),$$

for learning PRF evaluations. The set θ is used to keep track of the points that $\mathcal{A}$ can currently evaluate using constrained keys $\mathsf{CK}_C \leftarrow \mathcal{O}^\theta_\mathcal{C}(C)$, and the set φ keeps track of the points x where $\mathcal{A}$ has queried $y \leftarrow \mathcal{O}^\varphi_\mathcal{X}(x)$.

Additionally, we specify a bound r on the number of key constraint queries that can be run; this is included as an extra input to $\mathcal{O}^\theta_\mathcal{C}(\cdot)$. That is, there is an internal state in this oracle that monitors the number of queries that have been asked by $\mathcal{A}$. If r is exceeded then the oracle simply outputs $\perp$. Recall from Section II;1 that we write $\mathcal{O}^\theta_\mathcal{C}(\cdot\,; [r])$ when the oracle is limited in this way.

The entire (adaptive) security game is given in Figure V;1, and formal specification of security is given in Definition V;2.3.

```
$\mathsf{exp}^{\mathsf{cprf}}_{0,\mathcal{A}}(1^\lambda, 1^r)$
1 :  $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{cprf.Setup}(1^\lambda, 1^r, \mathcal{C})$;
2 :  $x^\dagger \leftarrow \mathcal{A}^{\mathcal{O}^{\varphi}_{\mathcal{X}}(\cdot), \mathcal{O}^{\theta}_{\mathcal{C}}(\cdot\,;\,[r])}(1^\lambda, 1^r, \mathsf{pp})$;
3 :  $y^\dagger \leftarrow \mathsf{cprf.Eval}(\mathsf{msk}, x^\dagger)$;
4 :  $b_{\mathcal{A}} \leftarrow \mathcal{A}^{\mathcal{O}^{\varphi}_{\mathcal{X}}(\cdot), \mathcal{O}^{\theta}_{\mathcal{C}}(\cdot\,;\,[r])}(1^\lambda, 1^r, \mathsf{pp}, x^\dagger, y^\dagger)$;
5 :  **if** $x^\dagger \in \varphi$ :
6 :    **return** $\perp$;
7 :  **for** $C \in \theta$ :
8 :    **if** $1 \leftarrow \mathsf{P}_C(x^\dagger)$ :
9 :      **return** $\perp$;
10 : **return** $b_{\mathcal{A}}$;
```

```
$\mathsf{exp}^{\mathsf{cprf}}_{1,\mathcal{A}}(1^\lambda, 1^r)$
1 :  $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{cprf.Setup}(1^\lambda)$; $f \leftarrow_{\$} \mathsf{prf}.\mathcal{F}$;
2 :  $x^\dagger \leftarrow \mathcal{A}^{\mathcal{O}^{\varphi}_{\mathcal{X}}(\cdot), \mathcal{O}^{\theta}_{\mathcal{C}}(\cdot\,;\,[r])}(1^\lambda, 1^r, \mathsf{pp})$;
3 :  $y^\dagger \leftarrow f(x^\dagger)$;
4 :  $b_{\mathcal{A}} \leftarrow \mathcal{A}^{\mathcal{O}^{\varphi}_{\mathcal{X}}(\cdot), \mathcal{O}^{\theta}_{\mathcal{C}}(\cdot\,;\,[r])}(1^\lambda, 1^r, \mathsf{pp}, x^\dagger, y^\dagger)$;
5 :  **if** $x^\dagger \in \varphi$ :
6 :    **return** $\perp$;
7 :  **for** $C \in \theta$ :
8 :    **if** $1 \leftarrow \mathsf{P}_C(x^\dagger)$ :
9 :      **return** $\perp$;
10 : **return** $b_{\mathcal{A}}$;
```

Figure V;1: CPRF indistinguishability game (adaptive).

Definition V;2.3 [CPRF security]

Let $\mathsf{exp}^{\mathsf{cprf}}_{b,\mathcal{A}}(1^\lambda, 1^r)$ be the experiments defined as in Figure V;1. We say that cprf is an r-key secure, *constrained* pseudorandom function (or a CPRF) if

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{cprf}(1^\lambda))) < \mathsf{negl}(\lambda),$$

holds for all PPT adversaries $\mathcal{A}$.

The game ultimately requires $\mathcal{A}$ to distinguish a PRF evaluation on a constrained input $x^\dagger$ (of the adversary's choice) from a uniformly distributed output. It concludes when the adversary submits a bit $b_{\mathcal{A}} \in \{0, 1\}$ indicating its decision. The formulation in Figure V;1 targets adaptive security, since all queries are made adaptively. We can modify to target selective security by specifying that some subset of the queries are specified by the adversary before the setup step is run (in step one).

m-KEY PRIVACY. As an additional requirement, we can specify that a CPRF is m-key *private* CPRF (or PCPRF) if the constrained keys for m different pairs of constraints are indistinguishable. We can define the security game as in Figure V;2 based on the indistinguishability model given in [55]. The explicit formalisation is given in Definition V;2.4, where $m = 1$. We only define the model for $m = 1$ because this is the only case that we consider in this chapter (see Remark V;2.1).

The definition that we use below is actually known as *weak* key privacy in [55], since the adversary is not permitted to make evaluation queries. Fortunately, weak key privacy is enough for most of the main applications of PCPRFs [55, 76]. Moreover, we are unable to prove the stronger guarantee where evaluation queries are permitted in the adaptive setting. We leave this open to future work. When we refer to m-key privacy in the sequel, we will always be referring to the weak key privacy security model.

```
$\mathsf{exp}^{\mathsf{pcprf}}_{b,\mathcal{A}}(1^\lambda)$
1 : $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{cprf.Setup}(1^\lambda);$
2 : $C_0, C_1 \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp});$
3 : $\mathsf{CK}_b \leftarrow \mathsf{cprf.Constrain}(\mathsf{pp}, \mathsf{msk}, C_b);$
4 : $b_{\mathcal{A}} \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}, \mathsf{CK}_b);$
```

Figure V;2: 1-key privacy in indistinguishability framework.

Definition V;2.4 [1-key privacy]

Let $\mathsf{exp}^{\mathsf{pcprf}}_{b,\mathcal{A}}(1^\lambda)$ denote the experiments from Figure V;2. We say that cprf satisfies weak 1-key privacy (or, that it is a PCPRF for 1 constrained key query) if

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{pcprf}(1^\lambda))) < \mathsf{negl}(\lambda)$$

holds for all PPT adversaries $\mathcal{A}$.

Note that, if $\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{pcprf}(1^\lambda))) = 0$ for all algorithms $\mathcal{A}$, then we say that cprf satisfies *perfect* weak 1-key privacy.

Remark V;2.1. *In Figure V;2 we choose to target the case of $m = 1$, since our construction only satisfies this requirement. To target m-key privacy, we would allow the adversary to query m pairs $(C_0^{(i)}, C_1^{(i)})$ for $i \in [m]$. To date, only the constructions of [55] and the LR CPRF of [58] satisfy privacy for $m > 1$.*

Simulation-based security. The work of [76] developed a simulation-based security framework, combining the two security properties from Definitions V;2.3 and V;2.4. In this framework, the simulator has no access to the constraint when answering queries using $\mathsf{cprf.Constrain}$, except for the size of the constraint circuit. It was shown by [76] that simulation security implies security in the weaker indistinguishability setting, but the reverse does not hold. We are unable to prove the security of our scheme in the simulation-based setting since the privacy property of our construction only holds for $m = 1$, while we permit $r \geq 1$ constraint queries.

V;3 Construction

V;3.1 Preparation

Before describing our construction we recall some notation that we defined in Section V;2.1. Initially, let $\mathbb{T}$ be a set such that $\mathbb{T} = \{(t_1, \ldots, t_r) \in [\ell]^r \mid t_1 \leq t_2 \leq \ldots \leq t_r\}$. Now, we compute $\mathbb{T} \leftarrow \mathsf{dedup}(\mathbb{T})$ to deduplicate the vectors in $\mathbb{T}$ that are equivalent after only considering their unique entries (such as the vectors $(1, 1, 2)$ and $(1, 2, 2)$). We do this to reduce the computational complexity of the evaluation procedure of the CPRF that we construct, but otherwise this is a cosmetic change.

For $\boldsymbol{b} \in \{0, 1\}^\ell$, we will write $\boldsymbol{b_t} \leftarrow \mathsf{reindex}(\boldsymbol{b}, \boldsymbol{t})$ to denote the vector that is reindexed with respect to the unique entries t_i in $\boldsymbol{t} \in \mathbb{T}$ (entries where $(i = 1) \vee (t_{i-1} < t_i)$). This means that if $\boldsymbol{t} = (1, 1, 2, 7, 8, 8, 9)$ and $\boldsymbol{b} \in \{0, 1\}^{10}$, then $\boldsymbol{b_t} = (b_1, b_2, b_7, b_8, b_9)$. We may abuse notation and write $x_{\boldsymbol{t}} \leftarrow \mathsf{reindex}(x, \boldsymbol{t})$ similarly, where $x \in \{0, 1\}^\ell$ is explicitly a bitstring.

Let z be the number of unique entries in $\boldsymbol{t}$. We write $z = \mathsf{unique}(\boldsymbol{t})$ for shorthand, then in our example for $\boldsymbol{t} = (1, 1, 2, 7, 8, 8, 9)$ above we would have $z = 5$.[4]

Intuitively, the reason why we require that $r = O(1)$ is because the the size of msk will end up being $\sum_{i=1}^{r} 2^i \binom{\ell}{i}$ elements. We provide more details on this in the technical overview that follows. By a corollary of Stirling's approximation, we have that $\binom{\ell}{i} \geq \ell^i / i!$. For security, we require that $\ell = \mathsf{poly}(\lambda)$, and thus $r = O(1)$ is necessary to ensure polynomial-time efficiency.

For $v \in \{0, 1, *\}^\ell$ and $x \in \{0, 1\}^\ell$, recall that the bit-fixing predicate $\mathsf{P}_v(x)$ outputs 1, iff $(x_i \overset{?}{=} v_i) \vee (v_i \overset{?}{=} *)$ for each $i \in [\ell]$; otherwise it outputs 0. We will use Γ_v to denote the set:

$$\Gamma_v = \left\{ (\boldsymbol{t}, \boldsymbol{b}) \;\middle|\; \begin{array}{l} \boldsymbol{t} \in \mathbb{T}, \\ z \leftarrow \mathsf{unique}(\boldsymbol{t}), \\ \boldsymbol{b} \in \{0,1\}^z, \\ v_{\boldsymbol{t}} \leftarrow \mathsf{reindex}(v, \boldsymbol{t}), \\ 1 \leftarrow \mathsf{P}_{v_{\boldsymbol{t}}}(\boldsymbol{b}). \end{array} \right\},$$

for $v \in \{0, 1, *\}^\ell$. We require that, for the set of r key constraint queries $\{v^{(\iota)}\}_{\iota \in [r]}$ made by the distinguishing adversary in $\mathsf{exp}_{b,\mathcal{D}}^{\mathsf{cprf}}(1^\lambda, 1^r)$, then $\exists$ a pair such that $(\boldsymbol{t}, \boldsymbol{b}) \notin \Gamma_{v^{(\iota)}}$, for each $\iota \in [r]$. Otherwise, there would be no valid constrained challenge point with which we could evaluate the pseudorandomness of the CPRF.

In addition, there must exist a vector $\boldsymbol{t}^\dagger$ such that the adversarial challenge input point $x^\dagger$ satisfies $(\boldsymbol{t}^\dagger, {x^\dagger}_{\boldsymbol{t}^\dagger}) \notin \Gamma_{v^{(\iota)}}$, for all $\iota \in [r]$ and $x_{\boldsymbol{t}^\dagger} \leftarrow \mathsf{reindex}(x^\dagger, \boldsymbol{t}^\dagger)$. If this was not satisfied, then the

[4] Where the unique entries are $(t_1, t_3, t_4, t_5, t_7)$.

challenge point would not be constrained since one of the queries to cprf.Constrain must be of the form $*^\ell$ (if no such $\boldsymbol{t}^\dagger$ exists). An example of a satisfactory string $x^\dagger$ is:

$$x^\dagger = (1 - v^{(1)}_{t^\dagger_1})(1 - v^{(2)}_{t^\dagger_2}) \cdots (1 - v^{(r)}_{t^\dagger_r}).$$

Note that we can reorder the queries $v^{(\iota)}$ wlog so that $x^\dagger \notin \Gamma_{v^{(\iota)}}$ for each $\iota \in [r]$.

V;3.2 Technical overview

To aid the understanding of our scheme, we first give an intuitive overview of the techniques that we use.

Lattice-based constructions

The idea for this work originates from the lattice-based CPRF for bit-fixing constraints of Canetti and Chen [76]. In this work, the adversary is allowed to query for one constrained key that is chosen selectively (rather than adaptively). The PRF is defined over an input $x \in \{0,1\}^\ell$, the master secret key is a set of Gaussian-distributed matrices $\{\boldsymbol{D}_{i,b}\}_{i \in [\ell], b \in \{0,1\}}$. These matrices are thought of as representatives of LWE secrets, the underlying technique is borrowed from the PRFs of [25, 53]. The constrained key is then some $v \in \{0,1,*\}^\ell$, where $\boldsymbol{D}_{i,v_i}$ is revealed if $v_i \in \{0,1\}$, and both $\{\boldsymbol{D}_i\}_{b \in \{0,1\}}$ are revealed if $v_i = *$ for each $i \in [\ell]$. Finally, newly sampled $\overline{\boldsymbol{D}_{i,1-v_i}} \leftarrow\!\!\$\; D_{\mathbb{Z}^{m \times m},\sigma}$ replace the matrices that are not learnt.[5] In the public parameters there is a matrix $\boldsymbol{A}$, and for an input x, the PRF evaluation is the rounded product $\boldsymbol{A} \cdot \prod_{i=1}^{\ell} \boldsymbol{D}_{i,x_i}$ wrt to some appropriate choice of integer $p > 0$.

The key observation of [76] is that pseudorandomness only has to hold for some challenge $x^\dagger$ where $(x^\dagger_j \neq v_j) \wedge (v_j \neq *)$ for $j \in [\ell]$. By definition, such a point must exist, otherwise $x^\dagger$ must be unconstrained. Then, when the PRF is evaluated at $x^\dagger$, the output includes the matrix $\boldsymbol{D}_{j,x^\dagger_j}$ that is not revealed in the constrained key (and thus to the adversary). As a result, their security proof relies on an LWE security reduction, where $\boldsymbol{D}_{j,x^\dagger_j}$ ultimately acts as an unknown LWE secret. It is also noted by [76] that a very similar argument can be used to show that the PRF of [53] is also a PCPRF for bit-fixing constraints. For circuit-based constraints, this proof technique does not apply since the matrices are no longer tied explicitly to one bit of the constraint query. In these cases, a more careful LWE argument is used in relation to the secret distribution that is considered.

Unfortunately, the analysis for bit-fixing does not follow for more than one key. It is entirely possible to choose two constrained keys that would reveal the entire set $\{\boldsymbol{D}_{i,b}\}_{i \in [\ell], b \in \{0,1\}}$, without

[5]This is not required for standard CPRF security, but only for the extra privacy property.

compromising all evaluation points.[6] Therefore, the LWE argument cannot be used since all the secrets are effectively made public. The main issue of this technique is that one bit of the PRF input is tied concretely to one bit of the master secret key. Consequently, when valid constraints reveal both components of the master secret key for each bit, security is effectively lost.

Our technique

To improve on the functionality of previous schemes, we design a CPRF construction that analyses r input bits at a time, for $r \geq 1$. This change is central to the collusion-resistant design that we achieve. We also depart from the lattice constructions above, effectively replacing the matrices above with keys to an underlying prf. It becomes clear that we require $r = O(1)$ later, as the size of the master secret key is $O(\mathsf{poly}(\lambda)^r)$.

Key generation. To be more precise, let prf be an underlying pseudorandom function, where $\mathsf{prf.Eval} : \{0,1\}^\lambda \times \{0,1\}^\ell \mapsto \{0,1\}^\nu$. Let $\boldsymbol{t} = (t_1, \ldots, t_r)$ be an ordered vector (i.e. $t_1 \leq t_2 \leq \ldots \leq t_r$) of r values taken from $[\ell]$, that is $\boldsymbol{t} \in [\ell]^r$. We associate the set $\mathbb{T}$ with all vectors $\boldsymbol{t}$ that satisfy this property. As before, let $z \leftarrow \mathsf{unique}(\boldsymbol{t})$ denote the number of unique entries in $\boldsymbol{t}$.

Before we progress, we reiterate that we compute $\mathbb{T} \leftarrow \mathsf{dedup}(\mathbb{T})$ to remove all vectors that are equivalent after considering only the unique entries in each $\boldsymbol{t} \in \mathbb{T}$. We see below that if we did not do this then we would compute $\mathsf{prf.Eval}$ on the same inputs multiple times. While the CPRF would still run in polynomial time, it results in a much less efficient construction. As a consequence, we perform this deduplication to arrive at CPRF with better computational complexity. When we deduplicate, only the vector with the highest lexicographic ordering is retained for each possible vector of unique entries.

The 'functional' master secret key is generated by running the following loop.

- For $\boldsymbol{t} \in \mathbb{T}$ and each $w \in \{0,1\}^z$ (where $z \leftarrow \mathsf{unique}(\boldsymbol{t})$: sample $K_{\boldsymbol{t},w} \leftarrow_\$ \mathsf{prf.Setup}(1^\lambda)$.

Clearly, we require that $r = O(1)$, otherwise this loop would run in super-polynomial time with respect to λ. Finally, let $\mathsf{msk} = \{K_{\boldsymbol{t},w}\}_{\boldsymbol{t} \in \mathbb{T}, w \in \{0,1\}^z}$. Moreover, since $\boldsymbol{t}$ is ordered, then $|\mathsf{msk}| = \sum_{i=1}^{r} 2^i \cdot \binom{\ell}{i}$.

We also run a separate invocation of the above to generate a 'dummy' set of PRF keys. That is, for $\boldsymbol{t} \in \mathbb{T}$ and $w \in \{0,1\}^z$: sample $\overline{K_{\boldsymbol{t},w}} \leftarrow_\$ \mathsf{prf.Setup}(1^\lambda)$. Let $\overline{\mathsf{msk}} = \{\overline{K_{\boldsymbol{t},w}}\}_{\boldsymbol{t} \in \mathbb{T}, w \in \{0,1\}^z}$.

[6]For example, choosing the constraints $v_1 = 1 * * * 1$ and $v_2 = 0 * * * 0$; where $x = 1 * * * 0$ is still constrained.

The entire master secret key is then given by the pair $(\mathsf{msk}, \overline{\mathsf{msk}})$. The need for the dummy keys becomes apparent soon, but essentially this is to allow us to satisfy 1-key privacy, when answering constraint queries.

Evaluation. To evaluate the CPRF on $x \in \{0,1\}^\ell$, then do the following.

- For each $\boldsymbol{t} \in \mathbb{T}$, let $x_{\boldsymbol{t}}$ be the string $x_{\boldsymbol{t}} \leftarrow \mathsf{reindex}(x, \boldsymbol{t})$ and $z \leftarrow \mathsf{unique}(\boldsymbol{t})$.
- Compute the XOR: $y \leftarrow \bigoplus_{\boldsymbol{t} \in \mathbb{T}} \mathsf{prf.Eval}(K_{\boldsymbol{t}, x_{\boldsymbol{t}}}, x)$.
- Output y.

In other words, evaluation requires $\sum_{i=1}^{r} 2^i \binom{\ell}{i}$ evaluations of the underlying PRF. Recall that $r = O(1)$ and so it also runs in polynomial time.

Constraining. To constrain the function we analyse bit-fixing predicates implied by strings $v \in \{0,1,*\}^\ell$. Here, we use the notation that $*$ is a wildcard character, and we say that $\mathsf{P}_v(\cdot)$ is a bit-fixing predicate with respect to v, if $1 \leftarrow \mathsf{P}_v(x)$ if and only if:

$$\bigwedge_{i=1}^{\ell} \left(\left(v_i \stackrel{?}{=} x_i \right) \bigvee \left(v_i \stackrel{?}{=} * \right) \right) = 1,$$

for $x \in \{0,1\}^\ell$.

Now let $v_{\boldsymbol{t}} \leftarrow \mathsf{reindex}(v, \boldsymbol{t})$ be defined as before, for a vector $\boldsymbol{t} = (t_1, \ldots, t_r) \in \mathbb{T}$. Then to produce a constrained key, CK^v for our CPRF we analyse each string $v_{\boldsymbol{t}}$ individually against all possible $w \in \{0,1\}^z$.

Recall that the definition of a CPRF states that evaluation should remain the same using CK_v, if $\mathsf{P}_v(x) = 1$. Before we describe constraining for general parameter settings, it may be helpful to consider a small example where $\ell = \nu = 8, r = 4, v = 100{*}{*}1{*}0$ and let $\boldsymbol{t} = (3,4,4,6)$. Then $v_{\boldsymbol{t}} = 0 * 1 \in \{0,1,*\}^r$ and so we need to be able to evaluate the CPRF on inputs $x \in \{0,1\}^\ell$ where $x_{\boldsymbol{t}} = 001$ or $x_{\boldsymbol{t}} = 011$.[7] To achieve this, we can let $\widetilde{K}^v_{\boldsymbol{t},001} = K_{\boldsymbol{t},001} \in \mathsf{msk}$ and $\widetilde{K}^v_{\boldsymbol{t},011} = K_{\boldsymbol{t},011} \in \mathsf{msk}$. However, we do not have to reveal any keys where $\mathsf{P}_{v_{\boldsymbol{t}}}(x_{\boldsymbol{t}}) = 0$, for example if $x_{\boldsymbol{t}} = 100$. In these occurrences, we let $\widetilde{K}^v_{\boldsymbol{t},w} = \overline{K_{\boldsymbol{t},w}} \in \overline{\mathsf{msk}}$. Finally, we let:

$$\mathsf{CK}_v = \left(\{\widetilde{K}_{\boldsymbol{t},001}\} \cup \{\widetilde{K}_{\boldsymbol{t},011}\} \cup \{\widetilde{K}_{\boldsymbol{t},w}\}_{w \notin \{001,011\}} \right)_{\boldsymbol{t} \in \mathbb{T}},$$

denote the entire constrained key.

In general, we define the constraining algorithm ($\mathsf{cprf.Constrain}$) to do the following.

[7]Note that we ignore the index at $t_3 = 4$ since this is a non-unique entry of $\boldsymbol{t}$ (where $t_3 = t_2$). This is essentially re-explaining the $\mathsf{reindex}()$ algorithm.

- For all $\boldsymbol{t} = (t_1, \dots, t_r) \in \mathbb{T}$: let $v_{\boldsymbol{t}} \leftarrow \mathsf{reindex}(v, \boldsymbol{t})$.
- For $w \in \{0,1\}^z$, where $z \leftarrow \mathsf{unique}(\boldsymbol{t})$, then if $\mathsf{P}_{v_{\boldsymbol{t}}}(w) = 1$: let $\widetilde{K}_{\boldsymbol{t},w} = K_{\boldsymbol{t},w}$.
- For $w \in \{0,1\}^z$, if $\mathsf{P}_{v_{\boldsymbol{t}}}(w) = 0$: let $\widetilde{K}_{\boldsymbol{t},w} = \overline{K_{\boldsymbol{t},w}}$.
- Output $\mathsf{CK}_v = \left(\widetilde{K}_{\boldsymbol{t},w}\right)_{\boldsymbol{t} \in \mathbb{T}, w \in \{0,1\}^z}$.

Constrained evaluation. An interesting property of our scheme is that constrained keys are essentially distributed the same as master secret keys where the dummy portion of the key is removed (i.e. $(\mathsf{msk}, \emptyset)$). This is due to the fact that all the underlying set elements are just keys sampled from the keyspace of prf. Therefore, the structural form of the constrained evaluation is identical to the real evaluation algorithm.

Proving security

The proof of security requires a number of different considerations for initially proving security in the selective sense; subsequently achieving adaptive security; and finally realising 1-key privacy.

Constrained pseudorandomness. The main goal of our work is to prove that our construction satisfies pseudorandomness on 'constrained points', after receiving $r = O(1)$ constrained keys. Let $v^{(1)}, \dots, v^{(r)}$ be the bit-fixing strings that the constrained keys are defined with respect to.

Our goal is to essentially show that there exists a $\boldsymbol{t}^\dagger = (t_1^\dagger, \dots, t_r^\dagger)$ such that $v^{(i)}_{t_i^\dagger} \neq *$, for each $i \in [r]$.[8] If this was not the case, then this implies that there exists $i \in [r]$ such that $v^{(i)} = ** \cdots *$ (i.e. the all wildcard string). If this was true, then any eventual challenge input point $x^\dagger$ would be unconstrained, and thus the security game is void.

For each $v^{(i)}$, let $\mathsf{CK}_{v^{(i)}} = \left(\widetilde{K}^{(i)}_{\boldsymbol{t},w}\right)_{\boldsymbol{t} \in \mathbb{T}, w \in \{0,1\}^z}$. Now consider the string:

$$w^\dagger = (1 - v^{(1)}_{t_1^\dagger},\ 1 - v^{(2)}_{t_2^\dagger},\ \cdots,\ 1 - v^{(r)}_{t_r^\dagger}),$$

and write $w^\dagger = (w_1^\dagger, \dots, w_r^\dagger)$, where each $v^{(i)}_{t_i^\dagger} \in \{0,1\}$. Compute $w^\dagger_{\boldsymbol{t}^\dagger} \leftarrow \mathsf{reindex}(w^\dagger, \boldsymbol{t}^\dagger)$; so that $w^\dagger_{\boldsymbol{t}^\dagger} \in \{0,1\}^z$. Then $\mathsf{P}_{v^{(i)\dagger}_{\boldsymbol{t}^\dagger}}(w^\dagger_{\boldsymbol{t}^\dagger}) = 0$ for all $i \in [r]$, and thus $\widetilde{K}^{(i)}_{\boldsymbol{t}^\dagger, w^\dagger} = \overline{K_{\boldsymbol{t}^\dagger, w^\dagger}}$. Therefore, the 'functional' key $K_{\boldsymbol{t}^\dagger, w^\dagger_{\boldsymbol{t}^\dagger}} \in \mathsf{msk}$ is never revealed by a constrained key query. This is the heart of our security proof.

[8] We can reorder the constraint queries so that this is satisfied.

Our security reduction uses the fact that constrained evaluations can be handled by an adversary that has access to an oracle for the underlying prf at a randomly chosen key $K^\dagger$. The adversary simulates the master secret key in its entirety, apart from for the key $K_{\boldsymbol{t}^\dagger, w^\dagger}$. Any input query x that uses this key can be answered by submitting x as an oracle query to the function $\mathsf{prf.Eval}(K^\dagger, x)$, where $K_{\boldsymbol{t}^\dagger, w^\dagger} = K^\dagger$ is implicitly chosen.

For the challenge constrained input $x^\dagger$, we require that $K_{\boldsymbol{t}^\dagger, x^\dagger_{\boldsymbol{t}}} = K^\dagger$ — this is why the selectivity of queries seems important initially. The adversary submits $x^\dagger$ to its challenge oracle and receives back $y^\dagger \leftarrow \mathsf{prf.Eval}(K^\dagger, x^\dagger)$ or $y^\dagger \leftarrow f(x^\dagger)$ for a uniformly sampled function $f : \{0,1\}^\ell \mapsto \{0,1\}^\nu$.[9]

The fact that the value $y \in \{0,1\}^\nu$ output by the CPRF is pseudorandom is inferred via the fact that $y^\dagger$ is XORed into y, and $y^\dagger$ is a pseudorandom output dependent solely on $x^\dagger$ (by the security of prf). See Theorem V;4.2 for the full proof of security, where Lemma V;4.1 focuses on the case of selective security.

ADAPTIVE SECURITY. Essentially, our construction arrives at adaptive security *for free*. Previous constructions incur sub-exponential security losses during the reduction from adaptive to selective security. Essentially all constructions use the technique where the adaptive adversary attempts to guess the challenge point $x^\dagger$ that the selective adversary uses. We are able to achieve adaptive security with a polynomial security loss (e.g. $1/\mathsf{poly}(\lambda)$): by simply guessing the key that is implicitly used by the adversary (i.e. $K_{\boldsymbol{t}^\dagger, w^\dagger}$). If this key is not eventually used by the challenge ciphertext, or it is revealed via a constrained key query, then the reduction aborts. This is because the entire proof hinges on the choice of this key, rather than the input itself. Since there are polynomially many keys (for $r = O(1)$), we can achieve adaptive security with only a $1/\mathsf{poly}(\lambda)$ probability of aborting. We handle this non-trivial abort case in the proof of Lemma V;4.2.

1-KEY PRIVACY. We obtain perfect weak 1-key privacy by returning to our observation about constrained evaluations. All constrained keys are essentially made up of PRF keys sampled from $\mathsf{prf.Setup}(1^\lambda)$. Therefore, if we refer to the indistinguishability security game for *weak* key privacy of Boneh et al. [55], then for any two constraint queries submitted by an adversary, the resulting key is distributed in exactly the same way in either case. See the proof of Lemma V;4.3 for more details. Targeting weak key privacy does not reduce the applications of the PCPRF that we construct [55, 76]

We cannot obtain key privacy for more than one query because two constrained keys would reveal where the constrained keys differ, since the underlying 'functional' keys have to be consistent across constrained keys. Therefore, receiving more than one constrained key, it will become obvious which keys correspond to which constraint. For a similar reason, we cannot obtain simulation-

[9]Note that we require a version of PRF security where challenge points and standard evaluation queries are distinct in the sense that evaluation queries are always answered by the actual PRF.

based security [76], because we would have to permit $O(1)$ constrained key queries for the pseudorandomness requirement and this would break the 1-key privacy of our scheme. Achieving strong key privacy also seems difficult in the adaptive security model, due to the presence of the evaluation queries.

V;3.3 Actual construction

Following the intuition in the previous section, we now describe our CPRF formally in Construction V;3.1. Recall the definition of $\mathbb{T}$; assume that $\mathbb{T} \leftarrow \mathsf{dedup}(\mathbb{T})$ before each of the CPRF algorithms is ran. Moreover, recall that Γ_v is defined as:

$$\Gamma_v = \left\{ (\boldsymbol{t}, \boldsymbol{b}) \;\middle|\; \begin{array}{l} \boldsymbol{t}{\in}\mathbb{T}, \\ z{\leftarrow}\mathsf{unique}(\boldsymbol{t}), \\ \boldsymbol{b}{\in}\{0,1\}^z, \\ v_{\boldsymbol{t}}{\leftarrow}\mathsf{reindex}(v,\boldsymbol{t}), \\ 1{\leftarrow}\mathsf{P}_{v_{\boldsymbol{t}}}(\boldsymbol{b}). \end{array} \right\}.$$

Construction V;3.1 [Our PCPRF]

Let λ be the security parameter, $r \in \mathbb{Z}$ be some positive constant, and let prf be a PRF (Definition II;4.2). Let $\mathcal{X} \in \{0,1\}^\ell$ be the input space, $\mathcal{Y} \in \{0,1\}^\mu$ be the output space of prf, and let $\mathcal{C}$ be the class of BitFix constraints $v \in \{0,1,*\}^\ell$.
Our construction, cprf, is as follows:

- $\mathsf{msk} \leftarrow \mathsf{cprf.Setup}(1^\lambda, 1^r)$: [Figure V;3];
- $y \leftarrow \mathsf{cprf.Eval}(\mathsf{msk}, x \in \mathcal{X})$: [Figure V;4];
- $\mathsf{CK}_v \leftarrow \mathsf{cprf.Constrain}(\mathsf{msk}, v \in \mathcal{C})$: [Figure V;5];
- $y \leftarrow \mathsf{cprf.CEval}(\mathsf{CK}_v, x \in \mathcal{X})$: [Figure V;6].

Remark V;3.1. *Note that constrained evaluation, on unconstrained inputs, is identical to the real evaluation algorithm using* msk. *Consequently, we may write* $\mathsf{cprf.Eval}(\mathsf{pp}, \mathsf{CK}_v, x)$ *for evaluation on some constrained key* $\mathsf{CK}_v \leftarrow \mathsf{cprf.Constrain}(\mathsf{pp}, \mathsf{msk}, v)$.

```
cprf.Setup($1^\lambda, 1^r$):
  • For each $\boldsymbol{t} \leftarrow \mathbb{T}$:
      – Let $z \leftarrow \mathsf{unique}(\boldsymbol{t})$;
      – For each $\boldsymbol{b} \in \{0,1\}^z$:
          * Sample keys $K_{\boldsymbol{t},\boldsymbol{b}}, \overline{K_{\boldsymbol{t},\boldsymbol{b}}} \leftarrow_{\$} (\mathsf{prf.Setup}(1^\lambda))^2$;
          * Let $\mathsf{msk}[\boldsymbol{t},\boldsymbol{b}] = K_{\boldsymbol{t},\boldsymbol{b}}$;
          * Let $\overline{\mathsf{msk}}[\boldsymbol{t},\boldsymbol{b}] = \overline{K_{\boldsymbol{t},\boldsymbol{b}}}$;
  • Output $(\emptyset, (\mathsf{msk}, \overline{\mathsf{msk}}))$.
```

Figure V;3: Setup algorithm for cprf. Note that the set $\{0,1\}^z$ can be iterated over, since $z \leq r = O(1)$. Since $\mathsf{pp} = \emptyset$ we omit further mention of it.

```
cprf.Eval($(\mathsf{msk}, \overline{\mathsf{msk}}), x \in \{0,1\}^\ell$):
  • For each $\boldsymbol{t} \leftarrow \mathbb{T}$:
      – Let $x_{\boldsymbol{t}} \leftarrow \mathsf{reindex}(x, \boldsymbol{t})$;
      – Let $K_{\boldsymbol{t},x_{\boldsymbol{t}}} \leftarrow \mathsf{msk}[\boldsymbol{t}, x_{\boldsymbol{t}}]$;
      – Compute $y_{\boldsymbol{t}}^x \leftarrow \mathsf{prf.Eval}(K_{\boldsymbol{t},x_{\boldsymbol{t}}}, x)$;
  • Output $y^x = \bigoplus_{\boldsymbol{t} \in \mathbb{T}} y_{\boldsymbol{t}}^x$.
```

Figure V;4: Evaluation algorithm for cprf.

V;4 Security proof

In this section, we provide the necessary proofs of correctness and security for our CPRF. To summarise, we show that our construction is a CPRF for $r = O(1)$ constraint queries in the adaptive security model. We also show that our construction satisfies 1-key privacy.

Theorem V;4.1 [Correctness]

Construction V;3.1 is perfectly *correct*.

Proof. Firstly, we know that each algorithm runs in polynomial-time since there are $\sum_{i=1}^{r} \binom{\ell}{i}$ vectors in $\mathbb{T}$, and for each of these vectors there are 2^i possible elements (or PRF keys) to operate over. Since $r = O(1)$, then clearly $2^i \cdot \binom{\ell}{i}$ is $\mathsf{poly}(\lambda)$ for all $i \in [r]$. Thus the sum of all r of these is also polynomial.

```
$\mathsf{cprf.Constrain}((\mathsf{msk}, \overline{\mathsf{msk}}), v \in \{0,1,*\}^\ell)$:
  • For each $\boldsymbol{t} \leftarrow \mathbb{T}$:
      – Let $z \leftarrow \mathsf{unique}(\boldsymbol{t})$;
      – For each $\boldsymbol{b} \leftarrow \{0,1\}^z$:
          -- Let $G^{(v)}_{\boldsymbol{t},\boldsymbol{b}} \leftarrow \emptyset$
          -- If $(\boldsymbol{t}, \boldsymbol{b}) \in \Gamma_v$:
                 $\mathsf{CK}_v[\boldsymbol{t}, \boldsymbol{b}] \leftarrow \mathsf{msk}[\boldsymbol{t}, \boldsymbol{b}]$;
          -- Else if $(\boldsymbol{t}, \boldsymbol{b}) \notin \Gamma_v$:
                 $\mathsf{CK}_v[\boldsymbol{t}, \boldsymbol{b}] \leftarrow \overline{\mathsf{msk}}[\boldsymbol{t}, \boldsymbol{b}]$;
  • Output $\mathsf{CK}_v$.
```

Figure V;5: Constraining algorithm for cprf.

```
$\mathsf{cprf.CEval}(\mathsf{CK}_v, x)$:
  • Output $\mathsf{cprf.Eval}((\mathsf{CK}_v, \emptyset), x)$.
```

Figure V;6: Constrained evaluation algorithm for cprf. Since CK_v and msk are perfectly indistinguishable (by Lemma V;4.3), we can just use CK_v as input to the $\mathsf{cprf.Eval}$ algorithm.

Let CK_v be some constrained key for $v \in \{0,1,*\}^\ell$ and let $x \in \{0,1\}^\ell$ be such that $\mathsf{P}_v(x) = 1$. Additionally let $\boldsymbol{t} \in \mathbb{T}$, $x_{\boldsymbol{t}} \leftarrow \mathsf{reindex}(x, \boldsymbol{t})$ and let $\widetilde{K_{\boldsymbol{t},x_{\boldsymbol{t}}}} = \mathsf{CK}_v[\boldsymbol{t}, x_{\boldsymbol{t}}]$ and $K_{\boldsymbol{t},x_{\boldsymbol{t}}} \leftarrow \mathsf{msk}[\boldsymbol{t}, x_{\boldsymbol{t}}]$. Then:

$$\begin{aligned}
\mathsf{cprf.CEval}(\mathsf{CK}_v, x) &= \bigoplus_{\boldsymbol{t}\in\mathbb{T}} y^x_{\boldsymbol{t}} = \bigoplus_{\boldsymbol{t}\in\mathbb{T}} \mathsf{prf.Eval}(\widetilde{K_{\boldsymbol{t},x_{\boldsymbol{t}}}}, x); \\
&= \bigoplus_{\boldsymbol{t}\in\mathbb{T}} \mathsf{prf.Eval}(K_{\boldsymbol{t},x_{\boldsymbol{t}}}, x); \\
&= \mathsf{cprf.Eval}(\mathsf{msk}, x).
\end{aligned}$$

The third equality follows since x is unconstrained and thus $\widetilde{K_{\boldsymbol{t},x_{\boldsymbol{t}}}} = K_{\boldsymbol{t},x_{\boldsymbol{t}}} \in \mathsf{msk}$. Therefore, $(\boldsymbol{t}, x_{\boldsymbol{t}}) \in \Gamma_v$ for all $\boldsymbol{t} \in \mathbb{T}$, and so $\widetilde{K_{\boldsymbol{t},x_{\boldsymbol{t}}}} \not\leftarrow \overline{\mathsf{msk}}$. We conclude that $\mathsf{CK}_v[\boldsymbol{t}, x_{\boldsymbol{t}}] \in \mathsf{msk}$, and the evaluation methods are identical. □

Theorem V;4.2 [PCPRF]

Construction V;3.1 is an r-key adaptively secure CPRF satisfying perfect weak 1-key privacy, assuming the existence of one-way functions, for $r = O(1)$ constraint queries and $\mathsf{poly}(\lambda)$ evaluation queries.

Proof. The proof of this theorem follows from the proofs of Lemma V;4.1, Lemma V;4.2 and Lemma V;4.3. The first two lemmas considers the pseudorandomness of constrained evaluations, the latter targets 1-key privacy.

First, we make some cosmetic modifications to the CPRF security game that allow us to consider bit-fixing predicates rather than general circuit-based predicates. That is, we use the oracle $\mathcal{O}^{\theta}_{\{0,1,*\}^{\ell}}(v)$ that outputs a constrained key CK_v for the bit-fixing string $v \in \{0,1,*\}^{\ell}$. We reiterate that the constrained key can be used to evaluate the PRF on an input x iff $1 \leftarrow \mathsf{P}_v(x)$.

We give a security reduction from the pseudorandomness of prf (Definition II;4.2) to the security of our scheme. We initially prove Lemma V;4.1, showing that our scheme is secure when the adversary queries are chosen selectively in the CPRF and PRF security games. We then give a reduction from the adaptive case to the case of an adaptively secure PRF incurring a polynomial security loss in Lemma V;4.2. To achieve this, we use the same proof technique as in the selective case, but introduce some extra abort steps. In theory, we could present the proof of adaptivity (Lemma V;4.2) independently of the selective security proof of selective security. However, presenting them separately helps to give better intuition when presenting our proof technique.

Let $v^{(1)}, v^{(2)}, \dots, v^{(r)} \in \{0,1,*\}^{\ell}$ denote the key constraint queries, and let $x^{\dagger} \in \{0,1\}^{\ell}$ denote the challenge input query made by $\mathcal{A}$. Additionally, let $x^{\dagger}_{\boldsymbol{t}^{\dagger}} \leftarrow \mathsf{reindex}(x^{\dagger}, \boldsymbol{t}^{\dagger})$ and $z \leftarrow \mathsf{unique}(\boldsymbol{t}^{\dagger})$. Since the query $x^{\dagger}$ is constrained, then necessarily $\mathsf{P}_{v^{(i)}}(x^{\dagger}) = 0$ for each $i \in [r]$. In the following claims, we consider a specific choice of $\boldsymbol{t}^{\dagger} \leftarrow \mathbb{T}$ where $(v^{(i)}_{t^{\dagger}_i} \neq *)$ for $i \in [r]$. There is at least one such $\boldsymbol{t}^{\dagger} \in \mathbb{T}$ for any r constraint queries, corresponding to the constrained input $x^{\dagger}$. Otherwise $x^{\dagger}$ would be unconstrained for at least one of $v^{(i)}$. In other words, then we can be sure that $(\boldsymbol{t}^{\dagger}, x^{\dagger}_{\boldsymbol{t}^{\dagger}}) \notin \Gamma_{v^{(i)}}$ for $i \in [r]$, by choosing:

$$x^{\dagger} = (1 - v^{(1)}_{t^{\dagger}_1}, 1 - v^{(2)}_{t^{\dagger}_2}, \cdots, 1 - v^{(r)}_{t^{\dagger}_r}).$$

Note that we reorder the constrained key queries, without loss of generality, so that $t_1 \leq t_2 \leq \ldots \leq t_r$.

Lemma V;4.1 [Selective pseudorandomness]

Let Λ denote Construction V;3.1. Then we have that

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{cprf}(1^{\lambda}))) < \max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{B}, \mathsf{prf}(1^{\lambda}))),$$

where: $\mathcal{A}, \mathcal{B}$ are PPT distinguishing algorithms for $b, c \in \{0,1\}$; $\mathcal{B}$ makes all queries in the prf security game selectively; and $\mathcal{A}$ makes at most $r = O(1)$ (selective) constraint queries, and $\mathsf{poly}(\lambda)$ (selective) evaluation queries.

Proof. Let $\mathcal{B}$ be a PPT distinguishing adversary that is interacting with the experiment $\mathsf{exp}^{\mathsf{prf}}_{c,\mathcal{B}}(1^\lambda)$, for $c \in \{0,1\}$ and for a pseudorandom function prf. Recall from Definition II;4.2 that in both experiments (for $c \in \{0,1\}$) the adversary has access to an evaluation oracle that takes $\mathsf{poly}(\lambda)$ evaluation queries $x \in \mathcal{X}$ from $\mathcal{B}$, and returning PRF evaluations on x with respect to a uniformly random choice of key $K^\dagger \in \mathcal{K}$. Denote the evaluation oracle in $\mathsf{exp}^{\mathsf{prf}}_{c,\mathcal{B}}(1^\lambda)$ by $\mathcal{O}^{\mathsf{prf}}_{\mathcal{X}}(\cdot)$. The difference between the two experiments is related to a specifically chosen challenge point $x^\dagger \leftarrow \mathcal{B}$: when $c = 0$, $\mathcal{B}$ receives $y^{\mathsf{prf}} \leftarrow \mathsf{prf.Eval}(K, x^\dagger)$; when $c = 1$, $\mathcal{B}$ receives $y^{\mathsf{prf}} \leftarrow_{\$} \mathcal{Y}$, where $\mathcal{Y}$ is the output space of prf.

Let $\mathcal{A}$ be a PPT distinguishing adversary interacting with $\mathsf{exp}^{\mathsf{cprf}}_{b,\mathcal{A}}(1^\lambda, 1^r)$, for $b \in \{0,1\}$, in the selective query model. We show that:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{cprf}(1^\lambda))) < \max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{B}, \mathsf{prf}(1^\lambda)))$$

and thus, the advantage of any $\mathcal{A}$ is bounded by a negligible function by Definition II;4.2.

Since all key queries submitted by $\mathcal{A}$ are chosen selectively, then we assume these are received in $\mathsf{exp}^{\mathsf{cprf}}_{b,\mathcal{A}}(1^\lambda, 1^r)$ in the first step and they are answered after setup is complete. The algorithm $\mathcal{B}$ simulates $\mathsf{exp}^{\mathsf{cprf}}_{b,\mathcal{A}}(1^\lambda, 1^r)$ for $\mathcal{A}$ as defined in Simulation V;4.1.

Simulation V;4.1 [prf adversary]

- $(\mathsf{msk}, \overline{\mathsf{msk}}) \leftarrow \Lambda.\mathsf{Setup}(1^\lambda, 1^r, 1^\ell)$ is carried out as normal.
- Constraint queries ($v^{(i)} \in \{0, 1, *\}^\ell$):
  - Return $\mathsf{CK}_{v^{(i)}} \leftarrow \Lambda.\mathsf{Constrain}((\mathsf{msk}, \overline{\mathsf{msk}}), v)$ to $\mathcal{A}$.
  - Construct the sets $\Gamma_{v^{(i)}}$ for each $i \in [r]$.
- Challenge query ($x^\dagger \in \{0, 1\}^\ell$):
  - If $\mathsf{P}_{v^{(i)}}(x^\dagger) = 1$ for some $i \in [r]$:
    - Return $\perp$.
  - Let $\boldsymbol{t}^\dagger \in \mathbb{T}$ be s.t. for $x^\dagger_{\boldsymbol{t}^\dagger} \leftarrow \mathsf{reindex}(x^\dagger, \boldsymbol{t}^\dagger)$, then $(\boldsymbol{t}^\dagger, x^\dagger_{\boldsymbol{t}^\dagger}) \notin \Gamma_{v^{(i)}}$ for all $i \in [r]$.
  - Computes $y^\dagger \leftarrow \Lambda.\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x^\dagger)$ as normal (Figure V;4), except that instead of using $y^\dagger_{\boldsymbol{t}^\dagger} \leftarrow \mathsf{prf}.\mathsf{Eval}(\mathsf{msk}[\boldsymbol{t}^\dagger, x^\dagger_{\boldsymbol{t}^\dagger}], x^\dagger)$, $\mathcal{B}$ does the following:
    - Submits $x^\dagger$ as the challenge input in $\mathsf{exp}^{\mathsf{prf}}_{c,\mathcal{B}}(1^\lambda)$, and receives $y^{\mathsf{prf}}_{\boldsymbol{t}^\dagger}$.
    - Sets $y^\dagger_{\boldsymbol{t}^\dagger} \leftarrow y^{\mathsf{prf}}_{\boldsymbol{t}^\dagger}$.
  - Return $y^\dagger$ to $\mathcal{A}$ as the challenge response.
- Input queries ($x \in \{0, 1\}^\ell$):
  - If $\mathsf{P}_{v^{(i)}}(x) = 1$ for some $i \in [r]$:
    - Returns $y \leftarrow \Lambda.\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x)$.
  - If $x = x^\dagger$: returns $\perp$.
  - If $\mathsf{P}_{v^{(i)}}(x) = 0$ for each $i \in [r]$:
    - Lets $x_{\boldsymbol{t}^\dagger} \leftarrow \mathsf{reindex}(x^\dagger, \boldsymbol{t}^\dagger)$, for the vector $\boldsymbol{t}^\dagger$ used in the response to the challenge query.
    - If $x_{\boldsymbol{t}^\dagger} \neq x^\dagger_{\boldsymbol{t}^\dagger}$: returns $y \leftarrow \Lambda.\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x)$.
    - Else: computes y as in $\Lambda.\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x)$, except that instead of using $\boldsymbol{y}_{\boldsymbol{t}^\dagger} \leftarrow \mathsf{prf}.\mathsf{Eval}(\mathsf{msk}[\boldsymbol{t}^\dagger, x_{\boldsymbol{t}^\dagger}], x)$, $\mathcal{B}$ does the following:
      - Submits x as an input query to $\mathcal{O}^{\mathsf{prf}}_{\mathcal{X}}(\cdot)$ and receives $y^{\mathsf{prf}}_{\boldsymbol{t}^\dagger}$.
      - Sets $y_{\boldsymbol{t}^\dagger} \leftarrow y^{\mathsf{prf}}_{\boldsymbol{t}^\dagger}$.
    - Return y to $\mathcal{A}$.

$[(c = 0) \implies (b = 0)]$: Firstly, we need to show that when $\mathcal{B}$ interacts with $\mathsf{exp}^{\mathsf{prf}}_{0,\mathcal{B}}(1^\lambda)$, then this perfectly simulates $\mathsf{exp}^{\mathsf{cprf}}_{0,\mathcal{A}}(1^\lambda, 1^r)$ for $\mathcal{A}$. The only difference between the simulations of

$\mathcal{B}$ and the original construction manifest themselves in the response to challenge and evaluation queries. Queries for constrained keys can be simulated just using the pair $(\mathsf{msk}, \overline{\mathsf{msk}})$. Therefore, we focus on the differences between the executions wrt the answers of the challenge and evaluation queries.

For *unconstrained* evaluation queries, $\mathcal{B}$ answers with the honest evaluation using the simulated msk. This secret key is distributed identically and so we can ignore these type of queries. Now, for the selectively-chosen challenge query $x^\dagger$, $\mathcal{B}$ uses the pair $(\boldsymbol{t}^\dagger, x^\dagger_{\boldsymbol{t}^\dagger}) \notin \Gamma_{v^{(i)}}$ in a different way. Recall that such a pair exists for all $i \in [r]$ since $x^\dagger$ is constrained with respect to each of the predicates $v^{(i)}$. For the output $y^\dagger$, $\mathcal{B}$ computes:

$$y^\dagger = y^{\mathsf{prf}}_{\boldsymbol{t}^\dagger} \oplus \left(\bigoplus_{\substack{\boldsymbol{t} \in \mathbb{T}, \\ \boldsymbol{t} \neq \boldsymbol{t}^\dagger}} \mathsf{prf.Eval}(\mathsf{msk}[\boldsymbol{t}^\dagger, x^\dagger_{\boldsymbol{t}^\dagger}], x^\dagger) \right), \tag{V;1}$$

where $y^{\mathsf{prf}}_{\boldsymbol{t}^\dagger} \leftarrow \mathsf{prf.Eval}(K^\dagger, x^\dagger)$ and $K^\dagger$ is the key that is chosen uniformly by the challenger in $\mathsf{exp}^{\mathsf{prf}}_{0,\mathcal{B}}(1^\lambda)$. That is, $\mathcal{B}$ sets $x^\dagger$ as its challenge input query in $\mathsf{exp}^{\mathsf{prf}}_{0,\mathcal{B}}(1^\lambda)$ and obtains the PRF evaluation $y^{\mathsf{prf}}_{\boldsymbol{t}^\dagger}$. Note that this is equivalent to the construction of Λ, where $\mathsf{msk}[\boldsymbol{t}^\dagger, x^\dagger_{\boldsymbol{t}^\dagger}] = K^\dagger$.

For the selectively-chosen *constrained* evaluation queries $x \in \mathcal{X}$, if $(\boldsymbol{t}^\dagger, x_{\boldsymbol{t}^\dagger}) \neq (\boldsymbol{t}^\dagger, x^\dagger_{\boldsymbol{t}^\dagger})$ then the queries are answered using the pair $(\mathsf{msk}, \overline{\mathsf{msk}})$ sampled by $\mathcal{B}$. Otherwise, if $(\boldsymbol{t}^\dagger, x_{\boldsymbol{t}^\dagger}) = (\boldsymbol{t}^\dagger, x^\dagger_{\boldsymbol{t}^\dagger})$, the input query results in an output that also uses the key at $\mathsf{msk}[\boldsymbol{t}^\dagger, x^\dagger_{\boldsymbol{t}^\dagger}]$. These queries are answered in a similar way to Equation (V;1):

$$y = y_{\boldsymbol{t}^\dagger} \oplus \left(\bigoplus_{\substack{\boldsymbol{t} \in \mathbb{T}, \\ \boldsymbol{t} \neq \boldsymbol{t}^\dagger}} \mathsf{prf.Eval}(\mathsf{msk}[\boldsymbol{t}^\dagger, x_{\boldsymbol{t}^\dagger}], x) \right), \tag{V;2}$$

where $y_{\boldsymbol{t}^\dagger} = \mathsf{prf.Eval}(K^\dagger, x) \leftarrow \mathcal{O}^{\mathsf{prf}}_{\mathcal{X}}(x)$. Note that this oracle uses the same PRF key as the challenge query, so this is still the same as the original Λ construction with $\mathsf{msk}[\boldsymbol{t}^\dagger, x^\dagger_{\boldsymbol{t}^\dagger}] = K^\dagger$.

As a result, $\mathcal{B}$ perfectly simulates $\mathsf{exp}^{\mathsf{cprf}}_{0,\mathcal{A}}(1^\lambda, 1^r)$ for a PPT algorithm $\mathcal{A}$, using access to the experiment $\mathsf{exp}^{\mathsf{prf}}_{0,\mathcal{B}}(1^\lambda)$.

$\underline{(c = 1) \implies (b = 1)}$: When $\mathcal{B}$ has access to $\mathsf{exp}^{\mathsf{prf}}_{1,\mathcal{B}}(1^\lambda)$, note that the answers to evaluation queries for unconstrained $x \in \mathcal{X}$ are the same. The answer $y^\dagger$ to the challenge input query takes the same form as in Equation (V;1), except that $y^{\mathsf{prf}}_{\boldsymbol{t}^\dagger} \leftarrow\!\!\$\ \mathcal{Y}$ is now sampled uniformly from the output space $\mathcal{Y}$ of prf. Therefore, the entire output $y^\dagger$ is distributed uniformly. Constrained evaluation queries are derived in the same way as the previous simulation, since $\mathcal{O}^{\mathsf{prf}}_{\mathcal{X}}(\cdot)$ always outputs real evaluations from prf regardless of the value of $c \in \{0, 1\}$.

Consequently, it is not hard to see that that $\mathcal{B}$ perfectly simulates $\mathsf{exp}^{\mathsf{cprf}}_{1,\mathcal{B}}(1^\lambda, 1^r)$ for $\mathcal{A}$, since $y^\dagger_{\mathbb{T}}$ is distributed uniformly and thus perfectly indistinguishable from the output of a uniform function. Moreover, evaluation queries are still answered by honest evaluations from the underlying prf.

Using the above two arguments we see that

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{cprf}(1^\lambda))) < \max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{B}, \mathsf{prf}(1^\lambda))) < \mathsf{negl}(\lambda).$$

In summary, we obtain the result of Lemma V;4.1 via a security reduction from the pseudorandomness of prf to the security of Λ. □

Lemma V;4.2 [Adaptive pseudorandomness]

Let Λ denote Construction V;3.1. Then the following inequality holds:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{cprf}(1^\lambda))) \leq \frac{1}{\mathsf{poly}(\lambda)} \max_{\mathcal{B}}(\mathsf{Adv}(\mathcal{B}, \mathsf{prf}(1^\lambda))),$$

where $\mathcal{A}$ is any PPT algorithm that makes all queries selectively, and $\mathcal{B}$ is any PPT algorithm making all queries adaptively in the prf security game.

Proof. The proof of this lemma essentially follows the same argument as in Lemma V;4.1 but where we provide extra abort steps in the proof and we use an adaptively-secure prf construction. All together this requires a hybrid argument.

An important part of the proof of Lemma V;4.1 is that the evaluation queries are chosen in the knowledge of the pair $(\boldsymbol{t}^\dagger, x^\dagger_{\boldsymbol{t}^\dagger})$ that is used for embedding the PRF evaluation from $\mathsf{exp}^{\mathsf{prf}}_{c,\mathcal{B}}(1^\lambda)$. As such, it seems that the challenge query needs to made selectively in order for the security proof to hold. Additionally, all other queries need to also be made selectively for this pair to be chosen. Fortunately, we show that the probability of the prf adversary, $\mathcal{B}$, aborting is actually $1/\mathsf{poly}(\lambda)$ and so the security loss introduced using this technique can be covered by a polynomial-time reduction.

Our hybrid argument has the following steps.

- H_0: This is the original CPRF security game, where $\mathcal{A}$ makes all queries adaptively.
- H_1: In this step, the PRF adversary $\mathcal{B}$ samples a random pair $(\boldsymbol{t}_{\mathcal{B}}, \boldsymbol{b}_{\mathcal{B}}) \leftarrow_\$ \mathbb{T} \times \{0,1\}^z$ where $z \leftarrow \mathsf{unique}(\boldsymbol{t}_{\mathcal{B}})$. Then $\mathcal{B}$ aborts *at the end of the game* if any of the following conditions are met:

  - for any bit-fixing vector $v^{(i)}$ chosen by $\mathcal{A}$ (for $i \in [r]$), the pair $t_{\mathcal{B}}, \boldsymbol{b}_{\mathcal{B}})$ satisfies $(\boldsymbol{t}_{\mathcal{B}}, \boldsymbol{b}_{\mathcal{B}}) \notin \Gamma_{v^{(i)}}$;
  - for the challenge query $x^{\dagger}$ chosen by $\mathcal{A}$, then $x^{\dagger}_{\boldsymbol{t}_{\mathcal{B}}} \leftarrow \mathsf{reindex}(x^{\dagger}, \boldsymbol{t}_{\mathcal{B}})$ satisfies $x^{\dagger}_{\boldsymbol{t}_{\mathcal{B}}} \neq \boldsymbol{b}_{\mathcal{B}}$;
  - if $(\boldsymbol{t}_{\mathcal{B}}, \boldsymbol{b}_{\mathcal{B}})$ chosen by $\mathcal{B}$ is not the *first pair* (with respect to some pre-defined order over $[\ell]^z \times \{0,1\}^z$ such as the lexicographic order) that does not violate the first two conditions. Note that it is possible to efficiently find such a pair by enumerating over $[\ell]^z \times \{0,1\}^z$ since $r = O(1)$.[10]

  When $\mathcal{B}$ aborts, it substitutes the guess output by $\mathcal{A}$ by a random bit.

- H_2: In this step, the adversary $\mathcal{B}$ aborts the security game immediately after any of the conditions above are violated.
- H_3: In this step, the adversary $\mathcal{B}$ answers queries from $\mathcal{A}$ in the manner defined in Simulation V;4.1.

It should be clear that, as long as the security loss in changing state between H_0 and H_2 can be kept to a polynomial function in the security parameter, then the transition from H_2 to H_3 can be bounded by the advantage required to bound the proof of Lemma V;4.1.

Claim V;4.2.1. *Let $\mathcal{A}$ be an adversary attempting to distinguish between H_0 and H_1 in the experiments $\mathsf{exp}^{\mathsf{H}_0,\mathsf{H}_1}_{b,\mathcal{A}}(1^\lambda)$. Then:*

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{H}_{0,1}(1^\lambda))) < \frac{\epsilon}{\sum_{j=1}^{r} 2^j \binom{\ell}{j}}$$

where ϵ is the advantage in winning in H_0.

Proof. Note that there at least a single pair $(\boldsymbol{t}, \boldsymbol{b})$ must satisfy the conditions in H_1, otherwise the security game would be vacuous because the challenge query would be unconstrained. The number of total pairs is $\sum_{j=1}^{r} 2^j \binom{\ell}{j}$, and since the conditions can be checked in $\mathsf{poly}(\lambda)$ time, then the probability of $\mathcal{B}$ aborting the security game is equal to $1 - 1/\sum_{j=1}^{r} 2^j \binom{\ell}{j}$. Therefore, the advantage that $\mathcal{B}$ has in winning the security game is $\epsilon/(\sum_{j=1}^{r} 2^j \binom{\ell}{j})$ where ϵ is the advantage that $\mathcal{A}$ has in winning in H_0 □

[10] One may wonder why the final condition for the abort is necessary, because the reduction in the proof of Lemma V;4.2 works even without it. This additional abort step is introduced to make the probability of abort to occur independently of the choice of the constrained key queries and the challenge query made by the adversary. Without this step, we cannot lower bound $|\Pr[\mathsf{H}_1] - 1/2|$. A similar problem was identified by Waters [292], who introduced the "artificial abort step" to resolve it. Our analysis here is much simpler because we can compute the abort probability exactly in our case.

Claim V;4.2.2. *Let $\mathcal{A}$ be an adversary attempting to distinguish between H_1 and H_2 in the experiments* $\mathsf{exp}_{b,\mathcal{A}}^{\mathsf{H}_1,\mathsf{H}_2}(1^\lambda)$*. Then:*

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{H}_{1,2}(1^\lambda))) = 0.$$

Proof. The step in H_2 is merely a syntactic change from H_1 and so the advantage is the same. □

Claim V;4.2.3. *Let $\mathcal{A}$ be an adversary attempting to distinguish between H_2 and H_3 in the experiments* $\mathsf{exp}_{b,\mathcal{A}}^{\mathsf{H}_2,\mathsf{H}_3}(1^\lambda)$*. Then:*

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{H}_{2,3}(1^\lambda))) < \mathsf{negl}(\lambda).$$

Proof. The proof of this claim is exactly the same as the proof of Lemma V;4.1, except with that $\mathcal{B}$ aborts if any of the conditions in H_1 are satisfied. As such, we do not reiterate the proof here. □

Finally, since the advantage of $\mathcal{A}$ of winning the CPRF security game in Simulation V;4.1 is $\mathsf{negl}(\lambda)$, then the probability of $\mathcal{A}$ winning the security game is

$$\epsilon = \max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{H}_0(1^\lambda))) \leq \sum_{j=1}^{r} 2^j \binom{\ell}{j} \left(\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{H}_3(1^\lambda))) + \mathsf{negl}(\lambda) \right) < \mathsf{negl}(\lambda)$$

and thus the proof of Lemma V;4.2 follows from the proofs of the above claims. Recall that the fact that

$$\sum_{j=1}^{r} 2^j \binom{\ell}{j} \left(\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{H}_3(1^\lambda))) + \mathsf{negl}(\lambda) \right) < \mathsf{negl}(\lambda)$$

is satisfied is because $\sum_{j=1}^{r} 2^j \binom{\ell}{j} = \mathsf{poly}(\lambda)$. □

Lemma V;4.3 [Perfect 1-key privacy]

Construction V;3.1 satisfies perfect weak 1-key privacy.

Proof. Notice that the master secret key is of the form:

$$(\{K_{\boldsymbol{t},w}\}, \{\overline{K_{\boldsymbol{t},w}}\})_{\boldsymbol{t}\in\mathbb{T}, w\in\{0,1\}^z},$$

where $K_{\boldsymbol{t},w}, \overline{K_{\boldsymbol{t},w}} \leftarrow \mathsf{prf.Setup}(1^\lambda)$. Let $v^{(0)}, v^{(1)} \in \{0,1,*\}^\ell$ be the two bit-fixing strings that the adversary $\mathcal{A}$ queries. Then, $\mathcal{A}$ receives either one of the following two distributions:

- $\left(\widetilde{K}^{(0)}_{\boldsymbol{t},w}\right)_{\boldsymbol{t}\in\mathbb{T},w\in\{0,1\}^z}$ where $\widetilde{K}^{(0)}_{\boldsymbol{t},w} = K_{\boldsymbol{t},w}$ if and only if $w \in \Gamma_{v^{(0)}}$, and $\widetilde{K}_{\boldsymbol{t},w} = \overline{K_{\boldsymbol{t},w}}$ otherwise.
- $\left(\widetilde{K}^{(1)}_{\boldsymbol{t},w}\right)_{\boldsymbol{t}\in\mathbb{T},w\in\{0,1\}^z}$ where $\widetilde{K}^{(1)}_{\boldsymbol{t},w} = K_{\boldsymbol{t},w}$ if and only if $w \in \Gamma_{v^{(1)}}$, and $\widetilde{K}_{\boldsymbol{t},w} = \overline{K_{\boldsymbol{t},w}}$ otherwise.

Notice that both the distributions are made up of entirely uniformly sampled keys in $\{0,1\}^\lambda$ for prf. Therefore, they are perfectly indistinguishable and the proof is complete. □

Finally, the proof of Theorem V;4.2 follows by the results of Lemma V;4.1, Lemma V;4.2 and Lemma V;4.3. The fact that we achieve this under the existence of one-way functions is because a pseudorandom function can be instantiated by a OWF directly from the results of [164, 178]. Thus the statement of Theorem V;4.2 follows immediately. □

V;5 Implementation

Our CPRF is constructed just from an underlying PRF. While we require a large number of PRF evaluations, since these operations are quite cheap, this implies that it may be plausible to run our construction quite cheaply.

Discussion of adaptive security parameters. In the proof of Lemma V;4.2, we incur a security loss of $1/\mathsf{poly}(\lambda)$. While this loss is small enough (asymptotically) to guarantee adaptive security with a polynomial security reduction, the impact of it on parameter settings will noticeably impact the practicality of our scheme. This is because we have to essentially satisfy security in a setting where the security parameter is $\lambda' = \lambda/\mathsf{poly}(\lambda)$ and $\mathsf{poly}(\lambda) = \sum_{j=1}^{r} 2^j \binom{\ell}{j}$; this quantity grows rapidly, even for small values of r.

For this reason, our implementation only targets security in the selective security setting. In this model, the reduction from the security of the underlying PRF to the security of the CPRF is tight. Subsequently, we can take the security of the scheme to be solely defined by the security parameter λ. This allows us to be flexible with the choices of ℓ and r, which consequently have no impact on the security reduction. Therefore, we only have to consider the efficiency of the scheme when choosing ℓ and r when we make our parameter settings below.

Parameter settings. Recall that $\ell = \mathsf{poly}(\lambda)$ and $r = O(1)$; we can calculate concretely how our choice of ℓ and r will influence the computational running times and costs of our scheme. For example, in $\mathsf{cprf.Setup}$ we have to sample $\sum_{j=1}^{r} 2^j \cdot \binom{\ell}{j}$ uniform PRF keys. This is going to be somewhat larger than ℓ^r. Therefore, if we take the security parameter to be 128 bits in length,

ℓ	4			8				
r	2	3	4	2	3	4	5	6
Setup	0.05	0.227	1.01	0.299	2	14.4	166.1	12356
Eval	0.035	0.076	0.141	0.134	0.517	1.5	3.74	9.303
Constrain	0.023	0.062	0.155	0.079	0.39	1.33	3.98	10.539

Table V;2: Benchmarks (ms) for cprf operations, for $\lambda = 80$ and $\ell \in \{4, 8\}$.

ℓ	16				80	
r	2	3	4	5	2	3
Setup	1.07	18.6	4387	291201	55.3	1562824
Eval	0.522	3.32	19	101	13.2	463
Constrain	0.301	2.62	19.9	106	7.46	373

Table V;3: Benchmarks (ms) for cprf operations, for $\lambda = 80$ and $\ell \in \{16, 80\}$.

and equivalently take $\ell = 128$ and $r = 4$, then we can expect total running costs that are $> 2^{27}$. Then, increasing r above 4 is going to be impractical to run.

Limiting ℓ so that we consider smaller inputs helps to alleviate the burden slightly, as we see later this enables running experiments for $r = 6$. Since we are interested in experimenting with as large r as possible we decide to give results with $\ell \in \{4, 8, 16, 80, 128\}$.

Some applications (such as those of ID-NIKE and broadcast encryption) may tolerate only short input lengths if the length of encoded data is small (for instance, encoding identities).

We choose to instantiate prf using the hmac algorithm, with the SHA-256 hash function. hmac is indeed both a secure MAC, and a secure PRF algorithm in the ROM (see Section II;4.6). An advantage of using this algorithm is that it is typically included in native cryptographic libraries.

Results. We answer the conjecture that our scheme is practically efficient in the affirmative, providing the first meaningful implementation of a CPRF for any number of collusions and predicates. Our implementation is a proof-of-concept but the cryptographic mechanism is remarkably simple, consisting of only 508 lines of code. It is written in the language Go to expose concurrency features that can speed up the key generation processes. The figures below were run on the following configuration: 256GB RAM; Intel(R) Xeon(R) CPU E5-2667 v2 @ 3.30GHz; maximum of 1 core using all 16 threads. Our source code will be made open source in the near future, but for now is available in Appendix A.[11] In Table V;2 and V;3, we provide benchmarks for $\lambda = 80$, for various settings of ℓ and r. In Tables V;4 and V;5, we provide similar benchmarks in the case of $\lambda = 128$.

[11]This work is currently under submission and so the source code is not yet public.

ℓ	4			8				
r	2	3	4	2	3	4	5	6
Setup	0.048	0.185	0.755	0.218	1.62	12.8	151	11989
Eval	0.038	0.083	0.176	0.142	0.49	1.5	3.78	9.91
Constrain	0.024	0.068	0.134	0.08	0.323	1.36	4.12	11.6

Table V;4: Benchmarks (ms) for cprf operations, for $\lambda = 128$ and $\ell \in \{4, 8\}$.

ℓ	16				128	
r	2	3	4	5	2	3
Setup	0.903	17.1	4237	293433	1346	54569004
Eval	0.531	3.56	19.9	94.2	33.7	1800
Constrain	0.295	2.63	19	113	19.8	1511

Table V;5: Benchmarks (ms) for cprf operations, for $\lambda = 128$ and $\ell \in \{16, 128\}$.

Discussion of results. It is clear that our results demonstrate an efficient CPRF scheme for small length inputs and very small settings of r. For $r < 6$, all algorithms run in less than a second (apart from for $\ell = 16$). The running times for $\lambda \in \{80, 128\}$ are virtually the same. For $(r = 6) \wedge (\ell = 8)$, setup takes approximately 12 seconds. Similarly, for $(r = 4) \wedge (\ell = 16)$ it takes approximately 4 seconds. For $(r = 5) \wedge (\ell = 16)$, setup is much more expensive — taking between 4 and 5 minutes. These findings suggest that we could easily increase λ to higher levels without much change in the running times.

The exponential dependency of ℓ on r becomes rapidly more apparent for higher settings of both. For small $\ell \in \{8, 16\}$, we can see noticeable increases in the time taken to run cprf.Setup between $r = 5$ and $r = 6$. In contrast, the differences between cprf.Eval and cprf.Constrain seem to follow a linear relation across all intervals. As a result, the runtimes of Eval and Constrain are much smaller in comparison to the setup. Fortunately, Setup is usually only run once for each instantiation of cprf. As such, tolerating an initial setup expense allows much more efficient CPRF evaluation and constraining algorithms.

If we move to larger values of $\ell \in \{80, 128\}$, then the running time for the master secret key generation becomes vastly inflated. For $(r = 3) \wedge (\ell = 80)$, cprf.Setup takes around 12 seconds already. For $(r = 3) \wedge (\ell = 128)$, cprf.Setup takes over 15 hours! Clearly, the latter case is completely unsustainable. But we think it provides a useful point-of-reference for where our scheme becomes practically inefficient to run.

Comparison with previous work. While previous constructions of CPRFs have not been implemented, the only constructions that allow for multiple constraint queries (and constrained keys to be defined wrt bit-fixing predicates) are reliant on multilinear maps and indistinguishability obfuscation. Even ignoring the security concerns regarding candidate constructions of these primitives, they are very inefficient. This means that any CPRF construction based on these primitives has almost no chance of being practical.

Lattice-based constructions (and the construction based on traditional groups [20]) only allow one constrained key to be learnt, so our scheme already compares favourably with these designs. Moreover, the lattice-based designs require very large parameters for establishing the LWE assumption that is required. Most designs take similar parameters choices as those made during the design of the lattice-based PRF of Boneh et al. [53]. These works require the LWE assumption to hold with a sub-exponential modulus-to-noise ratio; resulting in a very large modulus and large computational costs. Again, without corresponding implementations it is impossible to compare actual running times, but we can infer that our construction is likely to be much more efficient due to the simplicity of the primitives that we use.

V;6 Conclusion

To summarise this chapter, we have devised a new construction of a constrained PRF for the bit-fixing predicate. In comparison with previous research, our CPRF is the first to allow multiple ($r > 1$) constrained keys to be learnt during the security proof from standard assumptions and in the standard model. In fact, our construction requires only the existence of one-way functions, a much weaker assumption than *any* previous construction.

In addition, our CPRF is fully adaptively secure without the requirement for sub-exponential security losses and satisfies 1-key privacy. We have detailed a proof-of-concept implementation of our construction that demonstrates that our CPRF can be very efficient for small values of r.

V;7 Future work

We briefly discuss the possible avenues for future work, based on our research.

More collusions. If we could increase the number of collusions to something that was dependent on the security parameter alone, rather than the input length, then we would make significant progress to achieving a CPRF secure against any number of collusions. Our construction utilises a combinatorial technique that is inherently dependent on $\ell = \mathsf{poly}(\lambda)$ — this explains the dependence on constant-size r. If we could remove the dependence on ℓ, and instead only depend linearly on λ, then we could achieve logarithmic numbers. This could also lead to a much more efficient construction in complexity terms.

More expressive predicates. One of the most influential papers that we found for devising our construction was the attribute-based encryption scheme of Gorbunov et al. [169]. We can think of each of the vectors $\boldsymbol{t} \in \mathbb{T}$ as corresponding to a gate, and then the individual keys corresponding to the output wires of said gate. Viewing our construction like this enabled us to find that there would be certain keys, for bit-fixing predicates, that would never be given to the adversary.

Indeed, the construction of [169] targets policies that are expressed as circuits. A significant improvement on our work would potentially leverage their results even further, to obtain constrained keys that also correspond to bounded-depth circuit predicates, with collusion-resistance.

Practical explorations. Our construction could lead to small-scale implementations of primitives such as ID-NIKE.[12] Moreover, our benchmarks run in `Go` up to relatively small parameter settings, but an optimised implementation in a faster language (such as `C` or `Rust`) may capitalise on performance further. Consequently, this may allow us to reach higher values of r in our construction. Additionally, more implementations in higher-level languages (for instance, `JavaScript`) could enable us to deploy the CPRF as part of applications more widely (for example, in browser-based scenarios).

In some ways, a CPRF represents a non-interactive oblivious PRF. Oblivious PRFs (OPRFs) are incredibly useful primitives that are used in many practical scenarios (see Chapter VI for one such application). Non-interactivity is a highly sought after characteristic in cryptographic primitives, since communication times can be the ultimate bottleneck in cryptographic design. An interesting avenue of research would test whether our constrained PRF could generically replace OPRFs (in scenarios where the number of clients low, perhaps).

[12] Although, the goal with these applications is to achieve unbounded security, i.e. $r = \mathsf{poly}(\lambda)$.

VI Anonymously Authenticating Internet Traffic Using Verifiable Oblivious PRFs

imposter

PREFACE

In this chapter we will develop a practical protocol for anonymously authenticating clients to servers. The protocol itself enjoys similarities with previous work, though we provide optimisations that allow for more efficient communication and client computation. Our scheme is instantiated from elliptic curves, implicitly relying on a cryptographic primitive known as a *verifiable oblivious pseudorandom function*.

We use this scheme for providing clients with the ability to prove their 'honesty' in an anonymity-preserving way, to some appropriate server. This is achieved using the notion of 'anonymous' tokens that can be acquired by the client. These tokens attest to the client's honestly in the future.

Our solution solves a real-world problem, engendered by the increasing usage of content delivery networks (CDNs) for serving web content to clients. The problem originates from the desire of CDNs to prevent malicious clients from gaining access to web resources. Though client connections are rarely blocked altogether, they may have to complete some sort of challenge first. These tests — usually implemented as CAPTCHAs — are burdensome for users and, in some cases, are prone to implementation errors that can prevent access. We show, statistically, that users who browse anonymously (specifically via tools such as Tor or VPNs) are targeted disproportionately by such measures.

We develop compatible client and server implementations for our protocol. When deployed, these tools are compatible for proving honesty to the Cloudflare network, without sacrificing user anonymity. Our client deployment comprises a browser extension, known as Privacy Pass, that is compatible with the Chrome, Firefox and Tor web browsers. The server code is embedded into the Cloudflare access-control mechanism. A successful authentication allows the client to bypass challenges that are required by Cloudflare during browsing.

Privacy Pass was released in November 2017 and has since seen over 75000 downloads. We give browsing statistics showing that the extension has resulted in a notable decrease in the number of internet challenges that are served by Cloudflare. The network is significantly more accessible for all internet users, as a result.

This chapter is based heavily on the following paper [120] that was published at the Privacy Enhancing Technologies Symposium 2018 [PETS18].[1] We provide links to all of our deployment code (which is open-source and licensed under the BSD-3 clause); the browser extension can also be downloaded from the Chrome store and the Firefox add-ons store. The website for Privacy Pass is `https://privacypass.github.io`. Large parts of this work was completed while the thesis author undertook internships at Cloudflare during Jun-Sept 2016 and Jul-Oct 2017.

[1]Winner of the Andreas Pfitzmann Best Student Paper Award 2018 at PETS18.

Overview of contributions

- A new VOPRF protocol based on elliptic curves, with batched verifiability (Section VI;2.5).
- A 1-RTT anonymous authentication protocol based on our VOPRF (Section VI;3).
- A browser extension (Privacy Pass) compatible with the Chrome and Firefox browsers for client-side authentication operations. Additionally, a server-side deployment in the CDN Cloudflare, for interacting with users that have Privacy Pass installed (Section VI;5);
- Empirical analysis of Privacy Pass adoption, along with evidence supporting greater browser accessibility for users interacting with Cloudflare (Section VI;5.5).

Contents

VI;1 Introduction

An increasingly common trend for websites with globally high visitation is to use content delivery networks (CDNs) to host or cache their resources. A CDN does not directly act as a hosting provider, more as a 'reverse proxy' whereby websites pay for the CDN to help route their website content through to clients. This routing is primarily built with the intention of maximising the speed of displayed content appearing in client's browsers. For example, many CDNs replicate cacheable resources at many data centre locations across the globe. Doing this brings web resources closer to clients by ensuring a closer geographic proximity between clients and requested resources.

A second advantage of using a CDN it that it can provide security mechanisms on the edge network (i.e. before requests are received by the origin website). CDNs are typically well-resourced and thus have a much greater probability of successfully foiling coordinated attacks against websites. For example, for large-scale flooding attacks (such as those required for leveraging DDoS) CDNs can typically employ the sophistication of their own edge network to dissipate the attack across multiple locations. This prevents the attack from targeting a single entity. For small-scale attacks (such as spam and content scraping) CDNs can use their vast knowledge of the topology of the internet to determine whether connections look malicious, before allowing them access.

The advantages laid out above are quickly seeing website creators moving towards employing CDNs for distributing content. According to Cisco, CDNs will serve 71% of all internet browser traffic in 2021, up from 52% in 2016 [96]. Some of the most well-known CDNs include Akamai, Cloudflare, Fastly, and Amazon Cloudfront.

Methods of protection. In this chapter we focus on the security services offered by CDNs and the residual effects of employing over-zealous website protection. Typically the protection that we discussed is enforced via a variety of means. One of the most common methods is to use IP-based reputation checks on the IP addresses that send HTTP requests to the CDN. These checks involve analysing whether incoming IPs have been used for past malicious activity before granting access. If an IP address is deemed to have a 'poor' reputation, existing CDNs may use one of the following options [288] for guarding protected websites:

- block access altogether;
- issue human proof-of-work challenges to prevent access from bots (e.g. CAPTCHAs [3]);
- route requests through a web application firewall.

IP reputation checks may be made against publicly available databases, many of which are available online for free, for example [95, 97, 112]. Otherwise, CDNs may choose to maintain their own

IP reputation database, using data on attacks that they have witnessed to value the reputation of each address.

Incorrect classification. As with all protection mechanisms, there is the possibility of both false positives and negatives occurring when performing these checks. A *false positive* would be a malicious client — a client that intends to launch some sort of attack on a requested website or the CDN network — that makes requests from a IP address that is assigned a 'good' reputation. A *false negative* is an 'honest' client that is associated with an IP address that has a 'poor' reputation.

The case of false positives are not too worrisome, since a malicious client is assumed to eventually conduct some malicious behaviour that will then cause their IP address to have a 'poor' reputation. After this occurs they will be classified correctly — providing that they cannot quickly and easily switch IP addresses. False negatives are more difficult to justify, since honest users may be subjected to one of the 'blocking' mechanisms described above. Moreover, unless there is functionality for updating IP reputations regularly — including re-evaluation of the reputation to deem it honest, after an initial characterisation as malicious — then these effects may be long-standing. In particular, if users are accessing websites from static IP addresses, then they may remain a 'false negative' for a large period of time.

A more serious situation arises if a user shares a static IP address with multiple other users. In such a situation, any malicious activity by one user associated with the IP address is likely to result in worse browsing conditions for *all* of the users. In terms of concrete groups of users who may be negatively affected by such a security characterisation, we can immediately think of users of *anonymity* services such as Tor, Virtual Private Networks (VPNs) and I2P. These services use a finite list of static IP addresses whose size is orders of magnitude less than the number of users using the service.[2] Moreover, if the IP addresses of a service are publicly available, then it is also plausible that content providers can deliberately target these IP addresses to block access, if they so wish.[3]

The case of Tor and Cloudflare. While the above seems like conjecture, over the past few years it has become apparent that users of anonymity services experience much worse browsing conditions when interacting with CDNs [207]. The Cloudflare network uses a range of security mechanisms, although the most obvious to human users is the requirement to complete a Turing test in the form of a CAPTCHA [3], to access protected websites. This test is used if the IP address making the request has a poor reputation. While access is eventually granted by completing the CAPTCHA, there has been widespread coverage of issues that Tor users (in particular) face when attempting to access Cloudflare-protected websites [289]. The manifestations of these issues are myriad, but typically range from: vocal annoyance at the difficulty of solving these CAPTCHAs,

[2]In the case of Tor, there are typically >2000000 active users for >1000 exit nodes.

[3]This is a configuration available to Cloudflare customers.

to access being completely blocked by badly implemented CAPTCHAs and/or browser compatibility issues.[4]

Users of Tor/VPNs are often forced into using such tools so due to oppressive internet censorship or for protection of their own identity. These tools typically ensure a worsened browsing experience, due to the increased latency experienced while browsing. Consequently, access-limiting measures introduced by CDNs only further degrade the browsing experience for an important subset of users. To fully characterise the disparity in experiences between normal browsing and browsing via Tor, it is worth noting that 1.04% of the global traffic served by Cloudflare is challenged [98].[5] In comparison, the number of requests from Tor users challenged by Cloudflare sits at 17%. Cloudflare typically serves over 10 trillion requests per month [99] and approximately 0.05% of this traffic arrives over the Tor network [98]. As such, Cloudflare handles thousands of requests every second and billions of requests every month from the Tor network alone. This is not even accounting for the numbers of requests originating from VPNs/I2P shared networks operating under similar conditions. Reducing the workload for these users will clearly have a large, undeniably positive effect on their web accessibility.

Previous mitigations. Unfortunately, reducing the number of challenges by simply allowing more requests to access resources (or lowering the reputation threshold) is not economically feasible in the CDN model. Many customers employ CDNs for protecting their websites specifically and, in fact, prefer stronger security measures over greater user access. For instance, many Cloudflare customers deliberately opt to force Tor users to undergo stricter security measures [98].

A common method of reducing challenges for users who have previously completed one successfully is to use browser cookies. The client is given a cookie (of limited lifetime) when they complete a challenge successfully, these cookies are then redeemed in place of a future challenge, if the requirement arises. While this appears to solve the problem of reducing the number of challenges on the face of it, there are subtle issues that reduce the utility of the solution dramatically for certain users. More pertinently, the cookies could potentially be used to subvert the anonymity requirements of even those users that are accessing from anonymity-preserving services.

The first issue is that cookies are regularly cleared out during Tor browser sessions.[6] Therefore, cookies are unlikely to be useful for any meaningful length of time. For the second more devastating issue, using cookies across multiple domains provides an easy way of tracking user movements across the CDN edge network, thus breaking the anonymity models of Tor and VPNs. These issues suggest that a much more nuanced, potentially cryptographic, solution is required.

[4]See `https://trac.torproject.org/projects/tor/ticket/18361` for a collection of related issues.
[5]By 'challenged', we mean that they are shown some proof-of-work challenge before they can access web resources.
[6]Essentially when the Tor circuit configuration resets, which occurs every 10 minutes or so

VI;1.1 Our contribution

In this chapter, we develop a solution that works in tandem with any system that uses challenges to ascertain whether a client is honest. In particular, we grant an anonymous user, deemed to be honest at some point in the past, the ability to gain access to further web resources; without the need for more challenges. Our solution maintains anonymity of the user — so that CDNs cannot globally link requests and attempt to subvert anonymization measures. In essence, we devise a protocol for *anonymously authenticating* users for a predetermined number of future occasions, after an initial trusted setup.

Our protocol is built upon a cryptographic primitive known as an *oblivious pseudorandom function* [142]. An OPRF protocol allows a client to receive PRF evaluations at chosen inputs, from a server holding a PRF key. The security requirement is that the server does not learn the chosen PRF inputs, and the client does not learn the server's key. We give a specific instantiation of the primitive based on elliptic curves. We require an extra security property known as *verifiability*, an extra protocol message that attests to the server's honesty in evaluating the underlying PRF. In summary, a verifiable OPRF (VOPRF) provides a signing mechanism. We augment this VOPRF with the ability to also redeem tokens to the signer, much like a blind signature scheme to achieve authentication.[7] Constructions of VOPRFs already exist [194, 196, 277], we lay out our specific modifications shortly.

The core part of the contribution is an implementation of the anonymous authentication protocol into a meaningful client-server scenario. That is, firstly we develop an implementation of the client in the anonymous authentication protocol in a browser extension, named Privacy Pass, that is compatible with the Chrome and Firefox browsers (and also Tor browser). Secondly, we implement an example server-side implementation in the Cloudflare network. We use the anonymous authentication protocol to bypass CAPTCHAs required by Cloudflare for browsing protected sites, if the user has already completed a CAPTCHA in the past. That is, for each CAPTCHA solution, Cloudflare provides anonymous tokens that are used to bypass CAPTCHA solutions in the future.

We globally released a beta version of Privacy Pass in November 2017. We detail results from the deployment of the extension in Section VI;5 and how it has improved the global browsing experience for users interacting with the Cloudflare CDN.

VI;1.2 Related work on anonymous authentication

In this chapter, we focus on methods of anonymously authenticating users. Our goal is to develop practical solutions for such a mechanism that can be seamlessly embedded into the user browser.

[7]Note that we do not achieve an actual signature scheme, since verification is not a public operation.

The related work below reflects on existing research and why it is not exactly suitable for our requirements. We discuss previous research regarding the cryptographic primitives that are central to our design later in Section VI;2.

Anonymous e-cash. In essence, our solution grants a user anonymous tokens for each challenge that they solve; these tokens can be redeemed later to bypass future challenges. The setup is reminiscent of anonymous e-cash settings that were popularized by Chaum [81, 82]. Chaum showed that it was possible to modify textbook RSA such that a user could receive signatures from a signer on 'blinded' messages. The user is then able to 'unblind' the signature such that it has a valid signature for the unblinded message. The blinding prevents the signer from learning the underlying message. These techniques were used by Chaum et al. [85] for an untraceable e-cash system and more recently as the backbone of the taxable, anonymous e-cash service Taler [129], released by the GNU project in 2016. There have been numerous different blind signature schemes introduced since the blind-RSA variant [56, 147, 152, 155, 261, 269, 273], though these typically are much more conceptually involved and sometimes require more than two rounds of communication.

A blind signature scheme could also be leveraged to help solve the problem that we are faced with. However, our specific construction of a VOPRF is much more efficient than existing construction and we do not require the means for public verification of signature equivalents.

Anonymous blacklisting/whitelisting. Anonymous e-cash systems can be useful in granting privacy-preserving access to resources. An alternative method for providing similar functionality is the usage of anonymous blacklisting techniques [184]. In these schemes a user is asked to prove that they are not blacklisted (or that they are part of a whitelist) using some finite resource that they control. This is very similar to the situation that we consider, where owning a token is equivalent to being part of a whitelist or not being blacklisted. In fact, the work of Henry and Goldberg [184] is motivated by a similar scenario where honest users of Tor are denied access to resources. Additionally, they also explicitly consider the potential of using human challenges or CAPTCHAs as a method for whitelisting. Their formalization suggests numerous ways for maintaining an anonymous blacklist, including using blind RSA signatures. They also discuss 'Nymble-like' [183, 230] systems that leverage the use of a trusted third party for maintaining anonymity during access requests. A similar situation is considered by Liu et al. [229] when proposing their solution TorPolice, a privacy-preserving mechanism that enables access-control policy implementation wrt connections originating from the Tor network.

We ignore the 'Nymble-like' designs as they require an extra trusted party that we do not. While Henry and Goldberg [184] provide a formalization of multiple different anonymous blacklisting methods, with various pros and cons, there is little work detailing the scalability of such systems. Moreover, our solution results in a browser extension that is easily supported by simply running a small piece of code at the service provider. This scales much more efficiently than each service provider implementing bespoke blacklisting/whitelisting techniques. Our solution benefits all users, not just those originating from Tor, and so we provide a more general browser-based solution than TorPolice [229].

Privacy-preserving e-ticketing. Another diverging thread of literature focuses on 'privacy-preserving e-ticketing'. The works of Heydt-Benjamin et al. [185] and Sadeghi et al. [271] were the first to propose solutions to privacy-preserving ticketing for public transportation systems. These solutions were based on anonymous e-cash systems such as those listed above, and on physically unclonable functions. The work of Kerschbaum et al. [206] analysed Singapore's EZ-Link system and showed that it was easy to extract a traveller's travel records. They proposed an encrypted bill processing procedure that allows for privacy-preserving data mining analysis by the transport company.

While the above solutions are similar to the work that we describe, they are heavily targeted at public-transport applications. Additionally, data-mining analysis on the results of the transactions is the primary motivation of these works, which we do not share.

VI;1.3 Layout

In Section VI;2 we will introduce the cryptographic mechanisms that we will need for the rest of the work. This includes a construction of a VOPRF protocol from elliptic curve cryptography foundations. In Section VI;3 we show that these foundations can also be used to construct an anonymous authentication protocol. In Section VI;4 we will introduce the application that we're targeting: reducing the number of internet challenges for honest anonymous users. We will cover the browser extension Privacy Pass in Section VI;5 and an empirical analysis of the server-side deployment. We will also consider a number of plausible out-of-band attacks in Section VI;6, and their mitigation. By 'out-of-band' we mean attacks that fall outside of the security model that we consider. We conclude the chapter in Section VI;7.

VI;2 Preliminaries

Specific notation. Throughout, we will use the notation C and S to denote the client and server respectively in protocol interactions. We will refer to a 'user' or 'client' as the initiator of a connection to some website. We define the 'edge' server as the entity that decides whether a user's request will be allowed or denied. The edge server typically refers to the role of the CDN in the browsing scenario. We use the term 'challenge' to generically refer to some task that the edge provides to a user. This task is solved to prove that the client is acting in an 'honest' way (for CAPTCHAs this check simply proves the requests originates from a human). By an 'honest' user we simply mean a user that *should* be granted access to a website and conversely for a 'malicious' user. In Cloudflare lexicon, 'honest' users are human and 'malicious' users are bots. We will use the notion of a 'protected' page to refer to a website that uses the edge to supply users with challenge pages for granting access (i.e. a CDN customer).

Group instantiation. Most of the cryptographic operations in this chapter take place in a multiplicative, cyclic group $\mathbb{G} = \mathbb{G}(\lambda)$ of order p such that $\mathsf{bitlength}(p) = \mathsf{poly}(\lambda)$. However, when we implement the operations for the experimental section, we will implement $\mathbb{G}$ in the elliptic curve setting. Therefore, all operations in our implementation are handled in the setting where $\mathbb{G}$ is an additive group. As we mentioned previously, elliptic curves bring many efficiency benefits due to favourable parameter settings. We leverage these improvements for lowering the computational costs of our scheme.

VI;2.1 Non-interactive zero-knowledge (NIZK) proofs

A non-interactive zero-knowledge (NIZK) proof is a powerful primitive to allow attesting to knowledge without revealing the contents of the knowledge. Commonly, we have a *prover* $\mathbb{P}$ and a *verifier* $\mathbb{V}$, where $\mathbb{P}$ is attempting to convince $\mathbb{V}$ of some statement. The non-interactivity of the proof system is valuable in the sense that it can be publicly verified without the need for interacting with the prover, beyond sending the actual proof and any initial setup. We give a formalisation in Definition VI;2.1, relying on the fact that the verifier is honest. We do not give a formulation in the malicious verifier case, since this is beyond the scope of the work.

Definition VI;2.1 [NIZK proof system]

Let $\mathbb{P}$ be a *prover*, let $\mathbb{V}$ be a *verifier*, let $\mathcal{L}$ be a language with accompanying relation $\mathsf{P}_{\mathcal{L}}(\cdot,\cdot)$, and let $\mathcal{W}_{\mathcal{L}}$ be a generic set of witnesses for $\mathcal{L}$. Let $\mathsf{nizk} = (\mathsf{Setup}, \mathbb{P}, \mathbb{V})$ be a tuple of algorithms defined below.

- $\mathsf{crs} \leftarrow_{\$} \mathsf{nizk.Setup}(1^{\lambda})$: outputs a common-reference string $\mathsf{crs} \in \mathcal{Z}$.
- $\pi \leftarrow \mathsf{nizk}.\mathbb{P}(\mathsf{crs}, x, w)$: on input $\mathsf{crs} \in \mathcal{Z}$, a word $x \in \mathcal{L}$ and a witness $w \in \mathcal{W}_{\mathcal{L}}$; outputs a proof $\pi \in \Lambda$.
- $b \leftarrow \mathsf{nizk}.\mathbb{V}(\mathsf{crs}, x, \pi)$: on input $\mathsf{crs} \in \mathcal{Z}$, a word $x \in \mathcal{L}$ and a proof $\pi \in \Lambda$; outputs $b \in \{0,1\}$.

We now define security for nizk as the satisfaction of the following criteria.

Definition VI;2.2 [Security]

We say that nizk is a *non-interactive zero-knowledge proof system* (NIZK) for $\mathcal{L}$ (and honest verifier $\mathbb{V}$) if the following holds.

1. (*Perfect completeness*): Consider $x \in \mathcal{L}$ and $w \in \mathcal{W}_{\mathcal{L}}$, where $\mathsf{P}_{\mathcal{L}}(x, w) = 1$. Then:

$$\Pr\left[1 \leftarrow \mathbb{V}(\mathsf{crs}, x, \pi) \middle\| \substack{\mathsf{crs} \leftarrow\$ \mathsf{nizk.Setup}(1^\lambda), \\ \pi \leftarrow \mathbb{P}(\mathsf{crs}, x, w)}\right] = 1.$$

2. (*Computational soundness*): Consider $x \notin \mathcal{L}$. Then:

$$\Pr\left[1 \leftarrow \mathbb{V}(\mathsf{crs}, x, \pi) \middle\| \substack{\mathsf{crs} \leftarrow\$ \mathsf{nizk.Setup}(1^\lambda), \\ \pi \leftarrow \mathbb{P}^*(\mathsf{crs}, x)}\right] = \mathsf{negl}(\lambda).$$

   for every PPT algorithm $\mathbb{P}^*$.

3. (*Perfect zero-knowledge*): There exists a simulated setup algorithm $\mathsf{nizk.SimSetup}(1^\lambda)$ that outputs $\mathsf{crs}_{\mathsf{Sim}}$ and a trapdoor $\mathcal{T}$. Furthermore, there is an algorithm $\mathsf{nizk.Sim}(\mathsf{crs}_{\mathsf{Sim}}, \mathcal{T}, x)$ that outputs a simulated proof π_{Sim}. Then, firstly we have that:

$$\{\mathsf{crs} \leftarrow\$ \mathsf{nizk.Setup}(1^\lambda)\} \approx_p \{\mathsf{crs}_{\mathsf{Sim}} | (\mathsf{crs}_{\mathsf{Sim}}, \mathcal{T}) \leftarrow\$ \mathsf{nizk.SimSetup}(1^\lambda)\},$$

   holds unconditionally. Secondly, we require that the probability:

$$\Pr\left[1 \leftarrow \mathbb{V}(\mathsf{crs}_{\mathsf{Sim}}, x, \pi_{\mathsf{Sim}}) \middle\| \substack{x \in \mathcal{L}, \\ (\mathsf{crs}_{\mathsf{Sim}}, \mathcal{T}) \leftarrow\$ \mathsf{nizk.SimSetup}(1^\lambda), \\ \pi_{\mathsf{Sim}} \leftarrow \mathsf{nizk.Sim}(\mathsf{crs}_{\mathsf{Sim}}, \mathcal{T}, x)}\right] = 1,$$

   is satisfied. In other words, there is an indistinguishable and efficient simulation of the proof generation process for any word x.

The formalisation of NIZK proof systems was first introduced by [50], and constructions of such systems from general assumptions were introduced in [137]. The common-reference string is used as an indicator that a trusted setup occurs between the prover and the verifier. While this conception of NIZKs is fairly restrictive, it is enough for the use-case that we consider.

VI;2.2 Discrete log equivalence proofs

In this work, we will use a NIZK proof system known as a discrete log equivalence, or DLEQ, proof. This system is based on the well-known protocol of Schnorr [273] for proving knowledge of a discrete log exponent in the interactive setting. This means that we define the language $\mathcal{L}$ to be the pairs (x, y) where $y = x^k$ for a value k and where $\mathsf{crs} = (g, h)$ where $h = g^k$.

The Schnorr protocol allows a prover $\mathbb{P}$ to prove to a verifier $\mathbb{V}$ that they know the exponent in the RHS of the pair (g, g^r), in zero-knowledge. A DLEQ proof system first requires $\mathbb{P}$ to commit to some pair $(g, h = g^r) \in \mathbb{G}^2$ initially; this is the crs of the proof system.[8] The proof system allows $\mathbb{P}$ to non-interactively show that another pair $(x, y = x^r) \in \mathbb{G}^2$ is constructed wrt the same exponent $r \in \mathbb{Z}$. Non-interactivity is achieved in the standard way by using the Fiat-Shamir transform [139], at the cost of introducing random oracle evaluations. The Fiat-Shamir transform [139] allows conversion of an interactive proof system (where there is an explicit step where $\mathbb{V}$ 'challenges' $\mathbb{P}$) into a non-interactive proof system (where this step is removed). Informally speaking, this step is replaced with a step where $\mathbb{P}$ computes the random oracle over the entire transcript of the protocol so far.

The description of a DLEQ proof system is given below in Construction VI;2.1.

Construction VI;2.1 [DLEQ proof system]

Let $p > 0$ be some prime integer, let $H : \mathbb{Z}_p^6 \mapsto \mathbb{Z}_p$ be a random oracle, and let $\mathsf{DLEQ} = (\mathsf{Setup}, \mathbb{P}, \mathbb{V})$ be a tuple of algorithms. Define the algorithms in the following way.

- $\mathsf{crs} \leftarrow_\$ \mathsf{DLEQ.Setup}(1^\lambda, p, k)$: Takes $p, k \in \mathbb{Z}$ as input, and outputs $\mathsf{crs} = (\mathbb{G}, p, (g, h))$, where $\mathbb{G}$ is a finite group of prime-order p; $g \in \mathbb{G}$ is some randomly chosen generator; and $h = g^k$.
- $\pi \leftarrow \mathsf{DLEQ}.\mathbb{P}(\mathsf{crs}, (x, y), k)$: Takes as input $\mathsf{crs} = (\mathbb{G}, p, (g, h))$, a pair $(x, y) \in \mathbb{G}^2$ and the witness exponent k, and then:
  - samples a random nonce $t \leftarrow_\$ \mathbb{Z}_p$;
  - computes $a \leftarrow g^t$ and $b \leftarrow x^t$;
  - computes $c = H(g, h, x, y, a, b)$ and $s = t - ck \mod p$;[a]
  - returns $\pi = (c, s)$.
- $d \leftarrow \mathsf{DLEQ}.\mathbb{V}(\mathsf{crs}, (x, y), \pi)$: Takes crs, (x, y) as input as above, along with the proof $\pi = (c, s)$. Verifies π by then:
  - computing $a' = g^s h^c$ and $b' = x^s y^c$;
  - computing $c' = H(g, h, x, y, a', b')$;
  - returns $d = (c' \stackrel{?}{=} c) \in \{0, 1\}$.

[a]Note that s is not explicitly a group element of $\mathbb{G}$ and thus addition is carried out in the ring $\mathbb{Z}_p$.

[8]Throughout this section, we will always assume that $\mathbb{G}$ is multiplicative and thus isomorphic to $(\mathbb{Z}_p^*, \times)$

Intuitively, we obtain the fact that DLEQ is a NIZK proof system given the Fiat-Shamir transform, by the security of the Schnorr protocol [262]. However, for the completeness of this chapter and rather than deferring to this transformation, we give explicit security proofs below showing that each of the key tenets of Definition VI;2.2 hold for Construction VI;2.1.

Lemma VI;2.1 [Completeness]

Construction VI;2.1 satisfies perfect completeness.

Proof. Suppose that $\mathsf{crs} = (\mathbb{G}, p, (g, h)) \leftarrow \mathsf{DLEQ.Setup}(1^\lambda)$ and, as defined, $\pi = (c, s) \leftarrow \mathsf{DLEQ}.\mathbb{P}(\mathsf{crs}, (x, y), k)$ for $k \in \mathbb{Z}$, $x \in \mathbb{G}$ and $y = x^k$. Then during verification we have that $a' = g^s h^c$ and $b' = x^s y^c$. Essentially we just need to check that $a' = a$ and $b' = b$, where a, b are the values computed by $\mathsf{DLEQ}.\mathbb{P}$. If this holds, then $H(g, h, x, y, a, b) = H(g, h, x, y, a', b')$ and thus the verifier outputs $d = 1$.

Firstly, $a' = g^s h^c = g^{t-ck} g^{ck} = g^t = a$; secondly $b' = x^s y^c = x^{t-ck} x^{ck} = x^t = b$. □

Lemma VI;2.2 [Soundness]

Construction VI;2.1 satisfies computational soundness.

Proof. Consider a dishonest prover $\mathsf{DLEQ}.\mathbb{P}^*$ that succeeds with advantage ϵ in convincing the honest verifier $\mathsf{DLEQ}.\mathbb{V}$ that π is valid, without knowing the witness $k \in \mathbb{Z}$. We show that such a prover leads to an adversary $\mathcal{A}$ that breaks the discrete log assumption (Definition II;3.1).

To illustrate the reduction, let $\mathcal{A}$ be an adversary attempting to break $\mathsf{exp}^{\mathsf{dl}}_{\mathcal{A}}(1^\lambda, \mathbb{G})$ from Figure II;4. The trusted setup $\mathsf{DLEQ.Setup}(1^\lambda)$ is replaced with what $\mathcal{A}$ receives, i.e. $(\mathbb{G}, p, (g, g^k))$ where k is the challenge exponent. This is identically distributed to the crs received in the real construction. Then $\mathcal{A}$ runs $\mathsf{DLEQ}.\mathbb{P}^*(\mathsf{crs}, (x, y))$. In the third step, when $\mathbb{P}^*$ makes its query to the random oracle H, then $\mathcal{A}$ extracts the query and parses it as (g, h, x, y, a_1, b_1). If $h \neq g^k$, then $\mathcal{A}$ aborts the reduction.

When $\pi_1 = (c_1, s_1)$ is received from $\mathsf{DLEQ}.\mathbb{P}^*$, $\mathcal{A}$ runs $d_1 \leftarrow\!\!\$\, \mathsf{DLEQ}.\mathbb{V}(\mathsf{crs}, (x, y), \pi_1)$ and aborts if $d_1 \neq 1$. If $d_1 = 1$, $\mathcal{A}$ rewinds the execution of $\mathsf{DLEQ}.\mathbb{P}^*$ until before the third step is run and reprograms H to output a new random value $c_2 \leftarrow\!\!\$\, \mathbb{Z}_p$. Now, $\mathcal{A}$ plays the execution of $\mathsf{DLEQ}.\mathbb{P}^*(\mathsf{crs}, (x, y))$ again and receives $\pi_2 = (c_2, s_2)$. If $d_2 \leftarrow \mathsf{DLEQ}.\mathbb{V}(\mathsf{crs}, (x, y), \pi_2)$ and $d_2 \neq 1$ then abort.

If $(d_2 \overset{?}{=} 1)$, then notice that $a'_j = g^{s_j + c_j k}$ and $b'_j = x^{s_j + c_j k}$ for $j \in \{1, 2\}$; where $s_j = t - c_j k$. Therefore, computing $(s_1 - s_2) \cdot (c_2 - c_1)^{-1}$ reveals k. To finish the reduction, $\mathcal{A}$ outputs k in $\mathsf{exp}^{\mathsf{dl}}_{\mathcal{A}}(1^\lambda, \mathbb{G})$ and wins with probability ϵ. By the DL assumption, we have that $\epsilon \leq \mathsf{negl}(\lambda)$; thus the computational soundness of DLEQ follows. □

Lemma VI;2.3 [Zero-knowledge]

Construction VI;2.1 satisfies perfect zero-knowledge.

Proof. The zero-knowledge aspect of the proof system follows directly from the fact that for an honest verifier, the output of the random oracle can be programmed to output a randomly sampled value which the prover can incorporate into the proof π. That is, the simulation for DLEQ.$\mathbb{V}$ samples a random $c \leftarrow\!\!\$\, \mathbb{Z}_p$ and programs H to output c on the input (g, h, x, y, a, b) where $a = g^t \cdot h^{-c}$ and $b = x^t \cdot y^{-c}$, for $t \leftarrow\!\!\$\, \mathbb{Z}_p$. Then, the fact that the proof verifies correctly is trivial. □

The fact that DLEQ is a NIZK proof system follows immediately from Lemmas VI;2.1, VI;2.2 and VI;2.3.

VI;2.3 Batched DLEQ proofs

The above protocol works with the same secret witness k for fresh instantiations of the proof π. The work of Henry [182] shows that it is possible to batch the above DLEQ proof structure for a single-shot verification. This batching technique is adapted from long-standing methods that were devised by Bellare et al. in [34]. Briefly, for one common reference string and witness k, it is possible to show that multiple pairs $\{x_i, y_i\}_{i \in [m]}$ share the same DL exponent k with the single pair $(g, g^k) \in \mathsf{crs}$. The advantages of this batching are realised in both reduced communication and computational costs for the verifier. We give the batched formulation in Construction VI;2.2.

Construction VI;2.2 [Batched DLEQ proof system]

Let $\mathbb{G} = \mathbb{G}(\lambda)$ be a prime-order group, where $p > 0$ denotes the order, let $\widehat{H} : \mathbb{Z}_p^{2m+2} \mapsto \mathbb{Z}_p$ be a random oracle, let prg be a PRG (Definition II;4.1) where $\mathsf{prg}.\mathsf{Eval}(m)$ maps into $\mathbb{Z}_p^m$. Let DLEQ be defined as in Construction VI;2.1, and let $\widehat{\mathsf{DLEQ}}_m = (\mathsf{Setup}, \mathbb{P}, \mathbb{V})$ be the tuple of algorithms defined below.

- $\mathsf{crs} \leftarrow_\$ \widehat{\mathsf{DLEQ}}_m.\mathsf{Setup}(1^\lambda, p, k)$: Outputs $\mathsf{crs} \leftarrow \mathsf{DLEQ}.\mathsf{Setup}(1^\lambda, p, k)$.
- $\pi \leftarrow \widehat{\mathsf{DLEQ}}_m.\mathbb{P}(\mathsf{crs}, (x, y), k)$: Takes $\mathsf{crs} = (\mathbb{G}, p, (g, h))$, group elements $\{(x_i, y_i)\}_{i\in[m]} \in \mathbb{G}^{2m}$ and the witness exponent k as input, and then:
  - calculates a seed $\omega \leftarrow \widehat{H}(g, h, \{(x_i, y_i)\}_{i\in[m]})$, and runs $\mathsf{prg}.\mathsf{Seed}(\omega)$;
  - runs $c_1, \ldots, c_m \leftarrow_\$ \mathsf{prg}.\mathsf{Eval}(m)$
  - computes $\widehat{x} \leftarrow \prod_{i=1}^m x_i^{c_i}$ and $\widehat{y} \leftarrow \prod_{i=1}^m y_i^{c_i}$;
  - returns $\pi \leftarrow \mathsf{DLEQ}.\mathbb{P}(\mathsf{crs}, (\widehat{x}, \widehat{y}), k)$;
- $b \leftarrow \widehat{\mathsf{DLEQ}}_m.\mathbb{V}(\mathsf{crs}, \{(x_i, y_i)\}_{i\in[m]}, \pi)$: Takes crs, $\{(x_i, y_i)\}_{i\in[m]}$ as input, along with the proof $\pi = (c, s)$. Verifies π by:
  - computing $\omega' \leftarrow \widehat{H}(g, h, \{(x_i, y_i)\}_{i\in[m]})$, and running $\mathsf{prg}.\mathsf{Seed}(\omega')$;
  - running $c'_1, \ldots, c'_m \leftarrow_\$ \mathsf{prg}.\mathsf{Eval}(m)$;
  - computing $\widehat{x}' \leftarrow \prod_{i=1}^m x_i^{c'_i}$ and $\widehat{y}' \leftarrow \prod_{i=1}^m y_i^{c'_i}$;
  - returning $b \leftarrow \mathsf{DLEQ}.\mathbb{V}(\mathsf{crs}, (\widehat{x}', \widehat{y}'), \pi)$.

The proofs of soundness and zero-knowledge follow similar arguments to those that are shown in the proofs of Lemma VI;2.2 and Lemma VI;2.3, respectively. Moreover, we point the reader to the thesis of Henry [182, Theorem 3.17] for full proof of these properties. Completeness follows immediately from the fact that $y_i = x_i^k$ for each $i \in [m]$; consequently, $\widehat{y} = \widehat{x}^k$ and we can just compute the proof π using the single-instantiation of $\mathsf{DLEQ}.\mathbb{P}$.

Advantages of batching proofs. The advantages of providing the batched proofs lie in the reduction of the communicational overheads between the prover and the verifier. For computation, the work of $\mathsf{DLEQ}.\mathbb{P}$ increases slightly, but the work of $\mathsf{DLEQ}.\mathbb{V}$ reduces. We typically place more weight on reducing the work of the verifier as this is likely to be an under-resourced user, in comparison to the prover. For communication, we measure the size of the data sent from $\mathbb{P}$ to $\mathbb{V}$.

We measure m instantiations of DLEQ against one instantiation of $\widehat{\mathsf{DLEQ}}_m$, for $m \in \mathbb{Z}$; we ignore the cost of the random oracle evaluation as this would usually be replaced by an inexpensive hash function evaluation. We measure computational efficiency in terms of the number of modular exponentiations that have to be computed, as these are the most expensive operation.

In the DLEQ case, $\mathsf{DLEQ}.\mathbb{P}$ runs 2 modular exponentiations, 1 modular multiplication and 1 modular addition per evaluation. Thus the total cost is equivalent to $2m$ modular exponentiations. For verification, $\mathsf{DLEQ}.\mathbb{V}$ runs 4 modular exponentiations and 2 multiplications per evaluation, for a total of $4m$ modular exponentiations. The only elements communicated are the pair in the proof statement $\pi = (c, s)$. The pair consists of two group elements from $\mathbb{Z}_p$ and so the total cost is $2m$ group elements.

In the batched $\widehat{\mathsf{DLEQ}}_m$ case, $\mathsf{DLEQ}.\mathbb{P}$ runs $2m + 2$ modular exponentiations, since they also compute a single invocation of DLEQ. For verification, $\mathsf{DLEQ}.\mathbb{V}$ computes $2m + 4$ modular exponentiations — roughly half the unbatched case. The communication overhead reduces from $2m$ to 2 group elements and is no longer dependent on the number of evaluations. This is significantly more efficient, considering that m can grow polynomially.

VI;2.4 Verifiable oblivious pseudorandom functions

The main building block of our construction is a verifiable oblivious pseudorandom function (VOPRF), as introduced by Freedman et al. [142]. An oblivious pseudorandom function (OPRF) is a protocol between a server S and a client C. Let prf be some pseudorandom function as given in Definition II;4.2.

Then S runs $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{prf}.\mathsf{Setup}(1^\lambda)$ and holds the secret key msk. The client holds some element $x \in \mathcal{X}$ that they intend to use as an input. A VOPRF protocol allows C to learn the output $\mathsf{prf}.\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x)$, without learning the key msk. Likewise, S does not learn the client input x.

Notice that this resembles a secure computation interaction as defined in Section II;6. That is, an OPRF is a protocol ψ, taking input msk from S and input $x \in \mathcal{X}$ from C. We define $\mathsf{aux}_\mathsf{C} = \mathsf{aux}_\mathsf{S} = (\mathsf{prf}, \mathsf{pp})$ to be the auxiliary information. The outputs of the protocol are then defined as $\mu_\mathsf{C} = \mathsf{prf}.\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x)$ and $\mu_\mathsf{S} = \perp$, respectively for the client and server. The view is then constructed as in Definition II;6.1.

A VOPRF protocol extends the auxiliary inputs to also include a fixed descriptor h that serves as a commitment to the private key msk. In essence, h is part of the crs of a NIZK proof system, that allows proving that $y \leftarrow \mathsf{prf}.\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x)$ is obtained using the PRF evaluation. Then, the output μ_C is augmented with the proof π corresponding to the statement.

While our construction is very close to the principles held by VOPRF designs, we do not make actual black-box usage of the primitive. Therefore, it is not necessary to give a generic formalisation of the primitive for the purpose of this work. We devise security properties for our actual construction (which is based on the VOPRF design given here) in Section VI;3

VI;2.5 Elliptic curve VOPRF

We focus on the construction of VOPRFs from elliptic curves (EC-VOPRF), of which there are a variety of competing designs. Jarecki et al. [194] construct a password-protected secret sharing protocol from such a construction. A similar application is targeted in [195]. The work of Papadopoulos et al. [252] uses a similar construction for constructing an efficient DNSSEC protocol. Finally, the work of Burns et al. [72] constructs an EC-OPRF, i.e. *without* verifiability.

In this section, we will give a new construction of an EC-VOPRF protocol. This primitive will form the foundations of our anonymous authentication protocol. Our EC-VOPRF differs from previous constructions in that it ensures verifiability via the aforementioned batched DLEQ proofs from Section VI;2.2. Some of the previous works [194, 252] enshrines verifiability by providing a DLEQ proof for each individual evaluation. Our approach reduces communication overheads significantly (and computation overheads for the verifier) by taking advantage of the batching mechanism.

Let C be the client in our EC-VOPRF protocol, and let S be the server. Similarly to above, C holds a set of inputs $\{x_i\}_{i\in[m]} \in \{0,1\}^\lambda$. The protocol allows C to learn $\{\mathsf{prf}.\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x_i)\}_{i\in[m]}$ where msk is a secret key held by S.

For our prf construction, we let $\mathsf{msk} = k \in \mathbb{Z}_p$ be a randomly sampled integer and $\mathsf{pp} = (\mathbb{G}, p, (g, h))$ where $g \in \mathbb{G}$ a generator of the cyclic group $\mathbb{G}$; $h = g^k$; and $p > 0$ a large prime. Then we define $\mathsf{prf}.\mathsf{Eval}(\mathsf{pp}, k, z)$ to output z^k. Clearly, the output z^k is a randomly distributed element in $\mathbb{G}$; by random choice of the key k and the fact that $\mathbb{G}$ is cyclic. We do not give a formal proof of the fact that prf is a pseudorandom function as given in Definition II;4.2, since we use it as part of the non-black-box construction of a VOPRF that we highlighted above. Finally, $\mathsf{pp} = \mathsf{crs} \leftarrow_{\$} \widehat{\mathsf{DLEQ}}_m.\mathsf{Setup}(1^\lambda)$ is a valid common-reference string for a batched DLEQ proof setup, as given in Construction VI;2.2.[9]

Finally, we define a hash function $G : \{0,1\}^\lambda \mapsto \mathbb{G}^*$ that hashes binary strings into the non-identity elements $\mathbb{G}^*$ of the group $\mathbb{G}$.[10] We give the details of our design in Construction VI;2.3.

[9]We ignore the inclusion of p as an extra parameter as this is an assumed input in the DLEQ formulation anyway.

[10]We avoid the identity element since this element has order 1, and we assume that all other elements have order $p-1$.

```
$\psi^1_{EC}(\mathsf{C}, \mathsf{pp}, \{x_i\}_{i\in[m]})$
1 : $\varphi \leftarrow \emptyset$;
2 : $\{r_i\}_{i\in[m]} \leftarrow\$ \mathbb{Z}_p$;
3 : for $i \in [m]$ :
4 :     $\widetilde{x_i} \leftarrow G(x_i)^{r_i}$;
5 :     $\varphi[i] = (x_i, r_i)$;
6 : $\{\widetilde{x_i}\}_{i\in[m]} \rightarrow \mathsf{S}$;
7 : return $\varphi$;
```

```
$\psi^2_{EC}(\mathsf{S}, \widehat{\mathsf{DLEQ}}_m, \mathsf{pp}, \{\widetilde{x_i}\}_{i\in[m]}, k)$
1 : $\mathcal{W} \leftarrow \emptyset$;
2 : for $i \in [m]$ :
3 :     $\widetilde{y_i} \leftarrow \widetilde{x_i}^k$;
4 :     $\mathcal{W}[i] = \widetilde{y_i}$;
5 : $\pi \leftarrow \widehat{\mathsf{DLEQ}}_m.\mathbb{P}(\mathsf{pp}, \{(\widetilde{x_i}, \widetilde{y_i})\}_{i\in[m]}, k)$;
6 : $(\mathcal{W}, \pi) \rightarrow \mathsf{C}$;
```

Figure VI;1: Left: Step one of the EC-VOPRF protocol.
Right: Step two of the EC-VOPRF protocol.

```
$\psi^3_{EC}(\mathsf{C}, \widehat{\mathsf{DLEQ}}_m, \mathsf{pp}, \{(\widetilde{x_i}, r_i)\}_{i\in[m]}, (\mathcal{W}, \pi))$
1 : $\{y_i\}_{i\in[m]} \leftarrow \mathcal{W}$;
2 : $b \leftarrow \widehat{\mathsf{DLEQ}}_m.\mathbb{V}(\mathsf{pp}, \{(x_i, y_i)_{i\in[m]}\}, \pi)$;
3 : if $b \neq 1$ :
4 :     return $\perp$;
5 : for $i \in [m]$ :
6 :     $(x_i, r_i) \leftarrow \varphi[i]$;
7 :     $y_i \leftarrow \widetilde{y_i}^{1/r_i}$;
8 : return $\{y_i\}_{i\in[m]}$;
```

Figure VI;2: Step three of the EC-VOPRF protocol.

Construction VI;2.3 [Batched EC-VOPRF]

Let λ be the security parameter, $m \in \mathbb{Z}$ and let $\widehat{\mathsf{DLEQ}}_m$ be defined as in Construction VI;2.2. Let $\mathsf{crs} = (\mathbb{G}, p, (g, h)) \leftarrow \widehat{\mathsf{DLEQ}}_m.\mathsf{Setup}(1^\lambda)$ for a prime-order group $\mathbb{G}$. Let $G : \{0,1\}^\lambda \mapsto \mathbb{G}^*$ be a random oracle and let $x_i \in \{0,1\}^\lambda$ for $i \in [m]$. Then, let ψ_{EC} be the elliptic curve VOPRF protocol consisting of the steps:

- $(\mathsf{C} : \varphi,\ \mathsf{S} : \{\widetilde{x_i}\}_{i\in[m]}) \leftarrow \psi^1_{EC}(\mathsf{C}, \mathsf{pp}, \{x_i\}_{i\in[m]})$ (Figure VI;3[Left]);
- $(\mathsf{C} : (\mathcal{W}, \pi)) \leftarrow \psi^2_{EC}(\mathsf{S}, \widehat{\mathsf{DLEQ}}_m, \mathsf{pp}, \{\widetilde{x_i}\}_{i\in[m]}, k)$ (Figure VI;3[Right]);
- $(\mathsf{C} : \{y_i\}_{i\in[m]}) \leftarrow \psi^3_{EC}(\mathsf{C}, \widehat{\mathsf{DLEQ}}_m, \mathsf{pp}, \{(\widetilde{x_i}, r_i)\}_{i\in[m]}, (\mathcal{W}, \pi))$ (Figure VI;4).

As we mentioned previously, we make non-black-box usage of the EC-VOPRF protocol when constructing our anonymous authentication protocol. As a result, Construction VI;2.3 acts merely as intuition for the full protocol in Section VI;3. In particular, our eventual protocol will have two phases, a *signing phase* and a *redemption phase*. The signing phase is essentially instantiated using the protocol above.

With this in mind, we give informal arguments that correctness and security hold for the protocol. For correctness, notice that C indeed receives $\widetilde{y_i} = (G(x_i)^{r_i})^k$ from S, for each $i \in [m]$. Therefore, computing $y_i = \widetilde{y_i}^{1/r_i}$ gives $y_i = G(x_i)^k$ by the commutativity of the exponents. This is indeed the output of the evaluation $\mathsf{prf.Eval}(\mathsf{pp}, k, G(x_i))$, where $G(x_i)$ simply maps x_i to an element of $\mathbb{G}$.

For security, we essentially require that S cannot learn the input $G(x_i)$. However, note that C sends a blinded version $G(x_i)^{r_i}$ to S. Again, since $r_i \leftarrow_\$ \mathbb{Z}_p$ is randomly sampled, then $G(x_i)^{r_i}$ is randomly distributed in the cyclic group $\mathbb{G}$. As a result, it is easy to simulate this message for S without knowledge of $G(x_i)$. Secondly, we require that the client does not learn the value k which is true by the DL assumption (Definition II;3.1).

This VOPRF is almost identical (up to the batching of the DLEQ proof) to the design of Jarecki et al. [194]; hence, we point the reader to their paper for an idea of how the proof of security would be written. Our eventual authentication scheme has more expressive security properties that we discuss in Section VI;3.

Relation to blind authentication. The important takeaway point from the EC-VOPRF protocol is that it mimics the first stage of blind signature or MAC mechanisms, such as those proposed in [56, 81, 82, 147, 152, 155, 261, 269, 273]. For example, C eventually receives an elliptic curve point raised to the power of k — the server's signing key. Therefore, an EC-VOPRF protocol appears to provide a signing mechanism, but without any method for verification. In Section VI;3

we provide a method of (symmetric) verification that does not link any information to a particular signing phase. This ensures the anonymity of the user.

VI;2.6 Stateful data storage

Our protocol in Section VI;3 will rely on a stateful data storage mechanism. That is, the protocol will maintain storage for checking data across multiple invocations. This is necessary to prevent malicious activity across protocol instantiations. In particular, we look to protect phases of the protocol against adversaries that re-use inputs in more than one invocation. We give a formal definition of the functionality expected by such a data structure in Definition VI;2.3.

Definition VI;2.3 [Data store]

Let the tuple $\mathcal{D} = (\mathsf{Store}, \mathsf{Check})$ be a *data store*, define the algorithms as below.

- $\mathcal{D}.\mathsf{Store}(x)$: stores an item x.
- $\mathcal{D}.\mathsf{Check}(x)$: returns a bit $b \in \{0, 1\}$.

Let λ be the security parameter, we require that the following properties to hold:

- $\Pr[1 \leftarrow \mathcal{D}.\mathsf{Check}(x) | \mathcal{D}.\mathsf{Store}(x)] = 1$;
- $\Pr[0 \leftarrow \mathcal{D}.\mathsf{Check}(x) | !\mathcal{D}.\mathsf{Store}(x)] = \mathsf{negl}(\lambda)$.

The second property holds if x has never been stored in the data structure.

Bloom filters fulfil the requirements above (see Section III;3.1). The statefulness of the data storage mechanism is determined by the protocol that makes use of it. That is, if the contents of the data storage are carried over multiple protocol instantiations.

VI;2.7 CAPTCHAs

Later in this section, we consider CAPTCHAs (Completely Automated Public Turing Tests to Tell Computers and Humans Apart) as a plausible test that humans can solve in order to prove that they are not a bot. Such tests were first introduced by von Ahn et al. [3] and are commonly used when browsing the internet. This allows website providers to prevent bots from accessing web content. Typically requests from bots are linked to malicious behaviour such as spam and distributed denial of service attacks.

The work of [48] gave a formalisation of CAPTCHAs for the goal of designing human proof-of-work schemes with provable guarantees. The underlying characteristics were also captured in the formalisation of [184]. In this work, we will think of a CAPTCHA as a scheme with the following characteristics:

- $CHL \leftarrow \mathsf{CAPTCHA.Gen}()$: generates a CAPTCHA CHL;
- $SOLN \leftarrow \mathsf{CAPTCHA.Solve}(CHL)$: outputs a possible solution $SOLN$ to the challenge CHL;
- $\mathsf{Y/N} \leftarrow \mathsf{CAPTCHA.Verify}(SOLN)$: verifies a CAPTCHA solution, and outputs Y if it is correct (N otherwise).[11]

We do not place any formal guarantees on $\mathsf{CAPTCHA}$, except that we assume that no PPT algorithms can run $SOLN \leftarrow \mathsf{CAPTCHA.Solve}(CHL)$ such that $\mathsf{CAPTCHA.Verify}(SOLN)$ returns Y whp. In this work, we will require that $\mathsf{CAPTCHA.Solve}(CHL)$ can only be run by a human user. When it is ran, and if $\mathrm{Y} \leftarrow \mathsf{CAPTCHA.Verify}(SOLN)$, we say that $\mathsf{CAPTCHA}$ has been 'solved'.

VI;3 Anonymous authentication protocol

The core cryptographic component of this chapter is the anonymous authentication protocol that we propose. The protocol is built upon the foundations provided by the EC-VOPRF protocol in Construction VI;2.3.

Our protocol can be split into two phases, a *signing phase* and a *redemption phase*. In the signing phase of our protocol, the client receives a number of 'signed' tokens from the server — these are essentially outputs from the EC-VOPRF construction. In the redemption phase, the client constructs authentication messages that the server verifies. If verification passes, then the client is authenticated to the server. By anonymous authentication, we mean that any redemption phase cannot be linked to any initial signing phase by the server, i.e. it preserves the privacy of the client. We enshrine this security requirement along with some others into provable guarantees shortly. Our final design is similar to those in the works of [194, 195, 252] (especially the 2HashDH-NIZK construction of [194]), though we incorporate more efficient batched DLEQ proofs.

Our construction is fairly specific and so we choose not to formalise the requirements for a generic anonymous authentication protocol at this point. Instead, we give some security requirements that we believe to be important for such a protocol to achieve, see Section VI;3.2 for more details.

[11] We tacitly avoid including CHL as input since it is implied

Our construction. Let $m \in \mathbb{Z}$ be some integer, and let $\widehat{\mathsf{DLEQ}}_m$ be a batched DLEQ proof for m elements. Let λ be the security parameter, and let $\mathbb{G}$ be a multiplicative, cyclic group of prime order $p = \mathsf{poly}(\lambda)$ with a generator denoted by g. The secret key is chosen as an element $k \leftarrow\!\!\$\, \mathbb{Z}_p$ and a 'public key' is computed as $h = g^k$. We assume that $(\mathbb{G}, p, (g, h))$ are essentially acquired from the common-reference string $\mathsf{crs} \leftarrow\!\!\$\, \widehat{\mathsf{DLEQ}}_m.\mathsf{Setup}(1^\lambda, p, k)$. Let H_1, H_2 be two hash functions modelled as random oracles: $H_1 : \{0,1\}^\lambda \mapsto \mathbb{G}$ hashes into the non-identity elements $\mathbb{G}^*$ of the group $\mathbb{G}$, $H_2 : \{0,1\}^\lambda \times \mathbb{G} \mapsto \{0,1\}^\kappa$ hashes into strings of length $\kappa = \kappa(\lambda)$. Let mac denote a secure MAC algorithm (Definition II;4.9). We then set

$$\mathsf{pp} = ((\mathbb{G}, g, h), H_1, H_2, \mathsf{mac}, \mathsf{aux}),$$

for some arbitrary auxiliary data aux.

In what follows, we will denote the entire protocol construction by Γ. We will then partition Γ into signing and redemption phases and define a setup algorithm that outputs the necessary public parameters from above. Finally, we provide $\mathsf{C}[x]$ as input to a protocol step to denote that the client participates with the input x. The same notation also applies to the server.

Setup phase. The setup phase is trusted by both participants, C and S, and outputs necessary parameters for the subsequent phases of the protocol. We detail the design of the setup procedure in Construction VI;3.1.

Construction VI;3.1 [Setup]

Let $m \in \mathbb{Z}$ be some integer, let $p > 0$ be some large prime, let λ be the security parameter and let $\kappa = \mathsf{poly}(\lambda)$. Let $\widehat{\mathsf{DLEQ}}_m$ be a batched DLEQ proof for m elements.
Define $(\mathsf{pp}, k) \leftarrow\!\!\$\, \Gamma.\mathsf{Setup}(1^\lambda, 1^\kappa, m, p)$ to do the following:

- sample $k \leftarrow\!\!\$\, \mathbb{Z}_p$;
- run $(\mathbb{G}, p, (g, h)) = \mathsf{crs} \leftarrow\!\!\$\, \widehat{\mathsf{DLEQ}}_m.\mathsf{Setup}(1^\lambda, p, k)$;
- sample some auxiliary data $\mathsf{aux} \leftarrow \{0,1\}^*$;
- let $H_1 : \{0,1\}^\lambda \mapsto \mathbb{G}$ and $H_2 : \{0,1\}^\lambda \times \mathbb{G} \mapsto \{0,1\}^\kappa$ be random oracles;
- let mac be the description of a MAC scheme;
- let $\mathsf{pp} = (\mathbb{G}, p, (g, h), H_1, H_2, \mathsf{mac}, \mathsf{aux})$;
- output (pp, k).

It is assumed that both C and S will receive pp, and only S will receive k.

```
$\psi^1_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}])$
1 :  $\varphi \leftarrow \emptyset$;
2 :  $\{r_i\}_{i\in[m]} \leftarrow\$ \mathbb{Z}_p$;
3 :  for $i \in [m]$ :
4 :      $\widetilde{x_i} \leftarrow H_1(x_i)^{r_i}$;
5 :      $\varphi[i] \leftarrow (x_i, r_i)$;
6 :  $\mathsf{S} \leftarrow \{\widetilde{x_i}\}_{i\in[m]}$;
7 :  return $\varphi$;
```

```
$\psi^2_{\mathsf{sign}}(\mathsf{pp}, \mathsf{S}[\{\widetilde{x_i}\}_{i\in[m]}, k], \widehat{\mathsf{DLEQ}}_m)$
1 :  $\mathcal{W} \leftarrow \emptyset$;
2 :  for $i \in [m]$ :
3 :      $\widetilde{y_i} \leftarrow \widetilde{x_i}^k$;
4 :      $\mathcal{W}[i] = \widetilde{y_i}$;
5 :  $\pi \leftarrow \widehat{\mathsf{DLEQ}}_m.\mathbb{P}(\mathsf{pp}, \{(\widetilde{x_i}, \widetilde{y_i})\}_{i\in[m]}, k)$;
6 :  $\mathsf{C} \leftarrow (\mathcal{W}, \pi)$;
```

Figure VI;3: Left: Step one of the signing phase, initiated by C. We sometimes refer to the values in the set $\{\widetilde{x_i}\}_{i\in[m]}$ as 'blinded' tokens.
Right: Step two of the signing phase, initiated by S.
These steps are essentially identical to those in Figure VI;1.

```
$\psi^3_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\varphi, (\mathcal{W}, \pi)], \widehat{\mathsf{DLEQ}}_m)$
1 :  $\mathcal{R} \leftarrow \emptyset$;
2 :  $\{y_i\}_{i\in[m]} \leftarrow \mathcal{W}$;
3 :  $b \leftarrow \widehat{\mathsf{DLEQ}}_m.\mathbb{V}(\mathsf{pp}, \{(x_i, y_i)_{i\in[m]}\}, \pi)$;
4 :  if $b \neq 1$ :
5 :      return $\perp$;
6 :  for $i \in [m]$ :
7 :      $(x_i, r_i) \leftarrow \varphi[i]$;
8 :      $y_i \leftarrow \widetilde{y_i}^{1/r_i}$;
9 :      $\mathcal{R}[i] = (x_i, y_i)$;
10 : return $\mathcal{R}$;
```

Figure VI;4: Step three of the signing phase, initiated by C. There is no communication with S required in this step. It differs slightly with Figure VI;2 since the output $\mathcal{R}$ also holds x_i.

Signing phase. The signing phase of our protocol Γ is covered in Construction VI;3.2. We refer to this stage as signing though this should not be confused with a digital signature scheme.

Construction VI;3.2 [Signing phase]

Let m, p, λ, κ be defined as previously, and let $\widehat{\mathsf{DLEQ}}_m$ be defined as in Construction VI;2.2. Let $(\mathsf{pp}, k) \leftarrow\!\$\ \Gamma.\mathsf{Setup}(1^\lambda, 1^\kappa, m, p)$ be as in Construction VI;3.1. Let ψ_{sign} be the protocol between a client $\mathsf{C}[\{x_i\}_{i\in[m]}]$ and a server $\mathsf{S}[k]$, comprised of the following steps.

1. $(\mathsf{C} : \varphi, \mathsf{S} : \{\widetilde{x}_i\}_{i\in[m]}) \leftarrow \psi^1_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}])$ [Figure VI;3:Left].
2. $(\mathsf{C} : (\mathcal{W}, \pi)) \leftarrow \psi^2_{\mathsf{sign}}(\mathsf{pp}, \mathsf{S}[\{\widetilde{x}_i\}_{i\in[m]}, k], \widehat{\mathsf{DLEQ}}_m)$ [Figure VI;3:Right].
3. $(\mathsf{C} : \mathcal{R}) \leftarrow \psi^3_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\varphi, (\mathcal{W}, \pi)], \widehat{\mathsf{DLEQ}}_m)$ [Figure VI;4].

In summary, we write:

$$\mathcal{R} \leftarrow \Gamma.\psi_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}], \mathsf{S}[k])$$

to indicate a successful signing phase between C and S. If the signing phase is unsuccessful, then the output will be $\perp$ instead.

Notice, the similarities between the signing phase and the EC-VOPRF design of Construction VI;2.3. The only differences are that C now maintains a list $\mathcal{R}$ of future 'passes' which it will use in the redemption phase (Construction VI;3.3). We may also refer to the final output $\mathcal{R}$ as a set of 'tokens'.

Redemption phase. The redemption phase of our protocol is covered in Construction VI;3.3. For this redemption phase, we require some stateful data store $\mathcal{D}$ (Definition VI;2.3). This data store is initialised before any redemption phases are initiated and is carried over multiple invocations.

```
$\psi^1_{\mathsf{red}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}, \mathcal{R}])$
1 : $(x, y) \leftarrow \mathcal{R}.\mathsf{pop}()$;
2 : $K_x \leftarrow H_2(x, y)$;
3 : $\mathsf{data} \leftarrow x\|\mathsf{aux}$;
4 : $\tau \leftarrow \mathsf{mac.Tag}(K_x, \mathsf{data})$;
5 : $\mathsf{S} \leftarrow (x, \tau)$;
```

```
$\psi^2_{\mathsf{red}}(\mathsf{pp}, \mathsf{S}[(x, \tau), k], \mathcal{D})$
1 : $b \leftarrow \mathcal{D}.\mathsf{Check}(x)$;
2 : if $b \neq 1$ :
3 :    $\mathsf{C} \leftarrow 0$;
4 : else :
5 :    $y' \leftarrow H_1(x)^k$;
6 :    $K'_x \leftarrow H_2(x, y')$;
7 :    $\mathcal{D}.\mathsf{Store}(x)$;
8 :    $\mathsf{C} \leftarrow \mathsf{mac.Verify}(K'_x, x, \tau)$;
```

Figure VI;5: LEFT: Step one of the redemption phase, initiated by C.
RIGHT: Step two of the redemption phase, initiated by S. We say that C is successfully authenticated if $1 \rightarrow \mathsf{C}$ during ψ^2_{red}.

Construction VI;3.3 [Redemption phase]

Let $m, p, \lambda, \mathsf{mac}, \kappa, \mathcal{D}$ be defined as previously. Let $(\mathsf{pp}, k) \leftarrow \Gamma.\mathsf{Setup}(1^\lambda, 1^\kappa, m, p)$. Let $\{x_i\}_{i\in[m]}$ be a set of inputs taken from $\{0,1\}^\lambda$, s.t. $\mathcal{R} \leftarrow \Gamma.\psi_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}], \mathsf{S}[k])$.
Let ψ_{red} be the redemption phase of the protocol Γ between a client $\mathsf{C}[\{x_i\}_{i\in[m]}, \mathcal{R}]$ and a server $\mathsf{S}[k]$. The *redemption phase* is comprised of the following steps.

1. $(\mathsf{S} : (x, \tau)) \leftarrow \psi^1_{\mathsf{red}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}, \mathcal{R}])$ [Figure VI;5:LEFT].

2. $(\mathsf{C} : b) \leftarrow \psi^2_{\mathsf{red}}(\mathsf{pp}, \mathsf{S}[(x, \tau), k], \mathcal{D})$ [Figure VI;5:RIGHT].

In summary, we write:

$$b \leftarrow \Gamma.\psi_{\mathsf{red}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}, \mathcal{R}], \mathsf{S}[k], \mathcal{D}),$$

to indicate the response, b, received during a redemption phase between C and S. We say that $b = 1$ if the client is authenticated successfully, and $b = 0$ if unsuccessful.

VI;3.1 Correctness

We first argue that the protocol achieves the functionality that it sets out to. That is, for a set of tokens $\mathcal{R}$ obtained from ψ_{sign}, then any of these tokens can be used to authenticate successfully in ψ_{red}.

Theorem VI;3.1 [Correctness]

Let ψ_{sign} be the protocol defined in Construction VI;3.2, and let ψ_{red} be the protocol defined in Construction VI;3.3. Then we have that:

$$\Pr\left[1 \leftarrow \psi_{\mathsf{red}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}, \mathcal{R}], \mathsf{S}[k], \mathcal{D})\Big|_{\mathcal{R}\leftarrow\psi_{\mathsf{sign}}(\mathsf{pp},\mathsf{C}[\{x_i\}_{i\in[m]}],\mathsf{S}[k])}^{\{x_i\}_{i\in[m]}\in\{0,1\}^\lambda,\, k\in\mathbb{Z}_p,}\right] = 1,$$

for all $j \in [m]$ and all choices of inputs $\{x_i\}_{i\in[m]} \in \{0,1\}^\lambda$ and keys $k \in \mathbb{Z}_p$.

Proof. Let $\widetilde{x_i} = H_1(x_i)_i^r$ denote the blinded inputs that C sends to S in ψ_{sign}^1. When S receives $\widetilde{x_i}$, it first computes $\widetilde{y_i} = \widetilde{x_i}^k$. It then computes:

$$\pi \leftarrow \widehat{\mathsf{DLEQ}}_m.\mathbb{P}((\mathbb{G}, p, (g,h)), \{\widetilde{x_i}, \widetilde{y_i}\}_{i\in[m]}, k),$$

and sends $(\{\widetilde{y_i}\}_{i\in[m]}, \pi)$ back to C in ψ_{sign}^2.

Since π is the output of $\widehat{\mathsf{DLEQ}}_m.\mathbb{P}$, then $b \leftarrow \widehat{\mathsf{DLEQ}}_m.\mathbb{V}((\mathbb{G}, p, (g,h)), \{\widetilde{x_i}, \widetilde{y_i}\}_{i\in[m]})$ and $b = 1$ (by the completeness of $\widehat{\mathsf{DLEQ}}_m$). Therefore, the client computes $y_i \leftarrow \widetilde{y_i}^{1/r_i}$ and stores each pair (x_i, y_i) and the signing phase is complete.

For the redemption phase, C computes a shared key $K_{x_i} \leftarrow H_2(x_i, y_i)$ for mac, computes the MAC tag $\tau_i \leftarrow \mathsf{mac.Tag}(K_{x_i}, \mathsf{data})$ (where $\mathsf{data} = x_i \| \mathsf{aux}$ for $\mathsf{aux} \in \mathsf{pp}$) and sends (x_i, τ_i) to S in ψ_{red}^1. Given x_i, S calculates $H_1(x_i)^k = y_i$. S computes the same derived key $K_{x_i} \leftarrow H_2(x_i, H_2(x_i)^k)$ and the same tag τ since aux is used by both C and S. As such, MAC verification returns $1 \leftarrow \mathsf{mac.Verify}(K_{x_i}, x_i, \tau)$ with probability 1, and correctness is ensured. □

VI;3.2 Security

For considering the security of our protocol, proving meaningful guarantees using the approaches of secure computation seems difficult due to the fact that we have to consider security over different protocol instantiations and over multiple invocations. That is, the anonymity guarantee that is central to our work is only salient when considered over numerous redemption phases wrt one signing phase. Moreover, the guarantees only have utility in a setting where there have been numerous invocations of both protocols to guarantee that the anonymity sets are large enough.

```
$\mathsf{exp}^{\mathsf{ul}}_{b,\mathcal{A}}(1^\lambda, 1^\kappa, m, p)$
1 : $(\mathsf{pp}, k) \leftarrow\!\!\$\; \Gamma.\mathsf{Setup}(1^\lambda, 1^\kappa, m, p);$
2 : $\{x_i^{(0)}\}_{i\in[m]} \leftarrow\!\!\$\; (\{0,1\}^\lambda)^m;$
3 : $\{x_i^{(1)}\}_{i\in[m]} \leftarrow\!\!\$\; (\{0,1\}^\lambda)^m;$
4 : $\mathcal{R}_0 \leftarrow \Gamma.\psi_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\{x_i^{(0)}\}_{i\in[m]}], \mathcal{A}[k]);$
5 : $\mathcal{R}_1 \leftarrow \Gamma.\psi_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\{x_i^{(1)}\}_{i\in[m]}], \mathcal{A}[k]);$
6 : for $i \in [m]$ :
7 :     $c_i \leftarrow \Gamma.\psi_{\mathsf{red}}(\mathsf{C}[\{x_i^{(b)}\}_{i\in[m]}, \mathcal{R}_b], \mathcal{A}[k]);$
8 : $b_{\mathcal{A}} \leftarrow \mathcal{A}(1^\lambda, k, \mathcal{V}^{\mathcal{A}}_{\mathsf{sign}}(\{x_i^{(b)}\}_{i\in[m]}, k), \mathcal{V}^{\mathcal{A}}_{\mathsf{red}}(\{x_i^{(b)}\}_{i\in[m]}, \mathcal{R}_b, k));$
9 : return $b_{\mathcal{A}}$;
```

Figure VI;6: Decisional experiments for characterising the unlinkability of the protocol Γ. We use $\mathcal{A}$ to denote the adversarial server.

It is plausible that we could consider the security of the protocol in a universal composability setting [75] instead, but this would require a much more complex security analysis, as in [194].

We know exactly what security goals that we need to achieve for our scheme, due to the specific choice of application. As a consequence, we choose to prove that these properties hold individually, without delving into generic security models. These properties can be categorised as: (1) unlinkability; (2) resistance to one-more-token attacks; (3) key-consistency. We believe that this provides a clearer interface, with which we can represent the security properties fulfilled by our protocol.

Unlinkability

The notion of *unlinkability* captures the inability of an adversarial server S to link together a redemption phase of the protocol to any individual signing phase. A protocol satisfying unlinkability is necessary for meeting the anonymity guarantees that we require. This is the primary goal of our scheme.

We use $\mathcal{V}^{\mathsf{S}}_{\mathsf{sign}}[\{\widetilde{x}_i\}_{i\in[m]}, k]$ to denote the view that S witnesses during the signing phase

$$\mathcal{R} \leftarrow \Gamma.\psi_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}], \mathsf{S}[k]);$$

and likewise $\mathcal{V}^{\mathsf{C}}_{\mathsf{sign}}[\{x_i\}_{i\in[m]}]$ for the client C. We denote the views of the corresponding redemption phases for the server and the client by $\mathcal{V}^{\mathsf{S}}_{\mathsf{red}}[\mathcal{R}, k]$ and $\mathcal{V}^{\mathsf{C}}_{\mathsf{red}}[\mathcal{R}]$, similarly.

Definition VI;3.1 [*Unlinkability*]

Let pp,κ,m,p,mac,$\mathcal{D}$,$\mathsf{C}[\{x_i\}_{i\in[m]}]$ and $\mathsf{S}[k]$ be defined as previously. Let $\mathcal{R} \leftarrow \Gamma.\psi_{\mathsf{sign}}(\mathsf{C}[\{x_i\}_{i\in[m]}], \mathsf{S}[k])$ refer to the signing phase in Construction VI;3.2 of Γ. The redemption phase of Γ in Construction VI;3.3, is referred to by $b \leftarrow \Gamma.\psi_{\mathsf{red}}(\mathsf{pp}, \mathsf{C}[x], \mathsf{S}[k], \mathcal{D})$.
We say that Γ has *unlinkable* signing and redemption phases, if

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{ul}(1^\lambda, \Gamma))) < \mathsf{negl}(\lambda),$$

where $\mathcal{A}$ is any PPT algorithm and $\mathsf{exp}^{\mathsf{ul}}_{b,\mathcal{A}}(1^\lambda, 1^\kappa, m, p)$ is defined as in Figure VI;6.

We now give a proof that Γ unconditionally satisfies unlinkability. Intuitively, this follows since the view of $\mathcal{A}$ in the signing phase is made up of uniformly distributed elements.

Theorem VI;3.2 [Proof of unlinkability]

Γ is unconditionally unlinkable, i.e. $\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{ul}(1^\lambda, \Gamma))) = 0$ in the random oracle model.

Proof. First, consider the opposing views $\mathcal{V}^{\mathcal{A}}_{\mathsf{sign}}(\{x_i^{(b)}\}_{i\in[m]}, k)$ from the signing phase. These are the lists:

$$\{\mathsf{pp}, k, \{\widetilde{x_i}^{(c)}\}_{i\in m}, \mathcal{R}_b\}$$

for $b \in \{0, 1\}$. The only difference is therefore the set $\{\widetilde{x_i}^{(b)}\}_{i\in m}$, since $\mathcal{R}_b$ is also wholly dependent on this set. Note that each $\widetilde{x_i}^{(b)} = H_1(x_i^{(b)})^{r_i}$ for some $r_i \leftarrow\!\!\$\ \mathbb{Z}_p$. Since H_1 is a random oracle, then $H_1(x_i^{(b)})$ is randomly distributed in $\mathbb{G} \setminus \{1_{\mathbb{G}}\}$ (for the identity element $1_{\mathbb{G}}$). Moreover, the exponent r_i is sampled uniformly and only known to C. Therefore, each of these elements is distributed uniformly.

Secondly, consider the view

$$\mathcal{V}^{\mathcal{A}}_{\mathsf{red}}(\{x_i^{(b)}\}_{i\in[m]}, \mathcal{R}_b, k) = \{\mathsf{pp}, k, (\{x_i^{(b)}\}_{i\in[m]}, \{\tau_i^{(b)}\}_{i\in[m]})\},$$

received by $\mathcal{A}$, where $\tau_i^{(b)}$ is the MAC tag associated with $x_i^{(b)}$. Then, it is clear that each $\tau_i^{(b)}$ is only dependent on the other inputs in the view (since it is computed over only those values). As a result, the only values that are not already known by the server are in the set $x_i^{(b)}$. However, each $\widetilde{x_i}^{(b)}$ remains uniformly distributed — even in the presence of $x_i^{(b)}$ — due to the exponentiation with r_i, which is unknown, and since $\mathbb{G}$ is cyclic.

Therefore, both of $\widetilde{x_i}^{(b)}$, for $b \in \{0,1\}$, are uniformly distributed wrt to the view of $\mathcal{A}$ in the redemption phase. Thus unlinkability is ensured. □

One-more-token security

The notion of *one-more-token* security enshrines the inability of the client to access the secret signing key of the server in a 'meaningful' way. By meaningful, we mean being able to use knowledge of signed tokens, received during a signing phase, to sign more tokens. Intuitively, this property ensures the utility of the protocol, in that the clients cannot forge signed tokens that are not received during the signing phase.

Definition VI;3.2 [One-More-Token security]

Our scheme satisfies *one-more-token security* if a PPT client, after receiving m signed tokens from the server, cannot successfully redeem $m+1$ tokens, except with negligible probability in the random oracle model.

To prove that Γ achieves one-more-token security, we construct a reduction from the one-more-decryption security of the El Gamal encryption scheme (Assumption II;5.2) in Theorem VI;3.3 below. This is possible since the crs from the DLEQ proof mechanism is distributed in the same way as an El Gamal public key (Construction II;5.1).

Theorem VI;3.3 [One-more-token security]

If the one-more-decryption security (Definition II;5.2) of El Gamal holds, then Γ demonstrates one-more-token security.

Proof. Consider an adversarial client $\mathcal{B}$ that can get m tokens signed by the server in the signing phase $\mathcal{R} \leftarrow \Gamma.\psi_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}], \mathsf{S}[k])$, and then redeem $m+1$ tokens during the redemption phase $\Gamma.\psi_{\mathsf{red}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}, \mathcal{R}], \mathsf{S}[k], \mathcal{D})$. We will produce an adversary $\mathcal{A}$ who wins the one-more-decryption game against El Gamal.

$\mathcal{A}$ starts the game by being given $\mathbb{G}$, q, g, h, and $\{(C_i, D_i)\}_{i\in[m+1]}$ in $\mathsf{exp}^{\mathsf{omd}}_{\mathcal{A}}(1^\lambda, m)$ (Figure II;13). $\mathcal{A}$ plays the role of the server, and initiates the adversarial client $\mathcal{B}$. It uses $(\mathbb{G}, p, (g, h))$ as the group description (crs), where $h = g^k$ is the commitment to the signing key in the protocol. $\mathcal{A}$ also selects a random permutation π on $\mathbb{Z}_p$, but does not reveal it to $\mathcal{B}$.

$\mathcal{B}$ can make two kinds of queries: polynomially many random oracle queries to H_1, H_2, and $\widehat{H}$, and at most m token signing queries. Here, $\widehat{H}$ refers to the hash function that is used in the invocation of $\widehat{\mathsf{DLEQ}}_m$.

When $\mathcal{B}$ makes a token signing query with blinded token $\widetilde{x}_i$, $\mathcal{A}$ selects a random $F_i \leftarrow\!\!\$\, \mathbb{G}$, and sends the ciphertext $(\widetilde{x}_i, F_i)$ to the decryption oracle, which returns $M_i = F_i/\widetilde{x}_i{}^k$. $\mathcal{A}$ then returns $F_i/M_i = \widetilde{x}_i{}^k$ to $\mathcal{B}$. It provides a forged DLEQ proof, which it can do because it can program the responses of $\widehat{H}$ and by simulating the CRS.

When $\mathcal{B}$ makes a random oracle request for $H_1(t)$, $\mathcal{A}$ replies with $\prod_{j\in[m+1]} C_j^{\left(a^{j-1}\right)}$, where $a = \pi(t)$. Only with negligible probability $1/p$ will this value be the identity element of $\mathbb{G}$, in which case $\mathcal{A}$ aborts with failure.

When $\mathcal{B}$ makes a random oracle request for $H_2(t, B)$, $\mathcal{A}$ stores (t, B, Z) in a table for a randomly selected Z, and replies with Z. If the same $H_2(t, B)$ is queried again, the same Z will be returned.

When $\mathcal{B}$ makes a random oracle request for $\widehat{H}$, $\mathcal{A}$ programs it to forge the DLEQ proof, as above.

At the end of the game, $\mathcal{B}$ will redeem $m+1$ tokens $\{(t_i, \mathsf{mac.Tag}(H_2(t_i, B_i), t||\mathsf{aux}))\}_{i\in[m+1]}$. $\mathcal{A}$ uses the H_2 table to look up the value of B_i used to generate the MAC key in each token, and it will be the case that $B_i = H_1(t_i)^k = \prod_{j\in[m+1]} \left(C_j^k\right)^{\left(\pi(t_i)^{j-1}\right)}$ for each $i \in [m+1]$.

If $\boldsymbol{V}$ is the (Vandermonde, and thus invertible) matrix with $v_{ij} = \pi(t_i)^{j-1}$, and $\boldsymbol{U} = \boldsymbol{V}^{-1}$, then it will be the case that $\prod_{j\in[m+1]} B_j^{u_{ij}} = C_i^k$ for each $i \in [m+1]$. $\mathcal{A}$ then outputs $M_i = D_i/C_i^k$ for each $i \in [m+1]$ and wins the game. $\square$

Key-consistency

The final security property that we consider is known as *key-consistency*. Consider a single invocation of the setup $(\mathsf{pp}, k) \leftarrow\!\!\$\, \Gamma.\mathsf{Setup}(1^\lambda, 1^\kappa, m, p)$. We show that it is computationally infeasible for an adversarial server to initiate a signing phase with a different key $k' \neq k$ s.t. that the protocol completes. In particular, this property relies on the soundness of the batched DLEQ proof that we use.

The reason that we need to enforce this property is due to the fact that a plausible attack for deanonymising clients would be to initiate each signing phase with a different key pair. Then, during the redemption phase, it would become obvious which tokens had been signed during which signing phase. By forcing the key to be the same throughout each signing phase, we can enforce the unlinkability requirement in Definition VI;3.1. Our technique requires a trusted setup phase where the server commits to its own key. It then is forced to prove that each signing phase uses the same key as that is use din the commitment phase.

```
$\mathsf{exp}^{\mathsf{kc}}_{\mathcal{A}}(1^\lambda, 1^\kappa, m, p)$
1 : $(\mathsf{pp}, k) \leftarrow\$ \Gamma.\mathsf{Setup}(1^\lambda, 1^\kappa, m, p);$
2 : $\{x_i\}_{i\in[m]} \leftarrow\$ (\{0,1\}^\lambda)^m;$
3 : $k' \leftarrow\$ \mathcal{A}(1^\lambda, \mathsf{pp}, k);$
4 : $T \leftarrow \Gamma.\psi_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}], \mathcal{A}[k']);$
5 : if $T \neq \perp$:
6 :     return $1$;
7 : else :
8 :     return $0$;
```

Figure VI;7: Computational experiment for characterising the key-consistency of the protocol Γ.

Definition VI;3.3 [Key-consistency]

Let κ, p, $\mathsf{C}[\{x_i\}_{i\in[m]}]$ and $\mathsf{S}[k]$ be defined as previously.
We say that Γ demonstrates *key-consistency* if:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{kc}(1^\lambda, \Gamma))) < \mathsf{negl}(\lambda),$$

for all PPT adversarial servers $\mathcal{A}$, where $\mathsf{exp}^{\mathsf{kc}}_{\mathcal{A}}(1^\lambda, 1^\kappa, m, p)$ is defined as in Figure VI;7.

Theorem VI;3.4 [Key-consistency of Γ]

Under the computational soundness of $\widehat{\mathsf{DLEQ}}_m$, then Γ satisfies key-consistency.

Proof. The proof follows almost trivially by considering an adversary $\mathcal{B}$ looking to break the computational soundness of $\widehat{\mathsf{DLEQ}}_m$.

Firstly, let $\mathcal{A}$ be the adversary initiated by $\mathcal{B}$ that breaks $\mathsf{exp}^{\mathsf{kc}}_{\mathcal{A}}(1^\lambda, 1^\kappa, m, p)$ with advantage ϵ. Then, $\mathcal{B}$ constructs $\mathsf{pp} = (\mathbb{G}, p, (g, h), H_1, H_2, \mathsf{mac}, \mathsf{aux})$ by letting

$$(\mathbb{G}, p, (g, h)) = \mathsf{crs} \leftarrow \widehat{\mathsf{DLEQ}}_m.\mathsf{Setup}(1^\lambda, p, k)$$

where crs is the common-reference string that $\mathcal{B}$ receives, for the challenger's randomly chosen key $k \leftarrow\$ \mathbb{Z}_p$. The rest of the public parameters can just be instantiated as in $\Gamma.\mathsf{Setup}(1^\lambda, 1^\kappa, m, p)$.

Then, $\mathcal{B}$ simply plays the role of the client in $\Gamma.\psi_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}], \mathsf{S}[k])$ for some choice of input $\{x_i\}_{i\in[m]} \in (\{0,1\}^\lambda)^m$. Eventually, $\mathcal{A}$ responds with $(\mathcal{W}, \pi)$ where π corresponds to an attempted forged proof. Then, $\mathcal{B}$ simply responds with π to their soundness challenger, and wins with probability ϵ, corresponding to the probability that $\mathcal{B}$ accepts the signed tokens from $\mathcal{A}$.

By the soundness of $\widehat{\mathsf{DLEQ}}_m$, we therefore note that $\epsilon < \mathsf{negl}(\lambda)$ and the proof is complete. $\square$

VI;3.3 Other security notions

In this section, we have demonstrated that our protocol is secure wrt three security properties that we have pinpointed as key security goals for our construction. We decided on these goals as our main aim was to provide an anonymous mechanism for clients to authenticate to servers. Therefore, the notions of unlinkability and key consistency are absolutely necessary to achieve. The notion of one-more-token security provides a security guarantee to the server that its key has not been compromised.

It is highly likely that, for different scenarios, it may be advantageous to prove extra security properties that we have not included here. Valuable future work would perform deeper analysis of our protocol and provide a more generic view of what security can hold for both the server and the client. For instance, if we could provide a generic model by which we consider the security of multiple instantiations of our protocol, in a similar manner to the security analysis of internet and messaging protocols, then we could provide more unified security guarantees. A security analysis in the universal composability framework [75] would be equally enlightening.

VI;4 Reducing internet challenges for anonymous users

In this section, we introduce the main application of our work. We focus on the case of website accessibility via content delivery networks, for users of anonymity-reserving browsing tools. For ease of discussion, we refer to this entire set of users as 'Tor users'.

VI;4.1 Content delivery networks

A content delivery network (or CDN) allows website providers to deliver content to browsing clients with far greater efficiency. Website providers are customers of the CDN, and the CDN acts as a reverse proxy to each of their customers. When a client browser attempts to reach some website `W`, they will first send a HTTP request to the CDN. The CDN will then recover the contents of `W` from their own network before returning these resources to the client. Figure VI;8 provides a diagrammatic representation.

The efficiency comes from the fact that CDNs typically distribute content globally, storing static content at numerous data centres around the world. Therefore, client requests can be served from locations that are much closer in proximity, rather than the location of the actual website provider. Since a common bottleneck of communication is the speed of light (e.g. communication from

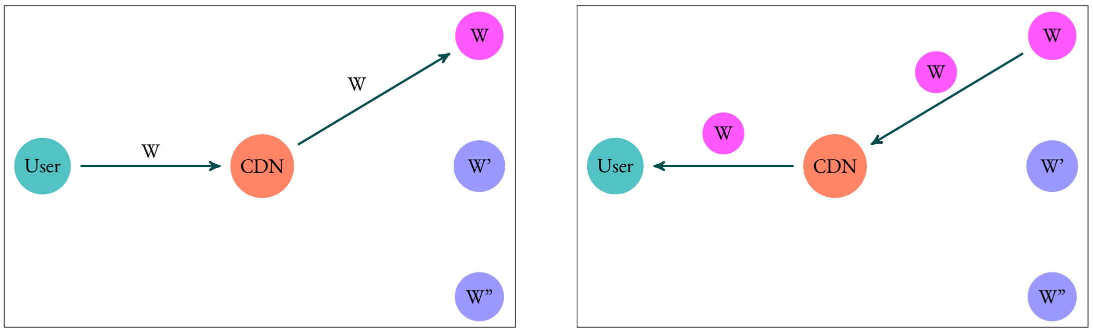


Figure VI;8: Diagrams of how CDN reverse proxy typically functions.

London to San Francisco will take at least 37ms), reducing the geographic distance between connections can greatly improve the browsing experience.

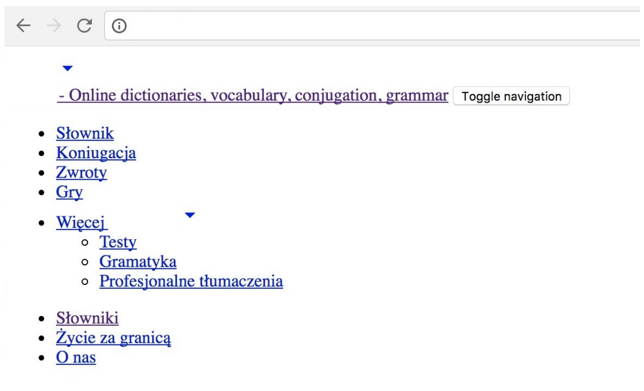
Another feature that comes with being a customer of a CDN is that knowledge derived on the edge network can be used to prevent malicious users from accessing a customer's content. This is commonly achieved using IP-based reputation challenge mechanisms for preventing those users from accessing web resources who are deemed malicious (or a bot, for example). That is, if the reputation is deemed to be poor, the CDN will ask the user to complete an extra challenge before they can gain access. We refer to customers of the CDN as CDN-protected websites. Such websites may also be referred to as 'origins'.

VI;4.2 Cloudflare

Cloudflare is a CDN that serves 10% of all internet traffic. It serves over 1 trillion requests per month, making it one of the largest CDN providers globally [98, 99].

This section introduces a case-study with the Cloudflare CDN. Recall, from the introduction that Cloudflare employs an IP-based reputation mechanism that disproportionately challenges Tor users to assert their honesty. These challenges take the form of CAPTCHAs and are required to be completed before access to protected web resources is granted.

Problems with CAPTCHAs. CAPTCHAs are generally accepted to be solvable by humans, but there are a variety of issues that can dramatically reduce accessibility in many cases. Firstly, CAPTCHAs provide a tangible annoyance within the browsing experience, potentially causing users to avoid solving them. Secondly, they may be hard to solve for users of certain disabilities (e.g. visual impairment). Thirdly, implementation and browser incompatibility issues can render CAPTCHAs unsolvable — even for humans. Moreover, if resources are not explicitly shown in the browser of the client, instead loaded from external websites that are also CDN-protected, then these will typically result in webpages that do not load correctly (see Figure VI;9). This is a result



- Online dictionaries, vocabulary, conjugation, grammar Toggle navigation

- Słownik
- Koniugacja
- Zwroty
- Gry
- Więcej
  - Testy
  - Gramatyka
  - Profesjonalne tłumaczenia
- Słowniki
- Życie za granicą
- O nas

Figure VI;9: A badly-rendered website with CSS files hosted on Cloudflare-protected domains.

of the user never having the opportunity to complete the CAPTCHA that is required, since they never navigate to the sub-resource URL directly.

Consequently, the browsing experience for Tor users can prove to be very restrictive. Indeed, since 17% of requests from the Tor network are challenged before access is granted [98], then this results in a huge number of requests having to solve challenges. The problem is magnified by the fact that cookies are not a viable alternative to bypassing these challenges. The functionality of Tor specifies that cookies and other potentially harmful data (wrt anonymity) are deleted at regular intervals. Moreover, generic cookies for multiple domains protected by a CDN could be used to track a user's activity over multiple domains — thus breaking the anonymity model.

In the next section we provide an overview of the architecture and workflow for accessing the Cloudflare network. We will then show how we can embed the protocol from Section VI;3 into this workflow, to reduce the burden of work on human users.

VI;4.3 Current workflow

To give better intuition for the problem that we are attempting to solve, we illustrate the workflow that is currently initiated between a user/client and the edge provider/CDN. As above, the model that we consider is heavily based on the architecture in the Cloudflare CDN. We will focus on illuminating the workflow for clients that are deemed to be 'malicious'.

Gauging reputation. Like many CDNs, Cloudflare uses a reputation gauging system that assigns scores based on activity witnessed from the client IP address. IP addresses are assigned malicious-reputation scores (from 0 to 100) suggesting the confidence that the IP address has been involved in malicious behaviour in the past. As an example, IP addresses that have been linked to 'bot-like' behaviour (such as spamming or DDoS attacks) will likely have a high score. However, other indicators taken from the client can also be used to change the score.

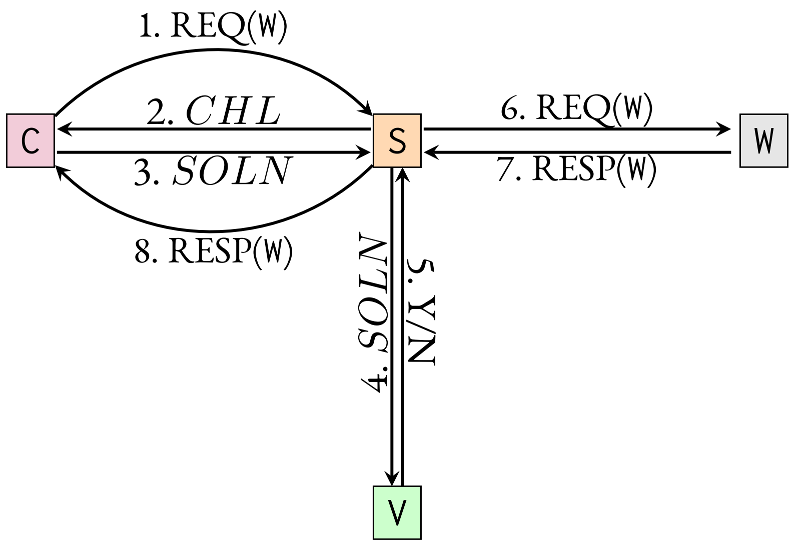


Figure VI;10: Current Cloudflare workflow for challenge system.

Deciding challenges. Let $0 < \eta < 100$ be some threshold malicious-reputation score. Simply, if a user C with reputation $\nu > \eta$ attempts to access some website protected by the edge (i.e. a customer of the edge) then C will always be shown a challenge.

While this simplification ignores other possibilities for routing requests through the CDN architecture, it helps us to generalize the workflow that we consider. Note that we are only interested in reducing the requirement for honest, human users to complete internet challenges. Other methods of request routing are not explicitly covered by our solution.

Allowing access. In Figure VI;10, we detail the workflow that occurs when a user C with reputation score $\nu_{\mathsf{C}} > \eta$ interacts with the edge provider S to access an *origin* website W. We also include a party V known as a validator; this entity validates challenge solutions for S. Here, it is assumed that S is the Cloudflare CDN, i.e. the edge server.

In the interaction, we assume that there are messages REQ(W) and RESP(W) that are used respectively for requesting and responding with content related to the origin W (mirroring HTTP requests/responses, for example). We use the notation of Section VI;2.7 for discussing CAPTCHAs: generation ($CHL \leftarrow \mathsf{CAPTCHA.Gen}()$); solutions ($SOLN \leftarrow \mathsf{CAPTCHA.Solve}(CHL)$); and validation (Y/N $\leftarrow \mathsf{CAPTCHA.Verify}(SOLN)$). In step 2, a CAPTCHA is generated and sent to the client. Step 3 requires the human client to solve CAPTCHA, this solution is sent to a 'verifier' V, that runs Y/N $\leftarrow \mathsf{CAPTCHA.Verify}(SOLN)$ to verify the solution. In the case of an incorrect solution (i.e. V returns 'N') then steps 6 onwards are not undertaken.

Remark VI;4.1. *Our solution is agnostic to the type of challenge used. We could swap out* CAPTCHA *for a different challenge of the provider's choice and the workflow would not change, as long as it still required some human intervention.*

A client in the current workflow has to compute a challenge solution for every request to S that they make. If there are multiple origins protected by S, then S could mitigate this by giving the client a method of authenticating in the future when a challenge is provided. In the HTTP setting, this is typically done using single-origin cookies. These cookies prevent a client from having to solve a challenge again for the same origin $\mathtt{W}$ for some predefined interval of time. The fact that they are single-origin means that the user cannot be tracked over their access to multiple domains. However, the same CDN serves a very large number of origins, and the client would still have to solve one challenge for each origin visited. Alternatively, S could use *cross-domain* cookies for multiple origins, although at the severe privacy cost of linking the client's requests across all origins. We view cross-domain privacy leaks as inadmissible, and do not consider them a viable solution.

VI;4.4 New workflow

In this section we show that the workflow shown in Figure VI;10 can be adapted to include the anonymous authentication protocol of Section VI;3. For a protocol step, ψ_{step}, we use the notation

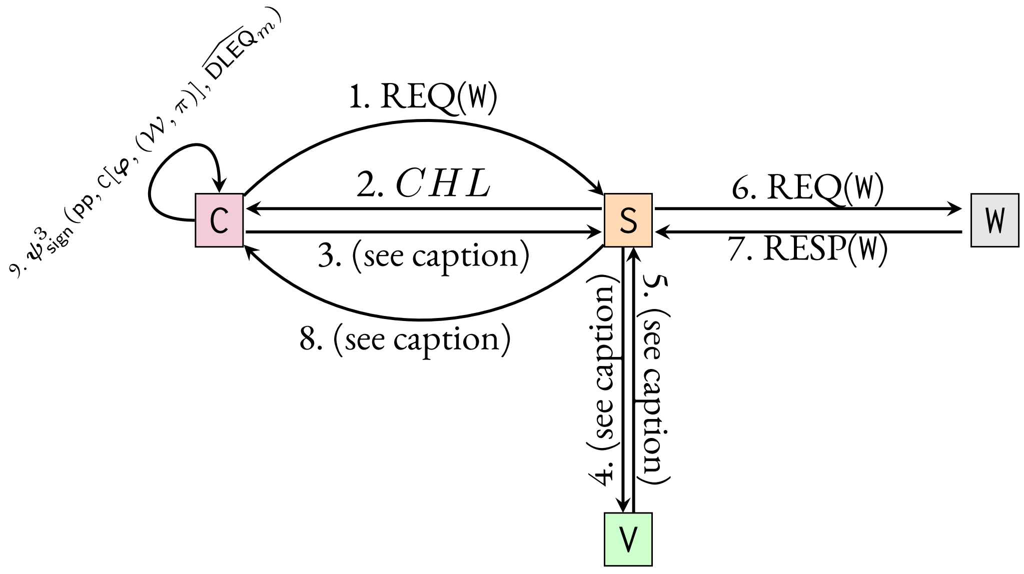
$$\mathsf{C} \xrightarrow{\psi_{\mathsf{step}}} \mathsf{S}$$

to indicate that C performs the protocol step and sends the specified output to S.

The previous workflow requires a user to solve a challenge for every access (modulo the use of single-domain cookies). To reduce the number of challenges exposed to the client without sacrificing the unlinkability requirement, we embed the protocol that we constructed in Section VI;3 into this workflow. This allows a client to receive m unlinkable tokens for every CAPTCHA solution. Each token can then be used to bypass challenges on future accesses. We present two high-level workflows in Figure VI;11 and Figure VI;12 embedding the protocols. The workflows are split into an issuance and redemption phases, mirroring the partitioning of the original protocol Γ.

Token issuance. At this point, the client is assumed to have no tokens that can be redeemed. The client is required to solve CHL as before, but when $SOLN$ is sent back, they also initiate the first step of the signing phase (ψ^1_{sign}) in Construction VI;3.2 and send the output to the server. The server validates $SOLN$. If this is successful it carries out the second step of the signing phase (ψ^2_{sign}) and returns the result back to the client, along with the resources they requested. Finally, the client runs the third step (ψ^3_{sign}); storing the outcome for future use.

Token redemption. In the token redemption step, when the client receives a request containing CHL, they instead initiate the first step of the redemption phase (ψ^1_{red}) and send the output to the server. The server runs the second step (ψ^2_{red}) and, if this passes correctly, they allow the client access to the requested resource by sending RESP($\mathtt{W}$).



$9.\ \psi^3_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\varphi, (\mathcal{W}, \pi)], \widehat{\mathsf{DLEQ}}_m)$
1. REQ(W)
2. CHL
3. (see caption)
8. (see caption)
6. REQ(W)
7. RESP(W)
4. (see caption)
5. (see caption)
C
S
W
V

Figure VI;11: Adapted workflow for token issuance. We assume that $(\mathsf{pp}, k) \leftarrow \Gamma.\mathsf{Setup}(1^\lambda, 1^\kappa, m, p)$ has been run as a trusted setup in the past. If N is returned in step 5 then subsequent steps are not run.
Step 3/4: $(SOLN,\ \Gamma.\psi^1_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}]))$.
Step 5: (Y, $\Gamma.\psi^2_{\mathsf{sign}}(\mathsf{pp}, \mathsf{S}[\{\widetilde{x}_i\}_{i\in[m]}, k], \widehat{\mathsf{DLEQ}}_m)$) / N.
Step 8: (RESP(W), $\Gamma.\psi^2_{\mathsf{sign}}(\mathsf{pp}, \mathsf{S}[\{\widetilde{x}_i\}_{i\in[m]}, k], \widehat{\mathsf{DLEQ}}_m)$).

In summary, for every signing workflow as in Figure VI;11 completed by a client C, it can then engage in m redemption workflows. These m redemptions provide unlinkable tokens (by Theorem VI;3.2), and do not require human participation to solve challenges. We can further augment successful redemptions with single-origin cookies. These cookies allow clients to bypass future challenges for the same domain without using more redemption tokens, and without client browsing being linkable across multiple domains.

Key rotation. It is important that the server S is able to implement a key rotation policy, both in the case that a key has exceeded its lifetime, and to reduce the potency of some attack vectors (Section VI;6.2). Key rotation in these workflows requires a new iteration of $\Gamma.\mathsf{Setup}$ to be run and distributed to each client, where k^* is the new secret key that is used. This is necessary so that the clients can verify the key-consistency proofs that are sent by S in the signing phase.

Our workflow in Figure VI;11 can be adapted so that the server can have multiple keys in use at any one time. This prevents key rotation from immediately rendering all possessed tokens useless. Essentially, S sends a list of public parameter commitments $\{\mathsf{pp}_j\}_j$ corresponding to all secret keys that it accepts. The client checks the list of commitments and uses a corresponding pair to verify the key-consistency proof that it receives in the later stages. When a key is phased out for good, the server simply removes the commitment from the acceptance list.

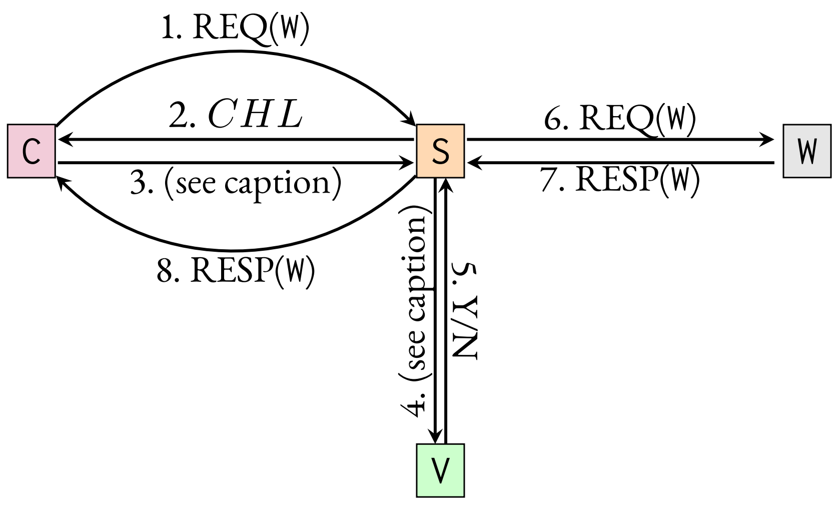


Figure VI;12: Adapted workflow for token redemption, with notation defined as in Figure VI;11.
Step 3/4: $\psi^1_{\mathsf{red}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}, \mathcal{R}])$.
In Step 5, V runs $b \leftarrow \psi^2_{\mathsf{red}}(\mathsf{pp}, \mathsf{S}[(x,\tau), k], \mathcal{D})$ and answers Y if $b = 1$ and N likewise.
If N is received from Step 5, then the subsequent steps do not run.

When multiple keys are in use, this acts to siphon the user base into smaller groups and thus increase the deanonymization potential. As a result, the list of keys that a service provider should be kept to a relatively low number, e.g. 2 or 3 at any one time. The speed of the rotation should also be kept at a relatively low frequency, e.g. once per month, or less.

VI;5 Privacy Pass deployment

To instantiate the workflows shown in Section VI;4.4 we have created a client-side browser extension named "Privacy Pass" and have written compatible server-side support in Go that can be run on edge servers. The browser extension handles the cryptographic operations that need to be carried out on the client side. Our browser extension is written in JavaScript[12] and is compatible with Chrome and Firefox. Additionally, the code is open source and available online.[13]

Privacy Pass carries out the cryptographic operations required in Figures VI;11 and VI;12, and augments HTTP requests from the client with this information. That is, the browser extension plays the role of C in the protocol Γ. Server-side support for Privacy Pass has been written into existing Cloudflare infrastructure, in order to reduce the number of CAPTCHAs that are faced by honest users. To achieve this, we run a new service on the Cloudflare edge network that acts as V in Γ. When Cloudflare receives messages from Privacy Pass, it routes the cryptographic material through to this service for processing and verification ($\mathsf{S} \rightarrow \mathsf{V}$).

[12]We make use of the `chrome.WebRequest` and `chrome.WebNavigation` frameworks for writing the extension.
[13]https://privacypass.github.io/

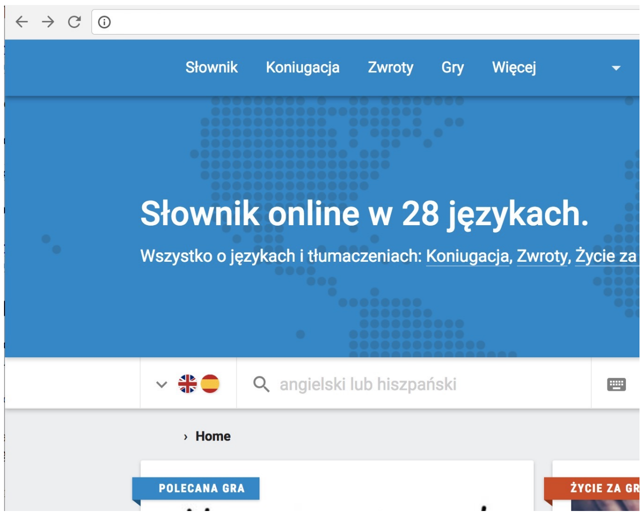

Słownik
Koniugacja
Zwroty
Gry
Więcej
Słownik online w 28 językach.
Wszystko o językach i tłumaczeniach: Koniugacja, Zwroty, Życie za
angielski lub hiszpański
› Home
POLECANA GRA
ŻYCIE ZA GR

Figure VI;13: The same website from Figure VI;9, when used with Privacy Pass.

Using this mechanism reduces the number of CAPTCHAs that need to be solved by a factor directly proportional to m — the number of signed tokens received for each initial solution. Currently m is set to 30, but the server supports up to $m = 100$. The value $m = 30$ was chosen because it provides a balance between usability, performance, and token hoarding (Section VI;6) considerations. As m rises, the latency rises since the server has to sign more tokens and compute a batched DLEQ proof over more values. Moreover, the client has to generate and unblind more tokens and also verify the batched proof. Since these client operations occur in the browser, and potentially halt the execution for a period, these operations need to be kept to a minimum.

A second benefit is that sub-resources hosted behind Cloudflare CAPTCHAs are accessible for clients — even though clients would not previously have been able to submit CAPTCHA solutions for these URLs. For example, in Figure VI;13 we show the website from Figure VI;9 after using Privacy Pass. In the old case, the user has no way of solving a CAPTCHA for the CSS files (other than locating and navigating to the URL), the website is displayed without the CSS styling and thus appears with clear errors. Privacy Pass is able to redeem signed tokens with these resources instead and thus the user is now able to view the website as it was intended. While this is undoubtedly an example of a flaw with the implementation of CAPTCHAs in general, Privacy Pass can be leveraged to make the browsing experience more pleasant and error-free for users.

We now give a brief account of how Privacy Pass executes the adapted workflows that we gave in Section VI;4.

VI;5.1 Signing

Firstly, assume that a client C (with sufficiently poor IP reputation $\nu_{\mathsf{C}} > \eta$ for some threshold η) is attempting to connect via HTTP(S) to a Cloudflare-protected website W, using a browser with Privacy Pass installed, and with no passes available. When a challenge CAPTCHA is solved by C then Privacy Pass first generates the 'blinded' tokens (that are sent to S in $\Gamma.\psi^1_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}])$) and appends these tokens to the body of the HTTP(S) request containing the solution. Privacy Pass analyses the structure of request URLs for determining when a Cloudflare CAPTCHA has been solved.

This amended request is sent asynchronously and the edge verifies whether the CAPTCHA solution is correct. If it is, the blinded tokens are signed and a key-consistency proof is generated as in $(\mathcal{W}, \pi) \leftarrow \Gamma.\psi^2_{\mathsf{sign}}(\mathsf{pp}, \mathsf{S}[\{\widetilde{x_i}\}_{i\in[m]}, k], \widehat{\mathsf{DLEQ}}_m)$. The edge now creates a new HTTP(S) response indicating that the solution was valid (i.e. status code 200) and returns a single-origin clearance cookie in the response header, along with the signed tokens and the proof in the response body.

Privacy Pass parses the signed tokens and the key-consistency proof, which is validated — this is where $\psi^3_{\mathsf{sign}}(\mathsf{pp}, \mathsf{C}[\varphi, (\mathcal{W}, \pi)], \widehat{\mathsf{DLEQ}}_m)$ is run. If the proof is valid, then the signed tokens are unblinded and stored for future use in the *local storage* of the browser. After storage is completed, Privacy Pass reloads the current page, which will now succeed because of the single-domain clearance cookie.

VI;5.2 Redemption

Assume that the client C has gained tokens as above and that it has attempted to access another edge-protected website W'. A CAPTCHA will be served and the header `cf-chl-bypass` will be present with value '1' in the edge response. Privacy Pass uses the presence of this header to initiate the token redemption procedure when it has signed tokens stored.[14]

Privacy Pass retrieves an unspent token pair (x, y) from local storage and starts forming a new HTTP request for W'. Recall that in $\psi^1_{\mathsf{red}}(\mathsf{pp}, \mathsf{C}[\{x_i\}_{i\in[m]}, \mathcal{R}])$ the client is required to construct a MAC tag over the original value x and some auxiliary data aux; the key for the MAC is derived from the server-signed part of the token pair that was retrieved (see $\psi^2_{\mathsf{red}}(\mathsf{pp}, \mathsf{S}[(x, \tau), k], \mathcal{D})$). The auxiliary data is generated from request binding data — precisely the contents of the 'Host' header for the request concatenated with the HTTP Path.[15] Then, the client derives the pair $(x, \tau) \leftarrow \mathsf{mac.Tag}(K_x, x)$ as the shared key, where $K_x = H_2(x, y)$. This shared key is ap-

[14]This header was also present in the first case but was not actually used.
[15]This is not quite a unique identifier for the request, but it is enough for preventing various MitM attacks.

	Operation	Timings (ms)
Client	Token generation (ψ^1_{sign})	$120 + 64 \cdot m$
	Verify DLEQ & store (ψ^3_{sign})	$220 + 110 \cdot m$
	Total signing request (ψ_{sign})	$340 + 180 \cdot m$
	Total redeem request (ψ_{red})	57
Server	Signing & DLEQ (ψ^2_{sign})	$0.36 + 0.75 \cdot m$
	Total signing	$1.48 + 0.87 \cdot m$
	Total redemption	0.8

Table VI;1: Benchmarks (ms) for operations that are instantiated using our protocol from Section VI;3. The value m is the number of tokens to be signed in a batch.

pended to the HTTP request as the value of a header named '`challenge-bypass-token`', and the request is then sent to the edge.

The edge verifies the value of the '`challenge-bypass-token`' header using the procedure in $b \leftarrow \psi^2_{\mathsf{red}}(\mathsf{pp}, \mathsf{S}[(x, \tau), k], \mathcal{D})$. If verification succeeds (i.e. $b == 1$), S serves the content of $\mathtt{W}'$ to C and a single-origin clearance cookie. The data storage $\mathcal{D}$ is implemented via a Bloom filter that holds all tokens that have been spent. This Bloom filter is scalable and increases in size as more tokens are stored. The contents are flushed when key rotation occurs.

VI;5.3 Cryptographic implementation

The random oracles in the construction are instantiated using the SHA-256 hash function. The prg used for constructing the batched proof is implemented using the SHAKE-256 Keccak-based design. All elliptic curve operations are performed using the NIST standardised curve P-256 [246]. We use a browser-based implementation of this pseudorandom generator for proof verification.[16] We make some modifications so that it can be run in the browser natively. All cryptographic operations are carried out in the browser using a local copy of the SJCL library,[17] and using the native `Go` crypto libraries for the server-side operations.

VI;5.4 Benchmarks

We illustrate the additional client and server overheads incurred due to the additional protocol structure. We compile our data from a series of benchmarks taken over the key cryptographic operations that are computed.

[16] `https://github.com/cryptocoinjs/keccak`
[17] https://crypto.stanford.edu/sjcl/

Operation	Size (bytes)
Signing request (C → S)	$57 + 63 \cdot m$
Signing response (S → C)	$295 + 121 \cdot m$
Redemption request (C → S)	396

Table VI;2: Size of additional information (bytes) for requests. m is the number of tokens to be signed in a batch.

We have the client generate m tokens at a time (for $m \in \{5, 10, 15, \ldots, 100\}$) and each of these is signed by the server. In addition, we include generation and checking of batch DLEQ proofs, and redemption of a single token. We repeated this experiment 100 times.

In Table VI;1 we provide a set of benchmarks for the operations that are computed for the client and server. In Table VI;2 we provide the size of the additional data for requests and responses due to Privacy Pass.[18] We do not include the redemption response size as Privacy Pass does not change this flow.

As is clear from our results, the overheads incurred from using Privacy Pass are quite acceptable. In particular, redemptions are fast for both the client and server and require very little added communication. For signing, the biggest costs appear to be the time taken for the client to generate requests and verifying the DLEQ proof in the server responses. On the other hand, the initial cost for acquiring signed tokens is warranted given that redemptions are so efficient and client proofs-of-work are avoided for the lifetime of the tokens. In summary, we view the initial cost of the signing phase as manageable, wrt the gains that are achieved in subsequent phases of browsing.

Finally, the additional communication load is less than 6 KB in total for $m = 30$, which is unlikely to trouble the browsing experience.

VI;5.5 Release and adoption results

We released the Chrome/Firefox browser extension Privacy Pass on 8 November 2017, and also released the open-source code for the extension, which would allow a user to locally install the extension in their browser. In the following we will detail various numbers acquired directly in partnership with Cloudflare [98] (we focus only on distinguishing requests coming from Tor and non-Tor users). There is a website for Privacy Pass[19] and the release was announced via a series of blog posts [115, 116, 284].

[18] The signing responses are about double the size of the requests, because for performance reasons, the (`JavaScript`) client compresses the elliptic curve points in the signing request, but the (`Go`) server does not compress the elliptic curve points in the response, lest the `JavaScript` client have to do expensive point decompressions.

[19] `https://privacypass.github.io`

The number of Chrome users by 28 November 2017 stood at 8499 and the number of Firefox users stood at 3489 — this does not include users manually installing the extension. By July 2018, this number had increased to 61578 Chrome users and 16375 Firefox users, demonstrating a 650% increase in total.

Averaging over any given 7 days in the November period, Cloudflare accepted 1.6 trillion requests with 780 million of those requests coming over the Tor network. Of these requests, 1.04% were challenged globally, with 17% of Tor traffic being challenged in the same period [98].[20] Fortunately, Privacy Pass alleviates the burden of these challenges by a factor of m for honest users who would usually complete the challenge. However, Privacy Pass may not result in a factor of m reduction in the total number of challenges witnessed — i.e. we may not see an optimal drop to $(17/m)\%$ of requests being served challenges in the Tor case.[21] This is due to the fact that these figures also include challenges that are served to requests from users that have no intention of solving the challenge (such as bots or content scrapers).

In terms of operating numbers, the number of redemption phases occurring peaked globally at 2000 per hour and for Tor users at 200 per hour. In July 2018, these peaks had increased to 3300 and 600, respectively. Additionally, there were 515 million requests containing single-origin, clearance cookies used to bypass challenges globally with 34.5 million of those occurring from the Tor network. As a result, 22.58% of requests coming from Tor users contain these clearance cookies. As mentioned in the previous sections, these cookies allow users to bypass CAPTCHAs for a single origin without having to spend more tokens. Using these cookies preserves tokens for use only on unseen domains, which is advantageous for users. They cannot be tracked across domains due to the single-origin policy.

We demonstrate a system that is clearly useful for clients with the extension installed. In the same time-frame, median request and response sizes for Tor users stood at 700–800 bytes and 5–6 KB respectively, while the median size of CAPTCHA submissions and responses were less than 1 KB.[22] Consequently, the additional protocol messages do not result in unmanageable request/response sizes.

Remark VI;5.1. *Note, that we do not record a drop in the actual number of CAPTCHAs served by Cloudflare. This is because Privacy Pass works to prevent the CAPTCHA being displayed to the user by immediately responding with a new HTTP request. This prevents the user from interacting with the challenge, but this occurs after it has been sent by the server. We take the number of redemptions and cookie usage as indicators for the decrease in CAPTCHA solutions.*

[20]By challenged, we mean the response to the request displays a Cloudflare CAPTCHA.

[21]We do not have explicit data for the VPN/I2P cases and so we focus on the case of reporting on Tor instead.

[22]These figures are slightly different to the benchmarks above as they include information specific to Cloudflare that we did not model in the benchmarking tests.

VI;5.6 Code and versioning

The code for Privacy Pass and the compatible server-side implementation is open-source.[23],[24] Version v1.0 can be downloaded from the Chrome[25] and Firefox extension stores.[26] The source code for Privacy Pass currently includes some updates that are not yet in the release version. We plan to release these updates officially in the near future. We regard Privacy Pass, and our protocol, as still in the beta phase of development and are welcoming external contributions.

VI;6 Potential attacks

The security properties that we proved in Section VI;3 do not consider a number of 'out-of-band' attacks that may still effectively subvert the security of the protocol, if mitigations are not considered. We say 'out-of-band' because they are not considered inside of the security model that we originally considered when we proved the protocol secure. We consider measures that can be put in place to limit the effectiveness of such attacks.

VI;6.1 Interception of signing requests

A characteristic of our protocol is that we do not currently encode any useful information into the tokens that are signed by the server. This prevents the client from generating tokens that may inadvertently associate their metadata tokens during redemption, and lead to potential deanonymization vectors. This means that tokens are not associated with any one client, increasing the effectiveness of monster-in-the-middle (MitM) adversaries; since transmitted tokens are not cryptographically linked to a specific protocol invocation.

For example, consider a MitM adversary $\mathcal{M}$ in the signing phase of the workflow in Figure VI;3. Consider $\mathcal{M}$ that hijacks the blinded tokens sent to the server, and simply forwards the challenge solution under their own identifier along with a set of their own generated tokens. If the challenge does not have some client-binding hijack-prevention mechanism, then $\mathcal{M}$ will receive signed tokens from the server without completing a challenge.

This attack is particularly effective as it removes the requirement for an adversary to authenticate to a service in order to receive tokens. For example, consider if the challenge mechanism was a CAPTCHA, and $\mathcal{M}$ was some bot that was ordinarily not able to solve it. This attack would allow $\mathcal{M}$ to receive tokens without computing an actual solution and then bypass future CAPTCHA

[23]`https://github.com/privacypass/challenge-bypass-extension`
[24]`https://github.com/privacypass/challenge-bypass-server`
[25]`https://tinyurl.com/privacypass`
[26]`https://addons.mozilla.org/en-US/firefox/addon/privacy-pass/`

invocations. In the case of Cloudflare, the CAPTCHA page is protected by TLS so this seems hard to hijack, however this remains a general attack vector.

While this attack is effective, it requires a renewable source of authentications/challenge solutions for it to carry on indefinitely. In this setting, even without Privacy Pass, such an adversary would essentially have access to a CAPTCHA solving farm and thus the need for acquiring tokens to authenticate would be made redundant anyway (except for a small gain in the speed at which authentication could be performed).

In contrast, the previous system required an *online* adversary looking to bypass a CAPTCHA maliciously. Using Privacy Pass, it becomes possible to intercept tokens and use them at a more convenient time in the near future (until the keys are rotated). As long as the maximum number of signed tokens is fairly small (in the case of Cloudflare only 100 tokens can be signed for each interaction), then the effectiveness of the attack is not especially increased.

VI;6.2 Token accumulation

A related attack avenue would be for clients to 'farm' signed tokens by repeatedly sending challenge solutions to invoke the signing phase of the protocol. This would allow the client to build up a stockpile of unused tokens that they could then redistribute amongst other clients or redeem tokens for a large period of time. Redistribution of token pairs (x_i, y_i) is possible due to the lack of data encoded into them. Building up a large store of tokens could be useful for launching DDoS attacks on service providers.

Clearly, solving challenges incurs much larger overheads than the token redemption process using the browser extension. Therefore, clients can invoke many redemptions in the same time that they could solve one challenge previously. It is important that pass verification is very efficient for the server to carry out, in order to reduce the threat of a DDoS attack against the server. Fortunately, verification of a token redemption occurs in less than 1 ms and a few hundred bytes of additional transmitted data. Additionally, carrying out these operations on powerful server hardware may lead to even faster running times. That being said, a client that amasses a huge number of tokens may still be able to cause problems.

For these attacks we implement a number of mitigations. Firstly, the low upper bound of 100 tokens signed at a time by Cloudflare means that it would take a large amount of effort to build up a stockpile of tokens large enough to launch a credible DDoS attack (1000000 tokens would require a minimum of 10000 CAPTCHA solutions).

Finally, we recommend regular key rotation so that signed tokens are implicitly related to epochs and are thus invalidated frequently. This prevents stockpiles of tokens from being useful for longer than the epoch they belong to. Ideally key rotation would occur over short time periods; a shorter time period directly reduces the number of useful tokens that can be stockpiled and redeemed. In our initial software distribution, key rotation is handled by updating the commitments directly in the browser extension. Moving forward, CDNs could upload their commitments to beacons or other globally visible and consistent locations, such as the Tor consensus.

VI;6.3 Token exhaustion

Privacy Pass uses a finite list of low-entropy characteristics to determine whether a token should be redeemed or not. In the case of Cloudflare CAPTCHAs, the extension looks for the presence of an HTTP response header and particular status code. Unfortunately, this means that it is easy to recreate the characteristics that are required by the extension to sanction a redemption.

To view the attack at its most powerful, consider a sub-resource that embeds itself widely on many webpages that can trigger token redemptions.[27] Such a resource would be able to drain the extension of all its tokens by triggering redemptions until all the tokens were used. While it is unclear why such an attack would be useful, it is important to acknowledge that it is indeed possible to carry out and would thus render the usage of Privacy Pass useless if the sub-resource was especially prevalent.

Our mitigation for this attack lies in the extension itself. First, the implementation of Privacy Pass prevents token redemptions occurring for the same URL in quick succession by keeping track of where spends have occurred. This data is refreshed when a single-domain cookie is removed to allow re-spending at the domain. This prevents a sub-resource from recursively draining tokens after each spend; it also spreads out the redemptions considerably. The sub-resource would have to force cookies to be removed and then schedule a page reload for each token spend.

A coarser method of stopping these attacks — that has not been implemented at time of writing in the Cloudflare deployment — would be to blacklist URLs from spending after one attempt and then refresh this state every time the extension reaches 0 tokens. While this would be a more effective countermeasure, it would also prevent a client from spending tokens on a URL, even if the cookie for the URL were removed. Other potential mitigations could include requiring the edge to sign the appropriate header, or the edge stripping the `cf-chl-bypass` header from origin responses.

VI;6.4 Time-based client deanonymization

As with all systems that require some sort of registration, there is the inherent problem that 'early adopter' clients are part of a much smaller anonymity set than is optimal for enforcing privacy

[27] For example, a popular analytics script or JQuery code.

constraints. In particular, consider a scenario where some user is amongst the first five to initiate a signing phase with some given CDN, or after key rotation has occurred. Then when a client comes to redeem a token, the provider will know that the client can be linked to one of the five only clients to have tokens signed. A malicious edge could also artificially reduce the size of the user base by only signing tokens for a small number of users, or by using artificially small key cycles.

A more detailed analysis may also take into account patterns of behaviour that lead to initiating signing and redemption phases in predictable ways. For instance, it may be possible to associate token redemptions to signing queries linked to a user profile that predictably asks for tokens to be signed at certain times.

This threat bypasses the formal unlinkability of the protocol, exploiting the fact that linkability is possible based on the timing of certain requests or via early registration. The pervasiveness of the potential to deanonymize clients this way means that, like many privacy enhancing technologies including Tor itself, such privacy solutions are only really effective when the number of users is quite large. As we mentioned previously, the user-base for Privacy Pass has been growing rapidly since it was first released. This suggests that linking browsing sessions is likely to incur significant overheads and high error rates.

VI;6.5 Double-spending protection

It is vital that services use their own private keys for signing tokens so that tokens cannot be spent multiple times across different services. Moreover, each service that supports Privacy Pass should implement double-spend protection to ensure that tokens cannot be spent multiple times across their own architecture. This is why we build the stateful data storage into our redemption protocol in Section VI;3. An effective way of managing a large double-spend index would be to use a hash table, Bloom filter or other similarly efficient data structure. In the Cloudflare case, a Bloom filter is used; and it is emptied every time key rotation occurs.

VI;7 Conclusion and future work

In conclusion, we have shown that we can embed an anonymity-preserving, cryptographic protocol for proving honesty into a browser-based workflow. The result is practical to run, in the sense that browsing is not affected by the extra computation. Moreover, the protocol allows users to bypass strenuous proof-of-work challenges that can make websites seem inaccessible. This is beneficial to users of anonymity services such as Tor and VPNs who are unfairly targeted by overzealous access-control mechanisms. Our implementation is completely open-source and is available online at `https://github.com/privacypass`. The official website for the extension can be found at `https://privacypass.github.io`.

VI;7.1 Future work

In terms of future work, we have a variety of focuses that target both improving the extension Privacy Pass and also the underlying cryptographic protocol.

Future integrations. For Privacy Pass, we hope to release a new official version in the next few months to the Chrome and Firefox stores that will aid integrating with more third parties, branching out from the Cloudflare integration. The intention with Privacy Pass was never to be bound to the Cloudflare architecture, and this is why we released the code for the server-side. We are investigating the possibility of integrating the server-side functionality with the CAPTCHA provider `FunCaptcha`.[28] Part of this release will include a new method for establishing configurations for different providers.

Better documentation. To aid the process of integrating new services, we also plan to document the features and triggers that are used by Privacy Pass for initiating the protocol design that we use. This will be a natural process when the new configuration process is released.

Verifiable key beacons. Currently the keys that are used for proof verification are stored in the code of Privacy Pass. We would like to support multiple keys being used at any one time to make key rotation simpler. We would also ideally like the keys to be placed in a publicly verifiable location where they could be accessed from. This would mean that key rotations would not require a code change — which will make the process much simpler.

IETF standardisation of EC-VOPRF authentication protocol. We are currently in the process of drafting an IETF standard document for the implementation of our protocol from Section VI;3.[29]

Post-quantum VOPRF. Exploring the possibility of instantiating our key exchange from cryptographic primitives that believed to be ‘post-quantum’ secure is an interesting pursuit. In particular, the method that we use is similar to the DH-style key exchange that is prominent throughout cryptographic literature. Therefore, it may be possible to achieve similar functionality using lattice-based or isogeny-based cryptosystems.

Browser-less solutions. Currently, requests that are sent via a browser-less mechanism to Cloudflare websites are blocked if a CAPTCHA is required, since the user has no ability to solve the challenge. Such browsing is common using tools such as `curl`, or by running browsers in head-less mode. Devising a solution that works outside of the browser architecture would also be a useful endeavour.

[28] `https://www.funcaptcha.com/`
[29] `https://datatracker.ietf.org/doc/draft-sullivan-cfrg-voprf/`

VII A Security Analysis of the GGH13 Graded Encoding Scheme Without Ideals

ignorance is bliss

Preface

In this chapter, we will analyse known constructions of branching program indistinguishability obfuscation candidates instantiated with a new graded encoding scheme. We provide a new variant of the candidate graded encoding scheme of Garg, Gentry and Halevi [148] that does not admit a polynomial-time cryptanalysis, under previously known techniques [15, 240]. Our main contribution is then to develop a non-trivial cryptanalytic technique for breaking this new scheme.

The purpose of this chapter is to illustrate structural vulnerabilities in the construction of [148]. It was widely believed that the weaknesses of [148] stemmed from the presence of a common generator g, that can be used to form ideals in the rings in which the encodings lie. This stemmed originally from a conjecture made in [176]. Our variant on the scheme removes the presence of g, and thus the possibility of launching previous attacks. However, we show that this removal does not provide security in the cases where the encoding scheme has potential applications. Specifically, we demonstrate attacks against the scheme when used to implement a large class of candidate branching program obfuscators.

We demonstrate tweaks to the cryptanalytic model used for analysing obfuscations candidates. We believe that the new model makes analysing security much more intuitive, where previous models relied on idealising graded encoding operations and abstracting away certain vulnerabilities.

There are some algebraic discrepancies between [148] and our new scheme that ensure that the attacks that we construct are not as devastating to security to those that have came previously. Indeed, there appear to be situations where our scheme admits no known attack, where [148] does. We leave further exploration of this discrepancy as interesting future work.

This chapter is based heavily on the material published in [8] [IMACC2017]. We include extra attacks that were added to the full version of the paper [9], after the date of publication. In addition, we make the correctness of our new scheme more concrete, where the previous work relied on approximations.

Overview of original contributions

- Construct a new cryptanalytic model for analysing the security of candidate indistinguishability obfuscators for branching programs. The sufficiency of the model is shown by adapting the known attacks against [148] successfully (Section VII;3).
- Construct a non-trivial variant of the graded encoding scheme proposed by Garg et al. [148] that is no longer susceptible to previously known 'annihilation attacks' [15, 240] in branching program scenarios [27, 253] (Section VII;5).

- Give novel variants of ‘annihilation attacks’ that break the security of our adapted graded encoding scheme (Section VII;6).

Contents

VII;1 Introduction

Over the past two decades (and more rapidly in the last five years), an emerging cornerstone of the literature on secure computation has been the study of program obfuscation. Program obfuscation is a hugely powerful cryptographic primitive that, in theory, allows evaluation of programs without revealing embedded secret data. In terms of practical usage, such a primitive would have several applications regarding digital rights management, whereby software vendors could provide packaged programs to users. The users could then run the program without seeing the source code. In a theoretical sense, the primitive is a crucial device for constructing cryptographic primitives that were previously thought to be impossible [272].

VII;1.1 Binary circuits

Prior to considering the possibility of obfuscating a program, we first consider the types of functions that we wish to obfuscate. Cryptographically speaking, a program is usually expressed as a circuit that can be evaluated in polynomial-time in length of the input. Such circuits must then have polynomial *depth* and *size*. The output of the circuit on a given input is taken to be the binary string $y \in \{0,1\}^{\ell_{\mathsf{out}}}$; taking bits from each of the output wires of the circuit. In this work, we do not consider arithmetic circuits, where input wires take values from $\mathbb{Z}$ and gates are ring additive and multiplicative operations.

Recall that the depth d of a circuit is the maximum length of a path from an input wire to an output wire, measured in the number of gates that need to be computed. We assume that all gates take two input wires (fan-in two) and have one output wire. We assume that input wires are used in multiple gates. We can construct circuits for arbitrary functions using only NAND gates. Unless specified otherwise, we will only consider circuits that output a bit $b \in \{0,1\}$, i.e. $\ell_{\mathsf{out}} = 1$.

VII;1.2 Circuit obfuscation

As we mentioned in Section II;6.2, defining a notion of security for a program/circuit obfuscator is a subtle process. We will introduce the security models that have been considered in the prior literature, along with the classes of circuits that are considered.

Virtual black-box security. Obfuscators that satisfy *virtual black-box* (VBB) security satisfy the most intuitive type of security. The security model specifies that the circuit can be simulated (in an indistinguishable manner) for all PPT adversaries by an algorithm that has only oracle access to the function that has been obfuscated. Therefore, any information that can be learnt from the obfuscated circuit, can also be learnt by just having oracle access to the function. As a result, any secret data in the function — not revealed by the output explicitly — is kept hidden.[1].

The study of VBB program obfuscation was initiated by Barak et al. [28]. Unfortunately, their seminal work showed that *virtual black-box* obfuscation was impossible for general functions, and thus circuits from $\mathcal{C}_{\mathsf{P/poly}}$. In other words, there exists a class of functions that can be expressed as polynomial-depth circuits, that provably reveal more information than a simulator with only oracle access to the function. Thus achieving meaningful definitions of program obfuscation requires a different approach, and has lead to several diverging strands of research.

The first focuses on bypassing the impossibility result by conversely exploring which classes of functions *can* be obfuscated in the VBB security model. Lynn et al. [232] and Canetti [74] gave point-function obfuscators in the random oracle model and based on collision-resistant hash functions respectively; while Wee [293] developed a point-function obfuscator based on strong one-way functions. Other work has seen developments in hyperplane obfuscators [77] based on strong DDH, and obfuscators for various families of evasive functions based on strong assumptions over MMAPs [26, 65]. There have been a number of advancements in obfuscators for evasive functions, more recently. The works of [44, 70, 172, 294] all achieving new constructions from lattice-based assumptions, or within the generic group model.

Indistinguishability obfuscation. While constructing VBB obfuscators for specific function classes is one way of avoiding the impossibility result of [28], another method is to rework the security model that is being considered. Barak et al. [28] surmised that a more achievable security model would ask adversaries to distinguish the obfuscation of two circuits that are functionally identical.[2] The security notion is known as indistinguishability obfuscation (IO), and is defined formally in Definition VII;1.1. It was shown by Goldwasser and Rothblum [168] that IO represents the 'best-possible' obfuscation that can be achieved.

[1]This security notion has obvious parallels with the malicious security model in multi-party computation, see Chapter III

[2]But the circuits themselves may be different in construction.

```
$\mathsf{exp}^{\mathsf{io}}_{b,\mathcal{C},\mathcal{A}}(1^\lambda, 1^d, 1^n, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$
1 : $C_0, C_1 \leftarrow \mathcal{A}_1(1^\lambda, 1^d, 1^n, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$;
2 : if $\exists\, x$ s.t. $\neg(C_0(x) \overset{?}{=} C_1(x))$ :
3 :     return $\perp$
4 : $\widetilde{C}_b \leftarrow \mathsf{io}(1^\lambda, 1^d, 1^n, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, C_b)$;
5 : $b_{\mathcal{A}} \leftarrow \mathcal{A}_2(1^\lambda, 1^d, 1^n, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, \widetilde{C}_b)$;
6 : return $b_{\mathcal{A}}$;
```

Figure VII;1: IO security model. We may also write $\mathsf{exp}^{\mathsf{io}}_{b,\mathcal{C},\mathcal{A}}(1^\lambda)$; omitting the input parameters where context is clear. A flaw of the IO security model is that the check in line 2 cannot be carried out in polynomial time, and thus simulation is unbounded. To mitigate this, $\mathcal{A}$ has to be restricted to only output functionally equivalent circuits.

Definition VII;1.1 [Indistinguishability obfuscation]

Consider a circuit family $\mathcal{C} = \mathcal{C}_{\ell_{\mathsf{in}},\ell_{\mathsf{out}},d,n}$ with input size ℓ_{in}, output size ℓ_{out}, depth d and size n. Let io be a PPT algorithm, which takes as input: a circuit $C \in \mathcal{C}$; a security parameter $\lambda \in \mathbb{N}$; the circuit parameters above; and outputs a boolean circuit $\widetilde{C}$ (possibly $\widetilde{C} \notin \mathcal{C}$). We say that $\mathsf{io}(\cdot)$ is an obfuscator for $\mathcal{C}$ if it satisfies the following properties:

1. *Functionality*: For all circuits $C \in \mathcal{C}$, we have

$$\Pr\Big[\forall x \in \{0,1\}^n \ : \ C(x) = \widetilde{C}(x) \leftarrow \mathsf{io}(1^\lambda, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}}, 1^d, 1^n, C)(x)\Big] = 1,$$

   where the probability is taken over the random coin tosses of $\mathsf{io}(\cdot)$.

2. *Polynomial slowdown*: For every $\lambda \in \mathbb{N}$ and $C \in \mathcal{C}$, the circuit $\widetilde{C}$ is of size at most $\mathsf{poly}(|C|, \lambda)$.

3. *Security*: We require that:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{io}(\cdot)(1^\lambda))) < \mathsf{negl}(\lambda).$$

   where the security of $\mathsf{io}(\cdot)$ is defined by the experiments $\mathsf{exp}^{\mathsf{io}}_{b,\mathcal{C},\mathcal{A}}(1^\lambda, 1^d, 1^n, 1^{\ell_{\mathsf{in}}}, 1^{\ell_{\mathsf{out}}})$ for a pair of (non-uniform) PPT algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and $\mathcal{A}_1$ outputs functionally equivalent circuits.

We may also abuse notation and say that $\mathsf{io}(\cdot)$ is an *IO obfuscator* for $\mathcal{C}$.

Utility of IO. While VBB obfuscation provided an intuitive description of the functionality that we expect from a program obfuscator, IO provides a more subtle description. In fact, it is not clear how sensitive information can be hidden by such an obfuscator. The only requirement is that the obfuscations of two functionally equivalent circuits are indistinguishable. So an important question is:

How do we utilise this security property for hiding secret information?

Sahai and Waters [272] gave a foundational work describing how IO can be used to achieve many seminal results in cryptography. For instance, they give a generic transformation from symmetric-key encryption to public-key encryption. They also gave the first construction of a deniable encryption scheme, amongst many other applications. The key feature of their work is to use a puncturing technique that omits certain secret information from the obfuscated circuit, and then replace the underlying circuit with an instantiation that does not use the secret information.

In summary, they produce canonicalised variants of binary circuits that are functionally equivalent but do not hold sensitive information (such as secret keys). Then showing that the obfuscation of the two circuits is indistinguishable allows the security proof to 'puncture' out the sensitive circuit[3] for the canonical version. Now the adversary no longer has any access to any secret information. This idea has been applied to many different strands of research, and has established IO as a fundamental cryptographic primitive. Examples of applications include [10, 14, 22, 46, 100, 136, 149, 150, 158], but this is by no means an exhaustive list.

VII;1.3 Candidate constructions of IO

While the applications above help to crystallise the importance of IO as a cryptographic primitive, the worth of it is only demonstrated in full by concrete instantiations of the primitive. The taxonomy of known constructions of IO can be split into the following categories.

Candidate constructions of IO via branching programs. Apart from the work of Goldwasser and Rothblum [168], the study of IO was largely silent until the candidate construction of Garg et al. [150]. This work constructs IO obfuscators for the circuit class $\mathcal{C}_{\mathsf{P/poly}}$. Firstly, the work considers circuits in the form of generalised matrix branching programs. It was shown by Barrington [29] that any binary circuit in $\mathcal{C}_{\mathsf{NC}^1}$ can be realised by a width-5 matrix branching program, requiring 4^d matrix multiplications. Firstly, [150] shows that a number of randomisation techniques appear to 'hide' the function, though the security assumptions are non-standard. Then they show that obfuscation for $\mathcal{C}_{\mathsf{P/poly}}$ can be achieved using FHE. We establish our notation for branching programs formally in Section VII;2.4.

More specifically, the randomisation techniques alter the structure of the individual matrices in the branching program without introducing a change in the output. The final step 'encodes' each entry of the matrices using a *graded encoding scheme* (GES). A GES is an approximation to a

[3]Using a puncturable PRF

multilinear map, roughly speaking.[4] While this technique was first used in [150], it has lead to a number of further constructions using essentially the same technique [12, 23, 27, 66, 153, 239, 253]. We give a full treatment of the abstract obfuscation model that we consider in VII;2.5, along with differences regarding each of the specific instantiations.

All-known candidates are proven secure under uber-assumptions that essentially posit that the scheme is secure. Moreover, the GES operations are idealised with security proven in what is known as the *ideal multilinear map model* [150]. Variations on this model have been investigated [153, 240], but they only differ in the types of queries that are permissible.

All known candidates for IO in this model are subject to polynomial-time cryptanalysis, though the attacks depend on the GES that is used to instantiate them. We give a brief overview of the current state of cryptanalysis of these IO candidates in Section VII;1.6. All the cryptanalytic techniques exploit the gaps between the idealised models of graded encoding schemes [15, 87, 88, 90, 103, 240, 256] and the existing instantiations.

Constructions for obfuscating programs directly. The works of Applebaum and Brakerski [16], Zimmerman [298] and Döttling et al. [132] construct program obfuscation that is directly applicable to arithmetic circuits. This helps to avoid the usage of Barrington's theorem and the subsequent blow-up in the branching program size. Broadly speaking, these candidates treat programs as arithmetic circuits and implement gate operations directly as GES operations. This allows the adversary to compute the circuits directly without having to perform matrix multiplications, allowing for much simpler constructions as a result. The constructions are much more nuanced, but still lack concrete proofs of security and admit their own polynomial-time cryptanalyses via insecurities in the underlying graded encoding schemes [101, 256].

Constructions from constant-degree, ideal multilinear maps. Recent work [13, 14, 222, 223, 224, 225] has established provably secure[5] instantiations of IO obfuscators from concrete instantiations of ideal multilinear maps [57]. These obfuscators all require constant-degree graded encodings, which may be easier to instantiate than more general, polynomial-depth graded encodings. The work of [224] requires an ideal construction of trilinear maps; and non-standard invocations of blockwise local PRGs. However, recent cryptanalysis suggests that even constructions of trilinear maps based on familiar assumptions evade us.[6] In a notable improvement, Ananth et al. [13] provide a tentative construction under bilinear maps and LWE, though their work also rests upon non-standard conjectures.

This avenue for achieving a construction of IO under plausible assumptions seems the most promising (at least to the thesis author), but this direction is tangential to the current chapter.

[4] We cover graded encoding schemes in more detail shortly.
[5] Albeit still under relatively poorly understood assumptions.
[6] Recent work from Huang [190] claims to construct candidate trilinear maps, but the claims are yet to be verified at the time of writing.

Constructions from 2-key PCPRFs. The work of Canetti and Chen [76] shows that IO can be achieved using a constrained PRF, that satisfies at least 2-key privacy. Unfortunately, all known constructions of constrained PRFs under standard assumptions only demonstrate 1-key privacy [21, 67, 68, 76, 255]. This includes the new construction that we presented in Chapter V. PCPRFs for m-key privacy (where $m \geq 2$) and bounded-depth circuit predicates are only available under primitives based on ideal multilinear maps [55].

VII;1.4 Graded encoding schemes

Graded encoding schemes provide a generalisation of the functionality provided by ideal multilinear maps (MMAPs), first established by Boneh and Silverberg [57]. A multilinear map is simply a generalisation of well-known cryptographic bilinear maps to larger linearity. That is, there are a set of source groups $\mathbb{G}_1, \ldots, \mathbb{G}_\kappa$, and a target group $\mathbb{G}_T$. Then a κ-degree (non-degenerate) map $e : \mathbb{G}_1 \times \ldots \times \mathbb{G}_\kappa \mapsto \mathbb{G}_T$, is a function that computes:

$$g_T \leftarrow e(g_1, \ldots, g_\kappa)$$

where $g_i \in \mathbb{G}_i$ is a generator for all $i \in [\kappa]$ and g_T is a generator of $\mathbb{G}_T$. The linearity of the function requires that, for $\alpha_i \in \mathbb{Z}$, then:

$$g_T^{\alpha_1 \ldots \alpha_\kappa} \leftarrow e(g_1^{\alpha_1}, \ldots, g_\kappa^{\alpha_\kappa}).$$

We can also define symmetric multilinear maps, where $\mathbb{G}_1 = \ldots = \mathbb{G}_\kappa$.

Arguably, κ-degree graded encoding schemes provide a much more expressive structure. Instead of defining a single map e, they provide functions e_V, where $V \subseteq [\kappa]$. Then:

$$g_V \leftarrow e_V(\{g_i\}_{i \in V});$$

and g_V can be used in future evaluations by functions e_W, where $V \subseteq W \subseteq [\kappa]$. In these cases, we treat g_V as an element corresponding to all the indices in V. Therefore, we can only evaluate:

$$g_W \leftarrow e_W(g_V, \{g_j\}_{j \in (W \setminus V)}),$$

i.e. we cannot re-use indices.

The name graded encoding comes from the fact that the elements of the scheme are treated as *encodings* that permit 'levelled' homomorphic operation in a similar manner as in a homomorphic encryption scheme. In our nomenclature, 'levels' correspond to the group structure that the encodings implicitly construct. The encodings are required to satisfy some notion of security (e.g. one-wayness).

The difference between a GES and a homomorphic encryption scheme is the fact that the encodings cannot be explicitly decrypted. In contrast, many applications of a GES — including those that are satisfied by MMAPs — only require the ability to determine whether an encoding encodes the same element or not. For a group setting, we can treat g^{α_1} as an encoding of the value $\alpha_1 \in \mathbb{Z}$. This is a deterministic encoding that immediately reveals when another encoding has the same value, thus providing an effective 'zero-test' whereby an evaluator can check whether an encoding encodes the value 0, or not.

Current instantiations of graded encodings permit only noisy approximations of such encodings, thus an encoding is essentially part of an equivalence class for a given value. As such, we must explicitly define a zero-test procedure that allows checking the equality of two encodings, to achieve full functionality. This zero-test procedure can only be carried out on encodings that are indexed by the top-level set $\mathcal{U} = [\kappa]$.

We explain the formal concept of such schemes more clearly in Section VII;2.1.

VII;1.5 Constructions and cryptanalysis of graded encoding schemes

The first GES was introduced by Garg, Gentry and Halevi [148], and will be known throughout as GGH. Their construction takes inspiration from the NTRU encryption scheme [186]. As mentioned above, GGH provides noisy approximations of graded encodings and thus gives zero-testing procedures for checking the equality of elements. They also show that extracting a canonical representation of the value of such encodings is possible. This is necessary for many applications.

Subsequently, there have essentially been two other candidates achieving the same functionality. First, Coron, Lepoint and Tibouchi [104] demonstrate a similar design but taken over the integers. Their scheme is based on the homomorphic encryption scheme of [128]. Secondly, Gentry, Gorbunov and Halevi [157] provide an altogether different structure where the graded operations align to computation over a directed graph. Their construction uses trapdoor sampling techniques of [4, 161, 237] for providing the graph structure. Again, they provide an explicit mechanism for testing whether the encodings of elements, encoded along the same path segments, encode the same value.

Similarly to IO, MMAPs have become a cornerstone of recent cryptographic constructions, the following [10, 54, 55, 58, 151, 297] naming just a few. In this chapter, we will focus on the security of the GGH GES in the IO framework.

Security. Proving security of the schemes is a tricky proposal. Firstly, the security assumptions that are required for graded encoding schemes can differ depending on the application. The most well-known assumption is the multilinear DDH (MDDH) assumption [57], a generalisation of the bilinear DDH assumption. We describe MDDH in Figure VII;2.1, the assumption is essential for constructing non-interactive key exchange (NIKE) [144, 148].

Indeed, proving that MDDH holds for any any of the three candidates is plausibly difficult. This is because the zero-testing procedure is implemented as a form of decryption with a *faulty* secret key. This faulty secret key is a public parameter and should only permit learning whether an encoding encodes the value of 0, or not. Proving security of the schemes with this parameter present is difficult since it inherently reveals some information about the secret parameters of the scheme. Indeed, this is where almost all cryptanalysis of graded encoding schemes originates.

A common feature of most applications is that encodings of zero have to be provided below the top-level of computation (indexed by a subset $T \subseteq \mathcal{U}$). These encodings of zero are used to rerandomise publicly revealed encodings and are vital for key applications such as non-interactive key exchange [104, 148]. Unlike these applications, IO constructions that are instantiated from a GES do not require the presence of such encodings of zero. IO only requires that the encodings of zero that can be found are indexed at the top-level of computation (e.g. by the set $\mathcal{U}$).

Zeroizing attacks. The cryptanalysis of [148] was initiated in the work itself, showing that a weak DL attack could be performed against encodings of certain structures. This was improved by Hu and Jia [189], showing that the secret parameters of the encoding scheme could be learnt due to the presence of low-level encodings of zero. Consequently, the MDDH assumption cannot hope to hold for GGH. We provide a better overview of the attack surface and methods in Section VII;2.1. The cryptanalysis of [104] was started by Cheon et al. [89]. Their attack is similarly as devastating and also makes use of the presence of lower-level encodings of zero.

The family of attacks described above are better known as *zeroizing* attacks. Similar variations on these attacks are also known against GGH15, breaking the prime application of NIKE [102]. We reiterate that the presence of lower-level encodings of zero is vital for the attack to succeed, thus these attacks do not work in the situation of IO, where they are not given to the adversary.

Lattice-subfield attacks on GGH. A different strain of attacks considered in [5, 91, 210] target the GGH GES. In particular, they show that the parameter choices made in [148], and subsequent optimisations [6, 219], allow the NTRU assumption to be broken in polynomial-time. This does not immediately break the GGH GES, but it does lead to a quantum polynomial-time attack for revealing the secret parameters of the scheme (alternatively a classical sub-exponential time attack). These attacks can be avoided by changing the dimension parameter in the GGH scheme.

VII;1.6 Cryptanalysis of branching program obfuscation

In candidate branching program obfuscators, the presence of lower-level encodings of zero are not required for functionality. In fact, the only encodings of zero that are necessary can be found only after encodings have reached the top-level of computation. Lower-level encodings of zero are vital to the functionality of zeroizing attacks and so vulnerabilities in candidate obfuscators do not arise. That being said, a branch of zeroizing-like attacks introduced by Coron et al. [101] can be used to break the candidate obfuscators of [16, 160, 298], when instantiated over the [104] GES. These attacks were later improved by Coron et al. [103]. We do not focus on these attacks explicitly as they do not seem to affect IO candidates that are built from GGH.

Most cryptanalysis of graded encoding schemes wrt the application of IO takes a slightly different flavour. Narrowing our focus to GGH, the attacks that we consider in this chapter focus on the ability to build algebraic relations during homomorphic computations, which can then be exposed after zero-testing. In short, the adversary in the $\mathsf{exp}^{\mathsf{io}}_{b,\mathcal{A}}(1^\lambda)$ security games constructs polynomials over functionally equivalent[7] branching program evaluations, rendering predictable zero-testing outputs for one program and not the other. These polynomials are known as *annihilating polynomials*, and it was shown by [15, 240] that such polynomials can be constructed in $\mathsf{poly}(\lambda)$ time for the obfuscators of [12, 23, 27, 64, 239, 253]. *Annihilation attacks* that distinguish IO obfuscated circuits using these polynomials provide the strongest set of known cryptanalysis against IO candidates.

Crucially, the attacks of [240] highlighted the insufficiency of the ideal multilinear map model (explained in Section VII;4) in abstracting the functionality of the GGH GES. In short, the zero-test oracle provides only a 'success' or 'failure' response in this ideal model. The [240] attack explicitly uses the output of the algebraic zero-testing procedure to launch their annihilation attacks. As a means of inspiring future constructions, [240] justifies using a 'weakened' MMAP model, where the adversary is able to submit polynomials evaluated over the responses from the zero-testing procedure. The adversary wins if they can successfully construct annihilating polynomials over the values of the zero-tested encodings. In this model, many candidates fail to provide security [12, 23, 27, 66, 239, 253].

The work of [240] only targets specific choices of branching programs, rather than the result of applying Barrington's theorem to circuits. Apon et al. [15] show how to expand the attacks to the scenario where the branching programs are the result of applying this theorem. Their analysis explicitly uses functionally equivalent branching programs that express a feature known as 'partial inequivalence'. The algebraic techniques that are used are essentially the same.

[7] The branching programs have to give the same output on all inputs for the security notion to hold.

Preventing annihilation attacks. The reason that annihilation attacks do not break all IO candidates (i.e. not [150]) instantiated over GGH is the additional structure provided in the obfuscation candidate. The [150] candidate uses extra randomisation features in the structure of the branching program that seemingly prevents formulating an annihilating polynomial in polynomial-time. In response to the work of [240], Garg et al. [153] construct an obfuscator with similar design choices as [150] with security proven in the weakened MMAP model, based on an assumption establishing that annihilating outputs from random branching program outputs is computationally difficult. The obfuscated branching programs have additional randomised structure embedded into them; alongside the functionality of the circuit. This additional randomisation prevents learning meaningful output using annihilating polynomials. Another construction by Döttling et al. [132] was also proven secure in the weakened model under similar assumptions, though it targets obfuscations of circuits directly, rather than using branching programs.

Expanding cryptanalysis. Subsequently, Chen, Gentry and Halevi [87] provide a more devastating attack that uses similar annihilating techniques, their methods provide polynomial-time cryptanalysis of the original obfuscator of [150], along with a restricted version of the [153] obfuscator (that is not covered by the given security proof). The technique only applies to branching programs that are 'input-partitionable'.

We do not expand on this cryptanalysis since it is out of the scope that we consider, but it seems to provide an initial criticism of the weakened MMAP model. In short, it appears to be insufficient for considering the security of IO candidates. Of particular relevance is the cryptanalysis of the [153] obfuscator, where the analysis only applies to single-input branching programs, rather than the dual-input programs considered in the actual work. This is important, since dual-input programs are not input-partitionable by definition. The weakened model specification does not consider the structure of branching programs. This highlights a way of prising the concrete security of IO candidates away from the model that they're considered in.

We also note that Fernando et al. [138] show that any input-partitioning branching program can be converted into an equivalent branching program that does not admit the partitioning behaviour, this means that the [87] can be heuristically prevented. Though, this does not give any additional commentary on the applicability of the weakened MMAP model as a valid mechanism for proving security.

Attacks in the weakened model. Finally, we can remark on two very recent proposals that categorically establish that the weakened graded encoding model is insufficient for considering security. Firstly, Cheon et al. [90] give a polynomial-time cryptanalysis of all known IO candidates based on GGH. Their work extends the subfield attacks of [5, 91, 210] into an attack on all IO candidates; using a generic technique for removing the level sets that encodings in the branching program are indexed by. The attacks eventually results in distinguishing the obfuscations using the

Obfuscator (from ggh)	quantum attack	classical attack
[12, 23, 27, 239, 253]	[15, 90, 240, 256]	[15, 90, 240]
[150] (without [138])	[87, 90]	[87, 90]
[150] (with [138])	[90]	[90]
[66]	[15, 90, 240]	[15, 90, 240]
[153]	[90, 256]	[90]

Table VII;1: Cryptanalysis of IO candidates from the GGH GES.

inherent algebraic structure of the GGH GES, as shown in [240]. Their attack works over general branching programs (e.g. from Barrington's theorem) and does not require the input-partitioning property. The only structure that is required is that the branching programs are *linear relationally inequivalent*. It should also be stated that the attacks can be prevented by adopting the parameter changes that prevent the original subfield attacks, or by taking $\lambda = \mathsf{poly}(\kappa)$.

Secondly, the work of Pellet-Mary [256] gives quantum polynomial-time attacks that specifically target the [132, 153] obfuscators. This attack is very different to the previous literature and uses the fact that zero-testing on the GGH GES can be done at levels *higher* than the top-level that is established. The only quantum part of the attack is the requirement of running a solver for the principal ideal problem, though this runs in classical sub-exponential time also [42, 108].

In summary, understanding and managing the broad overview of growing cryptanalytic literature on IO candidates instantiated over GGH can be a task of considerable burden. Therefore, we have provided a concise summary of the state of candidates and corresponding cryptanalysis in Table VII;1.

Clearly, there are no obfuscators standing that can be instantiated under the GGH GES, as it is specified in [148]. The attack of [90] features prominently but this cryptanalysis can be thwarted by parameter modification.

VII;1.7 Our contributions

In this chapter, we will provide a slightly different angle on the cryptanalysis of GGH. We focus on the statement of Halevi [176] who (roughly) proposed that:

> *"The core computational hardness problem for breaking the GGH GES, is to find a representative of the algebraic ideal $\langle g \rangle$ during computation."*

This ideal is present implicitly in all encodings from an instantiation of GGH. Finding a representative of this ideal is crucial to *all* of the cryptanalysis that we described above.[8] The most general method is to use the annihilation attacks of [240]. While [153] emerged as a method for immunising IO candidates against the possibility of annihilation attacks, our work originated from the exploration of attempting to prevent the attacks at the GES-level.

We construct a novel variant of GGH where the element g that is common to all encodings is replaced with an element that is common to only each level in the encoding scheme. This immediately prevents the possibility of finding ideals that can be used for distinguishing the obfuscation of branching programs under this new scheme. We call this new scheme "GGH without ideals" (GGHWOI).

Perhaps unsurprisingly, we show that there are vulnerabilities in the scheme which prevent it for being used in the context of a fully-fledged MMAP (i.e. using the hardness of MDDH). Of more interest is that we show that the new scheme admits vulnerabilities in the IO setting. Crucially, the vulnerabilities arise from a cryptanalytic method inspired by the algebraic cryptanalyses given in [15, 240]. Our analysis represents a non-trivial adaptation, since the original methods do not translate directly to our setting.

The conclusion that we derive is that the encodings from GGH-like schemes admit structural vulnerabilities that are insufficiently related to the presence of the common generator g. We show that these vulnerabilities are still present when the generator is replaced with non-common ring elements. We detail variations of the attacks given in [240] to break the obfuscators of [12, 23, 27, 66, 239, 253] when instantiated using GGHWOI in the weakened MMAP model. Our cryptanalysis works only in the symmetric GES setting (Section VII;2.1). The same cryptanalysis is not possible against the [150, 153] obfuscators for the same reasons as given in [240].

For analysing security, and as a sub-contribution, we interpret the weakened MMAP model as a game-based definition. The adversary now only has oracle access to the obfuscated programs, but has real access to the encodings of the graded encoding schemes. The model is sufficient since the previous attacks against GGH IO candidates can be instantiated without modification. Moreover, we believe that this model offers a more plausible attack surface, reducing the distance between actual constructions and the idealised models that they are considered in.

Finally for asymmetric graded encoding schemes, the level sets inhibit us from launching our attack; see Section VII;6 for more details. While it is likely that an adaptation exists that will map our attacks into the asymmetric case, we have been unable to find it. Currently, this appears to give a separation between the security of GGH and GGHWOI; since the attacks of [15, 240, 256] work regardless of the choice of level sets. We leave further exploration as potential future work.

[8]This element g is not to be confused with the group generators used in the abstract definition of graded encoding schemes that we gave previously.

Obfuscator (from gghwoi)	classical attacks (symmetric)	classical attacks (asymmetric)
[12, 23, 27, 66, 239, 253]	This chapter (based on [8, 9])	none
[150] (without [138])	[94]	[94]
[150] (with [138])	none	none
[153]	none	none

Table VII;2: Cryptanalysis of IO candidates from the gghwoi GES.

Discussion. Subsequent work by Chunsheng [94] expanded our results to the case of the [150] obfuscator, using techniques similar to the attacks of [87]. We believe that expanding the attack surface against gghwoi to all the cases where ggh is vulnerable would help us to better understand the weaknesses of the ggh GES. Another possibility is showing that the security of the two schemes are linked via a security reduction. As a summary, we provide a table of the current cryptanalysis of IO candidates when instantiated under gghwoi in Table VII;2.

We discuss the possibility (and failure) of our cryptanalysis applying to the more robust obfuscators of [150, 153] in Section VII;6.3. We also re-emphasise that we do not hold confidence in the gghwoi GES for maintaining security in the cases where no attacks are currently known, as we are unable to prove any meaningful security regarding the scheme.

VII;2 Preliminaries

VII;2.1 Multilinear maps and graded encodings

As we discussed in the introduction, multilinear maps provide a generalisation of the functionality provided by bilinear maps to κ-degree polynomials. Graded encoding schemes provide a more expressive functionality, effectively defining a levelled structure indexed by the set $[\kappa]$. Effectively, encodings $\$a$ and $\$b$ are indexed by subsets $S_a, S_b \subset [\kappa]$. Then, we can add $\$a + \$b = \$c$ where $S_a = S_b$, to get an encoding of the value $c = a + b$ indexed by $S_c = S_a = S_b$. In addition, if $S_a \cup S_b \subseteq [\kappa]$, then we can compute $\$d = \$a \cdot \$b$ to recover an encoding of $d = ab$ indexed by $S_c = S_a \cup S_b \subseteq [\kappa]$.

The final piece of functionality is known as zero-testing and is governed by a public parameter p_{zt}. We define a function $b \leftarrow \mathsf{ZeroTest}(\$x)$ for an encoding $\$x$ indexed by $[\kappa]$, where $b = 1$ when $x = 0$, with overwhelming probability.

All known instantiations of graded encoding schemes use additional noise to prevent encoded values from being reversed [104, 148, 157]. As a result, x admits many different encodings in the space $\mathcal{Y}$, dependent on the noise that is used to construct $\$x$. Additionally, the noise grows as operations are computed. Parameters are typically chosen so that noise growth permits a maximum of $\kappa - 1$ multiplications.

The example that we gave above corresponds to an *asymmetric GES*. In a *symmetric GES*, encodings are indexed by a value $T \in [\kappa]$ and encodings indexed by T_1, T_2 can be added if $T_1 = T_2$ and multiplied if $T_1 + T_2 \leq \kappa$. Zero-testing on an encoding is permitted when it is indexed by the set $\mathcal{U} = \kappa$. We will use the asymmetric notation when describing the GGH GES in the future. In this notation, recall that we have $\mathcal{U} = [\kappa]$ and $T \subseteq [\kappa]$. From now on, we will use $[a|T]$ to represent an encoding of a wrt $T \subseteq [\kappa]$.

Definition VII;2.1 [Graded encoding scheme (asymmetric)]

Let $\kappa = \mathsf{poly}(\lambda)$. We define an asymmetric graded encoding scheme, ges, as the tuple $(\mathsf{Setup}, \mathsf{Encode}, \mathsf{Add}, \mathsf{Mult}, \mathsf{ZeroTest})$. The algorithms are defined as below.

- $(\mathsf{msk}, \mathsf{pp}) \leftarrow \mathsf{ges.Setup}(1^\lambda, 1^\kappa)$: For 1^λ and multilinearity 1^κ; outputs a key pair $(\mathsf{msk}, \mathsf{pp}) \in \mathcal{K}_{\mathsf{msk}} \times \mathcal{K}_{\mathsf{pp}}$;
- $[a|T] \leftarrow \mathsf{ges.Encode}(\mathsf{msk}, a, T)$: For inputs $\mathsf{msk} \in \mathcal{K}_{\mathsf{msk}}$, $a \in \mathcal{X}$ and $T \subseteq [\kappa]$; outputs an encoding $[a|T] \in \mathcal{Y}$;
- $[c_+|T] \leftarrow \mathsf{ges.Add}(\mathsf{pp}, [a|T], [b|T])$: For inputs $\mathsf{pp} \in \mathcal{K}_{\mathsf{pp}}$, $[a|T] \in \mathcal{Y}$, $[b|T] \in \mathcal{Y}$ and where $T \subseteq [\kappa]$; outputs an encoding $[c_+|T] \in \mathcal{Y}$;
- $[c_\times|T] \leftarrow \mathsf{ges.Mult}(\mathsf{pp}, [a|T_1], [b|T_2])$: For inputs $\mathsf{pp} \in \mathcal{K}_{\mathsf{pp}}$, $[a|T_1] \in \mathcal{Y}$, $[b|T_2] \in \mathcal{Y}$ and where $T = T_1 \cup T_2 \subseteq [\kappa]$; outputs an encoding $[c_\times|T] \in \mathcal{Y}$;
- $b \leftarrow \mathsf{ges.ZeroTest}(\mathsf{pp}, [c|[\kappa]])$: For $\mathsf{pp} \in \mathcal{K}_{\mathsf{pp}}$ and $[c|[\kappa]] \in \mathcal{Y}$; outputs a bit $b \in \{0, 1\}$.

```
$\mathsf{exp}_{b,\mathcal{A}}^{\kappa\text{-mddh}}(1^\lambda, 1^\kappa)$
1 : $(\mathsf{msk}, \mathsf{pp}) \leftarrow \mathsf{ges.Setup}(1^\lambda, 1^\kappa)$;
2 : $\{[a_i|1] \leftarrow \mathsf{ges.Encode}(\mathsf{msk}, a_i \in \mathcal{X}, 1)\}_{i\in[\kappa+1]}$;
3 : if $b = 0$ :
4 :     $z \leftarrow \mathsf{ges.Encode}(\mathsf{msk}, \prod_{i=1}^{\kappa+1} a_i, \kappa)$;
5 : elseif $b = 1$ :
6 :     $z \leftarrow_\$ \mathcal{Y}$;
7 : $b_\mathcal{A} \leftarrow \mathcal{A}(\mathsf{pp}, \kappa, \{[a_i|1]\}_{i\in[\kappa+1]}, z)$;
8 : return $b_\mathcal{A}$;
```

Figure VII;2: Experiments for defining the κ-MDDH assumption. We use the symmetric GES representation for ease of notation, but asymmetric equivalents are also possible.

Definition VII;2.2 [Correctness]

Let $\kappa = \mathsf{poly}(\lambda)$, then we say that ges satisfies correctness iff the following statements are true.

$$\Pr\left[b = 1 \,\middle|\, \begin{array}{c} c{=}0,\ (\mathsf{msk},\mathsf{pp}){\leftarrow}\mathsf{ges.Setup}(\lambda,1^\kappa), \\ [c|[\kappa]]{\leftarrow}\mathsf{ges.Encode}(\mathsf{msk},c,[\kappa]), \\ b{\leftarrow}\mathsf{ges.ZeroTest}(\mathsf{pp},[c|[\kappa]]) \end{array}\right] > 1 - \mathsf{negl}(\lambda); \qquad \text{(VII;1)}$$

$$\Pr\left[b = 0 \,\middle|\, \begin{array}{c} c{\neq}0,\ (\mathsf{msk},\mathsf{pp}){\leftarrow}\mathsf{ges.Setup}(\lambda,1^\kappa), \\ [c|[\kappa]]{\leftarrow}\mathsf{ges.Encode}(\mathsf{msk},c,[\kappa]), \\ b{\leftarrow}\mathsf{ges.ZeroTest}(\mathsf{pp},[c|[\kappa]]) \end{array}\right] > 1 - \mathsf{negl}(\lambda); \qquad \text{(VII;2)}$$

$$\Pr\left[b = 1 \,\middle|\, \begin{array}{c} a_1,\dots,a_\ell{\in}X,\ C{\in}\mathcal{C},\ T_1,\dots,T_\ell{\subseteq}[\kappa], \\ c{\leftarrow}C(a_1,\dots,a_\ell), \mathcal{U}{=}C(T_1,\dots,T_\ell){=}[\kappa], \\ (\mathsf{msk},\mathsf{pp}){\leftarrow}\mathsf{ges.Setup}(\lambda,1^\kappa), \\ [-c|[\kappa]]{\leftarrow}\mathsf{ges.Encode}(\mathsf{msk},-c,[\kappa]), \\ \{[a_j|T_j]{\leftarrow}\mathsf{ges.Encode}(\mathsf{msk},a_j,T_j)\}_{j\in[\ell]}, \\ [c_\times|[\kappa]]{\leftarrow}\mathsf{ges.Eval}[\mathsf{pp},C,[a_1|T_1],\dots,[a_\ell|T_\ell]], \\ b{\leftarrow}\mathsf{ges.ZeroTest}(\mathsf{pp},\mathsf{ges.Add}(\mathsf{pp},[-c|[\kappa]],[c_\times|[\kappa]])) \end{array}\right] > 1 - \mathsf{negl}(\lambda); \qquad \text{(VII;3)}$$

We write $\mathsf{ges.Eval}(\mathsf{pp}, C, [a_1|T_1], \dots, [a_\ell|T_\ell])$ to denote the implementation of C over the underlying encoded values, using $\mathsf{ges.Add}$ and $\mathsf{ges.Mult}$.

Finally, for security we use the multilinear decisional diffie-hellman (MDDH) assumption as a key barometer for assessing the security of a GES. The work of [148] gives the κ-graded DDH assumption, asking for security on intermediate levels as well. We only specify the idealised MDDH assumption — but written using GES convention — since this is enough for most applications. For a specific choice of $\kappa = \mathsf{poly}(\lambda)$, we write κ-MDDH to refer to the κ-linear decisional diffie-hellman problem.

Assumption VII;2.1 [κ-MDDH]

Let $\kappa = \mathsf{poly}(\lambda)$, and let $\mathsf{exp}_{b,\mathcal{A}}^{\kappa\text{-mddh}}(1^\lambda, 1^\kappa)$ denote the experiments given in Figure VII;2. We say that a ges satisfies κ-MDDH security iff:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{ges}(1^\lambda, \kappa\text{-}\mathsf{mddh}))) < \mathsf{negl}(\lambda)$$

where $\mathcal{A}$ is any PPT algorithm.

Intuitively, the κ-MDDH problem is hard to solve because the adversary only has access to a GES with κ-linearity. Whereas the output is the multiplication of $\kappa + 1$ values when $b = 0$.

VII;2.2 The ggh graded encoding scheme

Let $\phi(X) = X^n + 1$ and let $\mathcal{R} = \mathbb{Z}[X]/(\phi(X))$; we let $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ be the quotient ring of $\mathcal{R}$ wrt a 'large' modulus $q \in \mathbb{N}$. Let $g \leftarrow\!\!\$\, \mathcal{R}^\times$ be a 'small' invertible element in $\mathcal{R}$. We require that the ideal $\langle g \rangle$ is prime with prime norm. Let $\mathcal{X} = \mathcal{R}_g$ be the plaintext space, and $\mathcal{Y} = \mathcal{R}_q$ be the encoding space. We specify the ggh graded encoding scheme in Construction VII;2.1.

Construction VII;2.1 [ggh GES]

We provide a description of the asymmetric GES, ggh, designed by [148].

- ggh.$\mathsf{Setup}(1^\lambda, 1^\kappa)$: Sample parameters q, n and let $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$. Sample $\{z_i\}_{i\in[\kappa]} \leftarrow\!\!\$\, \mathcal{R}_q$, $g \leftarrow\!\!\$\, \mathcal{R}^\times$. Sample $h \leftarrow\!\!\$\, \mathcal{R}_q$ s.t. $\|h\|_2 \approx \sqrt{q}$. Let $p_{zt} = h \cdot \prod_{i=1}^{\kappa} z_i \cdot g^{-1}$. Output $\mathsf{pp} = (p_{zt}, \mathcal{R}, q, n, \kappa)$ and $\mathsf{msk} = (\mathsf{pp}, g, h, \{z_i\}_{i\in[\kappa]})$.
- ggh.$\mathsf{Encode}(\mathsf{msk}, a \in \mathcal{R}_g, T \subseteq [\kappa])$: Sample $r \leftarrow\!\!\$\, \mathcal{R}$ as a 'small' element, let

$$[a|T] = (a + rg)/\prod_{i\in T} z_i \mod q,$$

  and output $v = [a|T]$.
- ggh.$\mathsf{Add}(\mathsf{pp}, [a|T], [b|T])$: Output $[a|T] + [b|T] \mod q$.
- ggh.$\mathsf{Mult}(\mathsf{pp}, [a|T_1], [b|T_2])$: Output $[a|T_1] \cdot [a|T_2] \mod q$.
- ggh.$\mathsf{ZeroTest}(\mathsf{pp}, [a|T])$: Let $\rho = p_{zt} \cdot [a|T] \mod q$ and output 1 if $\|\rho\|_2 < q^{0.75}$ and 0 otherwise.

Parameter settings. For the correctness requirements to follow, it is necessary that encodings of zero have ℓ_2 norms bounded by $q^{0.75}$. To achieve this, we can take:

- $n = \Omega(\kappa\lambda^2)$ (primarily to avoid sub-exponential principle ideal solvers [42]);
- sample each coefficient of $g \leftarrow_\$ \mathcal{R}(D_{\mathbb{Z}^n,\sigma})$ s.t. $\|g\|_2 = O(n\sqrt{\lambda})$ with overwhelming probability (i.e. a $(0, n\sqrt{\lambda})$-bounded discrete Gaussian distribution centred at 0, with $\sigma = \sqrt{n\lambda}$ by Lemma II;2.2);
- sample each coefficient of $r \leftarrow_\$ \mathcal{R}(D_{\mathbb{Z}^n,\sigma'})$ s.t. $\|r\|_2 = O(n^2\lambda)$ with overwhelming probability (i.e. a $(0, n^2\lambda)$-bounded discrete Gaussian centred at 0, with $\sigma' = \sigma = n^{1.5}\lambda$);
- each encoded value $a \in \mathcal{R}_g$;
- let $q \geq 2^{8\kappa\lambda} \cdot n^{O(\kappa)}$;[9]
- sample each coefficient of $h \leftarrow_\$ \mathcal{R}(D_{\mathbb{Z}^n,\sigma''})$ s.t. $\|h\|_2 = \Theta(\sqrt{nq})$ with overwhelming probability (i.e. a $(0, \sqrt{nq})$-bounded discrete Gaussian centred at 0, with $\sigma'' = \sqrt{q}$).

Let $\chi_g = \mathcal{R}(D_{\mathbb{Z}^n,\sigma})$, $\chi_r = \mathcal{R}(D_{\mathbb{Z}^n,\sigma'})$ and $\chi_h = \mathcal{R}(D_{\mathbb{Z}^n,\sigma''})$ from now on.

Lemma VII;2.1 [Correctness]

GGH is correct, wrt to the parameter choices made above.

Proof. Firstly, consider:

$$v \leftarrow \textsc{ggh}.\mathsf{Encode}(\mathsf{msk}, 0, [\kappa]),$$

then $v = rg/\prod_{i\in[\kappa]} z_i$ for $r \leftarrow_\$ \chi_r$, $g \leftarrow_\$ \chi_g$. Also, $p_{zt} = h \cdot \prod_{i\in[\kappa]} z_i \cdot g^{-1}$, for $h \leftarrow_\$ \chi_h$.

Then $b_v \leftarrow \textsc{ggh}.\mathsf{ZeroTest}(\mathsf{pp}, v)$ is the bit output by the zero-testing algorithm. Therefore, $p_{zt} \cdot v = rh$ and $\|r\|_2 \leq n^2\lambda$ and $\|h\|_2 \leq \sqrt{nq}$ whp. As such, $\|rh\|_2 \leq n^{2.5}\sqrt{q}$; and since $q > n^{O(\kappa)}$, then clearly we can choose $0.75c > 5$, let $q = n^{c\kappa}$. In this situation, $b_v = 1$ with the same high probability, and thus zero-testing is correct.

For the second inequality in Equation (VII;2) (and $v \leftarrow \textsc{ggh}.\mathsf{Encode}(\mathsf{msk}, a \neq 0, [\kappa])$), note that $t = p_{zt} \cdot v = ag^{-1}h + rh$. With overwhelming probability, g^{-1} is distributed uniformly in $\mathcal{R}_q$ and thus we have that $\|t\|_2 > q^{0.75}$, and thus $b_v = 0$.

[9]We can omit the factor of $2^{8\kappa\lambda}$ in the case of obfuscation [256], but it is necessary for re-randomisation purposes generally.

For the final inequality in Equation (VII;3), the only difference is that the magnitude of the elements that we consider are raised to the power of κ. Since q depends on κ also, then the same choice of c ensures that the correctness requirement is met. □

Zeroizing attacks. So-called *zeroizing* attacks, were first given by [189], but have become a prominent tool in cryptanalysis against all known GES candidates. The implications of these attacks is that the κ-MDDH problem is solvable in PPT for all of the current candidates.

Firstly we should note that [148, 189] showed that a weak form of the discrete log problem can be easily computed for GGH encodings. That is, for an encoding $[a|T]$, then we can find a' s.t. $a' \ \mathtt{mod} \ \langle g \rangle = a$. While this does not immediately allow us to break the hardness of κ-MDDH, it allows for breaking other assumptions (such as subgroup membership and decision linear). Hu and Jia [189] showed that this vulnerability could be expanded into a full attack on the κ-MDDH problem, using the presence of lower-level encodings of zero. Therefore, in general settings, we consider [148] to be insecure.

Zeroizing for IO. The reason that these attacks do not filter down to constructions of IO based on GGH, is due to the fact that there are no public encodings of zero provided to the adversary. These restricted GES instantiations were first coined as *multilinear jigsaw puzzles*. The only differences being that encodings of zero were only plausibly constructed at the top-level of computation, and that the only computations that are allowed are valid multilinear forms over the encodings. In fact, the idealised security models that IO constructions are proven secure within provide *only* this functionality, and nothing more. We describe these abstractions in Section VII;2.3.

VII;2.3 Idealised MMAP models

For proving the security of obfuscators, it has become necessary to abstract the functionality of multilinear maps and graded encoding schemes. In these models, it is possible to give security proofs that the candidates are secure (albeit under extra, largely experimental or contrived assumptions). Part of this necessity is that it is not clear what assumption should be satisfied by the underlying GES. In particular, while the κ-MDDH problem is used as a vital building block in [14, 222, 223, 224, 225], none of the GES candidates can be shown to provide security in this setting.

Informally, the idealised models provide the adversary with only oracle access to the GES algorithms. The oracle performs the necessary computations and returns random handles corresponding to the new encodings that the adversary has access to. A common aspect of the models is that the adversary wins automatically if it creates a valid encoding of zero indexed by a set that is below the top-level. This is to mimic the strength of zeroizing attacks [1, 23].

We provide the description of two models differing only in zero-testing functionality. The first model, the *ideal graded encoding model* proposed by [150], returns only a bit corresponding to whether the encoding encodes zero or not. In reality, the zero-testing algorithm returns an algebraic element — and extra computation is needed to determine whether this outputs corresponds to an encoding of zero or not. This extra functionality is captured in the *weakened graded encoding model*, proposed by [240]. The necessity for the weakened model will be made clearer in Section VII;4.

Henceforth, we will redefine the $\mathsf{ges.ZeroTest}$ algorithm as a combination of two separate algorithms $\mathsf{ges.ZeroTest}'(\mathsf{pp}, [a|T])$ and $\mathsf{ges.ZeroTest}''(\mathsf{pp}, \rho)$, where

$$\rho \leftarrow \mathsf{ges.ZeroTest}'(\mathsf{pp}, [a|[\kappa]]); \qquad \text{(VII;4)}$$

and

$$b \leftarrow \mathsf{ges.ZeroTest}''(\mathsf{pp}, \rho). \qquad \text{(VII;5)}$$

In particular, ρ is an algebraic element and b is the bit that is output by $\mathsf{ges.ZeroTest}$. That is, we define the zero-testing algorithm as:

$$b \leftarrow \mathsf{ges.ZeroTest}''(\mathsf{pp}, \mathsf{ges.ZeroTest}'(\mathsf{pp}, [a|[\kappa]])); \qquad \text{(VII;6)}$$

where $\rho = [a|[\kappa]] \cdot p_{zt}$ in the language of GGH.

In the following definitions, we define the oracles $\mathcal{O}^{setup}$, $\mathcal{O}^{enc}_{\mathcal{X}}$, $\mathcal{O}^{+}_{\mathcal{Y}^2}$, $\mathcal{O}^{\times}_{\mathcal{Y}^2}$ and $\mathcal{O}^{zt}_{\mathcal{Y}}$; corresponding to each of the algorithms defined in the tuple ges, respectively. The oracle definitions are given in Figure VII;3 and Figure VII;4.[10]

We also provide an alternative zero-testing $\mathcal{O}^{zt^\dagger}_{\mathcal{Y}}$ (Figure VII;4) and post-zero-testing oracles $\mathcal{O}^{p:zt}_{\mathcal{Y}^m}$ (Figure VII;5) that are used explicitly by the weakened model. Intuitively, the post-zero-testing oracle allows the adversary to submit polynomials Q evaluated over the algebraic elements $\rho^1, \ldots, \rho^m$ received from $\mathsf{ges.ZeroTest}'(\mathsf{pp}, [a^i|[\kappa]])$. The adversary succeeds if it finds polynomials that are non-zero in the encoding space $\mathcal{Y}$, and zero in the plaintext space $\mathcal{X}$. This specifically targets the case of GGH, where $\mathcal{X} = \mathcal{R}_g$ and $\mathcal{Y} = \mathcal{R}_q$, indicating that the monomials of $Q(\rho^1, \ldots, \rho^m)$ all contain a factor of g. This is broadly how [240] construct their annihilation attack, that we will explain in Section VII;4.

[10]We do not actually provide the adversary with access to the oracle $\mathcal{O}^{enc}_{\mathcal{X}}$ in the ideal models. This mirrors the functionality provided by the GGH GES.

```
$\mathcal{O}^{setup}(1^\lambda, 1^\kappa, 1^\ell, \{(a_1, T_1), \ldots, (a_\ell, T_\ell)\})$
1 :  $\mathsf{pp}^\dagger \leftarrow \mathsf{ges.Setup}(1^\lambda, 1^\kappa)$;
2 :  $R = \{\}$;
3 :  for $i \in [\ell]$ :
4 :     $c_i \leftarrow\!\$\ \{0,1\}^\lambda$;
5 :     $R[c_i] = (a_i, T_i, P_i = \{\})$;
6 :  $\mathsf{pp} = (\mathsf{pp}^\dagger, \kappa, \{c_i, T_i\}_{i\in[\ell]})$;
7 :  $\mathsf{msk} = (R, (\{a_i, T_i, P_i\}_{i\in[\ell]}), \mathsf{pp}^\dagger)$;
8 :  return $(\mathsf{pp}, \mathsf{msk})$;
```

```
$\mathcal{O}^{\circ\in\{+,\times\}}_{\mathcal{Y}2}(\mathsf{msk}, (c_1, T_1), (c_2, T_2))$
1 :  $\{(a_i, T_i, P_i) \leftarrow R[c_i]\}_{i\in\{1,2\}}$;
2 :  if $(\circ = +) \wedge ((T_1 \neq T_2) \vee (T_1 \not\subseteq [\kappa]))$ :
3 :     return $\perp$;
4 :  if $(\circ = \times) \wedge (T_1 \cup T_2 \not\subseteq [\kappa])$ :
5 :     return $\perp$;
6 :  $a = a_1 \circ a_2$; $c \leftarrow\!\$\ \{0,1\}^\lambda$;
7 :  $T = []$;
8 :  if $(\circ = +)$ :
9 :     $T = T_1$;
10 : if $(\circ = \times)$ :
11 :    $T = T_1 \cup T_2$;
12 : if $(a = 0) \wedge \neg(T \overset{?}{=} [\kappa])$ :
13 :    return "adversary wins";
14 : $P \leftarrow (a_1, a_2, \circ)$;
15 : $R[c] = (a, T, P)$;
16 : return $c$;
```

Figure VII;3: LEFT: Ideal GES model setup oracle.
RIGHT: Ideal GES model operation oracle for handling additions and multiplications. In this experiment, if the adversary creates a lower-level encoding of zero, they automatically win and the model is aborted (line 13). The set P keeps track of the operations that have been computed on each encoding.

```
$\mathcal{O}^{enc}_{\mathcal{X}}((a, T))$
1 :  if $T \not\subseteq [\kappa]$ :
2 :     return $\perp$;
3 :  $c \leftarrow\!\$\ \{0,1\}^\lambda$;
4 :  $R[c] = (a, T, \{\})$;
5 :  return $(c, T)$;
```

```
$\mathcal{O}^{zt}_{\mathcal{Y}}(\mathsf{msk}, c)$
1 :  $(a, T, P) \leftarrow R[c]$;
2 :  if $T \neq [\kappa]$ :
3 :     return $\perp$;
4 :  if $(a = 0)$ :
5 :     return $1$;
6 :  else :
7 :     return $0$;
```

```
$\mathcal{O}^{zt^\dagger}_{\mathcal{Y}}(\mathsf{pp}, c)$
1 :  $(a, T, P) \leftarrow R[c]$;
2 :  if $T \neq [\kappa]$ :
3 :     return $\perp$;
4 :  $p_{zt} \leftarrow \mathsf{pp}$;
5 :  if $Z = \emptyset$ :
6 :     $Z = \{\}$; $\mathsf{msk} \leftarrow Z$;
7 :  $\gamma^1(a) \leftarrow (a, T, P)$;
8 :  $c^\gamma \leftarrow\!\$\ \{0,1\}^\lambda$;
9 :  $Z[c^\gamma] = \gamma^1(a)$;
10 : if $(a = 0)$ :
11 :    return $(1, c^\gamma)$;
12 : else :
13 :    return $(0, c^\gamma)$;
```

Figure VII;4: LEFT: Ideal GES model encoding oracle.
CENTER: Ideal GES model zero-test oracle.
RIGHT: Weakened GES model zero-test oracle. The usage of $\gamma^1(a)$ is to present the adversary with a general target for annihilation. This target is dependent on the value of the final encoding, and the polynomial P evaluated over the underlying encoded values.

```
$\mathcal{O}^{p:zt}_{\mathcal{Y}^m}(\mathsf{pp}, (c^\gamma_1, \ldots, c^\gamma_m), Q)$
1 :  for $i \in [m]$ :
2 :    $\gamma^1_i(a) \leftarrow Z[c^\gamma_i]$;
3 :    if $\gamma^1_i(a) = \emptyset$ :
4 :      return $\perp$;
5 :  if $(Q \equiv 0 \in \mathcal{Y})$ :
6 :    return $0$;
7 :  elseif $(Q(\{\gamma^1_i(a)\}_{i\in[m]}) = 0 \in \mathcal{X})$ :
8 :    return $1$;
```

Figure VII;5: Post-zero-testing oracle.

Definition VII;2.3 [Ideal graded encoding model [150]]

The *ideal graded encoding model* supplies a PPT algorithm $\mathcal{A}$ access to the oracles:

- $\mathcal{O}^{setup}(1^\lambda, 1^\kappa, 1^\ell)$: Figure VII;3[LEFT];
- $\mathcal{O}^{\circ\in\{+,\times\}}_{\mathcal{Y}^2}(c^1, c^2)$: Figure VII;3[RIGHT];
- $\mathcal{O}^{zt}_{\mathcal{Y}}(c)$: Figure VII;4[CENTER].

We do not provide access to the oracle $\mathcal{O}^{enc}_{\mathcal{X}}$.

Definition VII;2.4 [Weakened graded encoding model [240]]

The *weakened graded encoding model* supplies a PPT algorithm $\mathcal{A}$ access to the oracles:

- $\mathcal{O}^{setup}(1^\lambda, 1^\kappa, 1^\ell, \{(a_1, T_1), \ldots, (a_\ell, T_\ell)\})$: Figure VII;3[LEFT];
- $\mathcal{O}^{\circ\in\{+,\times\}}_{\mathcal{Y}^2}(c^1, c^2)$: Figure VII;3[RIGHT];
- $\mathcal{O}^{zt^\dagger}_{\mathcal{Y}}(c)$: Figure VII;4[RIGHT];
- $\mathcal{O}^{p:zt}_{\mathcal{Y}^m}(\mathsf{pp}, \{c^\gamma_i\}_{i\in[m]}, Q)$: Figure VII;5.

We do not provide access to the oracle $\mathcal{O}^{enc}_{\mathcal{X}}$.

We should take some time to describe the use of $\gamma^1(a)$ in the oracles above. Concretely, when considering annihilation attacks, there are linear polynomials in some formal variables that have known coefficients. These polynomials are the target of annihilation attacks for all GES candidates, an example can be seen later in Equation (VII;10). The form of $\gamma^1(a)$ can be derived from the underlying encoded values and the polynomial that is used to construct the encoding, thus $\gamma^1(a)$ can be derived explicitly from the set (a, T, P), where P lists the underlying encoded values and operations, used to construct the encoding of a. Notice that $\mathcal{A}$ also knows P since it queries the operations itself.

We intentionally keep the description of the ideal models that we use generic, so that they may apply to different GES instantiations. It would undoubtedly be easier to derive these models from the specific instantiation of the GGH GES. The works of [30, 153, 235, 240] derive different ideal models depending on the GES candidate that is used. Fortunately, all the models can be reduced to a simplified model where the adversary simply has to specify a polynomial Q of the form described in Figure VII;5. Therefore, using the generalised model is applicable.

VII;2.4 Matrix branching programs

Let $q, \kappa, \ell_{\mathsf{in}}, \eta = \mathsf{poly}(\lambda)$ be parameters dependent on the security parameter λ, and let $w \in \mathbb{Z}$. Let $\mathsf{inp} : [\kappa] \mapsto [\ell_{\mathsf{in}}]^\eta$ be an 'input' function. Let $\{\boldsymbol{M}_{l,b_1,\ldots,b_\eta}\}$ be a set of matrices individually sampled from $\mathbb{Z}_q^{w\times w}$, for $b_1, \ldots, b_\eta \in \{0,1\}$ and $l \in [\kappa]$. Let $\boldsymbol{m}_0 \in \mathbb{Z}_q^w$, $\boldsymbol{m}_{\kappa+1} \in \mathbb{Z}_q^w$ be two vectors, these are known as 'bookends' and are used for guaranteeing a single element output. Define

$$\mathbb{M} := (\kappa, \ell_{\mathsf{in}}, \eta, w, \mathsf{inp}, \{\boldsymbol{M}_{l,b_1,\ldots,b_\eta}\}_{\substack{b_1,\ldots,b_\eta\in\{0,1\},\\ l\in[\kappa]}}, \boldsymbol{m}_0, \boldsymbol{m}_{\kappa+1}), \qquad \text{(VII;7)}$$

to be a matrix branching program (MBP) of length L, input length ℓ_{in}, width w and arity η.[11] We can evaluate $\mathbb{M}$ on inputs $x \in \{0,1\}^{\ell_{\mathsf{in}}}$ where $x_s = x[s]$ and we denote the output of such an evaluation by $\mathbb{M}(x) \in \{0,1\}$. Let $\boldsymbol{x}_{\mathsf{inp}(l)} = (x^1_{\mathsf{inp}(l)[1]}, \ldots, x^\eta_{\mathsf{inp}(l)[\eta]})$ where $\boldsymbol{x} = (x^1, \ldots, x^\eta) \in \{0,1\}^{\eta\ell_{\mathsf{in}}}$ is a vector of η specified inputs, we evaluate the branching program on input $\boldsymbol{x}$ by computing:

$$\mathbb{M}(\boldsymbol{x}) := \boldsymbol{m}_0^T \cdot \left(\prod_{l=1}^{\kappa} \boldsymbol{M}_{l,\boldsymbol{x}_{\mathsf{inp}(l)}}\right) \cdot \boldsymbol{m}_{\kappa+1}. \qquad \text{(VII;8)}$$

The input function inp chooses the bits in the input x that are examined at each layer l of the branching program. Clearly $|\mathsf{inp}(i)| = r$ where $\mathsf{inp}(i)[y]$ is equal to the y^{th} component of $\mathsf{inp}(i)$. In total, we have that the branching program contains $2^\eta\kappa$ matrices and 2 bookend vectors.

Using Barrington's theorem we can associate a circuit C with a branching program $\mathbb{M}_C$. This theorem is commonly used in the construction of IO candidates. The branching program (without bookend vectors) evaluates to the identity matrix on an input x if and only if $C(x) = 0$. The bookend vectors make it possible to structure a branching program $\mathbb{M}_C$ such that $\mathbb{M}(x) = 0$ when $C(x) = 0$ (e.g. see [12, 23, 27, 66, 239, 240, 253, 256]). We do not need to consider the

[11] We typically use $w = 5$ for the dimension of the matrices/vectors as this is sufficient for Barrington's theorem [29].

specifics of this transformation and so we will just formalise branching programs explicitly assuming that this can be done.

Definition VII;2.5 [Input function]

Let $\ell_{\mathsf{in}}, \kappa, \eta = \mathsf{poly}(\lambda)$ and $w \in \mathbb{Z}$. We say that $\mathsf{inp} : [\kappa]^w \mapsto [\ell_{\mathsf{in}}]^\eta$ is an η-input function if it maps all values in $[\kappa]^w$ to indices in $[\ell_{\mathsf{in}}]^w$.

Construction VII;2.2 [Matrix branching program]

Let $r, w = \mathsf{poly}(\lambda)$ and let $\mathsf{inp} : [\kappa]^\eta \mapsto [\ell_{\mathsf{in}}]^\eta$ be an η-input function (Definition VII;2.5). Then a *w-width, η-input branching program* $\mathbb{M}$, is a set denoted by

$$\mathbb{M} := (\kappa, \ell_{\mathsf{in}}, \eta, w, \mathsf{inp}, \{\boldsymbol{M}_{l,b_1,\ldots,b_\eta}\}_{\substack{b_1,\ldots,b_\eta \in \{0,1\},\\ l \in [\kappa]}}, \boldsymbol{m}_0, \boldsymbol{m}_{\kappa+1});$$

where κ is the 'depth' of $\mathbb{M}$; $\boldsymbol{M}_{l,b_1,\ldots,b_\eta} \in \mathbb{Z}_q^{w \times w}$ are square matrices of dimension w and $\boldsymbol{m}_0, \boldsymbol{m}_{\kappa+1} \in \mathbb{Z}_q^w$ are w-dimensional vectors.
We write $\mathbb{M}(x^1, \ldots, x^\eta)$, for $x^i \in \{0,1\}^{\ell_{\mathsf{in}}}$ and $i \in [\eta]$, to denote the evaluation

$$\mathbb{M}(x^1, \ldots, x^\eta) = \boldsymbol{m}_0^T \cdot \left(\prod_{i=1}^{\kappa} \boldsymbol{M}_{l, x^1_{\mathsf{inp}(l)[1]}, \ldots, x^\eta_{\mathsf{inp}(l)[\eta]}} \right) \cdot \boldsymbol{m}_{\kappa+1}.$$

This is the *evaluation* of the branching program.

Lemma VII;2.2 [Barrington's theorem [29]]

Let $C \in \mathcal{C}_{\mathsf{NC}^1}$, let $\boldsymbol{b} = (b^1, \ldots, b^\eta) \in \{0,1\}^\eta$ and let $\boldsymbol{x} = (x^1, \ldots, x^\eta) \in \{0,1\}^{\eta \ell_{\mathsf{in}}}$. Let $\mathsf{inp} : [\kappa] \mapsto [\ell_{\mathsf{in}}]^\eta$ be an input function. Then there exists an efficiently computable matrix branching program

$$\mathbb{M} = (\kappa, \ell_{\mathsf{in}}, \eta, 5, \mathsf{inp}, \{\boldsymbol{M}_{l,\boldsymbol{b}}\}_{l \in [\kappa], \boldsymbol{b} \in \{0,1\}^\eta}, \boldsymbol{m}_0, \boldsymbol{m}_{\kappa+1})$$

where $\boldsymbol{M}_{l,b} \in \mathbb{Z}_q^{5 \times 5}$, $\boldsymbol{m}_0, \boldsymbol{m}_{\kappa+1} \in \mathbb{Z}_q^5$; and s.t. $\mathbb{M}(\boldsymbol{x}) = 0$ iff $C(\boldsymbol{x}) = 0$; otherwise $\mathbb{M}(\boldsymbol{x}) \neq 0$.

If $\eta = 1$ then we say that $\mathbb{M}$ is a *single-input* branching program, and likewise *dual* if $\eta = 2$. There are no IO obfuscators requiring $\eta > 2$, so we will ignore branching programs that use such values in the future. We may omit mention of w in the future since it is fixed by Barrington's theorem.

Remark VII;2.1. *We say that $\mathbb{M}$ is a read-($\lfloor \kappa/\ell_{\mathsf{in}} \rfloor$) branching program if each bit of $\boldsymbol{x}$ is read $\kappa/\ell_{\mathsf{out}}$ times. We assume that $\mathsf{inp}(i) = i \mod \ell_{\mathsf{in}}$, i.e. that the input is read sequentially, and that $\kappa = 0 \mod \ell_{\mathsf{in}}$. Such a branching program is known as 'oblivious'. There are generic techniques for rewriting branching programs such that they satisfy these properties [87, 234]. In fact, and as noted by [234], the case of read-once single-input branching programs are considered a degenerate case for the attacks that we later describe. Therefore, we require that $\kappa = c\ell_{\mathsf{in}}$ where $c > 1$ is an integer.*

Finally, we explicitly the define the property of functional equivalence for branching programs.

Definition VII;2.6 [Functional equivalence]

We say that two branching programs, $\mathbb{M}$ and $\mathbb{M}'$, are *functionally equivalent* if, for all valid inputs $\boldsymbol{x} \in \{0,1\}^{\eta\ell_{\mathsf{in}}}$, then:

$$\Pr\big[\mathbb{M}'(\boldsymbol{x}) = 0 \,\big|\, \mathbb{M}(\boldsymbol{x}) = 0\big] = 1.$$

VII;2.5 Abstract obfuscators

In order to capture a broad class of obfuscators, we use an abstract obfuscator that mimics the functionality provided by *BGK-type* [240] obfuscators. In short, BGK-type obfuscators are those of [12, 23, 27, 66, 239, 253]. We only consider these obfuscators, since this set will be the focus of our security analysis in Section VII;6. It is also the same set of obfuscators considered by [240]. The abstract model was first alluded to in [101] and expanded in [15, 240, 256]. The abstract model concretely describes the functionality provided by [12, 23, 239] and can be augmented easily to describe [27, 66, 253].

For a circuit C, let $\mathbb{M} = (\kappa, \ell_{\mathsf{in}}, \eta, w, \{\boldsymbol{M}_{l,b}\}_{b\in\{0,1\},l\in[\kappa]}, \boldsymbol{m}_0, \boldsymbol{m}_{\kappa+1})$ be the corresponding η-input branching program obtained from Barrington's theorem (Lemma VII;2.2). Let ges be the GGH graded encoding scheme given in Construction VII;2.1. We describe our abstract obfuscator in Construction VII;2.3, based on the obfuscators of [240, 256].

Construction VII;2.3 [Abstract obfuscator]

Let $\mathbb{M}$ and ges be defined, as above; let $\boldsymbol{b} = (b^1, \ldots, b^\eta) \in \{0,1\}^\eta$. Our abstract obfuscation model uses the following steps.

1. Let $(\mathsf{msk}, \mathsf{pp}) \leftarrow \mathsf{ges.Setup}(1^\lambda, 1^{\kappa+2})$.
2. Embed the matrices $\{\boldsymbol{M}_{l,\boldsymbol{b}}\}_{\substack{\boldsymbol{b}\in\{0,1\}^\eta, \\ l\in[\kappa]}}$ into the ring $\mathcal{R}_g$.
3. Sample $\kappa + 1$ invertible matrices $\{\boldsymbol{K}_l\}_{l\in[\kappa+1]} \leftarrow_\$ \mathcal{R}_g$.
4. Let $\widehat{\boldsymbol{M}_{l,\boldsymbol{b}}} = \boldsymbol{K}_l \boldsymbol{M}_{l,\boldsymbol{b}} \boldsymbol{K}_{l+1}^{-1}, \widehat{\boldsymbol{m}}_0 = \boldsymbol{m}_0 \boldsymbol{K}_1^{-1}, \widehat{\boldsymbol{m}}_{\kappa+1} = \boldsymbol{K}_{\kappa+1} \boldsymbol{m}_{\kappa+1}$; this process is known as *Kilian randomisation*.
5. Sample $\alpha_{l,\boldsymbol{b}} \leftarrow_\$ \mathcal{R}_g^\times$ and compute $\widehat{\boldsymbol{M}_{l,\boldsymbol{b}}} = \alpha_{l,\boldsymbol{b}} \cdot \widehat{\boldsymbol{M}_{l,\boldsymbol{b}}}$; also sample $\alpha_0, \alpha_{\kappa+1}$ and compute $\widehat{\boldsymbol{m}}_0 = \alpha_0 \widehat{\boldsymbol{m}}_0, \widehat{\boldsymbol{m}}_{\kappa+1} = \alpha_{\kappa+1} \widehat{\boldsymbol{m}}_{\kappa+1}$. These are known as multiplicative bundling scalars. We let $\widehat{\mathbb{M}}$ be the branching program containing matrices $\widehat{\boldsymbol{M}}_{l,x_{\mathsf{inp}(l)}}$ and vectors $\widehat{\boldsymbol{m}}_0, \widehat{\boldsymbol{m}}_{\kappa+1}$.
6. Let $\mathcal{U}$, be some index set and let $\mathcal{T} \leftarrow \mathsf{partition}(\mathcal{U}, \kappa, \eta)$. Here, $\mathsf{partition}(\mathcal{U}, \kappa, \eta)$ is the algorithm that generates the set

$$\mathcal{T} = \{\{T_{l,\boldsymbol{b}}\}_{l\in[\kappa], \boldsymbol{b}\in\{0,1\}^t}, T_0, T_{\kappa+1}\};$$

s.t. $\{\{T_{l,\boldsymbol{x}_{\mathsf{inp}(l)}}\}_{l\in[\kappa]}, T_0, T_{\kappa+1}\}$ is a partition of $\mathcal{U}$ for each $\boldsymbol{x} \in \{0,1\}^{\eta\ell_{\mathsf{in}}}$. This is a *straddling sets partition*.
7. Let:

$$\widetilde{\mathbb{M}} = (\kappa, \ell_{\mathsf{in}}, \eta, \mathsf{inp}, \{\widetilde{\boldsymbol{M}}_{l,\boldsymbol{b}}\}_{\substack{l\in[\kappa] \\ \boldsymbol{b}\in\{0,1\}^\eta}}, \widetilde{\boldsymbol{m}}_0, \widetilde{\boldsymbol{m}}_{\kappa+1}) \leftarrow \widehat{\mathbb{M}}.\mathsf{Encode}(1^\lambda, 1^\kappa, \mathsf{ges}, \mathcal{T});$$

where $\widehat{\mathbb{M}}.\mathsf{Encode}(1^\lambda, 1^\kappa, \mathsf{ges}, \mathcal{T})$ is described in Figure VII;6.
8. Evaluate $C(\boldsymbol{x})$, for $x \in \{0,1\}^{\eta\ell_{\mathsf{in}}}$ by computing:

$$\widetilde{\mathbb{M}}(\boldsymbol{x}) = \widetilde{\boldsymbol{m}}_0{}^T \left(\prod_{l=1}^{\kappa} \widetilde{\boldsymbol{M}}_{l,\boldsymbol{x}_{\mathsf{inp}(l)}} \right) \widetilde{\boldsymbol{m}}_{\kappa+1},$$

and setting $C(\boldsymbol{x}) = 1 - b$ where $b \leftarrow \mathsf{ges.ZeroTest}(\mathsf{pp}, \widetilde{\mathbb{M}}(\boldsymbol{x}))$.

Unless we state otherwise, we will assume obfuscation of single-input programs as standard when considering cryptanalysis. For the case of annihilation attacks, dual-input programs tend not to prevent cryptanalysis [240] (unless the branching program is required to be input-partitioning [87]).

```
$\mathbb{M}.\mathsf{Encode}(1^\lambda, 1^\kappa, 1^\eta, \mathsf{ges}, \mathsf{msk}, T_0, T_{\kappa+1}, \{T_{l,\boldsymbol{b}}\}_{l\in[[\kappa]],\boldsymbol{b}\in\{0,1\}^\eta})$
1 : $\widetilde{\boldsymbol{M}}_{l,\boldsymbol{b}} \leftarrow \mathsf{ges.Encode}(\mathsf{msk}, \boldsymbol{M}_{l,\boldsymbol{b}}, T_{l,\boldsymbol{b}})$;
2 : $\widetilde{\boldsymbol{m}}_0 \leftarrow \mathsf{ges.Encode}(\mathsf{msk}, \boldsymbol{m}_0, T_0)$;
3 : $\widetilde{\boldsymbol{m}}_{\kappa+1} \leftarrow \mathsf{ges.Encode}(\mathsf{msk}, \boldsymbol{m}_{\kappa+1}, T_{\kappa+1})$;
4 : $\widehat{\mathbb{M}} = (\kappa, \ell_{\mathsf{in}}, \mathbb{M}.\eta, \mathbb{M}.w, \mathbb{M}.\mathsf{inp}, \{\widetilde{\boldsymbol{M}}_{l,\boldsymbol{b}}\}_{l\in[\kappa],\boldsymbol{b}\in\{0,1\}^\eta}, \widetilde{\boldsymbol{m}}_0, \widetilde{\boldsymbol{m}}_{\kappa+1})$;
```

Figure VII;6: Procedure for encoding a branching program using a GES. We slightly abuse notation, and note that running $\mathsf{ges.Encode}$ for a vector or matrix corresponds to running the algorithm for each individual entry.

From now on, for a branching program $\mathbb{M}$, we will write $\widetilde{\mathbb{M}} \leftarrow \mathsf{io}(1^\lambda, 1^\kappa, 1^\eta, \mathbb{M}, \mathcal{T}, \mathsf{ges})$ to denote the obfuscation of $\mathbb{M}$ using the abstract construction above. We may omit the parameters λ, κ, η, w as inputs if they are obvious from context (and since they are implied by the instantiations of $\mathsf{ges}, \mathbb{M}, \mathcal{T}$).

Correctness of evaluation. We can prove that the obfuscated program satisfies the *functionality* property of Definition VII;1.1.

Lemma VII;2.3 [Functionality]

Construction VII;2.3 satisfies the functionality requirement of Definition VII;1.1.

Proof. Recall that $C(x) = 1 - b$, where

$$b \leftarrow \mathsf{ges.ZeroTest}(\mathsf{pp}, \widetilde{\mathbb{M}}(\boldsymbol{x})).$$

Note that

$$\begin{aligned}
\widetilde{\mathbb{M}}(\boldsymbol{x}) &= \widetilde{\boldsymbol{m}}_0 \left(\prod_{i=1}^{\kappa} \widetilde{\boldsymbol{M}}_{l,\boldsymbol{x}_{\mathsf{inp}(l)}} \right) \widetilde{\boldsymbol{m}}_{\kappa+1}; \\
&= \left[\widehat{\boldsymbol{m}}_0{}^T \left(\prod_{i=1}^{\kappa} \widehat{\boldsymbol{M}}_{l,\boldsymbol{x}_{\mathsf{inp}(l)}} \right) \widehat{\boldsymbol{m}}_{\kappa+1} \middle| \, \mathcal{U} \right]; \\
&= \left[\boldsymbol{m}_0^T \cdot \left(\prod_{i=1}^{\kappa} \boldsymbol{M}_{l,\boldsymbol{x}_{\mathsf{inp}(l)}} \right) \cdot \boldsymbol{m}_{\kappa+1} \middle| \, \mathcal{U} \right] \text{ (by Kilian randomisation)}; \\
&= [0 \,|\, \mathcal{U}] \text{ iff } C(x) = 0.
\end{aligned}$$

where $\boldsymbol{x}_{\mathsf{inp}(l)} = (x^1_{\mathsf{inp}(l)}, \dots, x^\eta_{\mathsf{inp}(l)})$ and $\mathcal{U} = [\kappa]$. Therefore, zero-testing and inverting the bit will provide the correct outcome whp. □

Adapting model for all BGK-type obfuscators. Our abstract model concretely includes the obfuscators of [12, 23, 239]. To include the other obfuscators that we have mentioned previously, it is possible to introduce small changes to our model based on the analysis of [240, 256].

- [27]: This obfuscator comes with an additional (possibly non-zero) value q_{acc}. Now, the definition is modified s.t. $C(\boldsymbol{x}) = 0$ for an input $\boldsymbol{x} \in \{0,1\}^{\eta\ell_{\mathsf{in}}}$ iff $\mathbb{M}(x) = q_{acc}$; rather than 0 as in Lemma VII;2.2. Therefore, the scalars $\alpha_{l,\boldsymbol{b}}$ might change the value of the output. Fortunately, it was shown by [240, 256] that this change can be incorporated into the above model with little cost.
- [66]: This obfuscator is similar to the [27] design and fits into the model [240]. It should be noted that it does not use a straddling sets partitioning function, and thus can be instantiated using a symmetric GES.
- [253]: This obfuscator uses matrices of the form:

$$\boldsymbol{M}_{l,\boldsymbol{b}} \mapsto \begin{pmatrix} \boldsymbol{M}_{l,\boldsymbol{b}} & \\ & \boldsymbol{I} \end{pmatrix}$$

  for some appropriately-sized identity matrix $\boldsymbol{I}$. However, this property does not change the semantics of the multiplication or the output of the branching program wrt to the circuit evaluation $C(\boldsymbol{x})$. Therefore, this construction also fits inside of our model [240, 256].

Non-BGK-type obfuscators. We also describe the differences between our abstract model, and the obfuscators of [150, 153] that are considered not to be part of the model. Similarly to the [253] obfuscator, they use branching programs with extra matrices embedded in the bottom-right quadrant. That is, the individual matrices take the form:

$$\boldsymbol{M}_{l,\boldsymbol{b}} \mapsto \begin{pmatrix} \boldsymbol{M}_{l,\boldsymbol{b}} & \\ & \boldsymbol{L}_{l,\boldsymbol{b}} \end{pmatrix}$$

where $\boldsymbol{L}_{l,\boldsymbol{b}}$ is a random square matrix of dimension s. To get the correct output, the bookend vectors are then chosen to be of the form:

$$\boldsymbol{m}_0 = (a_1^0, \ldots, a_5^0, \underbrace{0, \ldots, 0}_{s/2}, \underbrace{\$, \ldots, \$}_{s/2}), \;\; \boldsymbol{m}_{\kappa+1} = (a_1^{\kappa+1}, \ldots, a_5^{\kappa+1}, \underbrace{\$, \ldots, \$}_{s/2}, \underbrace{0, \ldots, 0}_{s/2});$$

so that the random elements cancel out in the evaluation.[12] We show later that the inclusion of these random matrices makes it plausibly more difficult to launch an annihilation attack in Section VII;6 on GGH-like schemes. This was already noted by [240] for the specific GGH scheme.

[12]The [150] obfuscator also uses a dummy branching program for subtracting from the standard evaluation for learning the final result.

Security in idealised model. To derive security, candidate obfuscators implement the GES operations that are required using the oracles that are defined in the idealised models of Definitions VII;2.3 and VII;2.4.

In particular, in the idealised models, the insecurities of the graded encoding schemes are abstracted away. This makes formulating security proofs easier [27, 153], but ignores the possibility of cryptanalysis against actual instantiations of the GES. The works of [27, 153] formulate assumptions that are plausible in the idealised and weakened graded encoding models respectively.

Garg et al. [153] introduced the branching program unannihilatability assumption (BPUA). In the weakened graded encoding model, this states that it is computationally difficult for the adversary to launch annihilation attacks on algebraic elements that are recovered from arbitrary branching program evaluations for the [153] obfuscator. They show that the assumption is implied by the existence of PRFs that can be evaluated by circuits in $\mathcal{C}_{\mathsf{NC}^1}$. But, the constructions are still vulnerable to attacks that exploit insecurities of GGH rather than the ideal model [256].

Straddling sets. A common feature of the algorithm $\mathsf{partition}(\mathcal{U}, \kappa, \eta)$ is to output a set of level sets

$$\mathcal{T} = \{T_0, T_{\kappa+1}, \{T_{l,\boldsymbol{b}}\}_{l\in[\kappa],\boldsymbol{b}\in\{0,1\}^\eta}\}$$

s.t. for any input $\boldsymbol{x} \in \{0,1\}^{\eta\ell_{\mathsf{in}}}$, then

$$\mathcal{U} = T_0 \cup \left(\bigcup_{l=1}^{\kappa} T_{l,\boldsymbol{x}_{\mathsf{inp}(l)}}\right) \cup T_{\kappa+1}.$$

While the straddling sets requirement is more nuanced, we do not need to explicitly define it here. We only require the knowledge that we need to sample a new $z_{l,\boldsymbol{b}}$ in GGH for each $T_{l,\boldsymbol{b}}$. The creation of the explicit partition $\mathcal{T}$ is handled by the algorithm $\mathsf{partition}(\mathcal{U}, \kappa, \eta)$ that was defined previously.

Objectives for our abstraction. As with previous works, we will use this abstracted version of an obfuscator for demonstrating plausible attacks against graded encoding schemes. We will also give a brief overview of attacks against the GGH GES using this model in Section VII;4, this will provide a warm-up to the later sections.

VII;2.6 Annihilating polynomials

Here, we list definitions and key results taken from the work of Kayal [202] that are crucial to our security analysis. In short, we articulate the formalisation of expressing algebraic dependencies for a set of polynomials sampled from a particular field.

Definition VII;2.7 [Annihilating polynomial [202]]

Let $f = (f_1, \dots, f_k)$ be a vector of k polynomials of degree $\leq d$, where each $f_i \in \mathbb{F}[y_1, \dots, y_n]$ is an n-variate polynomial over $\mathbb{F}$. A non-zero polynomial $A(f_1, \dots, f_k) \in \mathbb{F}[y_1, \dots, y_n]$ is said to be an *annihilating polynomial* for f if $A(f_1, \dots, f_k) = 0$. The polynomials $f_1, \dots, f_k$ are said to be *algebraically dependent* if such an annihilating polynomial exists, otherwise they are *algebraically independent*.

Definition VII;2.8 [Algebraic rank]

Let $f = (f_1, \dots, f_k)$ be a vector of k polynomials as above, where $f' \subseteq f$ is a subset of algebraically independent polynomials of maximal size k'. That is for any $\overline{f} \notin (f_1, \dots, f_k)$ but where $\overline{f} \in \mathbb{F}[y_1, \dots, y_n]$, then the set $f' \cup f_{k+1}$ is algebraically dependent. Then the *algebraic rank* of f is equal to k'.

Lemma VII;2.4 [Existence (Lemma 4 [202])]

Over any field, and for any set of polynomials in n variables, the algebraic rank of this set is at most n. That is, any set of $n + 1$ polynomials with coefficients taken from this field are algebraically dependent.

Lemma VII;2.5 [Theorem 2 [202]]

Let $f_1, \dots, f_k \in \mathbb{F}[x_1, \dots, x_n]$ be a set of k polynomials in n variables over the field $\mathbb{F}$. Then this set of polynomials has algebraic rank k if and only if the Jacobian matrix, $\boldsymbol{J} f(x)$, taken from $f_1, \dots, f_k$, has rank k.

Corollary VII;2.1 [Detecting dependencies [31, 202]]

There exists a PPT algorithm that on input a set of k arithmetic circuits over a field $\mathbb{F}$, determines if the polynomials computed by these arithmetic circuits are algebraically dependent or not.

Remark VII;2.2. *The algorithm mentioned by Corollary VII;2.1 essentially requires submitting random values in place of the variables in the Jacobian matrix $\boldsymbol{J} f(x)$. By the Schwarz-Zippel lemma (Lemma II;2.1), the rank of the symbolic matrix is likely to be the same as the rank of the matrix evaluated on random inputs with high probability.*

VII;3 New game-based cryptanalytic models

Using the abstracted interface from Section VII;2.3, we develop a novel set of game-based security definitions for analysing the security of IO candidates. The purpose with these definitions is to work with a more concise description of security, allowing the adversary greater access to the underlying GES. In particular, we do not provide access to physical branching program that has been obfuscated. Instead, we only provide oracle access to the program, in the sense that the adversary sends inputs $\boldsymbol{x}$ and the obfuscators evaluates the program and returns the result.

In contrast, the encodings that the adversary receives are explicitly taken from the GES that we use. This means that the adversary explicitly receives algebraic elements resulting from after zero-testing the output of a program. In the previous models, the adversary only receives handles that correspond to such evaluations. Our technique allows us to unify the different idealised models into one security consideration, depending on the GES and IO candidates that are used.

Our new models. We present two security models in this section. The first (IND-BP) serves as a warm-up to the second (IND-OBF). We use the IND-OBF game as an alternative method for analysing the security of IO candidates associated with a GES instantiation. We later demonstrate attacks against the GGHWOI GES (Section VII;5) in both of the models, when instantiating the abstract obfuscator from Construction VII;2.3. The use of IND-BP is a weaker model that is not necessarily realistic for achieving meaningful security. More importantly, it helps to unify the annihilation attack methods, which can then be adapted for the stronger IND-OBF model. To justify that our model is sufficient we show that Construction VII;2.3 is also insecure in IND-OBF, when instantiated with GGH, via an equivalence relation.

VII;3.1 IND-BP

The IND-BP ($\mathsf{INDBP}^{\mathsf{ges}}_{\kappa,\eta}$) Security requirement intuitively asks an adversary $\mathcal{A}$ to submit two functionally equivalent branching programs $\mathbb{M}_0$ and $\mathbb{M}_1$ (length κ, arity η) to a challenger. The challenger samples a bit $b \leftarrow_{\$} \{0,1\}$ and computes

$$\widehat{\mathbb{M}} \leftarrow \mathbb{M}_b.\mathsf{Encode}(1^\lambda, 1^\kappa, \mathsf{ges}, \mathcal{T})$$

for $\mathcal{T} \leftarrow \mathsf{partition}(\mathcal{U}, \kappa, \eta)$.

The adversary is then given oracle access to the encoded branching program; receiving zero-tested outputs from it. We define this oracle formally in Figure VII;7[RIGHT]. The zero-tested outputs are specifically algebraic elements $\rho \leftarrow \mathsf{ges.ZeroTest}'(\mathsf{pp}, \widehat{\mathbb{M}}(\boldsymbol{x}))$. The adversary asks m queries and learns $\rho_1, \ldots, \rho_m$, along with $(b_1, \ldots, b_m) \leftarrow \mathsf{ges.ZeroTest}''(\mathsf{pp}, \{\rho_i\}_{i\in[m]})$.

```
$\mathsf{exp}^{\mathsf{indbp}}_{b,\mathcal{A}}(1^\lambda, 1^\kappa, 1^\eta, \mathsf{ges})$
1 :  $\kappa, \eta, \mathbb{M}_0, \mathbb{M}_1 \leftarrow \mathcal{A}(1^\lambda)$;
2 :  $(\mathsf{msk}, \mathsf{pp}) \leftarrow \mathsf{ges.Setup}(1^\lambda, 1^\kappa)$;
3 :  $\mathcal{U} \leftarrow_{\$} \{0,1\}^\lambda$;
4 :  $\mathcal{T} \leftarrow \mathsf{partition}(\mathcal{U}, \kappa, \eta)$;
5 :  $\widehat{\mathbb{M}}_b \leftarrow \mathbb{M}_b.\mathsf{Encode}(1^\lambda, 1^\kappa, 1^\eta, \mathsf{ges}, \mathsf{msk}, \mathcal{T})$;
6 :  $b_{\mathcal{A}} \leftarrow \mathcal{A}^{\mathcal{O}(\widehat{\mathbb{M}}_b(\cdot))}(1^\lambda, \mathsf{pp})$;
7 :  return $b_{\mathcal{A}}$;
```

```
$\mathcal{O}(\widehat{\mathbb{M}}(\boldsymbol{x}))$
1 :  $z \leftarrow \widehat{\mathbb{M}}(\boldsymbol{x})$;
2 :  $\rho \leftarrow \mathsf{ges.ZeroTest}'(\mathsf{pp}, z)$;
3 :  $b_z \leftarrow \mathsf{ges.ZeroTest}''(\mathsf{pp}, \rho)$;
4 :  return $(b_z, \rho)$;
```

Figure VII;7: LEFT: Experiments for analysing the security of an encoded branching program wrt IND-BP. RIGHT: Oracle for evaluating inputs $\boldsymbol{x} \in \{0,1\}^{\eta\ell_{\mathsf{in}}}$ during $\mathsf{exp}^{\mathsf{indbp}}_{b,\mathcal{A}}(1^\lambda, 1^\kappa, 1^\eta, \mathsf{ges})$. We write $\mathcal{O}_{\widehat{\mathbb{M}}(\boldsymbol{x})}$.

At this point the adversary can apply any generic technique to try and distinguish the two branching programs. For instance, they can compute a polynomial Q s.t. $Q(\rho_1, \ldots, \rho_m) \neq 0 \mod \mathcal{Y}$ and $Q(\rho_1, \ldots, \rho_m) \equiv 0 \mod \mathcal{X}$. This would be equivalent to winning in the weakened model. Then based on the result of this computation, they submit $b_{\mathcal{A}}$ to the challenger and win if $b_{\mathcal{A}} = b$. The similarities with the weakened graded encoding model are clear, though we allow $\mathcal{A}$ greater freedom.

Let $\mathsf{exp}^{\mathsf{indbp}}_{b,\mathcal{A}}(1^\lambda, 1^\kappa, 1^\eta, \mathsf{ges})$ be the decisional experiment where $\mathcal{A}$ attempts to succeed in the game above, and define it as in Figure VII;7[LEFT]. We give a formalisation of the IND-BP security requirement in Definition VII;3.1.

Definition VII;3.1 [$\mathsf{INDBP}^{\mathsf{ges}}_{\kappa,\eta}$]

We say that a graded encoding scheme ges satisfies $\mathsf{INDBP}^{\mathsf{ges}}_{\kappa,\eta}$ security, if:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{ges}(1^\lambda, \mathsf{indbp}))) < \mathsf{negl}(\lambda)$$

where $\mathcal{A}$ is any PPT algorithm. We may also write that ges satisfies IND-BP security, for short.

```
$\mathsf{exp}^{\mathsf{indobf}}_{b,\mathcal{A}}(1^\lambda, 1^\kappa, 1^\eta, \mathsf{ges}, \mathsf{io})$
1 :  $\kappa, \eta, \mathbb{M}_0, \mathbb{M}_1 \leftarrow \mathcal{A}(1^\lambda)$;
2 :  $(\mathsf{msk}, \mathsf{pp}) \leftarrow \mathsf{ges.Setup}(1^\lambda, 1^\kappa)$;
3 :  $\mathcal{U} \leftarrow_{\$} \{0,1\}^\lambda$;
4 :  $\mathcal{T} \leftarrow \mathsf{partition}(\mathcal{U}, \kappa, \eta)$;
5 :  $\widetilde{\mathbb{M}}_b \leftarrow \mathsf{io}(\mathbb{M}_b, \mathsf{ges}, \mathsf{msk}, \mathcal{T})$;
6 :  $b_{\mathcal{A}} \leftarrow \mathcal{A}^{\mathcal{O}(\widetilde{\mathbb{M}}_b(\cdot))}(1^\lambda, \mathsf{pp})$;
7 :  **return** $b_{\mathcal{A}}$;
```

Figure VII;8: Decisional experiment for analysing the security of an obfuscated branching program wrt IND-OBF. The oracle $\mathcal{O}(\widetilde{\mathbb{M}}(\cdot))$ is the same as in Figure VII;7.

VII;3.2 IND-OBF

Let $(\mathsf{ges}, \mathsf{io})$ be a valid pairing of a GES with an IO candidate.[13] The IND-OBF security model differs only from IND-BP only in that:

$$\mathbb{M}_b.\mathsf{Encode}(1^\lambda, 1^\kappa, 1^\eta, \mathsf{ges}, \mathsf{msk}, \mathcal{T}) \longmapsto \mathsf{io}(\mathbb{M}_b, \mathsf{ges}, \mathsf{msk}, \mathcal{T}),$$

in line 6 (Figure VII;3.1[LEFT]). Intuitively, this means that we also use extra randomisation mechanisms that are delivered in Construction VII;2.3. The result is a weaker security guarantee, since the adversary must also account for the randomisations when computing the polynomial Q. We redefine the experiment in Figure VII;8 for completeness, but $\mathcal{O}(\widetilde{\mathbb{M}}(\cdot))$ stays the same. We denote the new decisional experiment by $\mathsf{exp}^{\mathsf{indobf}}_{b,\mathcal{A}}(1^\lambda, 1^\kappa, 1^\eta, \mathsf{ges}, \mathsf{io})$.

We write $\mathsf{INDOBF}^{\mathsf{ges},\mathsf{io}}_{\kappa,\eta}$ to make the parameter settings of IND-OBF explicit.

Definition VII;3.2 [$\mathsf{INDOBF}^{\mathsf{ges},\mathsf{io}}_{\kappa,\eta}$]

We say that a pair $(\mathsf{ges}, \mathsf{io})$ satisfies $\mathsf{INDOBF}^{\mathsf{ges},\mathsf{io}}_{\kappa,\eta}$ security, if:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{ges}(1^\lambda, \mathsf{indobf}))) < \mathsf{negl}(\lambda)$$

where $\mathcal{A}$ is any PPT algorithm. We may also write that $(\mathsf{ges}, \mathsf{io})$ satisfies IND-OBF security.

Comparisons with idealised security models. It is clear that IND-OBF places a stronger security requirement on the pair $(\mathsf{ges}, \mathsf{io})$ than in Definition VII;2.3. Notice that the adversary receives the algebraic outputs of $\mathsf{ges.ZeroTest}'(\cdot)$ in IND-OBF but only the binary result of zero-testing in the ideal graded encoding model. For the weakened model (Definition VII;2.4) the

[13]By valid, we simply mean that functionality is satisfied when instantiating io with ges.

definitions seem much closer and it is not obvious whether the differences result in a separation. We show in Section VII;4 that the attacks of [240] can be carried out in either model, and so IND-OBF is sufficient for our cryptanalysis.

VII;4 Annihilation attacks against GGH

We give a brief overview of the *annihilation attacks* that break the abstract obfuscator from Construction VII;2.3, when instantiated using the GGH GES in the weakened graded encoding model. We first show that the attacks are possible in the weakened graded model (Definition VII;2.4) — demonstrating the result of [240] — before showing that these attacks translate naturally to the IND-OBF model.

VII;4.1 Intuition

Let $\rho = [0|\mathcal{U}]$ be a top-level zero-tested encoding acquired from the GGH GES. Let $\boldsymbol{a}$ be the vector of plaintexts $(a_1, \ldots, a_m) \in \mathcal{R}_g^m$, and $\boldsymbol{r} = (r_1, \ldots, r_m) \in \mathcal{R}$ be the independent randomness that is used in creating the set of encodings $v_1, \ldots, v_m$, that are used to construct ρ. We write P_{v^*} to denote the polynomial that is evaluated over $(v_1, \ldots, v_m)$ to give $v^* = \rho \cdot p_{zt}^{-1}$. Then we know that ρ takes the form:

$$\rho = \gamma^1(\boldsymbol{a}, \boldsymbol{r}) + \gamma^2(\boldsymbol{a}, \boldsymbol{r})g + \ldots + \gamma^\kappa(\boldsymbol{a}, \boldsymbol{r})g^{\kappa-1}; \qquad \text{(VII;9)}$$

since zero-testing removes a factor of g, the denominators z_i, and the value of the encoding is $P_{v^*}(a_1, \ldots, a_m) = 0$. We omit mention of the parameter h since the presence of this element does not change anything about our analysis.

Importantly, $\gamma^1(\boldsymbol{a}, \boldsymbol{r})$ is a polynomial that is linear in the variables taken from $\boldsymbol{r}$. The attack scenario assumes that the adversary knows $\boldsymbol{a}$, but $\boldsymbol{r}$ remains hidden since these are sampled obliviously for each encoding. The fact that γ^1 is linear in these variables, means that γ^1 is a κ-variate polynomial. In addition, let $M > \kappa$ and consider polynomials $P_{v^*}^1, \ldots, P_{v^*}^M$ s.t. $P_{v^*}^i(\boldsymbol{a}) = 0$ for all $i \in [M]$. Then, by Lemma VII;2.4, an annihilating polynomial would exist for the set $(\gamma_1^1, \ldots, \gamma_M^1) \in \mathcal{R}_g$, and this set has maximum algebraic rank equal to κ.

Let Q be an annihilating polynomial for $(\gamma_1^1, \ldots, \gamma_M^1)$. Then $Q(\rho_1, \ldots, \rho_M) \equiv 0 \mod \langle g \rangle$, since $Q(\gamma_1^1, \ldots, \gamma_M^1) \equiv 0 \mod \langle g \rangle$ and because all the other monomials contain factors of g. Moreover, whp under the Schwarz-Zippel lemma (Lemma II;2.1), $Q(\rho_1, \ldots, \rho_M) \neq 0 \mod q$. This is the basis of the attacks of [15, 87, 240].

Expanding the attacks to IO candidates. There are some problems that prevent us from immediately destroying the security of IO candidates based on GGH. These problems can be summarised as the following.

- Is it possible to create $M > m$ top-level encodings of zero?
- How do we learn $\langle g \rangle$?
- Can the polynomial Q be represented by a circuit that is evaluated in polynomial-time?
- Can the attack be adapted to include randomness that is introduced during obfuscation?

To take care of this analysis [240] consider a weaker obfuscation model where branching programs are obfuscated rather than circuits (via Barrington's theorem).[14] This significantly reduces the complexity of the obfuscated circuits. In this setting, Miles et al. [240] show that for single-input branching programs, a concrete instantiation of Q exists for the abstract obfuscator defined previously. They use degenerate branching programs that output 0 on every input, although this still constitutes a valid break. We detail the format of these attacks in the next section.

For the dual-input case, they show that an efficiently computable annihilating polynomial Q exists, though they do not give a concrete instantiation. The work of [15] extends the attacks to branching programs obtained via Barrington's theorem (and thus to circuits in $\mathcal{C}_{\mathsf{NC}^1}$). We do not give a detailed analysis of these extensions since the methods that are used are very similar to the original single-input case.

It should be reinforced that all the attacks require a heuristic step, where a basis of the ideal $\langle g \rangle$ is created. This heuristic step is essential and requires finding tM zero-tested encoding of zero, of the form laid out above. Then computing Q for t sets of size M should allow for computing a spanning set of vectors for $\langle g \rangle$ with non-negligible probability (in the choice of $t \in \mathbb{Z}$). A similar heuristic analysis was used in the work of [101]. The resulting spanning set can be used to mimic the step of checking whether $Q(\rho_1^\dagger, \ldots, \rho_m^\dagger) \equiv 0 \mod \langle g \rangle$ for some test vector $(\rho_1^\dagger, \ldots, \rho_M^\dagger)$. Indeed this analysis is possible, even with the additional randomness injected via the obfuscation techniques.

VII;4.2 Annihilation attacks on Construction VII;2.3 over GGH

We analyse security in the weakened graded encoding model, let $\widetilde{\mathbb{M}} \leftarrow \mathsf{io}(\mathbb{M}, \mathcal{T}, \mathsf{ges} = \mathrm{GGH})$ for a branching program $\mathbb{M}$ of width 5 and arity 1. Firstly, notice that the encoded matrices received can be written as

$$\widetilde{\boldsymbol{M}}_{l,\boldsymbol{b}} = \widehat{\boldsymbol{M}}_{l,\boldsymbol{b}} + g\boldsymbol{R}_{l,b}$$

[14] We also consider this setting.

for each (l, b) in the branching program $\widetilde{\mathbb{M}}$, and similarly for the vectors $\widetilde{\boldsymbol{m}}_0, \widetilde{\boldsymbol{m}}_{\kappa+1}$. We ignore the denominators of the encodings since these are completely removed after zero-testing. By a change of variables on the matrices $\boldsymbol{R}_{l,b}$, it is then possible to write

$$\widetilde{\boldsymbol{M}}_{l,\boldsymbol{b}} = \alpha_{l,b}\boldsymbol{K}_l(\boldsymbol{M}_{l,\boldsymbol{b}} + g\boldsymbol{R}_{l,b})\boldsymbol{K}_{l+1}^{-1}$$

for invertible matrices $\boldsymbol{K}_l \in \mathcal{R}_g^{5\times5}$, and scalars $\alpha_{l,b} \in \mathcal{R}_g$.

Evaluating the branching program gives:

$$\widetilde{\mathbb{M}}(\boldsymbol{x}) = \gamma^1(\boldsymbol{a}, \boldsymbol{r}) + \gamma^2(\boldsymbol{a}, \boldsymbol{r})g + \ldots + \gamma^{\kappa+2}(\boldsymbol{a}, \boldsymbol{r})g^{\kappa+1}$$

and, specifically

$$\gamma^1(\boldsymbol{a}, \boldsymbol{r}) = \prod_{i=0}^{\kappa+1} \alpha_{i,b}\boldsymbol{M}_0\left(\sum_{l=0}^{\kappa+1} \ldots \boldsymbol{M}_{l-1,x_{\mathsf{inp}(l-1)}}\boldsymbol{R}_{l,x_{\mathsf{inp}(l)}}\boldsymbol{M}_{l+1,x_{\mathsf{inp}(l+1)}} \ldots\right)\boldsymbol{M}_{\kappa+1}. \tag{VII;10}$$

We are abusing notation here so that

$$\alpha_{0,x_{\mathsf{inp}(0)}} = \alpha_0,\ \alpha_{\kappa+1,x_{\mathsf{inp}(\kappa+1)}} = \alpha_{\kappa+1},$$

and

$$\boldsymbol{m}_0 = \boldsymbol{M}_0,\ \boldsymbol{m}_{\kappa+1} = \boldsymbol{M}_{\kappa+1}.$$

Recall that $\mathsf{inp}(i) = i \mod \ell_{\mathsf{in}}$. The idea behind the attack of [240] is to use functionally equivalent branching programs $\mathbb{M}_0$ and $\mathbb{M}_1$, where $\mathbb{M}_0$ consists of all identity matrices. Then $\mathbb{M}_1$ comprises all identity matrices on the $b = 0$ branch, and on the $b = 1$ branch there is a set of indices S s.t., for $(i \mod \ell_{\mathsf{in}}) \in S$, then $\boldsymbol{M}_{i,1} \neq \boldsymbol{I}$ and $\boldsymbol{M}_{i,1}^{-1} = \boldsymbol{M}_{i,1}$ (i.e. the matrices are self-inverse). For example, take the matrices with 1's on the anti-diagonal and 0's elsewhere.

We define ℓ_{in} and κ s.t. $N \cdot \ell_{\mathsf{in}} = \kappa$ for $N \in 2\mathbb{N}$. Additionally, if $i \in S$ and $i < \ell_{\mathsf{in}}$, then we require that $u\ell_{\mathsf{in}} + i \in S$ for all $u \in [N-1]$. Finally, we set $\boldsymbol{M}_{i,1} = \overline{\boldsymbol{M}}$ to be the same self-inverse matrix for all $i \in S$. Since $\mathsf{inp}(l) = l \mod \ell_{\mathsf{in}}$, then these matrices cancel on any evaluation where $(l, b) \in S \times \{1\}$. To see this, note that if $(l, b) \in S \times \{1\}$, then there is a set $(l, l+\ell_{\mathsf{in}}, l+2\ell_{\mathsf{in}}, \ldots, l+(N-1)\ell_{\mathsf{in}})$ of indices corresponding to non-identity matrices. Since N is even, then these matrices will all be in the final product of the evaluation and will cancel out, since they are self-inverse. As a result, both branching programs evaluate to 0 on all inputs.

Now, reconsider the form of the polynomial γ^1 in Equation (VII;10) for an input x, where $\exists$ $(j, x_{\mathsf{inp}(j)}) \in S \times \{1\}$. For $\mathbb{M}_0$, this will evaluate to

$$\prod_{i=0}^{\kappa+1} \alpha_{i,b} \boldsymbol{M}_0 \left(\sum_{l=0}^{\kappa+1} \boldsymbol{R}_{l,x_{\mathsf{inp}(l)}} \right) \boldsymbol{M}_{\kappa+1};$$

but for $\mathbb{M}_1$, it will evaluate to:

$$\prod_{i=0}^{\kappa+1} \alpha_{i,b} \boldsymbol{M}_0 \left(\sum_{l=0}^{\kappa+1} \ldots \boldsymbol{M}_{l-1,x_{\mathsf{inp}(l-1)}} \boldsymbol{R}_{l,x_{\mathsf{inp}(l)}} \boldsymbol{M}_{l+1,x_{\mathsf{inp}(l+1)}} \ldots \right) \boldsymbol{M}_{\kappa+1}.$$

When $l \in S$, then this product no longer simplifies to $\boldsymbol{R}_{l,x_{\mathsf{inp}(l)}}$. There are now an odd number of matrices corresponding to the set S, and so this product is approximately equal to $\overline{\boldsymbol{M}} \boldsymbol{R}_{l,x_{\mathsf{inp}(l)}}$ or $\boldsymbol{R}_{l,x_{\mathsf{inp}(l)}} \overline{\boldsymbol{M}}$. This discrepancy allows the possibility of the distinguishing the two branching programs. In essence, by finding an annihilating polynomial Q that does not recover an element in $\langle g \rangle$ in the second case.

The attack in the weakened model. Assume that the adversary has knowledge of an annihilating polynomial for branching program evaluations consisting only of identity matrices (for example any input to $\mathbb{M}_0$), and denote this by Q^*. We should also state that this polynomial does not annihilate in the case where non-identity matrices are included in the product. Recall a valid annihilating polynomial for GGH, is one s.t. for $\{\rho_j \leftarrow \widetilde{\mathbb{M}}(x_j) \cdot p_{zt}\}_{j \in [M]}$; then $Q^*(\{\rho_j\}_{j \in [M]}) \in \langle g \rangle$. Let the set S considered above be s.t. $|S| = M \in \mathbb{N}$ — in [240] the attack works with $M = 3$. The attack against Construction VII;2.3 now proceeds as follows.

- Submit $(\mathbb{M}_0, \mathbb{M}_1)$ to the challenger in the weakened graded encoding model.
- Receive $\widetilde{\mathbb{M}}_c \leftarrow \mathsf{io}(\mathbb{M}_c, \mathsf{ges}, \mathcal{T})$, for $\mathcal{T} \leftarrow \mathsf{partition}(\mathcal{U}, \kappa, 1)$ and $c \leftarrow_{\$} \{0, 1\}$.
- For $j \in [tM]$, where $t = \mathsf{poly}(\lambda)$ and $M \in \mathbb{Z}$: let $x^j \in \ell_{\mathsf{in}}$ be s.t. $(l, x^j_{\mathsf{inp}(l)}) \in S \times \{0\}$ for all $l \in [\kappa]$, and compute $\rho_j \leftarrow \widetilde{\mathbb{M}}(x^j) \cdot p_{zt}$ using oracle access to the GES operations in Definition VII;2.4.
- For $i \in [t]$: evaluate $\delta_i \leftarrow Q^*(\{\rho_{i \cdot \iota}\}_{\iota \in [M]})$.
- Heuristically create a basis Δ_g of the ideal $\langle g \rangle$ using $\{\delta_i\}_{i \in [t]}$. This step is possible, if t is chosen large enough, since each δ_i consists of monomials that all contain a factor of g (by the choice of Q^*).
- Let $\{x^{\dagger,j}\}_{j \in [M]}$ be inputs s.t. $\exists\, (l, x^{\dagger,j}_{\mathsf{inp}(l)}) \in S \times \{1\}$. Let $\{\rho^{\dagger,j} \leftarrow \widetilde{\mathbb{M}}(x^{\dagger,j}) \cdot p_{zt}\}_{j \in [M]}$.
- Compute $\delta^\dagger \leftarrow Q^*(\{\rho^{\dagger,j}\})$.

- Finally test if $\delta^\dagger \overset{?}{\in} \langle g \rangle$ using Δ_g. If so: output $b = 0$, else: output $b = 1$ (where $\mathbb{M}_b$ is obfuscated).

It was shown by [240] (for a slightly less general case) that this attack is polynomial-time using a concrete choice of Q^* — allowing for the heuristic step. The works of [15, 87] both give similar attacks that equally use the heuristic step and implicit knowledge of Q^*.

Expanding to dual-input case. The choice of Q^* does not extend to the dual-input case, but [240] shows that finding a new valid choice is computationally feasible. Thus, there exists a choice of Q^* that can be computed by a polynomial-time circuit in the dual-input case, and the attack proceeds as before.

VII;4.3 Adapting the attacks to IND-OBF

Our first contribution is to show that the attack above applies in the IND-OBF security model (Definition VII;3.2). This merely shows us that our model is sufficient for capturing the weaknesses of GGH, but also justifies using this model in our later analysis. Our analysis shows that any adversary that finds a satisfying annihilating polynomial Q^* in the weakened model, also finds a satisfying polynomial in IND-OBF.

Lemma VII;4.1 [Sufficiency of IND-OBF]

Let $\mathcal{A}$ be an adversary that carries out the attack in Section VII;4.2 on the obfuscator io (Construction VII;2.3) instantiated under $\mathsf{ges} = \textsc{ggh}$ with non-negligible success probability. Then there is a PPT adversary $\mathcal{A}'$ that succeeds in $\mathsf{INDOBF}^{\mathsf{ges},\mathsf{io}}_{\kappa,\eta}$ with the same probability.

Proof. Notice that the attack above only evaluates inputs on the branching programs, and doesn't need access to the explicit matrices. In addition, the differences in the γ^1 polynomials between the two programs can be inferred by the original choice of $\mathbb{M}_0$ and $\mathbb{M}_1$.

Therefore, we can translate $\mathcal{A}$ in the attack above into a winning adversary $\mathcal{A}'$ in IND-OBF by specifying each input x^j as an input query to be used by $\mathcal{A}'$. Then $\mathcal{A}'$ returns the zero-tested results of the queries to $\mathcal{A}$, who then computes the basis Δ_g using the annihilating polynomial Q^* that it uses in the weakened model. Finally, $\mathcal{A}$ submits the input queries $x^{\dagger,j}$ to $\mathcal{A}'$, and receives more zero-tested results. It can then check offline that $\delta^\dagger \overset{?}{\in} \langle g \rangle$ and returns 0 to $\mathcal{A}'$ if "yes" and 1 to $\mathcal{A}'$ if "no". Then $\mathcal{A}'$ wins $\mathsf{exp}^{\mathsf{indobf}}_{b,\mathcal{A}'}(1^\lambda, 1^\kappa, 1, \mathsf{ges} = \textsc{ggh}, \mathsf{io})$ with the same probability. $\square$

We can then prove equivalence by noting that, in the weakened MMAP model, IND-OBF is at least as secure as the model itself. We do not state a specific lemma as the reduction is obvious since the adversary in the model can just play the role of the challenger in IND-OBF. Coupling this with Lemma VII;4.1 shows that the attack strategy in the weakened model is viable in IND-OBF, and thus IND-OBF is sufficient for considering annihilation attacks.

VII;5 GGH WITHOUT IDEALS

It is necessary to recall the original motivation of the chapter.

Can we introduce a change to GGH *to prevent annihilation attacks on BGK-type obfuscators?*

Approaching this question from the direction of modifying the IO candidate has given way to new candidates that are secure when instantiated using GGH [132, 153] (in the weakened security model). However, recent attacks have shown that inherent vulnerabilities with the GGH scheme mean that proof in this model are essentially invalid [90, 256]. Therefore, there is some utility in, instead, attempting to modify the structure of the GGH GES in order to prevent these attacks. This was attempted by [154], but this work was eventually subsumed into [153].

We will also revisit the hypothesis of Halevi [176].

"The core computational hardness problem for breaking GGH*, is to find a representative of the algebraic ideal $\langle g \rangle$ during computation."*

In this section we explore the possibility of either removing the presence of such algebraic ideals from GGH, or rendering the scheme such that finding any ideals is computationally difficult. We hope to do this without making the new encodings unrecognisable, in relation to the original GGH design.

Unfortunately, the presence of g in each encoding is critical to the zero-testing procedure, which is in turn critical to the GES functionality that is required. Therefore, when replacing g, it is also necessary to replace the zero-testing mechanism with a different design that makes different checks.

VII;5.1 Overview of GGH without ideals

One may think that it would be possible to change GGH so that different g's are used in every encoding or, less extremely, that a new g is used for each level. The resulting encodings would take the form

$$\frac{a_j + r_j g_{j'}}{z_i}$$

where either $j' = j$ in the former, or $j' = i$ in the latter; and $(i, j, j') \in [\kappa] \times [m] \times [m']$ for $m' = m$ or $m' = \kappa$ respectively. Unfortunately, when it comes to zero-testing, either change will result in incorrect evaluations. The natural way of adapting the zero-testing parameter p_{zt} is to let

$$p_{zt} = \frac{\prod_{i=1}^{\kappa} z_i \cdot h}{\prod_{j'=1}^{m'} g_{j'}};$$

but if $j' = j$, this requires specifying a concrete bound for m (either κ or m') and sampling each g'_j for $j \in [m]$. Moreover, in both cases, this adaptation does not maintain correctness.

For example, the polynomials $\gamma^{\iota}(\boldsymbol{a}, \boldsymbol{r})$ ($\iota \in [\kappa]$) will now be also defined over $\boldsymbol{g} = (g_{j'})_{j'}$. Consequently, each polynomial will contain a strict subset of the variables taken from $\boldsymbol{g}$, and thus when multiplying each with $\prod_{j'=1}^{m'} g_{j'}$, some inverted group elements will be multiplied with each monomial. Zero-testing correctness no longer holds since the inverse of these elements is uniform in $\mathcal{R}_q$, whp.

Modified encodings and operations. As a result, we need a different strategy when sampling the elements $g_{j'}$. Roughly speaking, our solution modifies the sampling of g_j so that they are, *very roughly*, sampled uniformly from a $(0, O(\sqrt[\kappa]{q}))$-bounded distribution and inverted prior to multiplication with r_j.[15] Let $\beta_{j'} = g_{j'}^{-1}$, then our encodings are are to be defined as:

$$\frac{a_j + r_j/\beta_{j'}}{z_i} \mod q.$$

To avoid the issue of having to bound m' from above, we set $j' = i$, $m' = \kappa$ and define β_i with respect to each level set. Operations over encodings are carried out in the same way.

We should further note that the plaintext space is no longer $\mathcal{R}_g$. In fact, we have no real restrictions on the plaintext space beyond the restriction that encodings are defined in $\mathcal{R}_q$. For the purpose of our analysis, we will assume that all top-level encoded values are subject to the same bounds as those implicitly set by χ_σ. Therefore, we can only encode small elements in $\mathcal{R}$, which is the same as in GGH.

[15] The parameter choices are actually far more nuanced and $q = \Omega(\prod_{j=1}^{\kappa} g_j)$.

Zero-testing. We modify the zero-testing parameter so that it is now composed of the product $p_{zt} = \prod_{i=1}^{\kappa} \beta_i z_i$. Then for a top-level encoding $v = (a + r/\beta_{\mathcal{U}})/z_{\mathcal{U}}$ where $\mathcal{U} = [\kappa]$, then

$$\rho = v \cdot p_{zt} = a\beta_{\mathcal{U}} + r$$

is the result of zero-testing. Our procedure now uses the fact that β_T is no longer small and thus $\|\rho\|_\infty$ is close to q when $a \neq 0$, and is roughly $O(\sqrt[\kappa+1]{q})$ smaller when $a = 0$. For a more general encoding, we now have that the polynomials γ^i are multiplied by $\kappa - i$ variables from $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_\kappa)$. Therefore, an encoding of zero has monomials with maximum degree $\kappa - 1$ and otherwise also includes a monomial with maximum degree κ. This fact is used to distinguish the values of encodings.[16]

We omit the use of the parameter h since the attacks that it seeks to prevent *appear* no longer applicable to our scheme.

Resistance of encodings to annihilation attacks. It should be clear that we immediately prevent unmodified versions of the attacks of [15, 87, 240]. We do not explicitly consider lattice subfield attacks, since these can be avoided by the particular choice of parameters. In the case of IO, the most potent attacks of [90] can be avoided by taking $\lambda = \mathsf{poly}(\kappa)$. The attacks of [256] appeared subsequent to the publication of the material considered in this chapter and so this has not been considered in our analysis. It's plausible that the quantum attacks shown there would also apply in our situation, though these would also need some modification.

Lattice-based cryptanalysis. Our primary focus is on attempting to generate a GGH-like algebraic structure that allows for the functionality of a GES, while preventing annihilation attacks. As we show later, we demonstrate that the GGH-like structure is inherently flawed in that even the above modifications do not prevent non-trivial variants of the annihilation attacks. Consequently, we do not perform a full cryptanalysis of our scheme wrt state-of-the-art lattice cryptanalysis. Clearly, if we could successfully describe such an algebraic structure that avoided these attacks, this analysis would become necessary.

VII;5.2 Actual scheme

Recall that $z \leftarrow_{\$} \mathcal{R}(\chi^n)$, for some distribution χ over $\mathbb{Z}$, indicates sampling n coefficients from χ and assigning each one to be a coefficient of the polynomial $z \in \mathcal{R}$. Sampling can also be done via the canonical embeddings approach.

Let $\mathcal{R}$, be the ring that we described for GGH. Let χ_σ be the $D_{\mathbb{Z},\sigma}$ distribution, centred at 0. Let μ_β be a parameter and let F_μ be a distribution sampling uniformly from the interval $[0, \mu_\beta]$.

[16]This can only be achieved by taking very different parameter sets to those originally used in GGH.

Finally, let B be the bound s.t. all samples from χ_σ^n are bounded above by B. Lemma II;2.2 implies that $B = \sigma\sqrt{n}$. Then the plaintext space of our scheme is set to be $\mathcal{X} = \mathcal{R}_B$.

We describe our construction below.

Construction VII;5.1 [GGH without ideals (GGHWOI)]

Let $\mathcal{R}, n, \sigma, \chi_\sigma, \mu_\beta, F_\mu, B, \mathcal{X}$ be described as above.

- GGHWOI.$\mathsf{Setup}(1^\lambda, 1^\kappa)$: Sample parameters μ_0, q and let $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$. Sample $\{z_i\}_{i\in[\kappa]} \leftarrow_\$ \mathcal{R}_q$, and $\{\beta_i\}_{i\in[\kappa]} \leftarrow_\$ \mathcal{R}(F_\mu)$. Let $p_{zt} = \prod_{i=1}^{\kappa} z_i\beta_i$. Output $\mathsf{pp} = (p_{zt}, \mathcal{R}, q, n, \kappa, \sigma, \mu_0)$ and $\mathsf{msk} = (\mathsf{pp}, \{\beta_i, z_i\}_{i\in[\kappa]})$.
- GGHWOI.$\mathsf{Encode}(\mathsf{msk}, a, T \subseteq [\kappa])$: Sample $r \leftarrow_\$ \chi_\sigma$ as a 'small' element, let
$$[a|T] = \frac{a + r/\prod_{i\in T}\beta_i}{\prod_{i\in T} z_i} \mod q,$$
and output $v = [a|T]$.
- GGHWOI.$\mathsf{Add}(\mathsf{pp}, [a|T], [b|T])$: Output $[a|T] + [b|T] \mod q$.
- GGHWOI.$\mathsf{Mult}(\mathsf{pp}, [a|T_1], [b|T_2])$: Output $[a|T_1] \cdot [a|T_2] \mod q$.
- GGHWOI.$\mathsf{ZeroTest}(\mathsf{pp}, [a|T])$: Let $\rho = p_{zt} \cdot [a|T] \mod q$ and output 1 if $\|\rho\|_\infty < \mu_0$, and 0 otherwise.

SYMMETRIC ENCODINGS. We can define a correspondingly symmetric variant of the scheme, by only sampling a single β and z. Then let $\mathcal{U} = \kappa$ and let index sets be defined as some integer $1 \le T \le \kappa$. For any such T we sample an encoding by computing $\beta_T = \beta^T$ and $z_T = z^T$.

PARAMETER SETTINGS. Before we go on to describe correctness, we should first elucidate the parameter settings that we make. This is required to ensure that we can pick the correct bounds for zero-testing. We establish the correctness of our scheme (based on these settings) in Lemma VII;5.1.

- $n = \kappa\lambda^2$;
- $\sigma = \lambda\sqrt{n}$;
- $B = \sigma\sqrt{n}$;
- $\mu_0 = 2^\kappa \kappa B^\kappa n^{2\kappa-3} B_\mu^{\kappa-1}$;
- $q = 4\mu_0$;

- $\mu_\beta = (7 \cdot (2B)^\kappa n^{\kappa-2})$

The parameter μ_0 is used as an implicit bound for top-level encodings of zero.

As we mentioned above, we do not consider lattice-based cryptanalysis of these parameters in this chapter. We only make this choice so that zero-testing is provably correct. Our main aim is to prevent the annihilation attacks that break GGH via construction of predictable algebraic relationships.

Lemma VII;5.1 [Correctness]

Construction VII;5.1 is correct, wrt to the parameter choices made above.

Proof. Firstly, let us consider the magnitude of some top-level encoding v created from lower-level encodings $v_1, \ldots, v_m$ corresponding to values $\boldsymbol{a} = (a_1, \ldots, a_m)$ and unique randomness $\boldsymbol{r} = (r_1, \ldots, r_m)$. By the structure of our encodings, we know that after zero-testing ($\rho = v \cdot p_{zt}$) we have:

$$\rho = P(\boldsymbol{a})\beta^{(\kappa)} + \gamma^1(\boldsymbol{a}, \boldsymbol{r})\beta^{(\kappa-1)} + \ldots + \gamma^{\kappa-1}(\boldsymbol{a}, \boldsymbol{r})\beta^{(1)} + \gamma^\kappa(\boldsymbol{r});$$

where $\beta^{(i)}$ is a sum of the $\binom{\kappa}{i}$ monomials of degree i when considering $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_\kappa)$ as formal variables. Moreover, the polynomial evaluation $P(\boldsymbol{a})$ corresponds to the value of the final encoding v.

Our first aim is to characterise the bounds on the magnitudes of top-level encodings of zero, this will help us to establish the value of μ_β. Our analysis of these bounds is not tight but allows us to demonstrate correctness. A top-level encoding of zero (after zero-testing) takes the form:

$$\rho_0 = \gamma^1(\boldsymbol{a}, \boldsymbol{r})\beta^{(\kappa-1)} + \ldots + \gamma^{\kappa-1}(\boldsymbol{a}, \boldsymbol{r})\beta^{(1)} + \gamma^\kappa(\boldsymbol{r}) \ \texttt{mod} \ q.$$

Note that every ring element in the entries of $\boldsymbol{a}$ and $\boldsymbol{r}$ has infinity norm bounded by B_μ. Moreover, each monomial in the polynomial γ^i (for all $i \in [\kappa]$) has combined degree κ in variables taken from $\boldsymbol{a}$ and $\boldsymbol{r}$. Also, γ^i has $\binom{\kappa}{i}$ such monomials. Therefore:

$$\begin{aligned} \|\rho_0\|_\infty &< n^{\kappa-1}B^\kappa \sum_{i=1}^{\kappa} \binom{\kappa}{i} \beta^{(\kappa-i)} < n^{2\kappa-3}B^\kappa B_\mu^{\kappa-1} \sum_{i=1}^{\kappa} \binom{\kappa}{i} \\ &< \kappa n^{2\kappa-3}B^\kappa B_\mu^{\kappa-1} \frac{2^\kappa}{\sqrt{\pi\kappa/2}} < \mu_0 = q/4; \end{aligned}$$

where the first bound is acquired via Corollary II;2.1. The second bound is obtained by taking the upper bound of each β variable, given by B. The third bound is an asymptotic bound derived from Stirling's approximation, when κ is sufficiently large.

Now, for encodings of non-zero, we essentially add $n^{\kappa-1}\|\beta^{\kappa}\|_\infty$ to $\|\rho_0\|_\infty$ (by Corollary II;2.1). Let $v_a = P(\boldsymbol{a})\beta^{\kappa} + \gamma^1(\boldsymbol{a},\boldsymbol{r})\beta^{(\kappa-1)} + \ldots \mod q$. Then, we have two requirements: (1) $\|v_a\|_\infty > q/4 \mod q$; and (2) $\|\beta^{\kappa}\|_\infty > q$. The first requirement guarantees that the correctness of zero-testing is maintained. The second ensures that modular reduction occurs for encodings of non-zero, for security. We can guarantee both requirements by simply ensuring that

$$7q/4 > n^{\kappa-1}\|\beta^{\kappa}\|_\infty > 5q/4. \tag{VII;11}$$

Considering $n^{\kappa-1}\beta^{\kappa}$ specifically, we need $n^{\kappa-1}\|\beta^{\kappa}\|_\infty > 5\mu_0$. Note that we must then satisfy:

$$\begin{aligned} n^{\kappa-1}\|\beta^{\kappa}\|_\infty &> 5\kappa n^{2\kappa-3}B^{\kappa}B_\mu^{\kappa-1}2^{\kappa};\\ \|\beta^{\kappa}\|_\infty &> 5\kappa n^{\kappa-2}B^{\kappa}B_\mu^{\kappa-1}2^{\kappa}; \end{aligned}$$

where we ignore the denominator $\sqrt{\pi\kappa/2}$. Taking $\beta^{\kappa-1}$ to be roughly equal to $B_\mu^{\kappa-1}$, then we finally have that:

$$\|\beta\|_\infty > 5\kappa n^{\kappa-2}B^{\kappa}2^{\kappa}.$$

Recall that we sample β uniformly from a distribution bounded above by $7\kappa n^{\kappa-2}B^{\kappa}2^{\kappa}$. Then

$$\Pr\left[\|\beta\|_\infty < 5\kappa n^{\kappa-2}B^{\kappa}2^{\kappa} \middle| \beta \leftarrow_{\$} F_\mu\right] = (5/7)^n;$$

and since $n = \mathsf{poly}(\kappa,\lambda)$, then this is negligible in the security parameter. Thus the lower bound of Equation (VII;11) is satisfied with overwhelming probability. The upper bound is satisfied trivially since the distribution only samples ring elements with coefficients smaller than this, whp. Therefore, $q > \|v_a\|_\infty > q/4 \mod q$ and thus the scheme is correct.

This covers the last two probabilities in the definition of correctness (Definition VII;2.2), the first follows from the fact that fresh encodings of top-level zeroes are inherently smaller than those constructed from lower-level encodings. Thus the zero-tested element is subject to the same bounds as above and the proof of correctness follows. $\square$

Clarification of correctness bounds from published work. The work that we base this chapter on gave much more approximate bounds [8, 9] — thus only a rough approximation of correctness was originally given in these works. The bounds that we describe here are markedly different due to the fact that we concretely set out a set of parameters that correctness holds for.

VII;5.3 Zeroizing attacks

We will first consider the applicability of our GES to general situations, i.e. in an attempt to prove the hardness of the κ-MDDH problem. Unfortunately, in these situations we require the presence of low-level encodings of zero and we note that trivial attacks against our scheme are possible in this scenario.

In particular, we allow there to be a set S_0 of indices such that $a_j \in \boldsymbol{a}$ and $j \in S_0$ indicates that $a_j = 0 \in \mathcal{R}_q$. Consider the magnitude of an encoding of zero $v_{i,\iota}$ at level 2 after a multiplication has occurred. Such an encoding takes the form:

$$v_{i,\iota} = \frac{a_i \beta_i r_\iota + a_\iota \beta_\iota r_i + r_i r_\iota}{\beta_i \beta_\iota}; \tag{VII;12}$$

and then consider the cases where $\iota \in S_0$ and $i \notin S_0$ wlog, and $i, \iota \in S_0$. In these two cases we actually have:

$$v_{i,\iota} = \frac{a_i \beta_i r_\iota + r_i r_\iota}{\beta_i \beta_\iota}; \tag{VII;13}$$

and

$$v'_{i,\iota} = \frac{r_i r_\iota}{\beta_i \beta_\iota}. \tag{VII;14}$$

respectively. Clearly, the choice of $\beta_i \leftarrow_{\$} \mathcal{R}(F_\mu)$ means that the difference in $\|v_{i,\iota}\|_\infty$ and $\|v'_{i,\iota}\|_\infty$ is noticeable. Therefore, the presence of lower-level zeroes allows us to dictate the magnitude of zero encodings obtained afterwards. We don't give a concrete attack, but it is not pessimistic to assume that the hardness of κ-MDDH can be compromised given this flaw.

VII;5.4 Annihilation attacks

We now consider the case of launching annihilation attacks against GGHWOI where no low-level encodings of zero are present. In fact, our encoding scheme trivially prevents the attacks since there are no efficiently realisable ideals.

Firstly, there is no formal variable taken from the sets $\boldsymbol{\beta}$, $\boldsymbol{a}$ and $\boldsymbol{r}$ that is contained in each monomial of the polynomial ρ_0. Therefore, finding an ideal would essentially amount to choosing one such variable and annihilating each monomial that didn't contain this variable. In our encodings, the number of polynomials is polynomial in κ and they are all of degree κ in unknown variables. It's highly likely that finding such a polynomial is computationally difficult [202].

The result is that, unmodified, the previous known attacks do not apply to our scheme. If we can provably achieve hardness against all known attacks, then we argue that a structural change in GGH encodings could result in a concrete security guarantee. On the other hand, adapting these attacks to GGHWOI would indicate that there is a structural vulnerability in the way that GGH-like encodings are created.

VII;6 Algebraic attacks on IO over GGHWOI

We respond to the final paragraphs of the previous section by confirming that there are structural flaws in the way that GGH-like encodings are formed, after computing IO obfuscated branching programs as in Construction VII;2.3. While the result itself is unsurprising — in the sense that there is already a wealth of cryptanalysis against GGH— the result implies that GGH is insecure for reasons that differ from current hypotheses. Our main finding is that the conjecture of Halevi does not hold: that the presence of the ideal $\langle g \rangle$ is *not* actually principal to the attacks against GGH. Our work suggests that the attacks are a manifestation of greater insecurities, that are still present even after making natural alterations to the scheme to remove $\langle g \rangle$ from the security analysis.

Essentially, our adapted annihilation attack considers the same polynomial γ^1 that is considered in the attacks of [240]. That is, let ρ_0 be a zero-tested, encoding of zero obtained from an obfuscated branching program. Then:

$$\rho_0 = \boxed{\gamma^1(\boldsymbol{a}, \boldsymbol{r})\beta^{(\kappa+1)}} + \ldots + \gamma^{\kappa+1}(\boldsymbol{a}, \boldsymbol{r})\beta^{(1)} + \gamma^{\kappa+2}(\boldsymbol{r}) \ \texttt{mod}\ q; \qquad \text{(VII;15)}$$

and we attempt to annihilate the polynomial γ^1 again. However, we cannot simply leverage the attack of [240] since $\gamma^1(\boldsymbol{a}, \boldsymbol{r})\beta^{(\kappa-1)}$ now consists of κ monomials of degree $\kappa + 1$ in variables taken from $\boldsymbol{\beta}$. Moreover, the heuristic step that was used before is now redundant due to the lack of a principal ideal to target.

We demonstrate implicit annihilating polynomials in the security models from Sections VII;3.1 and VII;3.2 to illustrate the effectiveness of our attacks.

Remark VII;6.1. *Our attacks work in the situation where the level sets are defined symmetrically, rather than asymmetrically. That is, $\mathcal{U} = \kappa + 2$, and $T_0 = T_{\kappa+1} = T_{l,b} = 1$. This is not exactly as defined in the level sets of [12, 23, 27, 239, 253], but aligns with the design of [66]. The former works use more granular level sets to implement straddling sets partitions over $\mathcal{U}$, to restrict 'mixed-input' attacks against their construction. We should emphasise that we do not use mixed-input attack techniques, but the structure of our encodings means that constructing attacks in this setting appears difficult. Intuitively this make sense, since γ^1 is no longer linear in unknown variables. Unfortunately, we have not been able to translate this difficulty into a meaningful security notion for asymmetric versions of our encodings. Therefore, we should emphasise this should not be taken as a security guarantee of any kind.*

VII;6.1 Attack in IND-BP

We provide a warm-up to the main contribution given in Section VII;6.2. This warm-up helps to give a gentler description to the attacks that we eventually use in the stronger IND-OBF security model. It is unsurprising that attacks exist in the IND-BP model; since it does not capture the extra randomisation procedures that are necessary for securing IO candidates. Although, the structure of our attacks in both models is very similar.

Theorem VII;6.2 [Attack in IND-BP]

Let $\mathsf{ges} = \textsc{gghwoi}$ in the symmetric setting, where $\mathcal{T} = \{T_0, T_1, \ldots, T_{\kappa+1}\}$ where $T_l = 1$, for each $l \in [0, \ldots, \kappa+1]$ and $\mathcal{U} = \kappa + 2$. Then we describe an algorithm $\mathcal{A}$, running in time $\mathsf{poly}(\lambda, \kappa)$ whp, s.t.:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{ges}(1^\lambda, \mathsf{indbp}))) \approx 1.$$

Proof. We split the proof of Theorem VII;6.2 into two claims; handling the single-input and dual-input cases of the attack. Firstly, to avoid confusion later on, we will explicitly consider the various knowns and unknowns available to the adversary $\mathcal{A}$ in $\mathsf{exp}^{\mathsf{indbp}}_{b,\mathcal{A}}(1^\lambda, 1^\kappa, 1^\eta, \mathsf{ges})$. Let $\boldsymbol{a} \in \mathcal{R}^{50\kappa+10}$ be the vector of encoded values taken from the entries of the original branching program; let $\boldsymbol{r} \in \mathcal{R}^{50\kappa+10}$ be the vector of unique small random elements that are used in each encoding; and let $\boldsymbol{\beta} \in \mathcal{R}^{\kappa}$ be the vector of level-specific sampled elements. The lengths of the first two vectors is decided due to using $w = 5$ branching programs, and since there are $2(\kappa+2)$ matrices and 2 bookend vectors.

Since the adversary chooses $\mathbb{M}_0, \mathbb{M}_1$, then it knows the vector $\boldsymbol{a}$ since each entry is encoded directly without additional randomness. It does not know the values of $\boldsymbol{r}$ or $\boldsymbol{\beta}$.

Claim VII;6.2.1. *The attack succeeds with probability close to* 1 *for* $\eta = 1$.

Proof. Let us consider obfuscated branching programs as described in the abstract model of Construction VII;2.3. Our attack technique is slightly more general in that we can consider generic pairs of branching programs, and thus we can mount our attacks immediately against branching programs that are derived from Barrington's theorem; as in [15].[17] Similarly to Equation (VII;10), we have that:

$$\boldsymbol{\beta}^{(\kappa+1)}\gamma^1 = \prod_{i=0}^{\kappa+1} \alpha_{i,b}\boldsymbol{M}_0\left(\sum_{l=1}^{\kappa} \boldsymbol{\beta}[l]\ldots \boldsymbol{M}_{l-1,x_{\mathsf{inp}(l-1)}}\boldsymbol{R}_{l,x_{\mathsf{inp}(l)}}\boldsymbol{M}_{l+1,x_{\mathsf{inp}(l+1)}}\cdots\right)\boldsymbol{M}_{\kappa+1};$$

[17] Although, we only consider attacks against similar choices of branching program to those made in [240] for simplicity.

where the only difference is the inclusion of the product $\boldsymbol{\beta}[l] = \prod_{i \neq l} \beta_i$, where $\beta[0] = 1$. It is important to understand that, using symmetrically defined encodings, we no longer use the more general indexing for sampling $\beta_{l,b}$ for each pair (l, b). That is, we only use one β s.t. $\beta_l = \beta$ for all $l \in [0, \dots, \kappa + 1]$ by utilising a symmetric variant of GGHWOI. Therefore, the products $\boldsymbol{\beta}[l]$ are equivalent for each l and we can rewrite γ^1 as:

$$\boldsymbol{\beta}^{(\kappa+1)}\gamma^1 = \beta^{\kappa+1} \prod_{i=0}^{\kappa+1} \alpha_{i,b} \boldsymbol{M}_0 \left(\sum_{l=1}^{\kappa} \dots \boldsymbol{M}_{l-1,x_{\mathsf{inp}(l-1)}} \boldsymbol{R}_{l,x_{\mathsf{inp}(l)}} \boldsymbol{M}_{l+1,x_{\mathsf{inp}(l+1)}} \cdots \right) \boldsymbol{M}_{\kappa+1}.$$

Recall from Section VII;4 that γ^1 is linear in the variables of $\boldsymbol{r}$. Recall, additionally, that $\mathsf{inp}(l) = l \mod \ell_{\mathsf{in}}$. The adversary $\mathcal{A}$ effectively has knowledge of the coefficients used in the monomials of γ^1 depending on the choice $b \in \{0, 1\}$ in $\mathsf{exp}^{\mathsf{indbp}}_{b,\mathcal{A}}(1^\lambda, 1^\kappa, 1, \mathsf{ges})$ (since they are only taken from $\boldsymbol{a}$). We now consider the game when $b = 0$ wlog.

Consider L inputs $(x_1, \dots, x_L)$ resulting in outputs $(\rho_1, \dots, \rho_L)$ that are received back from the challenge oracle in $\mathsf{INDBP}^{\mathsf{ges}}_{\kappa,\eta}$. Let $r^{l,b}_{i,j}$ be the $(i, j)^{\text{th}}$ entry of $\boldsymbol{R}_{l,b}$. Let $\boldsymbol{\nu}_t$ be the vector of dimension $50\kappa + 10$ containing the (polynomial) coefficients each $r^{l,b}_{i,j}$ in γ^1_t for some $t \in [L]$. We assume for this analysis that $x_1, \dots, x_n$ are chosen s.t. $\{\boldsymbol{\nu}^t\}_{t \in [L]}$ are an algebraically independent set of maximal size, thus appending another vector to the list creates an algebraically dependent set by Lemma VII;2.4. This is possible, since there are $50\kappa + 10$ possible variables in the branching program, and so we have a maximum size of $L = 50\kappa + 10$ algebraically independent basis vectors. Moreover, by the Schwarz-Zippel lemma (Lemma II;2.1), choosing a random set of inputs of size equal to the algebraic rank is likely to give an algebraically independent set, whp.

Now, choose some new input $x_\dagger$. Then clearly the coefficient vector $\boldsymbol{\nu}_\dagger$ corresponding to the zero-tested output $\rho_\dagger$ has to be algebraically dependent on $\boldsymbol{\nu}_1, \dots, \boldsymbol{\nu}_L$, again by Lemma VII;2.4. Therefore, using a form of "Gaussian elimination"[18] it is possible to compute a polynomial $Q^\dagger$, with 'coefficients' taken from $\boldsymbol{a}$ that create a new vector $\boldsymbol{\nu}_0$ where each entry is 0. In other words, $Q^\dagger$ is an annihilating polynomial. In fact, by Lemma VII;2.4, we know that $Q^\dagger$ exists because $L > 50\kappa + 10$.

Applying $Q^\dagger$ to the outputs $\rho_\dagger, \rho_1, \dots, \rho_L$ results in an output ρ'_0, where all the monomials in the sum of $\gamma^1_\dagger$ are eliminated. That is, the term $\gamma^1_\dagger \beta^{(\kappa+1)}$ is completely removed from the output ρ'_0. Then we can write ρ'_0 as:

$$\rho'_0 = Q^\dagger(\rho_0) \mod q; \tag{VII;16}$$

which is similar to ρ_0 in Equation (VII;15), except that the γ^1_0 term is removed. Thus ρ'_0 has a magnitude that is noticeably smaller in comparison in $\mathcal{R}$. In fact, if we use $\mathbb{M}_0, \mathbb{M}_1$ specified as in [240], then we can also use the same definition of the annihilating polynomial $Q^\dagger = Q^*$, since we are annihilating the γ^1 polynomial which takes the exact same structure. On the other hand, $Q^\dagger$ definitely exists due to the algebraic dependency over $\boldsymbol{\nu}_1, \dots, \boldsymbol{\nu}_L, \boldsymbol{\nu}_\dagger$.

[18]Obviously, standard Gaussian elimination does not work since the coefficients are polynomials taken from $\mathcal{R}$.

A final consideration is that we require that $Q^\dagger$ has small coefficients. In previous attacks, smallness is not important as they are only searching for elements that are multiples of g. Whereas our attack works by distinguishing annihilated polynomials based on differences in absolute magnitude. Fortunately, if $Q^\dagger = Q^*$ from [240], then it will definitely have small coefficients, since Q^* is chosen to have coefficients taken from $\boldsymbol{a}$, and thus the result remains of a similar magnitude (i.e. small). This is enough for the attack to work since we only have to demonstrate the attack for one pair of branching programs.

In the more general case where $Q^\dagger \neq Q^*$, then $Q^\dagger$ requires coefficients that are of a similar order to the magnitude of the encoded values. These ring elements are bounded by B, and thus comparatively smaller than each variable β. Therefore we will have that ρ_0' is noticeably smaller in the case of $\mathbb{M}_0$, than in the case of $\mathbb{M}_1$.

Indeed, to summarise the attack on general choices of b, the coefficients of $Q^\dagger$ that work in the case of $\mathbb{M}_0$ do not work in the case of $\mathbb{M}_1$ due to the different choice of self-inverse matrices. That is, when $\mathbb{M}_1$ is obfuscated, then γ_0^1 still exists in Equation (VII;16). Therefore, the adversary $\mathcal{A}$ in the experiment $\mathsf{exp}^{\mathsf{indbp}}_{b,\mathcal{A}}(1^\lambda, 1^\kappa, 1, \mathsf{ges})$ can fashion the attack against outputs assuming they are from $\mathbb{M}_0$. If the attack succeeds, it guesses $b = 0$ and otherwise $b = 1$.

The only thing left to check is that the attack runs in polynomial-time. This is true if the number of variables that are to be eliminated is $\mathsf{poly}(\lambda)$ and if the polynomials γ_t^1 are linear in unknowns. Fortunately, the number of such variables $r_{i,j}^{l,b}$ is $50\kappa + 10 = \mathsf{poly}(\lambda)$ and the latter is clearly true. Thus the attack is complete for $\eta = 1$, and succeeds with probability close to 1. □

Claim VII;6.2.2. *The attack succeeds with probability close to* 1 *for* $\eta = 2$.

Proof. For the dual-input case. We make the same observation as [240]: that the number of inputs needed to generate a linear dependence is still polynomial in κ. In fact, the number of variables just increases to $100\kappa + 10$, therefore we can find a viable $Q^\dagger$ in the same way as before, just using circa $2n$ inputs, rather than n.

Using the fact that an annihilating polynomial is shown to exist in this situation by [240], we can conclude the proof of this claim. □

The proof of Theorem VII;6.2 is concluded by the proof of Claims VII;6.2.1 and VII;6.2.2. □

Finally, we show that the attack strategy in the previous theorem, does not work in the case of the asymmetric description of GGHWOI.

Claim VII;6.2.3. *The simplified attack appears to no longer succeed when*

$$\mathcal{T} = \{T_0, T_{\kappa+1}, \{T_{l,b}\}_{l\in[\kappa],b\in\{0,1\}}\} \leftarrow \mathsf{partition}(\mathcal{U}, \kappa, \eta)$$

is used as a straddling sets partition of $\mathcal{U} = [\kappa]$, where $T_{l,b} \neq T_{l,1-b}$.

Proof. The term proof here is necessarily loose. We simply show that the attack as taken above, does not immediately result in the same output. Essentially, the γ^1 polynomial cannot be removed using the same "Gaussian elimination" strategy that we used before.

Considering more granular index sets that rely on asymmetric graded encoding schemes. For example, for a top-level set $\mathcal{U} = [0, \ldots, \kappa + 1]$, we have $\mathcal{T} \leftarrow \mathsf{partition}(\mathcal{U}, \kappa, 1)$. Therefore, we sample $\{\beta_0, \beta_{\kappa+1}\{\beta_{l,b}\}_{l\in[\kappa],b\in\{0,1\}}\}$ to allow for encoding wrt $\mathcal{T}$. Considering the polynomial γ^1 again:

$$\beta^{(\kappa+1)}\gamma^1 = \prod_{i=0}^{\kappa+1} \alpha_{i,b}\boldsymbol{M}_0\left(\sum_{l=1}^{\kappa} \boldsymbol{\beta}[l] \ldots \boldsymbol{M}_{l-1,x_{\mathsf{inp}(l-1)}}\boldsymbol{R}_{l,x_{\mathsf{inp}(l)}}\boldsymbol{M}_{l+1,x_{\mathsf{inp}(l+1)}} \cdots\right)\boldsymbol{M}_{\kappa+1};$$

the presence of $\boldsymbol{\beta}[l]$ means that we cannot explicitly use the same choice of Q^* that was made in Section VII;4. This is because, as well as being linear in each variable $r_{i,j}^{l,b}$ taken from the $(i,j)^{\text{th}}$ entry of the matrix $\boldsymbol{R}_{l,b}$, the monomials are degree $\kappa - 1$ in variables taken from $\boldsymbol{\beta}$. These variables are also chosen dependently on each input x to the branching program and are considered as another unknown for the adversary.

Thus the degree of the γ^1 polynomials is effectively increased to κ in the unknowns, and it was shown by [202] that finding annihilating polynomials over underlying cubic polynomials can be computationally difficult. Therefore, finding an annihilating polynomial over the γ^1 polynomials in this case also seems plausibly difficult. □

We remark that a possible attack strategy could be to use the techniques that we use in Theorem VII;6.3 later and similar to [256]. The goal would be to construct polynomials from the γ^1 polynomials, where each monomial contain the same factors from $\boldsymbol{\beta}$. Then we could apply the same attack strategy as before. For example, multiplying each γ^1 through by $(\prod_{i=0}^{\kappa+1} \beta_i) \cdot (\boldsymbol{\beta}[l])^{-1}$ would achieve this. Although, this seems difficult, without explicit knowledge of the product $(\prod_{i=0}^{\kappa+1} \beta_i)$, which is hidden in public zero-test parameter.

We conclude our analysis of the IND-BP case by remarking that we are unable make a security reduction from the structure of our zero-tested encodings $\rho_1, \ldots, \rho_L$ to the fact that annihilating in the asymmetric case seems difficult. It is hard to see how we can categorise all attack strategies into a unified description of an adversary, that can be thwarted by a falsifiable assumption.

We discuss later in Section VII;7 the possibility of making use of the branching program unannihilatability assumption [30, 153, 235] that is used in conjunction with the weakened graded encoding model, for proving the security of IO-like candidates. Although, we conclude that such an assumption relies inherently on annihilating generic branching program outputs directly, and so it doesn't seem to apply to our case.

VII;6.2 Attack in IND-OBF

We show how we can expand upon the attacks that we demonstrated in IND-BP, using similar techniques as those given in [15, 256].

Intuitively, the presence of the mixed-input scalars in io introduce an extra unknown to the equations that are dependent on the choice of input $x \in \{0,1\}^{\ell_{\mathsf{in}}}$. However, the product of $\alpha_{i,b}$ is multiplied to the entire polynomials ρ. Consequently, it is possible to introduce a method for computing new polynomials in $\mathcal{R}$ that demonstrate products of these scalars that are independent of the choice of x. As such, we are then able to proceed with the attack strategy of Theorem VII;6.2, albeit with a larger number of inputs that give top-level encodings of zero. Note that Kilian randomisation offers no extra security in this setting. Intuitively, this is because the randomisation is removed after the top-level of computation is reached.

Again, our attacks only work in the case where the level sets are defined symmetrically. This is unsurprising given that the attacks are derived from those in Theorem VII;6.2.

Theorem VII;6.3 [Attack in IND-OBF]

Let $\mathsf{ges} = $ GGHWOI, let io be an obfuscator following the structure of Construction VII;2.3 and let $\mathcal{T} = \{T_0, T_{\kappa+1}, T_l\}$ be s.t. where $T_0 = T_{\kappa+1} = T_l = 1$ and $\mathcal{U} = [\kappa+1] \cup \{0\}$. In other words, the GES is symmetric. Then we describe a algorithm $\mathcal{A}$, running in time $\mathsf{poly}(\lambda,\kappa)$ whp, s.t.:

$$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{ges}(1^\lambda, \mathsf{indobf}))) \approx 1$$

Proof. Again, we split the proof of this theorem into two separate claims, depending on the value of η that is considered. Also, the vectors $\boldsymbol{r}, \boldsymbol{\beta}$ are unknown and have the same dimensions. Importantly, the vector $\boldsymbol{a}$ is now strictly an unknown; since Kilian randomisation guarantees that the encoded values are randomised before encoding takes place. However, the coefficients of γ_t^1 in ρ_t can be treated as a known in $\mathsf{exp}^{\mathsf{indobf}}_{b,\mathcal{A}}(1^\lambda, 1^\kappa, 1^\eta, \mathsf{ges}, \mathsf{io})$ for $\mathcal{A}$ (depending on $b \in \{0,1\}$), as the randomisation has been removed at this point. This is enough for the attack to take place. We will finally note that the mixed input scalars $\{\alpha_{l,b'}\}_{l\in[\kappa],b'\in\{0,1\}}$ are unknown to the adversary, as well.

Claim VII;6.3.1. *The attack succeeds with probability close to* 1 *for* $\eta = 1$.

Proof. For some input x, then ρ, received by $\mathcal{A}$ from the oracle in $\mathsf{exp}^{\mathsf{indobf}}_{b,\mathcal{A}}(1^\lambda, 1^\kappa, 1, \mathsf{ges}, \mathsf{io})$, is:

$$\rho = \prod_{j=0}^{\kappa+1} \alpha_{j,x_{\mathsf{inp}(j)}} \left(\boxed{\gamma^1(\boldsymbol{a},\boldsymbol{r})\beta^{(\kappa+1)}} + \ldots + \gamma^{\kappa+1}(\boldsymbol{a},\boldsymbol{r})\beta^{(1)} + \gamma^{\kappa+2}(\boldsymbol{r}) \right) \mod q; \tag{VII;17}$$

where $\alpha_{j,b'}$ is an input-mixing scalar.[19] The boxed polynomial is the one that we seek to annihilate again. Notice that the attack from Theorem VII;6.2 no longer holds, since for any pair of inputs x^1, x^2, then the product $\alpha_{x^c} = \prod_{j=0}^{\kappa+1} \alpha_{j,x^c_{\mathsf{inp}(j)}}$ is different depending on the $c \in \{1,2\}$ that is used. Moreover, these scalars act as another unknown, and so γ^1 is no longer linear in unknown variables.

However, we show that we can choose pairs of inputs $x, \overline{x}$ resulting in encodings of zeros; and s.t. $\alpha_{x\overline{x}}$ is a product that is independent of the choice of the input pair.

First, take $x, \overline{x}$ such that $x_i = 1 - \overline{x}_i$ for all $i \in [\ell_{\mathsf{in}}]$. We require that $x, \overline{x}$ are both inputs that lead to top-level encodings of zeros, which is true for the choice of branching programs that we make in Theorem VII;6.2. This can be made to work for more general choices of branching programs, using the techniques of [15].

Now, consider the product $\rho_{x\overline{x}} = \rho_x \cdot \rho_{\overline{x}}$, where ρ_x and $\rho_{\overline{x}}$ are the respective outputs of program evaluations on $x, \overline{x}$, respectively. Then, we have

$$\rho_{x\overline{x}} = \alpha_{x\overline{x}} \left(\boxed{\gamma^1(\boldsymbol{a},\boldsymbol{r})\beta^{(2\kappa+1)}} + \ldots + \gamma^{2\kappa+1}(\boldsymbol{a},\boldsymbol{r})\beta^{(1)} + \gamma^{2\kappa+2}(\boldsymbol{r}) \right) \mod q; \tag{VII;18}$$

where $\alpha_{x\overline{x}} = \prod_{i=0}^{\kappa+1} \alpha_{i,0}\alpha_{i,1}$, which includes all scalars. Therefore, we have created a new algebraic element, multiplied by all input-mixing scalars in the io scheme. Moreover, the polynomial γ^1 is now quadratic in the unknown $\boldsymbol{r}$ variables.

Linearising about these quadratic variables, we create a new vector $\boldsymbol{\nu}_t$ containing $L = (50\kappa + 10)^2$ unknowns, and now we can think of γ^1 as a linear polynomial again. Since L is still $\mathsf{poly}(\kappa)$, then the attack from Theorem VII;6.2 still applies. Albeit, we now need $> L$ pairs of inputs $x, \overline{x}$ that lead to encodings of zero.

Fortunately, $\mathbb{M}_0$ and $\mathbb{M}_1$ output 0 on all inputs and thus the requirement is satisfied. We can complete the attack by computing the "Gaussian elimination" from before to recover $Q^\dagger$ (i.e. setting $Q^\dagger = Q^*$), using coefficients from $\boldsymbol{a}$, over $> L$ pairs of coefficient vectors $\boldsymbol{\nu}_t$. This recovers an algebraic element ρ'_0 of the form

$$\rho'_0 = Q^\dagger(\rho_{x\overline{x}}) \mod q;$$

[19] Again, we abuse notation and let $\alpha_0 = \alpha_{0,x_{\mathsf{inp}(0)}}$, and similarly for $j = \kappa + 1$.

which has infinity norm noticeably smaller than $\rho_{x\overline{x}}$. To translate this into a distinguishing attack, we simply compute $Q^\dagger$ wrt $\mathbb{M}_0$ rather than $\mathbb{M}_1$. Finally, we should re-emphasise that if $\mathbb{M}_0$ and $\mathbb{M}_1$ correspond to the choices of branching programs in [240], then we can simply use $Q^\dagger = Q^*$ again — where Q^* is the annihilating polynomial that they give.

In both cases, the coefficients of $Q^\dagger$ remain a similar magnitude to the elements of $\boldsymbol{a}$, which are small. Therefore, we can detect when γ_0^1 has been removed from ρ_0'; and also when it has not. As such, $\mathcal{A}$ outputs 0 if ρ_0' has noticeably smaller norm, and 1 otherwise. This succeeds in $\mathsf{exp}_{b,\mathcal{A}}^{\mathsf{indobf}}(1^\lambda, 1^\kappa, 1, \mathsf{ges}, \mathsf{io})$ with probability close to 1. Thus, the proof of Claim VII;6.3.1 is complete. $\square$

Claim VII;6.3.2. *The attack succeeds with probability close to* 1 *for* $\eta = 2$.

Proof. In the dual-input case, we create $(100\kappa + 10)^2$ unknowns and thus we can carry out the attacks in polynomial-time using a similar strategy to Claim VII;6.2.2. Thus the proof of Claim VII;6.3.2 is complete. $\square$

The proof of Theorem VII;6.3 is completed by the proofs of Claims VII;6.3.1 and VII;6.3.2. $\square$

VII;6.3 Other obfuscators

We quickly summarise why our attack fails to work in the cases of [150, 153]. Essentially, it is due to the same reasons that the [240] does not work against these obfuscators. Recalling the structure of the obfuscated branching programs from Construction VII;2.3, these two obfuscators use randomised matrices in the bottom-right quadrant of each matrix. Even though the bookend vectors remove this randomisation from the final output, the random entries in these matrices are embedded into the polynomial γ^1. Since the entries are chosen randomly during the obfuscation, these coefficients actually act as additional unknown variables when trying to construct $Q^\dagger$. Therefore, the polynomial γ^1 has some monomials that are of degree κ in unknowns, implying that computing $Q^\dagger$ would be difficult.

In fact, the obfuscator of [153] proves that annihilating the set of γ^1 polynomials is hard under the BPUA assumption (in the weakened graded encoding model). This assumption also implies that finding $Q^\dagger$ in our situation is hard as the form of γ^1 is the same. However, we stop short of formally proving this as this is not the central theme of this chapter.

Summary of cryptanalysis. We conclude this section by noting that we have demonstrated attacks against GGHWOI in the IND-OBF security game, over the obfuscator given in Construction VII;2.3 (in the case of symmetric graded level sets). We note that the attack does not work in the case of asymmetric level sets due to the reasons highlighted in the proof of Claim VII;6.2.3. In addition, the attacks do not work against obfuscators that are not covered by the abstract obfuscator.

VII;7 Discussion of results and future work

We conclude this chapter by summarising the contents of our contribution and discussing various avenues of interest that may arise from our research.

Our attacks demonstrate crucial flaws against GGH, that seem to manifest themselves from different peculiarities of the encoding structure. This is rather than the presence of $\langle g \rangle$, as hypothesised by Halevi [176]. We do this by producing a new scheme GGHWOI with the same algebraic structure as GGH, but without a reliance on a common element g. This new scheme remains insecure via similar cryptanalytic methods to those that break the security of GGH.

In this work, we are unable to identify exactly what this vulnerability is. However, we now highlight some aspects of our results that may prove useful in trying to decipher the vulnerability. We discuss interesting potential avenues for future research.

Presence of γ^1. Rather than the physical presence of the ideal $\langle g \rangle$, we speculate that the actual insecurity of the GGH GES can be traced to the algebraic make-up of the polynomial γ^1, in a zero-tested top-level encoding. In fact, in both GGH and GGHWOI, this polynomial is implicitly used as a way of hiding the secret parameters of the scheme. Unfortunately, this polynomial is only linear in unknown variables in both schemes — meaning that annihilation attacks based on the work of [202] will always be possible. We think that the structure of this polynomial is key to the vulnerabilities of GGH.

Non-linear γ^1 may prevent attacks. A peculiar aspect of GGHWOI is that the asymmetric levels of the scheme are made explicit in the zero-tested encodings of zero. This is not the same as GGH where the level sets are completely removed by zero-testing. This means that launching annihilation attacks against the asymmetric variant of the scheme is difficult, since the polynomial γ^1 becomes non-linear in unknowns.

Generalising this strategy to a method of preventing attacks, if it were possible to concretely adapt zero-tested encodings so that they were of $\mathsf{poly}(\kappa)$ degree in unknowns, may prevent annihilation-based methods. However, we are unable to come up with a firm security reduction in this case.

In GGHWOI in the asymmetric setting, this naturally happens due to the presence of each of the variables in $\boldsymbol{\beta}$ in each monomial of each polynomial. Is it possible to expand this to a new security assumption? Or more likely, does a new attack strategy allow us to bypass these terms in a similar tactic to that used in the proof of Theorem VII;6.3?

Establishing 'algebraic' equivalence of schemes. The work of Chunsheng [94] subsequently showed that it was possible to adapt the attacks of [87] to the GGHRSW [150] obfuscator, when instantiated using GGHWOI. This further highlights the similarities between the GGH and GGHWOI graded encoding schemes.

Further attempts to establish the attack of [256] against GGHWOI for instantiation [153] would highlight that our scheme is equivalent to GGH in terms of security analysis. Moreover, if we could show that a security proof of [153] exists under GGHWOI, then we would have a better idea of the equivalence of the two schemes. We think that it is plausible such a proof exists, since the branching program unannihilatability assumption seems to prevent the possibility of annihilating the γ^1 polynomial from GGH-like encodings. This attack strategy is also central to our analysis.

Further parameter analysis. We do not tackle a lattice-based cryptanalysis of our scheme, since if we show that GGHWOI is insecure in all of the same scenarios as GGH, then it will have little utility beyond demonstrating additional vulnerabilities of GGH. However, if we can show a meaningful separation between the two schemes (particularly in the case of straddling sets encodings), then it would be necessary to expose our scheme to more extensive cryptanalysis of our parameter settings.

We expect that a major insecurity of GGHWOI is likely to be that encodings of non-zero may not be uniformly distributed in $\mathcal{R}_q$, and modular reduction is only likely to occur once.

Furthermore, the bounds that we use for establishing correctness are *very* loose to account for the binomial expression. If we could make these bounds tighter, then we would almost certainly be able to develop a much more efficient construction.

VIII Conclusion

fear and loathing

In this thesis we have collated a wide-ranging analysis of the field of secure computation. Our works spans new theoretical constructions, novel implementations of these cryptographic primitives that can be run for large-scale computation, and cryptanalysis of candidate program obfuscators.

The intention of our work is to aid the ongoing transition of the science of secure computation from being an inherently theoretical pursuit, to one that can have real-world applications and impact. To corroborate this message, we have ensured that all the novel constructions that we have devised come equipped with implementations for adequate scenarios and security parametrisations. We additionally hope that our cryptanalytic techniques can help to advance our understanding of what security properties are required for non-interactive secure computation schemes, such as IO.

In terms of future research, there are a variety of topics that arise as a result of our findings. These range from devising more expressive theoretical constructions, to writing optimised versions of our implementations to make gains in computational running times. We will also continue to maintain our browser extension Privacy Pass so that the large user base continues to experience better accessibility wrt the Cloudflare network.

Appendices

A CPRF source code

relocation

This appendix provides the source code for the implementation taken from Chapter V. This source code will be made more widely available on publication of the work from this chapter.

This code is organised into a typical `Go` file structure, i.e. the parent folder is:

`~/go/src/github.com/alxdavids/owf-cprf/`

and contains packages `cprf` and `util` that are structured as children folders. The code is copied here without any changes to the original `Go` syntax. All code found here is licensed under the following BSD 3-Clause License found at `https://tinyurl.com/cprf-lic`.

Package `cprf`

Package `cprf` contains the source code for running the core CPRF algorithms, it also contains the source code for testing and benchmarking our software. For the CPRF functionality, the following functions map to their equivalents:

- `KeyGen` → cprf.Setup;
- `Eval` → cprf.Eval;
- `Constrain` → cprf.Constrain.

We also create 'structs' to establish the types of certain elements that we need:

- `MasterSecretKey` → cprf.msk;
- `Input` → $x \in \{0,1\}^\ell$;
- `Constraint` → $v \in \{0,1,*\}^\ell$.

Each of these structs also has a number of utility functions assigned to it for processing their data in the correct format. For constrained keys, we reuse the `MasterSecretKey` struct, since these keys have identical formatting.

We provide all the source code for this package below.

CPRF.GO

```go
package cprf

import (
	"crypto"
	"crypto/rand"
	"errors"
	"fmt"
	"github.com/alxdavids/owf-cprf/prf"
	"github.com/alxdavids/owf-cprf/util"
	"github.com/golang-collections/go-datastructures/bitarray"
	"math"
	"math/big"
	"sync"
)

type MasterSecretKey struct {
	indices   [][]*big.Int
	keys      map[string]([][]byte)
	dummyKeys map[string]([][]byte)
	ell, r    *big.Int
	hash      crypto.Hash
}

func (msk *MasterSecretKey) getKeysForIndex(index string) [][]byte {
	arr := msk.keys[index]
	return arr
}

func (msk *MasterSecretKey) getDummyKeysForIndex(index string)
[][]byte {
	arr := msk.dummyKeys[index]
	return arr
}

func (msk *MasterSecretKey) getStrIndex(i int) string {
	return util.JoinBigIntSlice(msk.indices[i])
}

func (msk *MasterSecretKey) getIndices() [][]*big.Int {
	return msk.indices
}

func (msk *MasterSecretKey) getKeysForInput(input *Input)
([][]byte, error) {
	if msk.ell.Cmp(input.ell) != 0 {
		return nil, errors.New(fmt.Sprintf("Incompatible input lengths
			for key: %v, and input: %v", msk.ell, input.ell))
```

```go
    }
    indices := msk.indices
    var inputKeys [][]byte
    for i := 0; i < len(indices); i++ {
        str := msk.getStrIndex(i)
        keys := msk.getKeysForIndex(str)
        reVal, err := input.reindex(indices[i])
        if err != nil {
            return nil, err
        }
        inputKeys = append(inputKeys, keys[reVal.Int64()])
    }
    return inputKeys, nil
}

func (msk *MasterSecretKey) getConstrainedKey(v *Constraint)
(map[string]([][]byte), error) {
    if msk.ell.Int64() != v.ell {
        return nil, errors.New(fmt.Sprintf("Incompatible input lengths
            for key: %v, and constraint: %v", msk.ell.Int64(), v.ell))
    }
    indices := msk.indices
    cKeys := make(map[string]([][]byte), len(msk.keys))
    for i := 0; i < len(indices); i++ {
        str := msk.getStrIndex(i)
        keys := msk.getKeysForIndex(str)
        dummyKeys := msk.getDummyKeysForIndex(str)
        reVals, err := v.reindex(indices[i])
        if err != nil {
            return nil, err
        }

        z := util.Unique(indices[i])
        total := int(math.Pow(float64(2), float64(z)))
        for ind := 0; ind < total; ind++ {
            found := false
            for l := 0; l < len(reVals); l++ {
                if int64(ind) == reVals[l].Int64() {
                    cKeys[str] = append(cKeys[str], keys[ind])
                    found = true
                    break
                }
            }
            if !found {
                cKeys[str] = append(cKeys[str], dummyKeys[ind])
            }
        }
    }

    return cKeys, nil
}

type Input struct {
    bits      bitarray.BitArray
    val, ell *big.Int
}

// Sets up a valid input for the CPRF based on the parameter choice
func (x *Input) SetParams(val, ell *big.Int) error {
    if val.Cmp(new(big.Int).Exp(big.NewInt(2), ell, nil)) != -1 {
        return errors.New("Chosen value is bigger than input length
            implies")
    }
    x.ell = ell
    x.val = val
    x.bits = bitarray.NewBitArray(ell.Uint64())
    for i := ell.Int64() - int64(1); i >= 0; i-- {
        pow := new(big.Int).Exp(big.NewInt(2), big.NewInt(i), nil)
        newVal := new(big.Int).Sub(val, pow)
        if newVal.Cmp(big.NewInt(0)) > -1 {
            x.bits.SetBit(uint64(i))
            val = newVal
        }
    }
    return nil
}

// Re-index the input with respect to the index vector that is being
// used
func (x *Input) reindex(indices []*big.Int) (*big.Int, error) {
    last := big.NewInt(-1)
    bits := x.bits
    z := util.Unique(indices)
    cnt := int64(z - 1)
    reVal := big.NewInt(0)
    for i := 0; i < len(indices); i++ {
        if last.Cmp(indices[i]) == 0 {
            continue
        }
        last = indices[i]

        b, err := bits.GetBit(indices[i].Uint64())
        if err != nil {
            return nil, err
        }

        if b {
            reVal = new(big.Int).Add(reVal,
                new(big.Int).Exp(big.NewInt(2), big.NewInt(cnt), nil))
        }
        cnt--
    }

    return reVal, nil
}

type Constraint struct {
    bits      bitarray.BitArray
    ell       int64
    wildcards map[int64](bool)
}

// Sets up a valid input for the CPRF based on the parameter choice
func (v *Constraint) SetParams(val, ell *big.Int, wilds []int64) error {
    if val.Cmp(new(big.Int).Exp(big.NewInt(2), ell, nil)) != -1 {
        return errors.New("Chosen value is bigger than input length
            implies")
    }
    v.ell = ell.Int64()
    v.bits = bitarray.NewBitArray(ell.Uint64())
    for i := ell.Int64() - int64(1); i >= 0; i-- {
        pow := new(big.Int).Exp(big.NewInt(2), big.NewInt(i), nil)
        newVal := new(big.Int).Sub(val, pow)
        if newVal.Cmp(big.NewInt(0)) > -1 {
            v.bits.SetBit(uint64(i))
            val = newVal
        }
    }

    wildcards := make(map[int64]bool)
    for i := 0; i < len(wilds); i++ {
        val := int64(wilds[i])
        if val > ell.Int64() {
            return errors.New(fmt.Sprintf("Wildcard value: %v, is out of
             bounds for ell: %v", val, ell))
        }
        wildcards[wilds[i]] = true
    }
    v.wildcards = wildcards
    return nil
}

// Reindex for constraints
func (v *Constraint) reindex(indices []*big.Int) ([]*big.Int, error) {
    last := big.NewInt(-1)
    bits := v.bits
    wilds := v.wildcards
```

```go
    z := util.Unique(indices)
    cnt := int64(z - 1)
    reVals := []*big.Int{big.NewInt(0)}
    for i := 0; i < len(indices); i++ {
        if last.Cmp(indices[i]) == 0 {
            continue
        }
        last = indices[i]

        w := wilds[indices[i].Int64()]
        b, err := bits.GetBit(indices[i].Uint64())
        if err != nil {
            return nil, err
        }

        if w {
            var newReVals []*big.Int
            for j := 0; j < len(reVals); j++ {
                reval := reVals[j]
                reval0 := reval
                reval1 := new(big.Int).Add(reval,
                    new(big.Int).Exp(big.NewInt(2), big.NewInt(cnt), nil))
                newReVals = append(newReVals, reval0)
                newReVals = append(newReVals, reval1)
            }
            reVals = newReVals
        } else if b {
            for j := 0; j < len(reVals); j++ {
                reVals[j] = new(big.Int).Add(reVals[j],
                        new(big.Int).Exp(big.NewInt(2), big.NewInt(cnt),
                        nil))
            }
        }
        cnt--
    }

    return reVals, nil
}

type KeyEntry struct {
    prfKey []byte
    j      int
    err    error
}

type KeyMap struct {
    index     string
    keys      [][]byte
    dummyKeys [][]byte
    err        error
}

// KeyGen: Master secret key generation
// @param {int} lambda Security parameter
// @param {*big.Int} ell    Input length
// @param {*big.Int} r      Number of Constraints
// @returns {map[string]([][]byte)} Master secret key
func KeyGen(lambda int, ell, r *big.Int, h crypto.Hash)
(*MasterSecretKey, error) {
    mskLen := new(big.Int).Exp(ell, r, nil)
    indices := util.IndexGen(ell, r, mskLen)
    cprfKeys := make(map[string]([][]byte), len(indices))
    cprfDummyKeys := make(map[string]([][]byte), len(indices))
    chanOut := make(chan *KeyMap)
    for i := 0; i < len(indices); i++ {
        go func(indexArr []*big.Int, lambda int) {
            z := util.Unique(indexArr)
            powz := math.Exp2(float64(z))
            km := &KeyMap{}
            km.keys = make([][]byte, int(powz))
            km.dummyKeys = make([][]byte, int(powz))
            chanKE := make(chan *KeyEntry)
            chanKD := make(chan *KeyEntry)
            for j := 0; j < int(powz); j++ {
                go func(lambda, j int) {
                    ke := &KeyEntry{}
                    prfKey := make([]byte, lambda)
                    _, err := rand.Read(prfKey)
                    if err != nil {
                        ke.err = err
                        chanKE <- ke
                    }
                    ke.prfKey = prfKey
                    ke.j = j
                    chanKE <- ke
                }(lambda, j)

                go func(lambda, j int) {
                    ke := &KeyEntry{}
                    dummyKey := make([]byte, lambda)
                    _, err := rand.Read(dummyKey)
                    if err != nil {
                        ke.err = err
                        chanKD <- ke
                    }
                    ke.prfKey = dummyKey
                    ke.j = j
                    chanKD <- ke
```

```go
                }(lambda, j)
            }

            var wg sync.WaitGroup
            wg.Add(2)
            go func(km *KeyMap, wg *sync.WaitGroup, powz float64,
            indexArr []*big.Int) {
                for l := 0; l < int(powz); l++ {
                    ke := <-chanKE
                    if ke.err != nil {
                        km.err = ke.err
                        break
                    }
                    km.keys[ke.j] = ke.prfKey
                }
                wg.Done()
            }(km, &wg, powz, indexArr)

            go func(km *KeyMap, wg *sync.WaitGroup, powz float64) {
                for l := 0; l < int(powz); l++ {
                    ke := <-chanKD
                    if ke.err != nil {
                        km.err = ke.err
                        break
                    }
                    km.dummyKeys[ke.j] = ke.prfKey
                }
                wg.Done()
            }(km, &wg, powz)

            wg.Wait()
            if km.err != nil {
                chanOut <- km
            }
            indexStr := util.JoinBigIntSlice(indexArr)
            km.index = indexStr
            chanOut <- km
        }(indices[i], lambda)
    }

    for i := 0; i < len(indices); i++ {
        km := <-chanOut
        if km.err != nil {
            return nil, km.err
        }
        cprfKeys[km.index] = km.keys
        cprfDummyKeys[km.index] = km.dummyKeys
    }
    msk := &MasterSecretKey{indices: indices, keys: cprfKeys,
        dummyKeys: cprfDummyKeys, ell: ell, r: r, hash: h}
    return msk, nil
}

// Eval: CPRF evaluation algorithm
// @param {*MasterSecretKey} msk
// @param {*Input} x
// @returns {[]byte} PRF evaluation
// @returns {error}
func Eval(msk *MasterSecretKey, x *Input) ([]byte, error) {
    keys, err := msk.getKeysForInput(x)
    if err != nil {
        return nil, err
    }
    out := big.NewInt(0)
    for i := 0; i < len(keys); i++ {
        newPRF := prf.InitPRF(msk.hash, keys[i])
        y := newPRF.Eval(x.val.Bytes())
        out = new(big.Int).Xor(out, new(big.Int).SetBytes(y))
    }
    return out.Bytes(), nil
}

// Constrain: CPRF constraining algorithm
// @param {*MasterSecretKey} msk
// @param {*Constraint} v
// @returns {*MasterSecretKey} Constrained key
// @returns {error}
func Constrain(msk *MasterSecretKey, v *Constraint)
(*MasterSecretKey, error) {
    cKeys, err := msk.getConstrainedKey(v)
    if err != nil {
        return nil, err
    }

    return &MasterSecretKey{keys: cKeys, ell: msk.ell, r: msk.r,
        indices: msk.indices, hash: msk.hash}, nil
}
```

CPRF_TEST.GO

```go
package cprf

import (
    "bytes"
    "crypto"
    "crypto/rand"
    "errors"
    "log"
    "math"
    "math/big"
    "testing"
)

var (
    lambda  int              = 8
    ell     *big.Int         = big.NewInt(4)
    r       *big.Int         = big.NewInt(3)
    xVal    *big.Int         = big.NewInt(13)
    h       crypto.Hash      = crypto.SHA256
    counts  map[string](int) = make(map[string]int, 16)
    lastKey *MasterSecretKey
)

// Test master secret key generation
func TestMSK(t *testing.T) {
    msk, err := KeyGen(lambda, ell, r, h)
    if err != nil {
        t.Fatalf("Key generation failed: %v", err)
    }
    if ell != msk.ell {
        t.Fatalf("ell: %v, msk.ell: %v", ell, msk.ell)
    }
    if r != msk.r {
        t.Fatalf("r: %v, msk.r: %v", r, msk.r)
    }

    // Check that the key map index vectors are ordered
    for ind := range msk.keys {
        last := 10
        for _, char := range ind {
            curr := int(char - '0')
            if curr == 76 {
                // this is a pipe character?
                continue
            }
            if curr > last {
                t.Fatalf("Last digit: %v, was smaller than current: %v",
                    last, curr)
            }
            last = curr
        }
    }

    // Check that the key map index is populated correctly
    for ind := range msk.keys {
        last := 10
        count := r.Int64()
        for _, char := range ind {
            curr := int(char - '0')
            if curr == 76 {
                // probably a pipe character
                continue
            }
            if curr == last {
                count--
            }
            last = curr
        }
        ele := msk.keys[ind]
        if len(ele) != int(math.Pow(2, float64(count))) {
            t.Fatalf("Incorrect length: %v, should be: %v", len(ele),
                int(math.Pow(2, float64(count))))
        }
    }

    // Check that the input keys are assigned correctly
    x := &Input{}
    x.SetParams(xVal, ell)
    inpKeys, err := msk.getKeysForInput(x)
    if err != nil {
        t.Fatalf("Input generation failed: %v", err)
    }

    err = lookForKey("3|2|0", msk, inpKeys)
    if err != nil {
        t.Fatalf("Possibly did not find correct key for input %v, with
            keys: %v, error: %v", "320", inpKeys, err)
    }
    err = lookForKey("2|1|1", msk, inpKeys)
    if err != nil {
        t.Fatalf("Possibly did not find correct key for input %v, with
            keys: %v, error: %v", "211", inpKeys, err)
    }
    err = lookForKey("0|0|0", msk, inpKeys)
    if err != nil {
        t.Fatalf("Possibly did not find correct key for input %v, with
            keys: %v, error: %v", "000", inpKeys, err)
    }
}

// Test that the constrained evaluation and standard evaluation
// mechanisms match up
func TestCEval(t *testing.T) {
    msk, err := KeyGen(lambda, ell, r, h)
    if err != nil {
        t.Fatalf("Key generation failed: %v", err)
    }

    x1 := &Input{}
    x1.SetParams(big.NewInt(7), ell)
    if err != nil {
        t.Fatalf("Input generation failed: %v", err)
    }
    x2 := &Input{}
    x2.SetParams(big.NewInt(3), ell)
    if err != nil {
        t.Fatalf("Input generation failed: %v", err)
    }
    x3 := &Input{}
    x3.SetParams(big.NewInt(8), ell)
    if err != nil {
        t.Fatalf("Input generation failed: %v", err)
    }

    v1 := &Constraint{}
    v1.SetParams(big.NewInt(6), ell, []int64{0, 3})
    ck1, err := Constrain(msk, v1)
    if err != nil {
        t.Fatalf("\nError occurred in constraining: %v\n", err)
    }
    v2 := &Constraint{}
    v2.SetParams(big.NewInt(1), ell, []int64{1})
    ck2, err := Constrain(msk, v2)
    if err != nil {
        t.Fatalf("\nError occurred in constraining: %v\n", err)
    }
    v3 := &Constraint{}
    v3.SetParams(big.NewInt(15), ell, []int64{0, 1, 2})
    ck3, err := Constrain(msk, v3)
    if err != nil {
        t.Fatalf("\nError occurred in constraining: %v\n", err)
    }

    cks := []*MasterSecretKey{ck1, ck2, ck3}
    inps := []*Input{x1, x2, x3}
    evals := []bool{true, false, false, false, true, false, false,
        false, true}
    for i := 0; i < len(inps); i++ {
        for j := 0; j < len(cks); j++ {
            yMsk, err := Eval(msk, inps[i])
            if err != nil {
                t.Fatalf("\nError occurred in evaluation: %v\n", err)
            }
            yCk, err := Eval(cks[j], inps[i])
            if err != nil {
                t.Fatalf("\nError occurred in evaluation: %v\n", err)
            }

            if evals[3*i+j] && !bytes.Equal(yMsk, yCk) {
                t.Fatalf("\nEvaluations should be the same but they are
                    not for i: %v, j: %v, yMsk: %v, yCk: %v", i, j,
                    yMsk, yCk)
            }

            if !evals[3*i+j] && bytes.Equal(yMsk, yCk) {
                t.Fatalf("\nEvaluations should be different but they are
                    the same for i: %v, j: %v, yMsk: %v, yCk: %v", i, j,
                    yMsk, yCk)
            }
        }
    }
}

func lookForKey(chosenIndex string, msk *MasterSecretKey,
inpKeys [][]byte) error {
    chkKeys := msk.getKeysForIndex(chosenIndex)
    var index int
    switch chosenIndex {
    case "3|2|0":
        index = 7
        break
    case "2|1|1":
        index = 2
        break
    case "0|0|0":
        index = 1
        break
    default:
        return errors.New("Incorrect index used")
    }
    chosen := chkKeys[index]
    found := false
    for i := 0; i < len(inpKeys); i++ {
        curr := inpKeys[i]
        if new(big.Int).SetBytes(chosen)
            .Cmp(new(big.Int).SetBytes(curr)) == 0 {
            found = true
            break
        }
    }

    if !found {
        return errors.New("Not found")
    }
    return nil
}

func benchKeyGen(b *testing.B, lambda int, ell, r *big.Int,
h crypto.Hash) {
    b.RunParallel(func(pb *testing.PB) {
        for pb.Next() {
            msk, err := KeyGen(lambda, ell, r, h)
            if err != nil {
                b.Fatalf("Key generation failed: %v", err)
            }
            lastKey = msk
        }
    })
}

func benchEval(b *testing.B, lambda int, ell, r *big.Int,
h crypto.Hash) {
    msk := lastKey
    max := new(big.Int).Exp(big.NewInt(2), ell, nil)
    randInt, err := rand.Int(rand.Reader, max)
    if err != nil {
        b.Fatalf("error in rand int: %v", err)
    }
    x := &Input{}
    x.SetParams(randInt, ell)
    b.ResetTimer()
    for i := 0; i < b.N; i++ {
        _, err := Eval(msk, x)
        if err != nil {
            b.Fatalf("\nError occurred in evaluation: %v\n", err)
        }
    }
}

func benchConstrain(b *testing.B, lambda int, ell, r *big.Int,
h crypto.Hash) {
    msk := lastKey
    max := new(big.Int).Exp(big.NewInt(2), ell, nil)
    randInt, err := rand.Int(rand.Reader, max)
    if err != nil {
        b.Fatalf("error in rand int: %v", err)
    }
    v := &Constraint{}
    v.SetParams(randInt, ell, []int64{0, 1, 3})
    b.ResetTimer()
    for i := 0; i < b.N; i++ {
        _, err := Constrain(msk, v)
        if err != nil {
            b.Fatalf("\nError occurred in constraining: %v\n", err)
        }
    }
}

func BenchmarkKeyGen1(b *testing.B) {
    if counts == nil {
        counts["1"] = 0
    }
    count := counts["1"]
    printBanner(80, 4, 2, count, b)
    b.ResetTimer()
    benchKeyGen(b, 10, big.NewInt(4), big.NewInt(2), crypto.SHA256)
    counts["1"] = count + 1
}
func BenchmarkEval1(b *testing.B) { benchEval(b, 10, big.NewInt(4),
    big.NewInt(2), crypto.SHA256) }
func BenchmarkConstrain1(b *testing.B) {
    benchConstrain(b, 10, big.NewInt(4), big.NewInt(2), crypto.SHA256)
}

func BenchmarkKeyGen2(b *testing.B) {
    if counts == nil {
        counts["2"] = 0
    }
    count := counts["2"]
    printBanner(80, 4, 3, count, b)
    b.ResetTimer()
    benchKeyGen(b, 10, big.NewInt(4), big.NewInt(3), crypto.SHA256)
    counts["2"] = count + 1
}
func BenchmarkEval2(b *testing.B) { benchEval(b, 10, big.NewInt(4),
    big.NewInt(3), crypto.SHA256) }
func BenchmarkConstrain2(b *testing.B) {
    benchConstrain(b, 10, big.NewInt(4), big.NewInt(3), crypto.SHA256)
}

func BenchmarkKeyGen3(b *testing.B) {
    if counts == nil {
        counts["3"] = 0
    }
    count := counts["3"]
    printBanner(80, 4, 4, count, b)
    b.ResetTimer()
    benchKeyGen(b, 10, big.NewInt(4), big.NewInt(4), crypto.SHA256)
    counts["3"] = count + 1
}
```

```
func BenchmarkEval3(b *testing.B) { benchEval(b, 10, big.NewInt(4),
    big.NewInt(4), crypto.SHA256) }
func BenchmarkConstrain3(b *testing.B) {
    benchConstrain(b, 10, big.NewInt(4), big.NewInt(4), crypto.SHA256)
}

func BenchmarkKeyGen4(b *testing.B) {
    if counts == nil {
        counts["4"] = 0
    }
    count := counts["4"]
    printBanner(80, 8, 2, count, b)
    b.ResetTimer()
    benchKeyGen(b, 10, big.NewInt(8), big.NewInt(2), crypto.SHA256)
    counts["4"] = count + 1
}
func BenchmarkEval4(b *testing.B) { benchEval(b, 10, big.NewInt(8),
    big.NewInt(2), crypto.SHA256) }
func BenchmarkConstrain4(b *testing.B) {
    benchConstrain(b, 10, big.NewInt(8), big.NewInt(2), crypto.SHA256)
}

func BenchmarkKeyGen5(b *testing.B) {
    if counts == nil {
        counts["5"] = 0
    }
    count := counts["5"]
    printBanner(80, 8, 3, count, b)
    b.ResetTimer()
    benchKeyGen(b, 10, big.NewInt(8), big.NewInt(3), crypto.SHA256)
    counts["5"] = count + 1
}
func BenchmarkEval5(b *testing.B) { benchEval(b, 10, big.NewInt(8),
    big.NewInt(3), crypto.SHA256) }
func BenchmarkConstrain5(b *testing.B) {
    benchConstrain(b, 10, big.NewInt(8), big.NewInt(3), crypto.SHA256)
}

func BenchmarkKeyGen6(b *testing.B) {
    if counts == nil {
        counts["6"] = 0
    }
    count := counts["6"]
    printBanner(80, 8, 4, count, b)
    b.ResetTimer()
    benchKeyGen(b, 10, big.NewInt(8), big.NewInt(4), crypto.SHA256)
    counts["6"] = count + 1
}
func BenchmarkEval6(b *testing.B) { benchEval(b, 10, big.NewInt(8),
    big.NewInt(4), crypto.SHA256) }
func BenchmarkConstrain6(b *testing.B) {
    benchConstrain(b, 10, big.NewInt(8), big.NewInt(4), crypto.SHA256)
}

func BenchmarkKeyGen7(b *testing.B) {
    if counts == nil {
        counts["7"] = 0
    }
    count := counts["7"]
    printBanner(80, 8, 5, count, b)
    b.ResetTimer()
    benchKeyGen(b, 10, big.NewInt(8), big.NewInt(5), crypto.SHA256)
    counts["7"] = count + 1
}
func BenchmarkEval7(b *testing.B) { benchEval(b, 10, big.NewInt(8),
    big.NewInt(5), crypto.SHA256) }
func BenchmarkConstrain7(b *testing.B) {
    benchConstrain(b, 10, big.NewInt(8), big.NewInt(5), crypto.SHA256)
}

func BenchmarkKeyGen8(b *testing.B) {
    if counts == nil {
        counts["8"] = 0
    }
    count := counts["8"]
    printBanner(80, 8, 6, count, b)
    b.ResetTimer()
    benchKeyGen(b, 10, big.NewInt(8), big.NewInt(6), crypto.SHA256)
    counts["8"] = count + 1
}
func BenchmarkEval8(b *testing.B) { benchEval(b, 10, big.NewInt(8),
    big.NewInt(6), crypto.SHA256) }
func BenchmarkConstrain8(b *testing.B) {
    benchConstrain(b, 10, big.NewInt(8), big.NewInt(6), crypto.SHA256)
}

func BenchmarkKeyGen22(b *testing.B) {
    if counts == nil {
        counts["22"] = 0
    }
    count := counts["22"]
    printBanner(80, 16, 2, count, b)
    b.ResetTimer()
    benchKeyGen(b, 10, big.NewInt(16), big.NewInt(2), crypto.SHA256)
    counts["22"] = count + 1
}
func BenchmarkEval22(b *testing.B) { benchEval(b, 10, big.NewInt(16),
    big.NewInt(2), crypto.SHA256) }
```

```
func BenchmarkConstrain22(b *testing.B) {
    benchConstrain(b, 10, big.NewInt(16), big.NewInt(2), crypto.SHA256)
}

func BenchmarkKeyGen23(b *testing.B) {
    if counts == nil {
        counts["23"] = 0
    }
    count := counts["23"]
    printBanner(80, 16, 3, count, b)
    b.ResetTimer()
    benchKeyGen(b, 10, big.NewInt(16), big.NewInt(3), crypto.SHA256)
    counts["23"] = count + 1
}
func BenchmarkEval23(b *testing.B) { benchEval(b, 10, big.NewInt(16),
    big.NewInt(3), crypto.SHA256) }
func BenchmarkConstrain23(b *testing.B) {
    benchConstrain(b, 10, big.NewInt(16), big.NewInt(3), crypto.SHA256)
}

func BenchmarkKeyGen24(b *testing.B) {
    if counts == nil {
        counts["24"] = 0
    }
    count := counts["24"]
    printBanner(80, 16, 4, count, b)
    b.ResetTimer()
    benchKeyGen(b, 10, big.NewInt(16), big.NewInt(4), crypto.SHA256)
    counts["24"] = count + 1
}
func BenchmarkEval24(b *testing.B) { benchEval(b, 10, big.NewInt(16),
    big.NewInt(4), crypto.SHA256) }
func BenchmarkConstrain24(b *testing.B) {
    benchConstrain(b, 10, big.NewInt(16), big.NewInt(4), crypto.SHA256)
}

func BenchmarkKeyGen25(b *testing.B) {
    if counts == nil {
        counts["25"] = 0
    }
    count := counts["25"]
    printBanner(80, 16, 5, count, b)
    b.ResetTimer()
    benchKeyGen(b, 10, big.NewInt(16), big.NewInt(5), crypto.SHA256)
    counts["25"] = count + 1
}
func BenchmarkEval25(b *testing.B) { benchEval(b, 10, big.NewInt(16),
    big.NewInt(5), crypto.SHA256) }
func BenchmarkConstrain25(b *testing.B) {
    benchConstrain(b, 10, big.NewInt(16), big.NewInt(5), crypto.SHA256)
}

func BenchmarkKeyGen9(b *testing.B) {
    if counts == nil {
        counts["9"] = 0
    }
    count := counts["9"]
    printBanner(128, 4, 2, count, b)
    b.ResetTimer()
    benchKeyGen(b, 16, big.NewInt(4), big.NewInt(2), crypto.SHA256)
    counts["9"] = count + 1
}
func BenchmarkEval9(b *testing.B) { benchEval(b, 16, big.NewInt(4),
    big.NewInt(2), crypto.SHA256) }
func BenchmarkConstrain9(b *testing.B) {
    benchConstrain(b, 16, big.NewInt(4), big.NewInt(2), crypto.SHA256)
}

func BenchmarkKeyGen10(b *testing.B) {
    if counts == nil {
        counts["10"] = 0
    }
    count := counts["10"]
    printBanner(128, 4, 3, count, b)
    b.ResetTimer()
    benchKeyGen(b, 16, big.NewInt(4), big.NewInt(3), crypto.SHA256)
    counts["10"] = count + 1
}
func BenchmarkEval10(b *testing.B) { benchEval(b, 16, big.NewInt(4),
    big.NewInt(3), crypto.SHA256) }
func BenchmarkConstrain10(b *testing.B) {
    benchConstrain(b, 16, big.NewInt(4), big.NewInt(3), crypto.SHA256)
}

func BenchmarkKeyGen11(b *testing.B) {
    if counts == nil {
        counts["11"] = 0
    }
    count := counts["11"]
    printBanner(128, 4, 4, count, b)
    b.ResetTimer()
    benchKeyGen(b, 16, big.NewInt(4), big.NewInt(4), crypto.SHA256)
    counts["11"] = count + 1
}
func BenchmarkEval11(b *testing.B) { benchEval(b, 16, big.NewInt(4),
    big.NewInt(4), crypto.SHA256) }
func BenchmarkConstrain11(b *testing.B) {
    benchConstrain(b, 16, big.NewInt(4), big.NewInt(4), crypto.SHA256)
```

```
}

func BenchmarkKeyGen12(b *testing.B) {
    if counts == nil {
        counts["12"] = 0
    }
    count := counts["12"]
    printBanner(128, 8, 2, count, b)
    b.ResetTimer()
    benchKeyGen(b, 16, big.NewInt(8), big.NewInt(2), crypto.SHA256)
    counts["12"] = count + 1
}
func BenchmarkEval12(b *testing.B) { benchEval(b, 16, big.NewInt(8),
    big.NewInt(2), crypto.SHA256) }
func BenchmarkConstrain12(b *testing.B) {
    benchConstrain(b, 16, big.NewInt(8), big.NewInt(2), crypto.SHA256)
}

func BenchmarkKeyGen13(b *testing.B) {
    if counts == nil {
        counts["13"] = 0
    }
    count := counts["13"]
    printBanner(128, 8, 3, count, b)
    b.ResetTimer()
    benchKeyGen(b, 16, big.NewInt(8), big.NewInt(3), crypto.SHA256)
    counts["13"] = count + 1
}
func BenchmarkEval13(b *testing.B) { benchEval(b, 16, big.NewInt(8),
    big.NewInt(3), crypto.SHA256) }
func BenchmarkConstrain13(b *testing.B) {
    benchConstrain(b, 16, big.NewInt(8), big.NewInt(3), crypto.SHA256)
}

func BenchmarkKeyGen14(b *testing.B) {
    if counts == nil {
        counts["14"] = 0
    }
    count := counts["14"]
    printBanner(128, 8, 4, count, b)
    b.ResetTimer()
    benchKeyGen(b, 16, big.NewInt(8), big.NewInt(4), crypto.SHA256)
    counts["14"] = count + 1
}
func BenchmarkEval14(b *testing.B) { benchEval(b, 16, big.NewInt(8),
    big.NewInt(4), crypto.SHA256) }
func BenchmarkConstrain14(b *testing.B) {
    benchConstrain(b, 16, big.NewInt(8), big.NewInt(4), crypto.SHA256)
}
```

```go

func BenchmarkKeyGen15(b *testing.B) {
    if counts == nil {
        counts["15"] = 0
    }
    count := counts["15"]
    printBanner(128, 8, 5, count, b)
    b.ResetTimer()
    benchKeyGen(b, 16, big.NewInt(8), big.NewInt(5), crypto.SHA256)
    counts["15"] = count + 1
}
func BenchmarkEval15(b *testing.B) { benchEval(b, 16, big.NewInt(8),
    big.NewInt(5), crypto.SHA256) }
func BenchmarkConstrain15(b *testing.B) {
    benchConstrain(b, 16, big.NewInt(8), big.NewInt(5), crypto.SHA256)
}

func BenchmarkKeyGen16(b *testing.B) {
    if counts == nil {
        counts["16"] = 0
    }
    count := counts["16"]
    printBanner(128, 8, 6, count, b)
    b.ResetTimer()
    benchKeyGen(b, 16, big.NewInt(8), big.NewInt(6), crypto.SHA256)
    counts["16"] = count + 1
}
func BenchmarkEval16(b *testing.B) { benchEval(b, 16, big.NewInt(8),
    big.NewInt(6), crypto.SHA256) }
func BenchmarkConstrain16(b *testing.B) {
    benchConstrain(b, 16, big.NewInt(8), big.NewInt(6), crypto.SHA256)
}

func BenchmarkKeyGen17(b *testing.B) {
    if counts == nil {
        counts["17"] = 0
    }
    count := counts["17"]
    printBanner(128, 16, 2, count, b)
    b.ResetTimer()
    benchKeyGen(b, 16, big.NewInt(16), big.NewInt(2), crypto.SHA256)
    counts["17"] = count + 1
}
func BenchmarkEval17(b *testing.B) { benchEval(b, 16, big.NewInt(16),
    big.NewInt(2), crypto.SHA256) }
func BenchmarkConstrain17(b *testing.B) {
    benchConstrain(b, 16, big.NewInt(16), big.NewInt(2), crypto.SHA256)
}

```

```go
func BenchmarkKeyGen18(b *testing.B) {
    if counts == nil {
        counts["18"] = 0
    }
    count := counts["18"]
    printBanner(128, 16, 3, count, b)
    b.ResetTimer()
    benchKeyGen(b, 16, big.NewInt(16), big.NewInt(3), crypto.SHA256)
    counts["18"] = count + 1
}
func BenchmarkEval18(b *testing.B) { benchEval(b, 16, big.NewInt(16),
    big.NewInt(3), crypto.SHA256) }
func BenchmarkConstrain18(b *testing.B) {
    benchConstrain(b, 16, big.NewInt(16), big.NewInt(3), crypto.SHA256)
}

func BenchmarkKeyGen19(b *testing.B) {
    if counts == nil {
        counts["19"] = 0
    }
    count := counts["19"]
    printBanner(128, 16, 4, count, b)
    b.ResetTimer()
    benchKeyGen(b, 16, big.NewInt(16), big.NewInt(4), crypto.SHA256)
    counts["19"] = count + 1
}
func BenchmarkEval19(b *testing.B) { benchEval(b, 16, big.NewInt(16),
    big.NewInt(4), crypto.SHA256) }
func BenchmarkConstrain19(b *testing.B) {
    benchConstrain(b, 16, big.NewInt(16), big.NewInt(4), crypto.SHA256)
}

func BenchmarkKeyGen20(b *testing.B) {
    if counts == nil {
        counts["20"] = 0
    }
    count := counts["20"]
    printBanner(128, 16, 5, count, b)
    b.ResetTimer()
    benchKeyGen(b, 16, big.NewInt(16), big.NewInt(5), crypto.SHA256)
    counts["20"] = count + 1
}
func BenchmarkEval20(b *testing.B) { benchEval(b, 16, big.NewInt(16),
    big.NewInt(5), crypto.SHA256) }
func BenchmarkConstrain20(b *testing.B) {
    benchConstrain(b, 16, big.NewInt(16), big.NewInt(5), crypto.SHA256)
}

func BenchmarkKeyGen30(b *testing.B) {
    if counts == nil {
        counts["30"] = 0
    }
    count := counts["30"]
    printBanner(80, 80, 2, count, b)
    b.ResetTimer()
    benchKeyGen(b, 10, big.NewInt(80), big.NewInt(2), crypto.SHA256)
    counts["30"] = count + 1
}
func BenchmarkEval30(b *testing.B) { benchEval(b, 10, big.NewInt(80),
    big.NewInt(2), crypto.SHA256) }
func BenchmarkConstrain30(b *testing.B) {
    benchConstrain(b, 10, big.NewInt(80), big.NewInt(2), crypto.SHA256)
}

func BenchmarkKeyGen31(b *testing.B) {
    if counts == nil {
        counts["31"] = 0
    }
    count := counts["31"]
    printBanner(80, 80, 3, count, b)
    b.ResetTimer()
    benchKeyGen(b, 10, big.NewInt(80), big.NewInt(3), crypto.SHA256)
    counts["31"] = count + 1
}
func BenchmarkEval31(b *testing.B) { benchEval(b, 10, big.NewInt(80),
    big.NewInt(3), crypto.SHA256) }
func BenchmarkConstrain31(b *testing.B) {
    benchConstrain(b, 10, big.NewInt(80), big.NewInt(3), crypto.SHA256)
}

func BenchmarkKeyGen32(b *testing.B) {
    if counts == nil {
        counts["32"] = 0
    }
    count := counts["32"]
    printBanner(80, 80, 4, count, b)
    b.ResetTimer()
    benchKeyGen(b, 10, big.NewInt(80), big.NewInt(4), crypto.SHA256)
    counts["32"] = count + 1
}
func BenchmarkEval32(b *testing.B) { benchEval(b, 10, big.NewInt(80),
    big.NewInt(4), crypto.SHA256) }
func BenchmarkConstrain32(b *testing.B) {
    benchConstrain(b, 10, big.NewInt(80), big.NewInt(4), crypto.SHA256)
}

func BenchmarkKeyGen40(b *testing.B) {
    if counts == nil {
        counts["40"] = 0
    }
    count := counts["40"]
    printBanner(128, 128, 2, count, b)
    b.ResetTimer()
    benchKeyGen(b, 16, big.NewInt(128), big.NewInt(2), crypto.SHA256)
    counts["40"] = count + 1
}
func BenchmarkEval40(b *testing.B) { benchEval(b, 16, big.NewInt(128),
    big.NewInt(2), crypto.SHA256) }
func BenchmarkConstrain40(b *testing.B) {
    benchConstrain(b, 16, big.NewInt(128), big.NewInt(2), crypto.SHA256)
}

func BenchmarkKeyGen41(b *testing.B) {
    if counts == nil {
        counts["41"] = 0
    }
    count := counts["41"]
    printBanner(128, 128, 3, count, b)
    b.ResetTimer()
    benchKeyGen(b, 16, big.NewInt(128), big.NewInt(3), crypto.SHA256)
    counts["41"] = count + 1
}
func BenchmarkEval41(b *testing.B) { benchEval(b, 16, big.NewInt(128),
    big.NewInt(3), crypto.SHA256) }
func BenchmarkConstrain41(b *testing.B) {
    benchConstrain(b, 16, big.NewInt(128), big.NewInt(3), crypto.SHA256)
}

func printBanner(lambda, ell, r, count int, b *testing.B) {
    if count < 1 {
        log.Printf("\n")
        log.Printf("lambda = %v; ell = %v; r = %v; \n", lambda, ell, r)
        log.Printf("------------------------------------------\n")
        log.Printf("\n")
    }
}
```

Package `prf`

Package `prf` contains the source code for the underlying PRF mechanism that we use in our construction. This code is very simple, using the `Go` native `crypto/hmac` package to implement an `hmac` PRF using the SHA-256 hash function. As we mentioned before, in the random oracle model this is a PRF. The source code follows below.

PRF.GO

```go
package prf

import (
    "crypto"
    "crypto/hmac"
    _ "crypto/sha256"
    "hash"
)

type PRF struct {
    F hash.Hash
}

func InitPRF(h crypto.Hash, k []byte) *PRF {
    hmac := hmac.New(h.New, k)
    return &PRF{F: hmac}
}

func (prf *PRF) Eval(x []byte) []byte {
    F := prf.F
    return F.Sum(x)
}
```

Package util

Package util contains all of the source code for utility functions that are used globally by the cprf package. The source code is given below.

utils.go

```go
package util

import (
    "math/big"
)

// Finds the number of unique indices in an integer array
func Unique(n []*big.Int) int {
    z := 1
    for i := 0; i < len(n)-1; i++ {
        if n[i].Cmp(n[i+1]) == 1 {
            z++
        }
    }
    return z
}

// generate viable indices for PRF keys
func IndexGen(l, r *big.Int, cprfKeyLen *big.Int) [][]*big.Int {
    arr := make([][]*big.Int, cprfKeyLen.Int64())
    rem := new(big.Int).Sub(cprfKeyLen, big.NewInt(1))
    rem64 := rem.Int64()
    for rem64 >= 0 {
        arr[rem64] = initZeroBigIntSlice(r.Int64())
        newRem := rem
        for i := r.Int64() - 1; i >= 0; i-- {
            div := new(big.Int).Exp(l, big.NewInt(i), nil)
            arr[rem64][i] = new(big.Int).Div(newRem, div)
            newRem = new(big.Int).Mod(newRem, div)
            if newRem.Cmp(big.NewInt(0)) == 0 {
                break
            }
        }
        arr[rem64] = quickSort(arr[rem64])
        rem = new(big.Int).Sub(rem, big.NewInt(1))
        rem64--
    }
    arrDedup := dedupIntSlice(arr, r.Int64())
    return arrDedup
}
```

```go

// Takes an array of integers and joins into a string
// Slices are assumed to be ordered
// pipes are used to separate indices in the string
func JoinBigIntSlice(s []*big.Int) string {
    res := ""
    for i := 0; i < len(s); i++ {
        c := s[i].String()
        res += c
        if i < len(s)-1 {
            res += "|"
        }
    }
    return res
}

func quickSort(data []*big.Int) []*big.Int {
    if len(data) > 1 {
        pivot := data[0]
        smaller := make([]*big.Int, 0, len(data))
        larger := make([]*big.Int, 0, len(data))
        equal := make([]*big.Int, 1, len(data))
        equal[0] = pivot
        for i := 1; i < len(data); i++ {
            if data[i].Cmp(pivot) == -1 {
                larger = append(larger, data[i])
            } else if data[i].Cmp(pivot) == 1 {
                smaller = append(smaller, data[i])
            } else if data[i].Cmp(pivot) == 0 {
                equal = append(equal, data[i])
            }
        }
        return append(append(quickSort(smaller), equal...),
            quickSort(larger)...)
    } else {
        return data
    }
}

// Deduplicate slices in a overarching slice
func dedupIntSlice(arr [][]*big.Int, r int64) [][]*big.Int {
    added := make([][]*big.Int, 0, len(arr))
    for i := 0; i < len(arr); i++ {
        slice := arr[i]
        foundSame := false
        for j := 0; j < len(added); j++ {
            if j == i {
                continue
            }

            if len(slice) != len(arr[j]) {
                continue
            }

            same := false
            for k := 0; k < len(added[j]); k++ {
                same = true
                if slice[k].Cmp(added[j][k]) != 0 {
                    same = false
                    break
                }
            }

            if same {
                foundSame = true
                break
            }
        }

        if !foundSame {
            added = append(added, arr[i])
        }
    }

    return added
}

func initZeroBigIntSlice(length int64) []*big.Int {
    s := make([]*big.Int, length)
    for i := 0; i < len(s); i++ {
        s[i] = big.NewInt(0)
    }
    return s
}
```

NOTATION	DESCRIPTION
λ	Security parameter
$[n], [n_1, n_2]$	$\{1, \ldots, n\}, \{n_1, \ldots, n_2\}$
$x_i; x \in \{0,1\}^*$	i^{th} bit in x
$x\vert_t; x \in \{0,1\}^\ell$	$x_t \cdots x_\ell$
$x\vert^t; x \in \{0,1\}^\ell$	$x_1 \cdots x_t$
$x\vert_T; x \in \{0,1\}^\ell; T = (t_i)_{i \in [m]}$	$x_{t_1} \cdots x_{t_m}$
$\mathsf{bitlength}(x)$	Bit length of $x \in \{0,1\}^*$
$S.\mathsf{pop}()$	Return $S[0]$ and move $S[i] \to S[i-1]$
$S.\mathsf{append}(x)$	Concatenate x to S
$\lvert S \rvert$	Cardinality of S
$x \stackrel{?}{=} y$	Returns 1 if x is equal to y; 0 otherwise
$\mathcal{A}, \mathcal{B}, \mathcal{D}$	PPT adversarial algorithms
$y \leftarrow \mathcal{D}(x)$	$\mathcal{D}$ returns y on input x
$\mathsf{negl}(\lambda)$	Negligible function
$\mathcal{C}_{\mathsf{NC}^1}$	Log-depth circuit class
$\mathcal{C}_{\mathsf{P/poly}}$	Poly-depth circuit class
$\mathsf{exp}^{\mathsf{prop}}_{b,\mathcal{A}}(1^\lambda, \mathsf{sch})$	Decisional experiments for prop and a scheme sch
$\mathsf{exp}^{\mathsf{prop}}_{\mathcal{A}}(1^\lambda, \mathsf{sch})$	Computational experiments for prop and a scheme sch
$\max_{\mathcal{A}}(\mathsf{Adv}(\mathcal{A}, \mathsf{prop}(1^\lambda, \mathsf{sch})))$	Advantage of any $\mathcal{A}$ succeeding in distinguishing experiments $\mathsf{exp}^{\mathsf{prop}}_{b,\mathcal{A}}(1^\lambda, \mathsf{sch})$, for a scheme sch
$X \approx_c Y$	Computationally indistinguishable distributions X, Y
$X \approx_s Y$	Statistically indistinguishable distributions X, Y
$X \approx_p Y$	Perfectly indistinguishable distributions X, Y
$\mathsf{P}_C()$	Predicate for $C \in \mathcal{C}$
$\mathcal{O}_{\mathcal{Y}}(f(\cdot))$	Oracle for f, with domain $\mathcal{Y}$
$\mathcal{A}^{\mathcal{O}_{\mathcal{Y}}(f(\cdot))}$	$\mathcal{A}$ has oracle access to f
$\mathcal{A}^{\mathcal{O}_{\mathcal{Y}}(f(\cdot), \boldsymbol{\varphi})}$	Oracle access where queries are stored in $\boldsymbol{\varphi}$
$\mathcal{A}^{\mathcal{O}_{\mathcal{Y}}(f(\cdot); [r])}$	Oracle access with query upper bound of $r \in \mathbb{N}$
$\mathtt{C}$, $\mathtt{S}$	Client and server
$\mathtt{C}[x]$	Client holding input x
$\boldsymbol{\psi}^1(\mathtt{C}, x)$	Protocol step one computed by client, with input x
pp	Public Parameters
$(\mathsf{pk}, \mathsf{sk})$	Typical asymmetric key pair
msk	Master secret key
$\boldsymbol{A}$, $\boldsymbol{v}$	Matrix and vector notation
v_i	$v_i = \boldsymbol{v}[i]$
$\mathbb{G}$	Cyclic group
p, q	Reserved for group orders or ring moduli
$\phi(X)$	Cyclotomic polynomial
$\mathcal{R}$	$\mathbb{Z}[X]/(\phi(X))$
$\mathcal{R}_q$	$\mathbb{Z}_q[X]/(\phi(X))$

Table A: Summary of notation

Acronyms

AHE	Additively Homomorphic Encryption
BF	Bloom filter
BitFix	Bit-fixing
BPUA	Branching Program Unannihilatability Assumption
CDN	Content Delivery Network
DCRA	Decisional composite residuocity assumption
DDH	Decisional Diffie-Hellman
DL	Discrete log
DSS	Digital Signature Scheme
FHE	Fully Homomorphic Encryption
GES	Graded encoding scheme
HE	Homomorphic Encryption
ID-NIKE	Identity-based non-interactive key exchange
iff	if and only if
IO	Indistinguishability obfuscation
LR	Left-right
LWE	Learning with errors
MDDH	Multilinear DDH
MHE	Multiplicatively Homomorphic Encryption
MMAP	Multilinear map
OPE	Oblivious Polynomial Evaluation
OPRF	Oblivious Pseudorandom Function
OT	Oblivious transfer
OWF	One-way function
PKE	Public-key Encryption
PPT	Probabilistic Polynomial Time
PRF	Pseudorandom Function
PSI	Private Set Intersection
PSI-CA	Private Set Intersection Cardinality
PSO	Private Set Operation
PSU	Private Set Union

PSU-CA	Private Set Union Cardinality
PSU/I-CA	Private Set Union/Intersection Cardinality
RLWE	Ring learning with errors
ROM	Random oracle model
RTT	Round trip time
s.t.	such that
SHE	Somewhat Homomorphic Encryption
SKE	Symmetric-key Encryption
VBB	Virtual black-box
VOPRF	Verifiable Oblivious Pseudorandom Function
whp	with high probability
wlog	without loss of generality
wrt	with respect to

Bibliography

[1] M. Abdalla, F. Benhamouda, and A. Passelègue. *Algebraic XOR-RKA-Secure Pseudorandom Functions from Post-Zeroizing Multilinear Maps*. Cryptology ePrint Archive, Report 2017/500. http://eprint.iacr.org/2017/500. 2017.

[2] S. Agrawal. *New Methods for Indistinguishability Obfuscation: Bootstrapping and Instantiation*. Cryptology ePrint Archive, Report 2018/633. https://eprint.iacr.org/2018/633. 2018.

[3] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. "CAPTCHA: Using Hard AI Problems for Security". In: *EUROCRYPT 2003*. Ed. by E. Biham. Vol. 2656. LNCS. Springer, Heidelberg, May 2003, pp. 294–311. doi: 10.1007/3-540-39200-9_18.

[4] M. Ajtai. "Generating Hard Instances of Lattice Problems (Extended Abstract)". In: *28th ACM STOC*. ACM Press, May 1996, pp. 99–108. doi: 10.1145/237814.237838.

[5] M. R. Albrecht, S. Bai, and L. Ducas. "A Subfield Lattice Attack on Overstretched NTRU Assumptions - Cryptanalysis of Some FHE and Graded Encoding Schemes". In: *CRYPTO 2016, Part I*. Ed. by M. Robshaw and J. Katz. Vol. 9814. LNCS. Springer, Heidelberg, Aug. 2016, pp. 153–178. doi: 10.1007/978-3-662-53018-4_6.

[6] M. R. Albrecht, C. Cocis, F. Laguillaumie, and A. Langlois. "Implementing Candidate Graded Encoding Schemes from Ideal Lattices". In: *ASIACRYPT 2015, Part II*. Ed. by T. Iwata and J. H. Cheon. Vol. 9453. LNCS. Springer, Heidelberg, Nov. 2015, pp. 752–775. doi: 10.1007/978-3-662-48800-3_31.

[7] M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. W. Postlethwaite, F. Virdia, and T. Wunderer. "Estimate All the {LWE, NTRU} Schemes!" In: *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*. 2018, pp. 351–367. doi: 10.1007/978-3-319-98113-0_19. url: https://doi.org/10.1007/978-3-319-98113-0%5C_19.

[8] M. R. Albrecht, A. Davidson, and E. Larraia. "Notes on GGH13 Without the Presence of Ideals". In: *16th IMA International Conference on Cryptography and Coding*. Ed. by M. O'Neill. Vol. 10655. LNCS. Springer, Heidelberg, Dec. 2017, pp. 135–158.

[9] M. R. Albrecht, A. Davidson, E. Larraia, and A. Pellet–Mary. *Notes On GGH13 Without The Presence Of Ideals*. Cryptology ePrint Archive, Report 2017/906. http://eprint.iacr.org/2017/906. 2017.

[10] M. R. Albrecht, P. Farshim, D. Hofheinz, E. Larraia, and K. G. Paterson. "Multilinear Maps from Obfuscation". In: *TCC 2016-A, Part I*. Ed. by E. Kushilevitz and T. Malkin. Vol. 9562. LNCS. Springer, Heidelberg, Jan. 2016, pp. 446–473. DOI: 10.1007/978-3-662-49096-9_19.

[11] M. R. Albrecht, R. Player, and S. Scott. "On the concrete hardness of Learning with Errors". In: *J. Mathematical Cryptology* 9.3 (2015), pp. 169–203. URL: http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml.

[12] P. V. Ananth, D. Gupta, Y. Ishai, and A. Sahai. "Optimizing Obfuscation: Avoiding Barrington's Theorem". In: *ACM CCS 14*. Ed. by G.-J. Ahn, M. Yung, and N. Li. ACM Press, Nov. 2014, pp. 646–658. DOI: 10.1145/2660267.2660342.

[13] P. Ananth, A. Jain, D. Khurana, and A. Sahai. *Indistinguishability Obfuscation Without Multilinear Maps: iO from LWE, Bilinear Maps, and Weak Pseudorandomness*. Cryptology ePrint Archive, Report 2018/615. https://eprint.iacr.org/2018/615. 2018.

[14] P. Ananth and A. Sahai. "Projective Arithmetic Functional Encryption and Indistinguishability Obfuscation from Degree-5 Multilinear Maps". In: *EUROCRYPT 2017, Part I*. Ed. by J. Coron and J. B. Nielsen. Vol. 10210. LNCS. Springer, Heidelberg, Apr. 2017, pp. 152–181. DOI: 10.1007/978-3-319-56620-7_6.

[15] D. Apon, N. Döttling, S. Garg, and P. Mukherjee. "Cryptanalysis of Indistinguishability Obfuscations of Circuits over GGH13". In: *ICALP 2017*. Ed. by I. Chatzigiannakis, P. Indyk, F. Kuhn, and A. Muscholl. Vol. 80. LIPIcs. Schloss Dagstuhl, July 2017, 38:1–38:16. DOI: 10.4230/LIPIcs.ICALP.2017.38.

[16] B. Applebaum and Z. Brakerski. "Obfuscating Circuits via Composite-Order Graded Encoding". In: *TCC 2015, Part II*. Ed. by Y. Dodis and J. B. Nielsen. Vol. 9015. LNCS. Springer, Heidelberg, Mar. 2015, pp. 528–556. DOI: 10.1007/978-3-662-46497-7_21.

[17] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. "High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority". In: *ACM CCS 16*. Ed. by E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi. ACM Press, Oct. 2016, pp. 805–817. DOI: 10.1145/2976749.2978331.

[18] G. Asharov, S. Halevi, Y. Lindell, and T. Rabin. "Privacy-Preserving Search of Similar Patients in Genomic Data". In: *PoPETs* 2018.4 (2018), pp. 104–124. DOI: 10.1515/popets-2018-0026. URL: https://doi.org/10.1515/popets-2018-0034.

[19] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. "More Efficient Oblivious Transfer Extensions". In: *Journal of Cryptology* 30.3 (2017), pp. 805–858.

[20] N. Attrapadung, T. Matsuda, R. Nishimaki, S. Yamada, and T. Yamakawa. "Constrained PRFs for NC^1 in Traditional Groups". In: *CRYPTO 2018, Part II*. Ed. by H. Shacham and A. Boldyreva. Vol. 10992. LNCS. Springer, Heidelberg, Aug. 2018, pp. 543–574. DOI: 10.1007/978-3-319-96881-0_19.

[21] N. Attrapadung, T. Matsuda, R. Nishimaki, S. Yamada, and T. Yamakawa. *Constrained PRFs for NC1 in Traditional Groups*. Cryptology ePrint Archive, Report 2018/154. https://eprint.iacr.org/2018/154. 2018.

[22] S. Badrinarayanan, V. Goyal, A. Jain, and A. Sahai. "Verifiable Functional Encryption". In: *ASIACRYPT 2016, Part II*. Ed. by J. H. Cheon and T. Takagi. Vol. 10032. LNCS. Springer, Heidelberg, Dec. 2016, pp. 557–587. DOI: 10.1007/978-3-662-53890-6_19.

[23] S. Badrinarayanan, E. Miles, A. Sahai, and M. Zhandry. "Post-zeroizing Obfuscation: New Mathematical Tools, and the Case of Evasive Circuits". In: *EUROCRYPT 2016, Part II*. Ed. by M. Fischlin and J.-S. Coron. Vol. 9666. LNCS. Springer, Heidelberg, May 2016, pp. 764–791. DOI: 10.1007/978-3-662-49896-5_27.

[24] A. Banerjee, G. Fuchsbauer, C. Peikert, K. Pietrzak, and S. Stevens. "Key-Homomorphic Constrained Pseudorandom Functions". In: *TCC 2015, Part II*. Ed. by Y. Dodis and J. B. Nielsen. Vol. 9015. LNCS. Springer, Heidelberg, Mar. 2015, pp. 31–60. DOI: 10.1007/978-3-662-46497-7_2.

[25] A. Banerjee, C. Peikert, and A. Rosen. "Pseudorandom Functions and Lattices". In: *EUROCRYPT 2012*. Ed. by D. Pointcheval and T. Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 719–737. DOI: 10.1007/978-3-642-29011-4_42.

[26] B. Barak, N. Bitansky, R. Canetti, Y. T. Kalai, O. Paneth, and A. Sahai. "Obfuscation for Evasive Functions". In: *TCC 2014*. Ed. by Y. Lindell. Vol. 8349. LNCS. Springer, Heidelberg, Feb. 2014, pp. 26–51. DOI: 10.1007/978-3-642-54242-8_2.

[27] B. Barak, S. Garg, Y. T. Kalai, O. Paneth, and A. Sahai. "Protecting Obfuscation against Algebraic Attacks". In: *EUROCRYPT 2014*. Ed. by P. Q. Nguyen and E. Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 221–238. DOI: 10.1007/978-3-642-55220-5_13.

[28] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. "On the (im)possibility of obfuscating programs". In: *J. ACM* 59.2 (2012), 6:1–6:48.

[29] D. A. M. Barrington. "Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1". In: *J. Comput. Syst. Sci.* 38.1 (1989), pp. 150–164. DOI: 10.1016/0022-0000(89)90037-8. URL: http://dx.doi.org/10.1016/0022-0000(89)90037-8.

[30] J. Bartusek, J. Guan, F. Ma, and M. Zhandry. *Preventing Zeroizing Attacks on GGH15*. Cryptology ePrint Archive, Report 2018/511. https://eprint.iacr.org/2018/511. 2018.

[31] W. Baur and V. Strassen. “The Complexity of Partial Derivatives”. In: *Theor. Comput. Sci.* 22 (1983), pp. 317–330. DOI: 10.1016/0304-3975(83)90110-X. URL: http://dx.doi.org/10.1016/0304-3975(83)90110-X.

[32] D. Beaver. “Precomputing Oblivious Transfer”. In: *CRYPTO’95*. Ed. by D. Coppersmith. Vol. 963. LNCS. Springer, Heidelberg, Aug. 1995, pp. 97–109. DOI: 10.1007/3-540-44750-4_8.

[33] M. Bellare, R. Canetti, and H. Krawczyk. “Keying Hash Functions for Message Authentication”. In: *CRYPTO’96*. Ed. by N. Koblitz. Vol. 1109. LNCS. Springer, Heidelberg, Aug. 1996, pp. 1–15. DOI: 10.1007/3-540-68697-5_1.

[34] M. Bellare, J. A. Garay, and T. Rabin. “Fast Batch Verification for Modular Exponentiation and Digital Signatures”. In: *EUROCRYPT’98*. Ed. by K. Nyberg. Vol. 1403. LNCS. Springer, Heidelberg, May 1998, pp. 236–250. DOI: 10.1007/BFb0054130.

[35] M. Bellare and P. Rogaway. “Random Oracles are Practical: A Paradigm for Designing Efficient Protocols”. In: *ACM CCS 93*. Ed. by V. Ashby. ACM Press, Nov. 1993, pp. 62–73. DOI: 10.1145/168588.168596.

[36] M. Bellare and P. Rogaway. “The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs”. In: *EUROCRYPT 2006*. Ed. by S. Vaudenay. Vol. 4004. LNCS. Springer, Heidelberg, May 2006, pp. 409–426. DOI: 10.1007/11761679_25.

[37] A. Ben-Efraim, Y. Lindell, and E. Omri. “Optimizing Semi-Honest Secure Multiparty Computation for the Internet”. In: *ACM CCS 16*. Ed. by E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi. ACM Press, Oct. 2016, pp. 578–590. DOI: 10.1145/2976749.2978347.

[38] M. Ben-Or, S. Goldwasser, and A. Wigderson. “Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)”. In: *20th ACM STOC*. ACM Press, May 1988, pp. 1–10. DOI: 10.1145/62212.62213.

[39] J. Benaloh. “Dense Probabilistic Encryption”. In: *In Proceedings of the Workshop on Selected Areas of Cryptography*. 1994, pp. 120–128.

[40] *Benchmarking Paillier Encryption*. https://medium.com/snips-ai/benchmarking-paillier-encryption-15631a0b5ad8. Accessed: 2018-05-25.

[41] D. J. Bernstein. “Curve25519: New Diffie-Hellman Speed Records”. In: *PKC 2006*. Ed. by M. Yung, Y. Dodis, A. Kiayias, and T. Malkin. Vol. 3958. LNCS. Springer, Heidelberg, Apr. 2006, pp. 207–228. DOI: 10.1007/11745853_14.

[42] J.-F. Biasse, T. Espitau, P.-A. Fouque, A. Gélin, and P. Kirchner. “Computing Generator in Cyclotomic Integer Rings - A Subfield Algorithm for the Principal Ideal Problem in $L_{|\Delta_{\mathbb{K}}|}(\frac{1}{2})$ and Application to the Cryptanalysis of a FHE Scheme”. In: *EUROCRYPT 2017, Part I*. Ed. by J. Coron and J. B. Nielsen. Vol. 10210. LNCS. Springer, Heidelberg, Apr. 2017, pp. 60–88. DOI: 10.1007/978-3-319-56620-7_3.

[43] A. Bishop, L. Kowalczyk, T. Malkin, V. Pastro, M. Raykova, and K. Shi. *A Simple Obfuscation Scheme for Pattern-Matching with Wildcards*. Cryptology ePrint Archive, Report 2018/210. `https://eprint.iacr.org/2018/210`. 2018.

[44] A. Bishop, L. Kowalczyk, T. Malkin, V. Pastro, M. Raykova, and K. Shi. "A Simple Obfuscation Scheme for Pattern-Matching with Wildcards". In: *CRYPTO 2018, Part III*. Ed. by H. Shacham and A. Boldyreva. Vol. 10993. LNCS. Springer, Heidelberg, Aug. 2018, pp. 731–752. DOI: `10.1007/978-3-319-96878-0_25`.

[45] N. Bitansky. "Verifiable Random Functions from Non-interactive Witness-Indistinguishable Proofs". In: *TCC 2017, Part II*. Ed. by Y. Kalai and L. Reyzin. Vol. 10678. LNCS. Springer, Heidelberg, Nov. 2017, pp. 567–594. DOI: `10.1007/978-3-319-70503-3_19`.

[46] N. Bitansky, O. Paneth, and D. Wichs. "Perfect Structure on the Edge of Chaos - Trapdoor Permutations from Indistinguishability Obfuscation". In: *TCC 2016-A, Part I*. Ed. by E. Kushilevitz and T. Malkin. Vol. 9562. LNCS. Springer, Heidelberg, Jan. 2016, pp. 474–502. DOI: `10.1007/978-3-662-49096-9_20`.

[47] M. Blanton and E. Aguiar. "Private and oblivious set and multiset operations". In: *ASIACCS 12*. Ed. by H. Y. Youm and Y. Won. ACM Press, May 2012, pp. 40–41.

[48] J. Blocki and H.-S. Zhou. "Designing Proof of Human-Work Puzzles for Cryptocurrency and Beyond". In: *TCC 2016-B, Part II*. Ed. by M. Hirt and A. D. Smith. Vol. 9986. LNCS. Springer, Heidelberg, Oct. 2016, pp. 517–546. DOI: `10.1007/978-3-662-53644-5_20`.

[49] B. H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors". In: *Commun. ACM* 13.7 (1970), pp. 422–426.

[50] M. Blum, P. Feldman, and S. Micali. "Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract)". In: *20th ACM STOC*. ACM Press, May 1988, pp. 103–112. DOI: `10.1145/62212.62222`.

[51] D. Boneh, E.-J. Goh, and K. Nissim. "Evaluating 2-DNF Formulas on Ciphertexts". In: *TCC 2005*. Ed. by J. Kilian. Vol. 3378. LNCS. Springer, Heidelberg, Feb. 2005, pp. 325–341. DOI: `10.1007/978-3-540-30576-7_18`.

[52] D. Boneh, S. Kim, and D. J. Wu. "Constrained Keys for Invertible Pseudorandom Functions". In: *TCC 2017, Part I*. Ed. by Y. Kalai and L. Reyzin. Vol. 10677. LNCS. Springer, Heidelberg, Nov. 2017, pp. 237–263. DOI: `10.1007/978-3-319-70500-2_9`.

[53] D. Boneh, K. Lewi, H. W. Montgomery, and A. Raghunathan. "Key Homomorphic PRFs and Their Applications". In: *CRYPTO 2013, Part I*. Ed. by R. Canetti and J. A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 410–428. DOI: `10.1007/978-3-642-40041-4_23`.

[54] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman. “Semantically Secure Order-Revealing Encryption: Multi-input Functional Encryption Without Obfuscation”. In: *EUROCRYPT 2015, Part II*. Ed. by E. Oswald and M. Fischlin. Vol. 9057. LNCS. Springer, Heidelberg, Apr. 2015, pp. 563–594. DOI: 10.1007/978-3-662-46803-6_19.

[55] D. Boneh, K. Lewi, and D. J. Wu. “Constraining Pseudorandom Functions Privately”. In: *PKC 2017, Part II*. Ed. by S. Fehr. Vol. 10175. LNCS. Springer, Heidelberg, Mar. 2017, pp. 494–524. DOI: 10.1007/978-3-662-54388-7_17.

[56] D. Boneh, B. Lynn, and H. Shacham. “Short Signatures from the Weil Pairing”. In: *ASIACRYPT 2001*. Ed. by C. Boyd. Vol. 2248. LNCS. Springer, Heidelberg, Dec. 2001, pp. 514–532. DOI: 10.1007/3-540-45682-1_30.

[57] D. Boneh and A. Silverberg. “Applications of Multilinear Forms to Cryptography”. In: *Contemporary Mathematics* 324 (2003), pp. 71–90.

[58] D. Boneh and B. Waters. “Constrained Pseudorandom Functions and Their Applications”. In: *ASIACRYPT 2013, Part II*. Ed. by K. Sako and P. Sarkar. Vol. 8270. LNCS. Springer, Heidelberg, Dec. 2013, pp. 280–300. DOI: 10.1007/978-3-642-42045-0_15.

[59] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. H. M. Smid, and Y. Tang. “On the false-positive rate of Bloom filters”. In: *Inf. Process. Lett.* 108.4 (2008), pp. 210–213.

[60] E. Boyle, S. Goldwasser, and I. Ivan. “Functional Signatures and Pseudorandom Functions”. In: *PKC 2014*. Ed. by H. Krawczyk. Vol. 8383. LNCS. Springer, Heidelberg, Mar. 2014, pp. 501–519. DOI: 10.1007/978-3-642-54631-0_29.

[61] Z. Brakerski. “Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP”. In: *CRYPTO 2012*. Ed. by R. Safavi-Naini and R. Canetti. Vol. 7417. LNCS. Springer, Heidelberg, Aug. 2012, pp. 868–886. DOI: 10.1007/978-3-642-32009-5_50.

[62] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. “(Leveled) fully homomorphic encryption without bootstrapping”. In: *ITCS 2012*. Ed. by S. Goldwasser. ACM, Jan. 2012, pp. 309–325. DOI: 10.1145/2090236.2090262.

[63] Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé. “Classical hardness of learning with errors”. In: *45th ACM STOC*. Ed. by D. Boneh, T. Roughgarden, and J. Feigenbaum. ACM Press, June 2013, pp. 575–584. DOI: 10.1145/2488608.2488680.

[64] Z. Brakerski and G. N. Rothblum. “Black-box obfuscation for d-CNFs”. In: *ITCS 2014*. Ed. by M. Naor. ACM, Jan. 2014, pp. 235–250. DOI: 10.1145/2554797.2554820.

[65] Z. Brakerski and G. N. Rothblum. “Obfuscating Conjunctions”. In: *CRYPTO 2013, Part II*. Ed. by R. Canetti and J. A. Garay. Vol. 8043. LNCS. Springer, Heidelberg, Aug. 2013, pp. 416–434. DOI: 10.1007/978-3-642-40084-1_24.

[66] Z. Brakerski and G. N. Rothblum. “Virtual Black-Box Obfuscation for All Circuits via Generic Graded Encoding”. In: *TCC 2014*. Ed. by Y. Lindell. Vol. 8349. LNCS. Springer, Heidelberg, Feb. 2014, pp. 1–25. DOI: 10.1007/978-3-642-54242-8_1.

[67] Z. Brakerski, R. Tsabary, V. Vaikuntanathan, and H. Wee. “Private Constrained PRFs (and More) from LWE”. In: *TCC 2017, Part I*. Ed. by Y. Kalai and L. Reyzin. Vol. 10677. LNCS. Springer, Heidelberg, Nov. 2017, pp. 264–302. DOI: 10.1007/978-3-319-70500-2_10.

[68] Z. Brakerski and V. Vaikuntanathan. “Constrained Key-Homomorphic PRFs from Standard Lattice Assumptions - Or: How to Secretly Embed a Circuit in Your PRF”. In: *TCC 2015, Part II*. Ed. by Y. Dodis and J. B. Nielsen. Vol. 9015. LNCS. Springer, Heidelberg, Mar. 2015, pp. 1–30. DOI: 10.1007/978-3-662-46497-7_1.

[69] Z. Brakerski and V. Vaikuntanathan. “Efficient Fully Homomorphic Encryption from (Standard) LWE”. In: *52nd FOCS*. Ed. by R. Ostrovsky. IEEE Computer Society Press, Oct. 2011, pp. 97–106. DOI: 10.1109/FOCS.2011.12.

[70] Z. Brakerski, V. Vaikuntanathan, H. Wee, and D. Wichs. “Obfuscating Conjunctions under Entropic Ring LWE”. In: *ITCS 2016*. Ed. by M. Sudan. ACM, Jan. 2016, pp. 147–156. DOI: 10.1145/2840728.2840764.

[71] J. Brickell and V. Shmatikov. “Privacy-Preserving Graph Algorithms in the Semi-honest Model”. In: *ASIACRYPT 2005*. Ed. by B. K. Roy. Vol. 3788. LNCS. Springer, Heidelberg, Dec. 2005, pp. 236–252. DOI: 10.1007/11593447_13.

[72] J. Burns, D. Moore, K. Ray, R. Speers, and B. Vohaska. *EC-OPRF: Oblivious Pseudorandom Functions using Elliptic Curves*. Cryptology ePrint Archive, Report 2017/111. http://eprint.iacr.org/2017/111. 2017.

[73] J. Camenisch and G. M. Zaverucha. “Private Intersection of Certified Sets”. In: *FC 2009*. Ed. by R. Dingledine and P. Golle. Vol. 5628. LNCS. Springer, Heidelberg, Feb. 2009, pp. 108–127.

[74] R. Canetti. “Towards Realizing Random Oracles: Hash Functions That Hide All Partial Information”. In: *CRYPTO’97*. Ed. by B. S. Kaliski Jr. Vol. 1294. LNCS. Springer, Heidelberg, Aug. 1997, pp. 455–469. DOI: 10.1007/BFb0052255.

[75] R. Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols”. In: *42nd FOCS*. IEEE Computer Society Press, Oct. 2001, pp. 136–145. DOI: 10.1109/SFCS.2001.959888.

[76] R. Canetti and Y. Chen. “Constraint-Hiding Constrained PRFs for NC^1 from LWE”. In: *EUROCRYPT 2017, Part I*. Ed. by J. Coron and J. B. Nielsen. Vol. 10210. LNCS. Springer, Heidelberg, Apr. 2017, pp. 446–476. DOI: 10.1007/978-3-319-56620-7_16.

[77] R. Canetti, G. N. Rothblum, and M. Varia. “Obfuscation of Hyperplane Membership”. In: *TCC 2010*. Ed. by D. Micciancio. Vol. 5978. LNCS. Springer, Heidelberg, Feb. 2010, pp. 72–89. DOI: 10.1007/978-3-642-11799-2_5.

[78] W. Castryck, I. Iliashenko, and F. Vercauteren. “Provably Weak Instances of Ring-LWE Revisited”. In: *EUROCRYPT 2016, Part I*. Ed. by M. Fischlin and J.-S. Coron. Vol. 9665. LNCS. Springer, Heidelberg, May 2016, pp. 147–167. DOI: 10.1007/978-3-662-49890-3_6.

[79] H. Cavusoglu, B. Mishra, and S. Raghunathan. “The effect of internet security breach announcements on market value”. In: *Intl. J. of Electronic Commerce* 9.1 (2004), pp. 70–104.

[80] A. Cerulli, E. D. Cristofaro, and C. Soriente. “Nothing Refreshes Like a RePSI: Reactive Private Set Intersection”. In: *ACNS 18*. Ed. by B. Preneel and F. Vercauteren. Vol. 10892. LNCS. Springer, Heidelberg, July 2018, pp. 280–300. DOI: 10.1007/978-3-319-93387-0_15.

[81] D. Chaum. “Blind Signature System”. In: *CRYPTO’83*. Ed. by D. Chaum. Plenum Press, New York, USA, 1983, p. 153.

[82] D. Chaum. “Blind Signatures for Untraceable Payments”. In: *CRYPTO’82*. Ed. by D. Chaum, R. L. Rivest, and A. T. Sherman. Plenum Press, New York, USA, 1982, pp. 199–203.

[83] D. Chaum, C. Crépeau, and I. Damgård. “Multiparty Unconditionally Secure Protocols (Extended Abstract)”. In: *20th ACM STOC*. ACM Press, May 1988, pp. 11–19. DOI: 10.1145/62212.62214.

[84] D. Chaum, I. Damgård, and J. van de Graaf. “Multiparty Computations Ensuring Privacy of Each Party’s Input and Correctness of the Result”. In: *CRYPTO’87*. Ed. by C. Pomerance. Vol. 293. LNCS. Springer, Heidelberg, Aug. 1988, pp. 87–119. DOI: 10.1007/3-540-48184-2_7.

[85] D. Chaum, A. Fiat, and M. Naor. “Untraceable Electronic Cash”. In: *CRYPTO’88*. Ed. by S. Goldwasser. Vol. 403. LNCS. Springer, Heidelberg, Aug. 1990, pp. 319–327. DOI: 10.1007/0-387-34799-2_25.

[86] H. Chen, K. Laine, and P. Rindal. “Fast Private Set Intersection from Homomorphic Encryption”. In: *ACM CCS 17*. Ed. by B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu. ACM Press, Oct. 2017, pp. 1243–1255. DOI: 10.1145/3133956.3134061.

[87] Y. Chen, C. Gentry, and S. Halevi. “Cryptanalyses of Candidate Branching Program Obfuscators”. In: *EUROCRYPT 2017, Part III*. Ed. by J. Coron and J. B. Nielsen. Vol. 10212. LNCS. Springer, Heidelberg, Apr. 2017, pp. 278–307. DOI: 10.1007/978-3-319-56617-7_10.

[88] Y. Chen, V. Vaikuntanathan, and H. Wee. “GGH15 Beyond Permutation Branching Programs: Proofs, Attacks, and Candidates”. In: *CRYPTO 2018, Part II*. Ed. by H. Shacham and A. Boldyreva. Vol. 10992. LNCS. Springer, Heidelberg, Aug. 2018, pp. 577–607. DOI: 10.1007/978-3-319-96881-0_20.

[89] J. H. Cheon, K. Han, C. Lee, H. Ryu, and D. Stehlé. “Cryptanalysis of the Multilinear Map over the Integers”. In: *EUROCRYPT 2015, Part I*. Ed. by E. Oswald and M. Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 3–12. DOI: 10.1007/978-3-662-46800-5_1.

[90] J. H. Cheon, M. Hhan, J. Kim, and C. Lee. *Cryptanalyses of Branching Program Obfuscations over GGH13 Multilinear Map from the NTRU Problem*. Cryptology ePrint Archive, Report 2018/408. https://eprint.iacr.org/2018/408. 2018.

[91] J. H. Cheon, J. Jeong, and C. Lee. *An Algorithm for NTRU Problems and Cryptanalysis of the GGH Multilinear Map without a Low Level Encoding of Zero*. Cryptology ePrint Archive, Report 2016/139. http://eprint.iacr.org/2016/139. 2016.

[92] K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell, and A. Nof. *Fast Large-Scale Honest-Majority MPC for Malicious Adversaries*. Cryptology ePrint Archive, Report 2018/570. https://eprint.iacr.org/2018/570. 2018.

[93] T. Chou and C. Orlandi. “The Simplest Protocol for Oblivious Transfer”. In: *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*. 2015, pp. 40–58. DOI: 10.1007/978-3-319-22174-8_3. URL: https://doi.org/10.1007/978-3-319-22174-8_3.

[94] G. Chunsheng. *Multilinear Maps Using a Variant of Ring-LWE*. Cryptology ePrint Archive, Report 2017/342. http://eprint.iacr.org/2017/342. 2017.

[95] Cisco. *Cisco Talos Intelligence*. https://talosintelligence.com/. Accessed Jul 2018. 2018.

[96] Cisco. *The Zettabyte Era: Trends and Analysis*. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html. Accessed Sep 2017. 2017.

[97] Citrix. *Citrix IP reputation*. https://docs.citrix.com/en-us/netscaler/11/security/reputation/ip-reputation.html. Accessed Jul 2018. 2018.

[98] Cloudflare. Personal communication. 2017.

[99] Cloudflare. *Cloudflare case studies*. Website. https://www.cloudflare.com/case-studies/. Accessed Nov 2017. 2017.

[100] A. Cohen, J. Holmgren, R. Nishimaki, V. Vaikuntanathan, and D. Wichs. "Watermarking cryptographic capabilities". In: *48th ACM STOC*. Ed. by D. Wichs and Y. Mansour. ACM Press, June 2016, pp. 1115–1127. DOI: 10.1145/2897518.2897651.

[101] J.-S. Coron, C. Gentry, S. Halevi, T. Lepoint, H. K. Maji, E. Miles, M. Raykova, A. Sahai, and M. Tibouchi. "Zeroizing Without Low-Level Zeroes: New MMAP Attacks and their Limitations". In: *CRYPTO 2015, Part I*. Ed. by R. Gennaro and M. J. B. Robshaw. Vol. 9215. LNCS. Springer, Heidelberg, Aug. 2015, pp. 247–266. DOI: 10.1007/978-3-662-47989-6_12.

[102] J.-S. Coron, M. S. Lee, T. Lepoint, and M. Tibouchi. "Cryptanalysis of GGH15 Multilinear Maps". In: *CRYPTO 2016, Part II*. Ed. by M. Robshaw and J. Katz. Vol. 9815. LNCS. Springer, Heidelberg, Aug. 2016, pp. 607–628. DOI: 10.1007/978-3-662-53008-5_21.

[103] J.-S. Coron, M. S. Lee, T. Lepoint, and M. Tibouchi. "Zeroizing Attacks on Indistinguishability Obfuscation over CLT13". In: *PKC 2017, Part I*. Ed. by S. Fehr. Vol. 10174. LNCS. Springer, Heidelberg, Mar. 2017, pp. 41–58. DOI: 10.1007/978-3-662-54365-8_3.

[104] J.-S. Coron, T. Lepoint, and M. Tibouchi. "Practical Multilinear Maps over the Integers". In: *CRYPTO 2013, Part I*. Ed. by R. Canetti and J. A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 476–493. DOI: 10.1007/978-3-642-40041-4_26.

[105] J.-S. Coron, T. Lepoint, and M. Tibouchi. "Scale-Invariant Fully Homomorphic Encryption over the Integers". In: *PKC 2014*. Ed. by H. Krawczyk. Vol. 8383. LNCS. Springer, Heidelberg, Mar. 2014, pp. 311–328. DOI: 10.1007/978-3-642-54631-0_18.

[106] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing. *SPDZ2k: Efficient MPC mod 2^k for Dishonest Majority*. Cryptology ePrint Archive, Report 2018/482. https://eprint.iacr.org/2018/482. 2018.

[107] R. Cramer, I. Damgård, and J. B. Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015. ISBN: 9781107043053. URL: http://www.cambridge.org/de/academic/subjects/computer-science/cryptography-cryptology-and-coding/secure-multiparty-computation-and-secret-sharing?format=HB&isbn=9781107043053.

[108] R. Cramer, L. Ducas, C. Peikert, and O. Regev. "Recovering Short Generators of Principal Ideals in Cyclotomic Rings". In: *EUROCRYPT 2016, Part II*. Ed. by M. Fischlin and J.-S. Coron. Vol. 9666. LNCS. Springer, Heidelberg, May 2016, pp. 559–585. DOI: 10.1007/978-3-662-49896-5_20.

[109] E. D. Cristofaro, P. Gasti, and G. Tsudik. "Fast and Private Computation of Cardinality of Set Intersection and Union". In: *CANS 12*. Ed. by J. Pieprzyk, A.-R. Sadeghi, and M. Manulis. Vol. 7712. LNCS. Springer, Heidelberg, Dec. 2012, pp. 218–231. DOI: 10.1007/978-3-642-35404-5_17.

[110] E. Cristofaro, S. Jarecki, J. Kim, and G. Tsudik. "Privacy-Preserving Policy-Based Information Transfer". In: *Proceedings of the 9th International Symposium on Privacy Enhancing Technologies*. PETS '09. Seattle, WA: Springer-Verlag, 2009, pp. 164–184. ISBN: 978-3-642-03167-0. DOI: 10.1007/978-3-642-03168-7_10. URL: http://dx.doi.org/10.1007/978-3-642-03168-7_10.

[111] *Cyber-security Information Sharing Partnership (CiSP)*. https://www.ncsc.gov.uk/cisp. Accessed: 2018-05-29.

[112] Cyren. *Cyren IP reputation check*. http://www.cyren.com/security-center/ip-reputation-check. Accessed Jul 2018. 2018.

[113] I. Damgård, S. Faust, and C. Hazay. "Secure Two-Party Computation with Low Communication". In: *TCC 2012*. Ed. by R. Cramer. Vol. 7194. LNCS. Springer, Heidelberg, Mar. 2012, pp. 54–74. DOI: 10.1007/978-3-642-28914-9_4.

[114] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryption". In: *CRYPTO 2012*. Ed. by R. Safavi-Naini and R. Canetti. Vol. 7417. LNCS. Springer, Heidelberg, Aug. 2012, pp. 643–662. DOI: 10.1007/978-3-642-32009-5_38.

[115] A. Davidson. *Privacy Pass - "The Math"*. Website. https://blog.cloudflare.com/privacy-pass-the-math/. Accessed Sep 2018. November 09 2017.

[116] A. Davidson. *Privacy Pass: A browser extension for anonymous authentication*. Website. https://medium.com/@alxdavids/privacy-pass-6f0acf075288. Accessed Sep 2018. November 09 2017.

[117] A. Davidson and C. Cid. "An Efficient Toolkit for Computing Private Set Operations". In: *ACISP 17, Part II*. Ed. by J. Pieprzyk and S. Suriadi. Vol. 10343. LNCS. Springer, Heidelberg, July 2017, pp. 261–278.

[118] A. Davidson and C. Cid. *Computing Private Set Operations with Linear Complexities*. Cryptology ePrint Archive, Report 2016/108. http://eprint.iacr.org/2016/108. 2016.

[119] A. Davidson, G. Fenn, and C. Cid. "A Model for Secure and Mutually Beneficial Software Vulnerability Sharing". In: *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*. WISCS '16. Vienna, Austria: ACM, 2016, pp. 3–14. ISBN: 978-1-4503-4565-1. DOI: 10.1145/2994539.2994547.

[120] A. Davidson, I. Goldberg, N. Sullivan, G. Tankersley, and F. Valsorda. "Privacy Pass: Bypassing Internet Challenges Anonymously". In: *PoPETs* 2018.3 (2018), pp. 164–180. DOI: 10.1515/popets-2018-0026. URL: https://doi.org/10.1515/popets-2018-0026.

[121] A. Davidson, S. Katsumata, R. Nishimaki, and S. Yamada. *Constrained PRFs for Bit-fixing from OWFs with Constant Collusion Resistance*. Cryptology ePrint Archive, Report 2018/982. https://eprint.iacr.org/2018/982. 2018.

[122] E. De Cristofaro, J. Kim, and G. Tsudik. "Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model". In: *ASIACRYPT 2010*. Ed. by M. Abe. Vol. 6477. LNCS. Springer, Heidelberg, Dec. 2010, pp. 213–231. DOI: 10.1007/978-3-642-17373-8_13.

[123] E. De Cristofaro and G. Tsudik. "Practical Private Set Intersection Protocols with Linear Complexity". In: *FC 2010*. Ed. by R. Sion. Vol. 6052. LNCS. Springer, Heidelberg, Jan. 2010, pp. 143–159.

[124] S. K. Debnath and R. Dutta. "Efficient Private Set Intersection Cardinality in the Presence of Malicious Adversaries". In: *ProvSec 2015*. Ed. by M. H. Au and A. Miyaji. Vol. 9451. LNCS. Springer, Heidelberg, Nov. 2015, pp. 326–339. DOI: 10.1007/978-3-319-26059-4_18.

[125] S. K. Debnath and R. Dutta. "Secure and Efficient Private Set Intersection Cardinality Using Bloom Filter". In: *ISC 2015*. Ed. by J. Lopez and C. J. Mitchell. Vol. 9290. LNCS. Springer, Heidelberg, Sept. 2015, pp. 209–226. DOI: 10.1007/978-3-319-23318-5_12.

[126] A. W. Dent. *A Note On Game-Hopping Proofs*. Cryptology ePrint Archive, Report 2006/260. http://eprint.iacr.org/2006/260. 2006.

[127] D. Derler, T. Jager, D. Slamanig, and C. Striecks. "Bloom Filter Encryption and Applications to Efficient Forward-Secret 0-RTT Key Exchange". In: *EUROCRYPT 2018, Part III*. Ed. by J. B. Nielsen and V. Rijmen. Vol. 10822. LNCS. Springer, Heidelberg, Apr. 2018, pp. 425–455. DOI: 10.1007/978-3-319-78372-7_14.

[128] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. "Fully Homomorphic Encryption over the Integers". In: *EUROCRYPT 2010*. Ed. by H. Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 24–43. DOI: 10.1007/978-3-642-13190-5_2.

[129] F. Dold and C. Grothoff. "GNU Taler: Ethical Online Payments for the Internet Age". In: *ERCIM News* 2016.106 (2016). URL: http://ercim-news.ercim.eu/en106/special/gnu-taler-ethical-online-payments-for-the-internet-age.

[130] C. Dong, L. Chen, and Z. Wen. "When private set intersection meets big data: an efficient and scalable protocol". In: *ACM CCS 13*. Ed. by A.-R. Sadeghi, V. D. Gligor, and M. Yung. ACM Press, Nov. 2013, pp. 789–800. DOI: 10.1145/2508859.2516701.

[131] C. Dong and G. Loukides. “Approximating Private Set Union/Intersection Cardinality With Logarithmic Complexity”. In: *IEEE Transactions on Information Forensics and Security* 12.11 (2017), pp. 2792–2806.

[132] N. Döttling, S. Garg, D. Gupta, P. Miao, and P. Mukherjee. *Obfuscation from Low Noise Multilinear Maps*. Cryptology ePrint Archive, Report 2016/599. `http://eprint.iacr.org/2016/599`. 2016.

[133] R. Egert, M. Fischlin, D. Gens, S. Jacob, M. Senker, and J. Tillmanns. “Privately Computing Set-Union and Set-Intersection Cardinality via Bloom Filters”. In: *ACISP 15*. Ed. by E. Foo and D. Stebila. Vol. 9144. LNCS. Springer, Heidelberg, June 2015, pp. 413–430. DOI: `10.1007/978-3-319-19962-7_24`.

[134] T. ElGamal. “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”. In: *CRYPTO’84*. Ed. by G. R. Blakley and D. Chaum. Vol. 196. LNCS. Springer, Heidelberg, Aug. 1984, pp. 10–18.

[135] S. Even, O. Goldreich, and A. Lempel. “A Randomized Protocol for Signing Contracts”. In: *CRYPTO’82*. Ed. by D. Chaum, R. L. Rivest, and A. T. Sherman. Plenum Press, New York, USA, 1982, pp. 205–210.

[136] P. Farshim, J. Hesse, D. Hofheinz, and E. Larraia. “Graded Encoding Schemes from Obfuscation”. In: *PKC 2018, Part II*. Ed. by M. Abdalla and R. Dahab. Vol. 10770. LNCS. Springer, Heidelberg, Mar. 2018, pp. 371–400. DOI: `10.1007/978-3-319-76581-5_13`.

[137] U. Feige, D. Lapidot, and A. Shamir. “Multiple Non-Interactive Zero Knowledge Proofs Based on a Single Random String (Extended Abstract)”. In: *31st FOCS*. IEEE Computer Society Press, Oct. 1990, pp. 308–317. DOI: `10.1109/FSCS.1990.89549`.

[138] R. Fernando, P. M. R. Rasmussen, and A. Sahai. “Preventing CLT Attacks on Obfuscation with Linear Overhead”. In: *ASIACRYPT 2017, Part III*. Ed. by T. Takagi and T. Peyrin. Vol. 10626. LNCS. Springer, Heidelberg, Dec. 2017, pp. 242–271. DOI: `10.1007/978-3-319-70700-6_9`.

[139] A. Fiat and A. Shamir. “How to Prove Yourself: Practical Solutions to Identification and Signature Problems”. In: *CRYPTO’86*. Ed. by A. M. Odlyzko. Vol. 263. LNCS. Springer, Heidelberg, Aug. 1987, pp. 186–194. DOI: `10.1007/3-540-47721-7_12`.

[140] FIPS. *FIPS PUB 198-1: The Keyed-Hash Message Authentication Code (HMAC)*. Publication. `https://csrc.nist.gov/csrc/media/publications/fips/198/1/final/documents/fips-198-1_final.pdf`. Accessed Aug 2018. 1997.

[141] T. K. Frederiksen, T. P. Jakobsen, J. B. Nielsen, P. S. Nordholt, and C. Orlandi. “MiniLEGO: Efficient Secure Two-Party Computation from General Assumptions”. In: *EUROCRYPT 2013*. Ed. by T. Johansson and P. Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 537–556. DOI: `10.1007/978-3-642-38348-9_32`.

[142] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. "Keyword Search and Oblivious Pseudorandom Functions". In: *TCC 2005*. Ed. by J. Kilian. Vol. 3378. LNCS. Springer, Heidelberg, Feb. 2005, pp. 303–324. DOI: 10.1007/978-3-540-30576-7_17.

[143] M. J. Freedman, K. Nissim, and B. Pinkas. "Efficient Private Matching and Set Intersection". In: *EUROCRYPT 2004*. Ed. by C. Cachin and J. Camenisch. Vol. 3027. LNCS. Springer, Heidelberg, May 2004, pp. 1–19. DOI: 10.1007/978-3-540-24676-3_1.

[144] E. S. V. Freire, D. Hofheinz, E. Kiltz, and K. G. Paterson. "Non-Interactive Key Exchange". In: *PKC 2013*. Ed. by K. Kurosawa and G. Hanaoka. Vol. 7778. LNCS. Springer, Heidelberg, Feb. 2013, pp. 254–271. DOI: 10.1007/978-3-642-36362-7_17.

[145] J. Freudiger, E. De Cristofaro, and A. E. Brito. "Controlled Data Sharing for Collaborative Predictive Blacklisting". In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Ed. by M. Almgren, V. Gulisano, and F. Maggi. Cham: Springer International Publishing, 2015, pp. 327–349.

[146] K. B. Frikken. "Privacy-Preserving Set Union". In: *ACNS 07*. Ed. by J. Katz and M. Yung. Vol. 4521. LNCS. Springer, Heidelberg, June 2007, pp. 237–252. DOI: 10.1007/978-3-540-72738-5_16.

[147] G. Fuchsbauer, C. Hanser, C. Kamath, and D. Slamanig. *Practical Round-Optimal Blind Signatures in the Standard Model from Weaker Assumptions*. Cryptology ePrint Archive, Report 2016/662. http://eprint.iacr.org/2016/662. 2016.

[148] S. Garg, C. Gentry, and S. Halevi. "Candidate Multilinear Maps from Ideal Lattices". In: *EUROCRYPT 2013*. Ed. by T. Johansson and P. Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 1–17. DOI: 10.1007/978-3-642-38348-9_1.

[149] S. Garg, C. Gentry, S. Halevi, and M. Raykova. "Two-Round Secure MPC from Indistinguishability Obfuscation". In: *TCC 2014*. Ed. by Y. Lindell. Vol. 8349. LNCS. Springer, Heidelberg, Feb. 2014, pp. 74–94. DOI: 10.1007/978-3-642-54242-8_4.

[150] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. "Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits". In: *54th FOCS*. IEEE Computer Society Press, Oct. 2013, pp. 40–49. DOI: 10.1109/FOCS.2013.13.

[151] S. Garg, C. Gentry, S. Halevi, A. Sahai, and B. Waters. "Attribute-Based Encryption for Circuits from Multilinear Maps". In: *CRYPTO 2013, Part II*. Ed. by R. Canetti and J. A. Garay. Vol. 8043. LNCS. Springer, Heidelberg, Aug. 2013, pp. 479–499. DOI: 10.1007/978-3-642-40084-1_27.

[152] S. Garg and D. Gupta. "Efficient Round Optimal Blind Signatures". In: *EUROCRYPT 2014*. Ed. by P. Q. Nguyen and E. Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 477–495. DOI: 10.1007/978-3-642-55220-5_27.

[153] S. Garg, E. Miles, P. Mukherjee, A. Sahai, A. Srinivasan, and M. Zhandry. “Secure Obfuscation in a Weak Multilinear Map Model”. In: *TCC 2016-B, Part II*. Ed. by M. Hirt and A. D. Smith. Vol. 9986. LNCS. Springer, Heidelberg, Oct. 2016, pp. 241–268. DOI: 10.1007/978-3-662-53644-5_10.

[154] S. Garg, P. Mukherjee, and A. Srinivasan. *Obfuscation without the Vulnerabilities of Multilinear Maps*. Cryptology ePrint Archive, Report 2016/390. http://eprint.iacr.org/2016/390. 2016.

[155] S. Garg, V. Rao, A. Sahai, D. Schröder, and D. Unruh. “Round Optimal Blind Signatures”. In: *CRYPTO 2011*. Ed. by P. Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 630–648. DOI: 10.1007/978-3-642-22792-9_36.

[156] C. Gentry. “Fully homomorphic encryption using ideal lattices”. In: *41st ACM STOC*. Ed. by M. Mitzenmacher. ACM Press, May 2009, pp. 169–178. DOI: 10.1145/1536414.1536440.

[157] C. Gentry, S. Gorbunov, and S. Halevi. “Graph-Induced Multilinear Maps from Lattices”. In: *TCC 2015, Part II*. Ed. by Y. Dodis and J. B. Nielsen. Vol. 9015. LNCS. Springer, Heidelberg, Mar. 2015, pp. 498–527. DOI: 10.1007/978-3-662-46497-7_20.

[158] C. Gentry, S. Halevi, M. Raykova, and D. Wichs. “Outsourcing Private RAM Computation”. In: *55th FOCS*. IEEE Computer Society Press, Oct. 2014, pp. 404–413. DOI: 10.1109/FOCS.2014.50.

[159] C. Gentry, S. Halevi, and V. Vaikuntanathan. “A Simple BGN-Type Cryptosystem from LWE”. In: *EUROCRYPT 2010*. Ed. by H. Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 506–522. DOI: 10.1007/978-3-642-13190-5_26.

[160] C. Gentry, A. B. Lewko, A. Sahai, and B. Waters. “Indistinguishability Obfuscation from the Multilinear Subgroup Elimination Assumption”. In: *56th FOCS*. Ed. by V. Guruswami. IEEE Computer Society Press, Oct. 2015, pp. 151–170. DOI: 10.1109/FOCS.2015.19.

[161] C. Gentry, C. Peikert, and V. Vaikuntanathan. “Trapdoors for hard lattices and new cryptographic constructions”. In: *40th ACM STOC*. Ed. by R. E. Ladner and C. Dwork. ACM Press, May 2008, pp. 197–206. DOI: 10.1145/1374376.1374407.

[162] C. Gentry, A. Sahai, and B. Waters. “Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based”. In: *CRYPTO 2013, Part I*. Ed. by R. Canetti and J. A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 75–92. DOI: 10.1007/978-3-642-40041-4_5.

[163] S. Goel and H. A. Shawky. “Estimating the market impact of security breach announcements on firm values”. In: *Information & Management* 46.7 (2009), pp. 404–410.

[164] O. Goldreich, S. Goldwasser, and S. Micali. “How to construct random functions”. In: *J. ACM* 33.4 (1986), pp. 792–807.

[165] O. Goldreich, S. Micali, and A. Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *19th ACM STOC*. Ed. by A. Aho. ACM Press, May 1987, pp. 218–229. DOI: 10.1145/28395.28420.

[166] S. Goldwasser and S. Micali. "Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information". In: *14th ACM STOC*. ACM Press, May 1982, pp. 365–377. DOI: 10.1145/800070.802212.

[167] S. Goldwasser, S. Micali, and C. Rackoff. "The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract)". In: *17th ACM STOC*. ACM Press, May 1985, pp. 291–304. DOI: 10.1145/22145.22178.

[168] S. Goldwasser and G. N. Rothblum. "On Best-Possible Obfuscation". In: *TCC 2007*. Ed. by S. P. Vadhan. Vol. 4392. LNCS. Springer, Heidelberg, Feb. 2007, pp. 194–213. DOI: 10.1007/978-3-540-70936-7_11.

[169] S. Gorbunov, V. Vaikuntanathan, and H. Wee. "Attribute-based encryption for circuits". In: *45th ACM STOC*. Ed. by D. Boneh, T. Roughgarden, and J. Feigenbaum. ACM Press, June 2013, pp. 545–554. DOI: 10.1145/2488608.2488677.

[170] L. Gordon, M. Loeb, and W. Lucy-shyn. "Sharing information on computer systems security: An economic analysis". In: *Journal of Accounting and Public Policy* 22.6 (2003), pp. 461–485.

[171] R. Goyal, S. Hohenberger, V. Koppula, and B. Waters. "A Generic Approach to Constructing and Proving Verifiable Random Functions". In: *TCC 2017, Part II*. Ed. by Y. Kalai and L. Reyzin. Vol. 10678. LNCS. Springer, Heidelberg, Nov. 2017, pp. 537–566. DOI: 10.1007/978-3-319-70503-3_18.

[172] R. Goyal, V. Koppula, and B. Waters. "Lockable Obfuscation". In: *58th FOCS*. IEEE Computer Society Press, 2017, pp. 612–621. DOI: 10.1109/FOCS.2017.62.

[173] M. D. Green and I. Miers. "Forward Secure Asynchronous Messaging from Puncturable Encryption". In: *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 305–320. DOI: 10.1109/SP.2015.26.

[174] J. Grossklags, N. Christin, and J. Chuang. "Secure or insure?: a game-theoretic analysis of information security games". In: *Proceedings of the 17th international conference on World Wide Web*. ACM. 2008, pp. 209–218.

[175] F. Günther, B. Hale, T. Jager, and S. Lauer. "0-RTT Key Exchange with Full Forward Secrecy". In: *EUROCRYPT 2017, Part III*. Ed. by J. Coron and J. B. Nielsen. Vol. 10212. LNCS. Springer, Heidelberg, Apr. 2017, pp. 519–548. DOI: 10.1007/978-3-319-56617-7_18.

[176] S. Halevi. *Graded Encoding, Variations on a Scheme*. Cryptology ePrint Archive, Report 2015/866. http://eprint.iacr.org/2015/866. 2015.

[177] S. Halevi, C. Hazay, A. Polychroniadou, and M. Venkitasubramaniam. *Round-Optimal Secure Multi-Party Computation*. Cryptology ePrint Archive, Report 2017/1056. http://eprint.iacr.org/2017/1056. 2017.

[178] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. "A Pseudorandom Generator from any One-way Function". In: *SIAM Journal on Computing* 28.4 (1999), pp. 1364–1396.

[179] K. Hausken. "Information sharing among firms and cyber attacks". In: *Journal of Accounting and Public Policy* 26.6 (2007), pp. 639–688.

[180] C. Hazay. "Oblivious Polynomial Evaluation and Secure Set-Intersection from Algebraic PRFs". In: *TCC 2015, Part II*. Ed. by Y. Dodis and J. B. Nielsen. Vol. 9015. LNCS. Springer, Heidelberg, Mar. 2015, pp. 90–120. DOI: 10.1007/978-3-662-46497-7_4.

[181] C. Hazay and K. Nissim. "Efficient Set Operations in the Presence of Malicious Adversaries". In: *Journal of Cryptology* 25.3 (2012), pp. 383–433. DOI: 10.1007/s00145-011-9098-x.

[182] Henry, Ryan. "Efficient Zero-Knowledge Proofs and Applications". http://hdl.handle.net/10012/8621. PhD thesis. University of Waterloo, 2014.

[183] R. Henry and I. Goldberg. "Extending Nymble-like Systems". In: *2011 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2011, pp. 523–537. DOI: 10.1109/SP.2011.17.

[184] R. Henry and I. Goldberg. "Formalizing Anonymous Blacklisting Systems". In: *2011 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2011, pp. 81–95. DOI: 10.1109/SP.2011.13.

[185] T. S. Heydt-Benjamin, H.-J. Chae, B. Defend, and K. Fu. "Privacy for Public Transportation". In: *Privacy Enhancing Technologies: 6th International Workshop (PET 2006)*. Ed. by G. Danezis and P. Golle. Springer, 2006, pp. 1–19. ISBN: 978-3-540-68793-1. DOI: 10.1007/11957454_1. URL: https://doi.org/10.1007/11957454_1.

[186] J. Hoffstein, J. Pipher, and J. H. Silverman. "NTRU: A Ring-Based Public Key Cryptosystem". In: *Lecture Notes in Computer Science*. Springer-Verlag, 1998, pp. 267–288.

[187] D. Hofheinz, A. Kamath, V. Koppula, and B. Waters. *Adaptively Secure Constrained Pseudorandom Functions*. Cryptology ePrint Archive, Report 2014/720. http://eprint.iacr.org/2014/720. 2014.

[188] F. Hormozdiari, J. W. J. Joo, A. Wadia, F. Guan, R. Ostrovsky, A. Sahai, and E. Eskin. "Privacy preserving protocol for detecting genetic relatives using rare variants". In: *Bioinformatics* 30.12 (2014), pp. 204–211. DOI: 10.1093/bioinformatics/btu294. URL: http://dx.doi.org/10.1093/bioinformatics/btu294.

[189] Y. Hu and H. Jia. "Cryptanalysis of GGH Map". In: *EUROCRYPT 2016, Part I*. Ed. by M. Fischlin and J.-S. Coron. Vol. 9665. LNCS. Springer, Heidelberg, May 2016, pp. 537–565. DOI: 10.1007/978-3-662-49890-3_21.

[190] M. A. Huang. “Trilinear maps for cryptography”. In: *CoRR* abs/1803.10325 (2018). arXiv: 1803.10325. URL: http://arxiv.org/abs/1803.10325.

[191] Y. Huang, D. Evans, and J. Katz. “Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?” In: *NDSS 2012*. The Internet Society, Feb. 2012.

[192] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. “Extending Oblivious Transfers Efficiently”. In: *CRYPTO 2003*. Ed. by D. Boneh. Vol. 2729. LNCS. Springer, Heidelberg, Aug. 2003, pp. 145–161. DOI: 10.1007/978-3-540-45146-4_9.

[193] M. O. Jackson and Y. Zenou. “Games on networks”. In: *Handbook of Game Theory* 4 (2014).

[194] S. Jarecki, A. Kiayias, and H. Krawczyk. “Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model”. In: *ASIACRYPT 2014, Part II*. Ed. by P. Sarkar and T. Iwata. Vol. 8874. LNCS. Springer, Heidelberg, Dec. 2014, pp. 233–253. DOI: 10.1007/978-3-662-45608-8_13.

[195] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. “Highly-Efficient and Composable Password-Protected Secret Sharing (Or: How to Protect Your Bitcoin Wallet Online)”. In: *EuroS&P*. IEEE, 2016, pp. 276–291.

[196] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. “TOPPSS: Cost-Minimal Password-Protected Secret Sharing Based on Threshold OPRF”. In: *ACNS 17*. Ed. by D. Gollmann, A. Miyaji, and H. Kikuchi. Vol. 10355. LNCS. Springer, Heidelberg, July 2017, pp. 39–58. DOI: 10.1007/978-3-319-61204-1_3.

[197] S. Jarecki and X. Liu. “Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection”. In: *TCC 2009*. Ed. by O. Reingold. Vol. 5444. LNCS. Springer, Heidelberg, Mar. 2009, pp. 577–594. DOI: 10.1007/978-3-642-00457-5_34.

[198] S. Jarecki and X. Liu. “Fast Secure Computation of Set Intersection”. In: *SCN 10*. Ed. by J. A. Garay and R. D. Prisco. Vol. 6280. LNCS. Springer, Heidelberg, Sept. 2010, pp. 418–435. DOI: 10.1007/978-3-642-15317-4_26.

[199] B. Johnson, J. Grossklags, N. Christin, and J. Chuang. “Uncertainty in interdependent security games”. In: *Decision and Game Theory for Security*. Springer, 2010, pp. 234–244.

[200] C. Jost, H. Lam, A. Maximov, and B. Smeets. *Encryption Performance Improvements of the Paillier Cryptosystem*. Cryptology ePrint Archive, Report 2015/864. http://eprint.iacr.org/2015/864. 2015.

[201] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. 2nd. Chapman & Hall/CRC, 2014. ISBN: 1466570261, 9781466570269.

[202] N. Kayal. “The Complexity of the Annihilating Polynomial”. In: *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*. IEEE Computer Society, 2009, pp. 184–193. DOI: 10.1109/CCC.2009.37. URL: http://dx.doi.org/10.1109/CCC.2009.37.

[203] M. Keller, E. Orsini, and P. Scholl. “Actively Secure OT Extension with Optimal Overhead”. In: *CRYPTO 2015, Part I*. Ed. by R. Gennaro and M. J. B. Robshaw. Vol. 9215. LNCS. Springer, Heidelberg, Aug. 2015, pp. 724–741. DOI: 10.1007/978-3-662-47989-6_35.

[204] F. Kerschbaum. “Outsourced private set intersection using homomorphic encryption”. In: *ASIACCS 12*. Ed. by H. Y. Youm and Y. Won. ACM Press, May 2012, pp. 85–86.

[205] F. Kerschbaum. “Public-Key Encrypted Bloom Filters with Applications to Supply Chain Integrity”. In: *Data and Applications Security and Privacy XXV - 25th Annual IFIP WG 11.3 Conference, DBSec 2011, Richmond, VA, USA, July 11-13, 2011. Proceedings*. Vol. 6818. Lecture Notes in Computer Science. Springer, 2011, pp. 60–75.

[206] F. Kerschbaum, H. W. Lim, and I. Gudymenko. “Privacy-preserving Billing for e-Ticketing Systems in Public Transportation”. In: *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*. WPES ’13. Berlin, Germany: ACM, 2013, pp. 143–154. ISBN: 978-1-4503-2485-4. DOI: 10.1145/2517840.2517848. URL: http://doi.acm.org/10.1145/2517840.2517848.

[207] S. Khattak, D. Fifield, S. Afroz, M. Javed, S. Sundaresan, D. McCoy, V. Paxson, and S. J. Murdoch. “Do You See What I See? Differential Treatment of Anonymous Users”. In: *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. 2016. URL: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/do-you-see-what-i-see-differential-treatment-anonymous-users.pdf.

[208] M. H. R. Khouzani, V. Pham, and C. Cid. “Strategic Discovery and Sharing of Vulnerabilities in Competitive Environments”. In: *GameSec 2014, Los Angeles, CA. Nov 6-7, 2014. Proceedings*. 2014, pp. 59–78.

[209] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias. “Delegatable pseudorandom functions and applications”. In: *ACM CCS 13*. Ed. by A.-R. Sadeghi, V. D. Gligor, and M. Yung. ACM Press, Nov. 2013, pp. 669–684. DOI: 10.1145/2508859.2516668.

[210] P. Kirchner and P.-A. Fouque. “Revisiting Lattice Attacks on Overstretched NTRU Parameters”. In: *EUROCRYPT 2017, Part I*. Ed. by J. Coron and J. B. Nielsen. Vol. 10210. LNCS. Springer, Heidelberg, Apr. 2017, pp. 3–26. DOI: 10.1007/978-3-319-56620-7_1.

[211] L. Kissner and D. X. Song. “Privacy-Preserving Set Operations”. In: *CRYPTO 2005*. Ed. by V. Shoup. Vol. 3621. LNCS. Springer, Heidelberg, Aug. 2005, pp. 241–257. DOI: 10.1007/11535218_15.

[212] N. Koblitz. “Hyperelliptic Cryptosystems”. In: *Journal of Cryptology* 1.3 (1989), pp. 139–150.

[213] V. Kolesnikov and R. Kumaresan. “Improved OT Extension for Transferring Short Secrets”. In: *CRYPTO 2013, Part II*. Ed. by R. Canetti and J. A. Garay. Vol. 8043. LNCS. Springer, Heidelberg, Aug. 2013, pp. 54–70. DOI: 10.1007/978-3-642-40084-1_4.

[214] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. “Efficient Batched Oblivious PRF with Applications to Private Set Intersection”. In: *ACM CCS 16*. Ed. by E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi. ACM Press, Oct. 2016, pp. 818–829. DOI: 10.1145/2976749.2978381.

[215] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu. “Practical Multi-party Private Set Intersection from Symmetric-Key Techniques”. In: *ACM CCS 17*. Ed. by B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu. ACM Press, Oct. 2017, pp. 1257–1272. DOI: 10.1145/3133956.3134065.

[216] H. Krawczyk, M. Bellare, and R. Canetti. *IETF-RFC2104: HMAC: Keyed-Hashing for Message Authentication*. IETF RFC. https://tools.ietf.org/html/rfc2104. Accessed Aug 2018. 2008.

[217] H. Kunreuther and G. Heal. “Interdependent security”. In: *Journal of risk and uncertainty* 26.2-3 (2003), pp. 231–249.

[218] K. Lai and M. Goemans. “The knapsack problem and fully polynomial time approximation schemes (FPTAS)”. In: (2006).

[219] A. Langlois, D. Stehlé, and R. Steinfeld. “GGHLite: More Efficient Multilinear Maps from Ideal Lattices”. In: *EUROCRYPT 2014*. Ed. by P. Q. Nguyen and E. Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 239–256. DOI: 10.1007/978-3-642-55220-5_14.

[220] A. Laszka, M. Felegyhazi, and L. Buttyan. “Survey of interdependent information security games”. In: *ACM Computing Surveys* 47.2 (2015), p. 23.

[221] S. Laube and R. Böhme. “Mandatory security information sharing with authorities: Implications on investments in internal controls”. In: *Proceedings of the 2nd ACM Workshop on Information Sharing and Collaborative Security*. ACM. 2015, pp. 31–42.

[222] H. Lin. “Indistinguishability Obfuscation from Constant-Degree Graded Encoding Schemes”. In: *EUROCRYPT 2016, Part I*. Ed. by M. Fischlin and J.-S. Coron. Vol. 9665. LNCS. Springer, Heidelberg, May 2016, pp. 28–57. DOI: 10.1007/978-3-662-49890-3_2.

[223] H. Lin. “Indistinguishability Obfuscation from SXDH on 5-Linear Maps and Locality-5 PRGs”. In: *CRYPTO 2017, Part I*. Ed. by J. Katz and H. Shacham. Vol. 10401. LNCS. Springer, Heidelberg, Aug. 2017, pp. 599–629. DOI: 10.1007/978-3-319-63688-7_20.

[224] H. Lin and S. Tessaro. “Indistinguishability Obfuscation from Trilinear Maps and Block-Wise Local PRGs”. In: *CRYPTO 2017, Part I*. Ed. by J. Katz and H. Shacham. Vol. 10401. LNCS. Springer, Heidelberg, Aug. 2017, pp. 630–660. DOI: 10.1007/978-3-319-63688-7_21.

[225] H. Lin and V. Vaikuntanathan. “Indistinguishability Obfuscation from DDH-Like Assumptions on Constant-Degree Graded Encodings”. In: *57th FOCS*. Ed. by I. Dinur. IEEE Computer Society Press, Oct. 2016, pp. 11–20. DOI: 10.1109/FOCS.2016.11.

[226] Y. Lindell and B. Pinkas. “A Proof of Security of Yao’s Protocol for Two-Party Computation”. In: *Journal of Cryptology* 22.2 (2009), pp. 161–188.

[227] Y. Lindell and B. Pinkas. “Privacy Preserving Data Mining”. In: *Journal of Cryptology* 15.3 (2002), pp. 177–206.

[228] D. Liu, Y. Ji, and V. Mookerjee. “Knowledge sharing and investment decisions in information security”. In: *Decision Support Systems* 52.1 (2011), pp. 95–107.

[229] Z. Liu, Y. Liu, P. Winter, P. Mittal, and Y. Hu. “TorPolice: Towards enforcing service-defined access policies for anonymous communication in the Tor network”. In: *25th IEEE International Conference on Network Protocols, ICNP 2017*. 2017, pp. 1–10. DOI: 10.1109/ICNP.2017.8117564. URL: https://doi.org/10.1109/ICNP.2017.8117564.

[230] P. Lofgren and N. Hopper. “BNymble: More Anonymous Blacklisting at Almost No Cost (A Short Paper)”. In: *FC 2011*. Ed. by G. Danezis. Vol. 7035. LNCS. Springer, Heidelberg, Feb. 2012, pp. 268–275.

[231] A. López-Alt, E. Tromer, and V. Vaikuntanathan. “On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption”. In: *44th ACM STOC*. Ed. by H. J. Karloff and T. Pitassi. ACM Press, May 2012, pp. 1219–1234. DOI: 10.1145/2213977.2214086.

[232] B. Lynn, M. Prabhakaran, and A. Sahai. “Positive Results and Techniques for Obfuscation”. In: *EUROCRYPT 2004*. Ed. by C. Cachin and J. Camenisch. Vol. 3027. LNCS. Springer, Heidelberg, May 2004, pp. 20–39. DOI: 10.1007/978-3-540-24676-3_2.

[233] V. Lyubashevsky, C. Peikert, and O. Regev. “On Ideal Lattices and Learning with Errors over Rings”. In: *EUROCRYPT 2010*. Ed. by H. Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 1–23. DOI: 10.1007/978-3-642-13190-5_1.

[234] F. Ma and M. Zhandry. *Encryptor Combiners: A Unified Approach to Multiparty NIKE, (H)IBE, and Broadcast Encryption*. Cryptology ePrint Archive, Report 2017/152. http://eprint.iacr.org/2017/152. 2017.

[235] F. Ma and M. Zhandry. *New Multilinear Maps from CLT13 with Provable Security Against Zeroizing Attacks*. Cryptology ePrint Archive, Report 2017/946. http://eprint.iacr.org/2017/946. 2017.

[236] C. A. Meadows. "A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party". In: *Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 7-9, 1986*. 1986, pp. 134–137. DOI: 10.1109/SP.1986.10022. URL: http://dx.doi.org/10.1109/SP.1986.10022.

[237] D. Micciancio and C. Peikert. "Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller". In: *EUROCRYPT 2012*. Ed. by D. Pointcheval and T. Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 700–718. DOI: 10.1007/978-3-642-29011-4_41.

[238] D. Micciancio and O. Regev. "Worst-Case to Average-Case Reductions Based on Gaussian Measures". In: *45th FOCS*. IEEE Computer Society Press, Oct. 2004, pp. 372–381. DOI: 10.1109/FOCS.2004.72.

[239] E. Miles, A. Sahai, and M. Weiss. *Protecting obfuscation against arithmetic attacks*. Cryptology ePrint Archive, Report 2014/878. http://eprint.iacr.org/2014/878. 2014.

[240] E. Miles, A. Sahai, and M. Zhandry. "Annihilation Attacks for Multilinear Maps: Cryptanalysis of Indistinguishability Obfuscation over GGH13". In: *CRYPTO 2016, Part II*. Ed. by M. Robshaw and J. Katz. Vol. 9815. LNCS. Springer, Heidelberg, Aug. 2016, pp. 629–658. DOI: 10.1007/978-3-662-53008-5_22.

[241] V. S. Miller. "Use of Elliptic Curves in Cryptography". In: *CRYPTO'85*. Ed. by H. C. Williams. Vol. 218. LNCS. Springer, Heidelberg, Aug. 1986, pp. 417–426. DOI: 10.1007/3-540-39799-X_31.

[242] S. Nagaraja, P. Mittal, C. Hong, M. Caesar, and N. Borisov. "BotGrep: Finding P2P Bots with Structured Graph Analysis". In: *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*. 2010, pp. 95–110.

[243] M. Naor and B. Pinkas. "Oblivious Transfer and Polynomial Evaluation". In: *31st ACM STOC*. ACM Press, May 1999, pp. 245–254. DOI: 10.1145/301250.301312.

[244] M. Naor and O. Reingold. "Number-theoretic constructions of efficient pseudo-random functions". In: *J. ACM* 51.2 (2004), pp. 231–262.

[245] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. "Location Privacy via Private Proximity Testing". In: *NDSS 2011*. The Internet Society, Feb. 2011.

[246] National Institute of Standards and Technology. *FIPS PUB 186-2: Digital Signature Standard (DSS)*. 2000. URL: http://www.itl.nist.gov/fipspubs/fip186-2.pdf.

[247] *NIST key length security guidelines*. http://www.keylength.com/en/4/. Accessed: 2018-05-22.

[248] *NIST National Vulnerability Database, Vulnerabilitity Metrics*. https://nvd.nist.gov/vuln-metrics/cvss. Accessed: 2018-05-30.

[249] *OASIS open standards*. https://www.oasis-open.org/. Accessed: 2018-05-29.

[250] J. P. Farwell and R. Rohozinski. "Stuxnet and the Future of Cyber War". In: 53 (Feb. 2011), pp. 23–40.

[251] P. Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *EUROCRYPT'99*. Ed. by J. Stern. Vol. 1592. LNCS. Springer, Heidelberg, May 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16.

[252] D. Papadopoulos, D. Wessels, S. Huque, M. Naor, J. Včelák, L. Reyzin, and S. Goldberg. *Making NSEC5 Practical for DNSSEC*. Cryptology ePrint Archive, Report 2017/099. http://eprint.iacr.org/2017/099. 2017.

[253] R. Pass, K. Seth, and S. Telang. "Indistinguishability Obfuscation from Semantically-Secure Multilinear Encodings". In: *CRYPTO 2014, Part I*. Ed. by J. A. Garay and R. Gennaro. Vol. 8616. LNCS. Springer, Heidelberg, Aug. 2014, pp. 500–517. DOI: 10.1007/978-3-662-44371-2_28.

[254] C. Peikert and A. Rosen. "Efficient Collision-Resistant Hashing from Worst-Case Assumptions on Cyclic Lattices". In: *TCC 2006*. Ed. by S. Halevi and T. Rabin. Vol. 3876. LNCS. Springer, Heidelberg, Mar. 2006, pp. 145–166. DOI: 10.1007/11681878_8.

[255] C. Peikert and S. Shiehian. "Privately Constraining and Programming PRFs, the LWE Way". In: *PKC 2018, Part II*. Ed. by M. Abdalla and R. Dahab. Vol. 10770. LNCS. Springer, Heidelberg, Mar. 2018, pp. 675–701. DOI: 10.1007/978-3-319-76581-5_23.

[256] A. Pellet-Mary. *Quantum Attacks against Indistinguishablility Obfuscators Proved Secure in the Weak Multilinear Map Model*. Cryptology ePrint Archive, Report 2018/533. https://eprint.iacr.org/2018/533. 2018.

[257] C. E. Phillips Jr, T. Ting, and S. A. Demurjian. "Information sharing and security in dynamic coalitions". In: *Proceedings of the seventh ACM symposium on Access control models and technologies*. ACM. 2002, pp. 87–96.

[258] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. "Phasing: Private Set Intersection Using Permutation-based Hashing". In: *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, 2015, pp. 515–530. ISBN: 978-1-931971-232. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/pinkas.

[259] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder. "Efficient Circuit-Based PSI via Cuckoo Hashing". In: *EUROCRYPT 2018, Part III*. Ed. by J. B. Nielsen and V. Rijmen. Vol. 10822. LNCS. Springer, Heidelberg, Apr. 2018, pp. 125–157. DOI: 10.1007/978-3-319-78372-7_5.

[260] B. Pinkas, T. Schneider, and M. Zohner. "Faster Private Set Intersection Based on OT Extension". In: *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*. 2014, pp. 797–812.

[261] D. Pointcheval and J. Stern. "Provably Secure Blind Signature Schemes". In: *ASIACRYPT'96*. Ed. by K. Kim and T. Matsumoto. Vol. 1163. LNCS. Springer, Heidelberg, Nov. 1996, pp. 252–265. DOI: 10.1007/BFb0034852.

[262] D. Pointcheval and J. Stern. "Security Proofs for Signature Schemes". In: *EUROCRYPT'96*. Ed. by U. M. Maurer. Vol. 1070. LNCS. Springer, Heidelberg, May 1996, pp. 387–398. DOI: 10.1007/3-540-68339-9_33.

[263] M. O. Rabin. *How To Exchange Secrets with Oblivious Transfer*. Cryptology ePrint Archive, Report 2005/187. http://eprint.iacr.org/2005/187. 2005.

[264] O. Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: *37th ACM STOC*. Ed. by H. N. Gabow and R. Fagin. ACM Press, May 2005, pp. 84–93. DOI: 10.1145/1060590.1060603.

[265] A. C. D. Resende and D. F. Aranha. *Unbalanced Approximate Private Set Intersection*. Cryptology ePrint Archive, Report 2017/677. http://eprint.iacr.org/2017/677. 2017.

[266] P. Rindal and M. Rosulek. "Improved Private Set Intersection Against Malicious Adversaries". In: *EUROCRYPT 2017, Part I*. Ed. by J. Coron and J. B. Nielsen. Vol. 10210. LNCS. Springer, Heidelberg, Apr. 2017, pp. 235–259. DOI: 10.1007/978-3-319-56620-7_9.

[267] P. Rindal and M. Rosulek. "Malicious-Secure Private Set Intersection via Dual Execution". In: *ACM CCS 17*. Ed. by B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu. ACM Press, Oct. 2017, pp. 1229–1242. DOI: 10.1145/3133956.3134044.

[268] R. L. Rivest, A. Shamir, and L. M. Adleman. "A Method for Obtaining Digital Signature and Public-Key Cryptosystems". In: *Communications of the Association for Computing Machinery* 21.2 (1978), pp. 120–126.

[269] M. Rückert. "Lattice-Based Blind Signatures". In: *ASIACRYPT 2010*. Ed. by M. Abe. Vol. 6477. LNCS. Springer, Heidelberg, Dec. 2010, pp. 413–430. DOI: 10.1007/978-3-642-17373-8_24.

[270] *S.754 - To improve cybersecurity in the United States through enhanced sharing of information about cybersecurity threats, and for other purposes.* https://www.congress.gov/bill/114th-congress/senate-bill/754. Accessed: 2018-05-29.

[271] A.-R. Sadeghi, I. Visconti, and C. Wachsmann. "User Privacy in Transport Systems Based on RFID E-Tickets". In: *Proceedings of the 1st International Workshop on Privacy in Location-Based Applications (PilBA)* (2008).

[272] A. Sahai and B. Waters. "How to use indistinguishability obfuscation: deniable encryption, and more". In: *46th ACM STOC*. Ed. by D. B. Shmoys. ACM Press, May 2014, pp. 475–484. DOI: 10.1145/2591796.2591825.

[273] C.-P. Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *CRYPTO'89*. Ed. by G. Brassard. Vol. 435. LNCS. Springer, Heidelberg, Aug. 1990, pp. 239–252. DOI: 10.1007/0-387-34805-0_22.

[274] C.-P. Schnorr and M. Jakobsson. "Security of Signed ElGamal Encryption". In: *ASIACRYPT 2000*. Ed. by T. Okamoto. Vol. 1976. LNCS. Springer, Heidelberg, Dec. 2000, pp. 73–89. DOI: 10.1007/3-540-44448-3_7.

[275] J. H. Seo, J. H. Cheon, and J. Katz. "Constant-Round Multi-party Private Set Union Using Reversed Laurent Series". In: *PKC 2012*. Ed. by M. Fischlin, J. Buchmann, and M. Manulis. Vol. 7293. LNCS. Springer, Heidelberg, May 2012, pp. 398–412. DOI: 10.1007/978-3-642-30057-8_24.

[276] C. Shapiro and H. R. Varian. *Information rules: a strategic guide to the network economy*. Harvard Business Press, 1999.

[277] M. Shirvanian, S. Jarecki, H. Krawczyk, and N. Saxena. *SPHINX: A Password Store that Perfectly Hides Passwords from Itself*. Cryptology ePrint Archive, Report 2018/695. https://eprint.iacr.org/2018/695. 2018.

[278] P. W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[279] V. Shoup. *Sequences of games: a tool for taming complexity in security proofs*. Cryptology ePrint Archive, Report 2004/332. http://eprint.iacr.org/2004/332. 2004.

[280] M. Slikker, H. Norde, and S. Tijs. "Information sharing games". In: *International Game Theory Review* 5.01 (2003), pp. 1–12.

[281] D. Stehlé and R. Steinfeld. "Making NTRU as Secure as Worst-Case Problems over Ideal Lattices". In: *EUROCRYPT 2011*. Ed. by K. G. Paterson. Vol. 6632. LNCS. Springer, Heidelberg, May 2011, pp. 27–47. DOI: 10.1007/978-3-642-20465-4_4.

[282] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov. "The First Collision for Full SHA-1". In: *CRYPTO 2017, Part I*. Ed. by J. Katz and H. Shacham. Vol. 10401. LNCS. Springer, Heidelberg, Aug. 2017, pp. 570–596. DOI: 10.1007/978-3-319-63688-7_19.

[283] D. R. Stinson. "Universal Hashing and Authentication Codes". In: *CRYPTO'91*. Ed. by J. Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, Aug. 1992, pp. 74–85. DOI: 10.1007/3-540-46766-1_5.

[284] N. Sullivan. *Cloudflare supports Privacy Pass*. Website. https://blog.cloudflare.com/cloudflare-supports-privacy-pass/. Accessed Sep 2018. November 09 2017.

[285] *The Go Programming Language*. https://golang.org/. Accessed: 2018-05-25.

[286] *ThreatExchange*. https://developers.facebook.com/programs/threatexchange. Accessed: 2018-05-29.

[287] *ThreatStream*. https://www.anomali.com/. Accessed: 2018-05-29.

[288] Tor. *List Of Services Blocking Tor*. https://trac.torproject.org/projects/tor/wiki/org/doc/ListOfServicesBlockingTor. Accessed Sep 2017. 2017.

[289] Tor. *The Problem with Cloudflare*. Website. https://blog.torproject.org/trouble-cloudflare. Accessed Jul 2018. 2017.

[290] X. Wang, A. J. Malozemoff, and J. Katz. "Faster Secure Two-Party Computation in the Single-Execution Setting". In: *EUROCRYPT 2017, Part III*. Ed. by J. Coron and J. B. Nielsen. Vol. 10212. LNCS. Springer, Heidelberg, Apr. 2017, pp. 399–424. DOI: 10.1007/978-3-319-56617-7_14.

[291] X. Wang, S. Ranellucci, and J. Katz. "Global-Scale Secure Multiparty Computation". In: *ACM CCS 17*. Ed. by B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu. ACM Press, Oct. 2017, pp. 39–56. DOI: 10.1145/3133956.3133979.

[292] B. R. Waters. "Efficient Identity-Based Encryption Without Random Oracles". In: *EUROCRYPT 2005*. Ed. by R. Cramer. Vol. 3494. LNCS. Springer, Heidelberg, May 2005, pp. 114–127. DOI: 10.1007/11426639_7.

[293] H. Wee. "On obfuscating point functions". In: *37th ACM STOC*. Ed. by H. N. Gabow and R. Fagin. ACM Press, May 2005, pp. 523–532. DOI: 10.1145/1060590.1060669.

[294] D. Wichs and G. Zirdelis. "Obfuscating Compute-and-Compare Programs under LWE". In: *58th FOCS*. IEEE Computer Society Press, 2017, pp. 600–611. DOI: 10.1109/FOCS.2017.61.

[295] Q. Xiong and X. Chen. "Incentive Mechanism Design Based on Repeated Game Theory in Security Information Sharing". In: *2nd International Conference on Science and Social Research (ICSSR 2013)*. Atlantis Press. 2013.

[296] A. C.-C. Yao. “Protocols for Secure Computations (Extended Abstract)”. In: *23rd FOCS*. IEEE Computer Society Press, Nov. 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38.

[297] M. Zhandry. *How to Avoid Obfuscation Using Witness PRFs*. Cryptology ePrint Archive, Report 2014/301. http://eprint.iacr.org/2014/301. 2014.

[298] J. Zimmerman. “How to Obfuscate Programs Directly”. In: *EUROCRYPT 2015, Part II*. Ed. by E. Oswald and M. Fischlin. Vol. 9057. LNCS. Springer, Heidelberg, Apr. 2015, pp. 439–467. DOI: 10.1007/978-3-662-46803-6_15.