IMBALANCED CRYPTOGRAPHIC PROTOCOLS

by
Gijs Van Laer

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy

Baltimore, Maryland
October, 2022

Abstract

Efficiency is paramount when designing cryptographic protocols, heavy mathematical operations often increase computation time, even for modern computers. Moreover, they produce large amounts of data that need to be sent through (often limited) network connections. Therefore, many research efforts are invested in improving efficiency, sometimes leading to imbalanced cryptographic protocols. We define three types of imbalanced protocols, computationally, communicationally, and functionally imbalanced protocols.

Computationally imbalanced cryptographic protocols appear when optimizing a protocol for one party having significantly more computing power. In *communicationally imbalanced cryptographic protocols* the messages mainly flow from one party to the others. Finally, in *functionally imbalanced cryptographic protocols* the functional requirements of one party strongly differ from the other parties.

We start our study by looking into laconic cryptography, which fits both the computational and communicational category. The emerging area of laconic cryptography involves the design of two-party protocols involving a sender and a receiver, where the receiver's input is *large*. The key efficiency requirement is that the protocol communication complexity must be independent of the receiver's input size. We show a new way to build laconic OT based on the new notion of *Set Membership Encryption* (SME) – a new member in the area of laconic cryptography. SME allows a sender to encrypt to one recipient from a universe of receivers, while using a small digest from a large subset of receivers. A recipient is only able to decrypt

the message if and only if it is part of the large subset.

As another example of a communicationally imbalanced protocol we will look at NIZKs. We consider the problem of proving in zero-knowledge the existence of exploits in executables compiled to run on real-world processors.

Finally, we investigate the problem of constructing law enforcement access systems that mitigate the possibility of unauthorized surveillance, as a functionally imbalanced cryptographic protocol. We present two main constructions. The first construction enables *prospective* access, allowing surveillance only if encryption occurs after a warrant has been issued and activated. The second allows *retrospective* access to communications that occurred prior to a warrant's issuance.

Thesis Readers

Dr. Matthew D. Green (Primary Advisor)
Associate Professor
Department of Computer Science
Johns Hopkins University

Dr. Abhishek Jain
Associate Professor
Department of Computer Science
Johns Hopkins University

Dr. J. Ayo Akinyele
CEO/Co-Founder
Bolt Labs Holdings, Inc.

Acknowledgments

The road to graduating in a PhD program is very different for everyone pursuing this goal. The reason I started this path is because when you want to work on applied cryptography, at this level of detail, there is no easy way to do this. The few companies working on applied cryptography almost always require a PhD, so it has been on my list of goals for a while. Even though I do not necessarily want to work at such company, it is out of pure interest that I wanted to get my hands dirty with real applied cryptography. Besides the real research work, I had the good fortune to get the chance to implement some of the cryptographic protocols as part of that research and during my work as a cryptographic engineer at Bolt Labs, which only further enriched my knowledge in this field. The real hard-core research has not always been (and maybe still isn't) my forte, but when I have to dive into the details of a construction of some cryptographic primitive based on some mathematical equations, I really enjoy it. So, if you ask me, has this part of my career been hard? Yes. Did I enjoy it and would I do it again? Absolutely. Nevertheless, I am happy to be at the end of this journey. New opportunities will open up in the near future, different challenges, new things to learn.

Of course, this is a pursuit that you cannot do on your own. I was extremely lucky to be advised by Matthew Green. Even though I sometimes needed days, if not weeks to understand one of your passionate explanations about some esoteric cryptographic system, I've learned a lot by doing so. Thank you for being patient with me while I was working through these things. Next, it was Abhishek Jain's enthusiasm in some of the problems I've been working

on that kept me going, his insights and broad knowledge of the field were invaluable. I would like to thank you both for the amazing guidance, learnings, and opportunities.

Thank you to my friend and coach, Roel Buyzen, without whom I would have crashed and burned long time ago. Thanks for all the amazing conversations about how to "just get the work done", or the conversations that were just more monologues; me complaining and ordering my thoughts. All these conversations were indispensable.

I would also like to thank Gabriel Kaptchuk, whom I have had the opportunity to work with on almost all of the projects and papers I worked on. Your relentless enthusiasm and never tiring in all of your work, never less than 5 projects on your plate at the same time. Likewise, I would like to thank the amazing colleagues in the lab that were always open for a good old nerdy conversation about deep cryptographic details, Arka Choudhuri, Gabrielle Beck, Alishah Chator, Aarushi Goel, Zhenzhong Jin, Tushar Jois, Stephan Kemper, and Max Zinkus. And more recently, and unfortunately due to the pandemic we had less chance to collaborate, but also a thank you to Harry Eldridge, Aditya Hegde, and Pratyush Tiwari.

Without the great team at Bolt Labs where I had the opportunity to apply the research directly into practice, I definitely would not have made the end of this PhD. Thank you to Ayo Akinyele for giving me this opportunity, and being a mentor along the way. Thanks to Colleen Swanson, Marcella Hastings, Solomon Akinyele, San Tran, and Vidya Akavoor, for being such a great team and always open for a discussion about cryptography, code, how to run a business, or just any random subject.

Also, thanks to my business partner, Lars Veelaert, for listening to me rambling about some cryptography from time to time. For building a business together while I was still on this path to graduation, for patiently waiting for me to be done with this PhD. Well, I am ready now, let's build this successful startup that we have been talking about and working on.

Last, but definitely not least, I would like to thank my friends and family for their support, in particular, my wife, Cathy, without whom I would not even have made it to the US for starting the PhD. You've always supported me unconditionally, and would take care of our children as the best mother in the world while I had to work on this little dissertation. Also, thanks to my two kids, Linus and Juno, who, let's be honest, actually wrote this dissertation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the design of cryptographic protocols, efficiency is paramount. Often, the heavy mathematical computations that are involved when performing a cryptographic protocol lead to very high computation time, this pushes modern computers to their limits. This problem gets exacerbated when the same cryptographic computation needs to be performed multiple times. Moreover, some protocols output large chunks of data that often have to be sent to another party over a (sometimes limited) network connection. Therefore, much research effort is focused on optimizing both the communication and computational cost of protocols. Often, a trade-off needs to be made between communication and computational cost, where a certain choice for one or the other can highly depend on the specific applications of the protocol.

In exploratory initial work, many protocols are optimized in such a way that *asymptotic computational efficiency* is minimized, some of the research includes lower bounds on said efficiency. In practice, however, finding the best asymptotic solution is not always desirable. Often, new implementations of existing protocols are found to have worse asymptotic efficiency, but orders of magnitude better real-world efficiency. In this work, we are more interested in the concrete optimizations, rather than the asymptotic efficiency of the protocols.

Different participants of a cryptographic protocol can have very different requirements,

needs, and limitations. For example one of the parties can have significantly more computation power, more storage, or faster network connection. Moreover, a participant can have very different security or privacy needs than the others. Many cryptographic protocols are specifically optimized for such settings, therefore, we introduce the term of *imbalanced cryptographic protocols* to describe the protocols that accommodate these different or imbalanced expectations of the participants. Imbalanced protocols try to strike the exact trade-off between theoretical optimizations and concrete efficiency in order to optimize the protocol in a real-world situation. Furthermore, we define the following three categories of imbalanced protocols:

Computationally imbalanced protocols: In this setting, one of the participants has significantly more computation power. Therefore, it makes sense to optimize a protocol by moving the heaviest part of the protocol to the participant with the most computing power, *e.g.* in the common case of running a protocol between a client and server. Here, the server often has large computational capacity while the client can be more limited, *e.g.* a mobile phone. We introduce the term *computationally imbalanced cryptographic protocol* to describe this type of protocols. This could lead to a theoretically less efficient protocol, but can be much more practical given this difference in computing power.

Communicationally imbalanced protocols: Also communicationally, protocols can be very imbalanced, meaning most of the data flows from one party to the others and not necessarily in the other direction. The most extreme example is a non-interactive protocol. In that case, there is only one round of communication going from one party to another, and no data is flowing back. Nevertheless, a protocol with many rounds can still be imbalanced when the difference in the amount of data that is flowing from one party to the other is significantly different.

Functionally imbalanced protocols: The last category is a little different, as this is

not necessarily about the efficiency of a protocol. But rather about the abilities of the different parties with respect to the data that has been shared. For example, you can imagine a protocol where a certain party can decrypt a ciphertext without restrictions, while the other party needs to comply with certain extra rules or can only obtain a function over the plaintext and not the plaintext itself.

Note that these types of protocols are merely a categorization to talk about general properties of the protocols. They are not strictly defined by certain numbers or bounds, but allow us to understand the nature of certain optimizations within protocol research. Nevertheless, we look into specific instances of these protocols and their applications in this dissertation, examining some of the properties that these protocols can have. Now that we have defined these categories, we study different protocols that are part of one or more of these categories.

We explore these different categories based on three different protocols and their applications. First, we look into laconic cryptography which is situated within two of our categories, computationally and communicationally imbalanced protocols. Next, we look at non-interactive proofs of knowledge of an exploit in software binaries for real-world processors as an application of a communicationally imbalanced protocol. Finally, we look at a functionally imbalanced protocol by looking at abuse resistant law enforcement access systems. Now, we briefly introduce all three of these protocols and their applications.

1.1 Laconic Cryptography

First, we look at the area of laconic cryptography, an area that tries to excel in both the computational as well as the communicational imbalanced categories. Recently, Cho *et al.* [59] introduced the elegant notion of laconic oblivious transfer, in laconic cryptography researchers try to heavily optimize the communication for one of the parties in an interactive protocol even if that party has a large input to the protocol. The first laconic protocol is laconic

Oblivious Transfer (ℓOT). Here, a receiver has a large database of selection bits and a sender has two labels for every position in the database. Based upon the selection bit at every position in the database, the sender would like to send one of the two labels. Furthermore, without the receiver revealing their selection bits and without the sender revealing the second label corresponding to the opposite bit. To make this protocol laconic, the receiver should only send little information to the sender, *i.e.* much smaller than the full size of the database. Moreover, the sender's computation time should be kept low as well, *i.e.* independent of the receiver's database size.

Later, other laconic protocols were introduced such as laconic private set intersection [8, 72, 73, 91, 168], where the receiver tries to find the intersection of their large set with a smaller set from a sender, again without revealing any extra information except the intersection itself. Similarly, to make this protocol laconic, the goal is to keep the communication from receiver to sender small and the senders computation time low.

In this dissertation we look specifically at improving laconic Oblivious Transfer. Previous improvements to regular Oblivious Transfer, called OT extensions, focused around reducing the computational complexity of multiple OT interactions, but still require interactive communication that grows linearly with the number of invocations. Laconic Oblivious Transfer tries to also reduce that communicational overhead, starting with the work of Cho *et al.*. However, the work of Cho *et al.* left much room for improvement. This is due to their use of complex garbled circuits. This only leads to a theoretical and asymptotic optimized solution, but nowhere near practical, even though the underlying primitive was improved as part of several follow up works [47, 62, 70, 71, 90, 92, 107].

Later, the work of Goyal *et al.* [108] and Alamati *et al.* [9] introduced a new balance in asymptotic efficiency leading towards a concretely efficient construction of laconic OT.

Set Membership Encryption. In Chapter 3, we introduce a new laconic primitive called

set membership encryption (SME). In this primitive, which can be viewed as a derivative of a public key broadcast encryption system, a sender can encrypt a message to a receiver, but takes an extra value as input to the encryption function. This value is a digest of a subset of the full set of possible receivers. The receiver can decrypt the message if and only if they were part of the subset that was used to compute the digest. The security property captures exactly this notion by disallowing decryption when the receiver is not in the set, even when the secret keys of all parties are revealed to the adversary, which makes this quite an interesting and strong primitive. We show how to construct this primitive based on two specific broadcast encryption (BE) systems. The BE scheme has double purpose and we have to use it as an identity-based encryption scheme as well to achieve our specific set membership security property.

Next, we show how we can use SME to construct laconic OT. Therefore, we seem to be able to leverage BE and IBE to construct SME. However, we stress that this construction is making non-black box use of the primitives, it does not appear to be possible to generalize this transformation. Nevertheless, the resulting schemes turn out to be quite efficient in comparison with previous work.

We compare the efficiency of our derived laconic OT construction with previous work on laconic OT. Compared to Cho *et al.*, our scheme shows an enormous improvement, which is in line with Goyal *et al.* and Alamati *et al.*. However, we also show a significant concrete improvement over these works.

1.2 Efficient Non-Interactive Proofs

Second, we will be looking into protocols that reduce the communication cost to achieve a communicationally imbalance protocol. Communication cost can be reduced in two ways: on one hand, one can reduce the number of rounds of communication, i.e. reducing the number

of messages going from one party to another. The ultimate goal of this reduction being one round of communication: this is called a *non-interactive protocol*. Although, non-interactivity obviously cannot be achieved for every protocol, it can be achieved in unexpected places. The best known example of a non-interactive protocol is in the setting of zero-knowledge proofs. Here, a prover can convince a verifier of the veracity of a statement without revealing the witness for that statement. Even without changing the number of rounds, a second way to optimize communication is by reducing the *size* of the messages. Again, theoretically this can be done by trying to achieve lower asymptotic efficiency, but this is not always the best concrete solution. We won't be studying how to construct such non-interactive protocol, but we will look into a very practical application, *i.e.* proving knowledge of an exploit in an existing binary for a real-world processor.

In Chapter 4 of this work, we will look at an extreme case of a communicationally imbalanced cryptographic protocol, by optimizing the number of communication rounds to only one, leading to a non-interactive protocol. The most famous of such protocols is the non-interactive zero knowledge proof (NIZK), which has been studied thoroughly and is still a very active field of research. In these protocols, a prover tries to convince a receiver of the fact that a statement is in an NP-language, without revealing anything about the witness. The communication only happens from the prover to the receiver to achieve non-interactiveness.

Efficient Proofs of Software Exploitability for Real-world Processors. Instead of studying general NIZKs or improving the existing NIZK protocols, we take a very practical approach to see how very complicated versions of such proofs are indeed practical. We consider the problem of proving in zero-knowledge the existence of vulnerabilities in executables compiled to run on real-world processors. We demonstrate that it is practical to prove knowledge of real exploits for real-world processor architectures without the need for source code and without limiting our consideration to narrow vulnerability classes. To achieve

this, we devise a novel circuit compiler and a toolchain that produces highly optimized, non-interactive zero-knowledge proofs for programs executed on the MSP430, an ISA commonly used in embedded hardware. Our toolchain employs a highly optimized circuit compiler and a number of novel optimizations to construct efficient proofs for program binaries. We use techniques that were set out by Ben-Sasson *et al.* [29, 30, 32], as well as the KKW [132] MPC-in-the-head based proof system, to build such practical NIZK prover and verifier. To demonstrate the capability of our system, we test our toolchain by constructing proofs for challenges in the Microcorruption capture the flag exercises [3].

1.3 Abuse Resistant Law Enforcement Access Systems

Now that we have explored two protocols in more detail, a laconic OT system and the practicality of a NIZK system, we can look at a way of combining several protocols including the once studied before. By doing so we are looking into functionally imbalanced protocols. Rather than optimizing for efficiency, we try to build a protocol where all different parties have very different requirements and expectations of the protocol.

The increasing deployment of end-to-end encrypted communications services has ignited a debate between technology firms and law enforcement agencies over the need for lawful access to encrypted communications. Unfortunately, existing solutions to this problem suffer from serious technical risks, such as the possibility of operator abuse and theft of escrow key material. In this work we investigate the problem of constructing law enforcement access systems that mitigate the possibility of unauthorized surveillance. We first define a set of desirable properties for an abuse-resistant law enforcement access system (ARLEAS), and motivate each of these properties. We then formalize these definitions in the Universal Composability (UC) framework [53], and present two main constructions that realize this definition. The first construction enables *prospective* access, allowing surveillance only if

encryption occurs after a warrant has been issued and activated. The second, more powerful construction, allows *retrospective* access to communications that occurred prior to a warrant's issuance. To illustrate the technical challenge of constructing the latter type of protocol, we conclude by investigating the minimal assumptions required to realize these systems.

Prospective ARLEAS. As mentioned, in the prospective case, messages send through an end-to-end encrypted messaging system can be accessed by law enforcement when they have a valid warrant for these specific messages at the time that they are sent. It is clear that this relaxes the requirements a little in comparison with access to all messages, including messages that were sent in the past. Therefore, it is to be expected that this primitive is going to be easier to construct than the more general retrospective case. Indeed, we show two possible ways to construct such prospective ARLEAS system.

We first show that this can be constructed efficiently using lossy encryption and efficient simulation sound NIZKs, with the limitation that warrants must explicitly specify the identities of users being targeted for surveillance. We then show a generalization of this construction such that warrants can be arbitrary predicates to be evaluated over each message metadata; our construction of this generalization relies on non-interactive secure computation [126]. It is in these constructions that we see the benefits of previous chapters, where practicality of large NIZKs becomes important, and the use of laconic OT can improve non-interactive secure computation.

Retrospective ARLEAS. We show how to realize ARLEAS that admits *retrospective* access, while still maintaining the auditability and detectability requirements of the system. The novel idea behind our construction is to use secure *proof-of-publication ledgers* to condition cryptographic escrow operations. The cryptographic applications of proof-of-publication ledgers have recently been explored (under slightly different names) in several works [60, 106, 130, 175]. Such ledgers may be realized using recent advances in consensus

networking, a subject that is part of a significant amount of research.

Lower Bound. Finally, we investigate the *minimal* assumptions for realizing retrospective access in an accountable law enforcement access system. As a concrete result, we present a lower-bound proof that any protocol realizing retrospective ARLEAS implies the existence of an extractable witness encryption scheme for some language $\mathcal{L}$ which is related to the ledger functionality and policy functions of the system. While this proof does not imply that all retrospective ARLEAS realizations require extractable witness encryption for general languages (*i.e.,* it may be possible to construct languages that have trivial EWE realizations), it serves as a guidepost to illustrate the barriers that researchers may face in seeking to build accountable law enforcement access systems.

1.4 Organization of This Work

In Chapter 2 we will introduce the necessary notation and definitions needed for the other chapters, this includes the introduction of some assumptions regarding bilinear groups. Next, in Chapter 3, we talk about laconic cryptography, introduce the new notion set membership encryption, and show how to build laconic OT from it. Then, in Chapter 4, we investigate how to build proofs for the knowledge of exploits in binaries compiled for real-world processors. And finally, we introduce law enforcement access systems that mitigate the possibility of unauthorized surveillance in Chapter 5.

Chapter 2

Preliminaries

Before diving into the main chapters of this dissertation, we introduce some common notation, definitions, and assumptions.

Notation. Let λ be an adjustable security parameter and $\mathsf{negl}(\lambda)$ be a negligible function in λ. We use $\|$ to denote concatenation, $\overset{c}{\approx}$ to denote computational indistinguishability. We will write $x \leftarrow \mathsf{Algo}(\cdot)$ to say that x is a specific output of running the algorithm Algo on specific inputs and will write $x \in \mathsf{Algo}(\cdot)$ to indicate that x is an element in the output distribution of Algo, when run with honest random coins. We write $\mathsf{Algo}^{\mathsf{Par}}$ to say that the algorithm Algo is parameterized by the algorithm Par. We write Algo^{D} to say that the algorithm Algo has random read access to the set D. We denote $[n] = \{1, \ldots, n\}$ and for a bit b we write $\bar{b}$ to denote $1 - b$. We will write $x \xleftarrow{\$} S$ when x gets randomly sampled from the set S, we assume the sampling is uniformly random unless otherwise specified. For ease of presentation, in all asymptotic efficiency notations we will ignore the security parameter and will assume it appears in all of them. We use $[b]$ for a share of a bit b, similarly we will use $[\![x]\!]$ for an arithmetic share of an element x in the respective arithmetic ring.

2.1 Definitions

In this section we show a few definitions that we will be needing in the main chapters of this dissertation.

2.1.1 Laconic Oblivious Transfer

We now give the following formal definition as presented by Cho *et al.*, we add the requirement that the size of the database is known when generating the common reference string, as well as some changes to the efficiency requirements.

Definition 1 (laconic OT) *A laconic OT (ℓOT) scheme consists of four algorithms* crsGen, Hash, Send, *and* $\mathsf{Receive}$.

$\mathsf{crsGen}(1^\lambda, \ell) \to \mathsf{crs}$. *This algorithm takes as input the security parameter λ and the size of the database ℓ.*

$\mathsf{Hash}(\mathsf{crs}, D) \to (\mathsf{digest}, \hat{D})$. *This algorithm takes as input a common reference string* crs *and a database $D \in \{0,1\}^\ell$ and outputs a digest* digest *of the database and a state $\hat{D}$.*

$\mathsf{Send}(\mathsf{crs}, \mathsf{digest}, L, m_0, m_1) \to c$. *This algorithm takes as input a common reference string* crs*, a digest* digest*, a database location $L \in [\ell]$, and two labels m_0 and m_1. It outputs a ciphertext c.*

$\mathsf{Receive}^{\hat{D}}(\mathsf{crs}, c, L) \to m$. *This algorithm takes as input a common reference string* crs*, a ciphertext c, and a database position L. Moreover, it has random read access to $\hat{D}$. It outputs a label m.*

This scheme should have the following properties:

Correctness: *For any database D of size $\ell = \mathsf{poly}(\lambda)$, for any polynomial function $\mathsf{poly}(\cdot)$, any database location $L \in [\ell]$, and any pair of labels $(m_0, m_1) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$, it*

holds that

$$\Pr\left[m = m_{D[L]} \;\middle|\; \begin{array}{ccc} \mathsf{crs} & \leftarrow & \mathsf{crsGen}\left(1^\lambda, \ell\right) \\ \left(\mathsf{digest}, \hat{D}\right) & \leftarrow & \mathsf{Hash}\left(\mathsf{crs}, D\right) \\ c & \leftarrow & \mathsf{Send}\left(\mathsf{crs}, \mathsf{digest}, L, m_0, m_1\right) \\ m & \leftarrow & \mathsf{Receive}^{\hat{D}}\left(\mathsf{crs}, c, L\right) \end{array}\right] = 1,$$

where the probability is taken over the random choices made by crsGen *and* Send.

Sender Privacy Against Semi-Honest Receivers: *There exists a PPT simulator* $\mathcal{S}$ *such that for any database of size at most* $\ell = \mathsf{poly}(\lambda)$ *for any polynomial function* $\mathsf{poly}(\cdot)$*, any memory location* $L \in [\ell]$*, and any pair of labels* $(m_0, m_1) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$*, let* $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda, \ell)$ *and* $\mathsf{digest} \leftarrow \mathsf{Hash}(\mathsf{crs}, D)$*, it holds that*

$$\left(\mathsf{crs}, \mathsf{Send}\left(\mathsf{crs}, \mathsf{digest}, L, m_0, m_1\right)\right) \overset{c}{\approx} \left(\mathsf{crs}, \mathcal{S}\left(D, L, m_{D[L]}\right)\right).$$

Receiver Privacy: *There exists a PPT simulator* $\mathcal{S}$ *such that for any database of size at most* $\ell = \mathsf{poly}(\lambda)$ *for any polynomial function* $\mathsf{poly}(\cdot)$*, let* $\mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda, \ell)$*, it holds that*

$$\left(\mathsf{crs}, \mathsf{Hash}(\mathsf{crs}, D)\right) \overset{c}{\approx} \left(\mathsf{crs}, \mathcal{S}\left(1^\lambda\right)\right).$$

Efficiency: *The length of* digest *is a fixed polynomial in* λ*, independent of the size of the database. Moreover, the algorithm* Hash *runs in time* $|D| \cdot \mathsf{poly}\left(\log|D|, \lambda\right)$*,* Send *runs in time* $\mathsf{poly}\left(\log|D|, \lambda\right)$*, and* Receive *runs in time* $\mathcal{O}\left(|D|, \lambda\right)$*.*[1]

2.1.2 Laconic Private Set Intersection

We now give the following formal definition for a laconic private set intersection scheme as presented by Alamati *et al.* [9].

Definition 2 (laconic PSI) *A laconic private set intersection (ℓPSI) scheme consists of four algorithms* $\mathsf{crsGen}, R_1, S,$ *and* R_2*.*

[1]We slightly relax the efficiency in comparison with the definition given in Cho et al. [59] This slightly worse asymptotic receiver time results in constructions with much better concrete efficiency.

$\mathsf{crsGen}(1^\lambda, \ell) \to \mathsf{crs}$. *This algorithm takes as input the security parameter* λ *and the maximum size of the receiver set* ℓ.

$R_1(\mathsf{crs}, S_R) \to (\mathsf{digest}, \mathsf{st})$. *This algorithm takes as input a common reference string* crs *and a receiver set* S_R *and outputs a digest* digest *of the database and state* st.

$S(\mathsf{crs}, \mathsf{digest}, S_S) \to c$. *This algorithm takes as input a common reference string* crs, *a digest* digest, *and a sender set* S_S. *It outputs a ciphertext* c.

$R_2(\mathsf{crs}, c, \mathsf{st}) \to \mathcal{I}$. *This algorithm takes as input a common reference string* crs, *a ciphertext* c, *and state* st. *It outputs a set* $\mathcal{I}$.

This scheme should have the following properties:

Correctness: *For all* $S_R, S_S \subseteq [\ell]$, *we have*

$$\Pr\left[S_R \cap S_S = R_2(\mathsf{crs}, c, \mathsf{st}) \;\middle|\; \begin{array}{cc} \mathsf{crs} & \leftarrow \mathsf{crsGen}(1^\lambda, \ell) \\ (\mathsf{digest}, \mathsf{st}) & \leftarrow R_1(\mathsf{crs}, S_R) \\ c & \leftarrow S(\mathsf{crs}, \mathsf{digest}, S_S) \end{array} \right] = 1.$$

Efficiency: *There exists a fixed polynomial* $\mathsf{poly}(\cdot)$ *such that the length of* digest *and the running time of* S *are at most* $\mathsf{poly}(\lambda, \log |S_R|)$.

Security is defined in Canetti's UC framework [53], by using the ideal functionality defined in Figure 2-1.

PSI functionality ($\mathcal{F}_{\mathrm{PSI}}$).

The functionality $\mathcal{F}_{\mathrm{PSI}}$ is parameterized by a universe Ω

Setup phase R sends (sid, S_R) to $\mathcal{F}_{\mathrm{PSI}}$ where $S_R \subseteq \Omega$. It ignores future messages from R with the same sid.

Send phase S sends $(\mathsf{sid}, i, S_S \subseteq \Omega)$ to $\mathcal{F}_{\mathrm{PSI}}$. $\mathcal{F}_{\mathrm{PSI}}$ sends $(\mathsf{sid}, i, S_R \cap S_S)$ to R. It ignores future messages from S with the same sid and $i \in N$

Figure 2-1. The ideal functionality for laconic PSI ($\mathcal{F}_{\mathsf{PSI}}$)

2.1.3 Broadcast Encryption

Broadcast encryption systems [82] allow for a sender to broadcast encrypted messages on a broadcast channel, while the message is only intended for a dynamically chosen subset $S \subseteq [n]$ of all users. The system will assure that a ciphertext is created that can only be decrypted by anyone in the set S.

For completeness we show the definition for adaptive broadcast encryption based on the definition shown by Gentry and Waters [94].

Definition 3 (Broadcast Encryption) *A broadcast encryption scheme consists of four randomized algorithms:*

- $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}\left(1^\lambda, n\right)$. *It takes as input the security parameter λ and the maximum number of receivers n. It outputs a public key* pk *and a master secret key* msk.
- $C \leftarrow \mathsf{Encrypt}\left(\mathsf{pk}, M, S\right)$. *This algorithm takes as input a public key* pk, *a message M, and a subset $S \subseteq [n]$. It outputs a ciphertext C, which can be broadcasted.*
- $K_k \leftarrow \mathsf{KeyGen}\left(k, \mathsf{pk}, \mathsf{msk}\right)$. *This algorithm takes as input an index $k \in [n]$, a public key* pk, *and a master secret key* msk. *It outputs a private key K_k.*
- $M \leftarrow \mathsf{Decrypt}\left(\mathsf{pk}, K, S, k, C\right)$. *This algorithm takes as input a public key* pk, *a private key K, a subset $S \subseteq [n]$, an index $k \in [n]$, and the ciphertext C. It outputs a plaintext message M.*

This scheme should have the following properties:

Correctness: *For all $S \subseteq [n]$ and all $i \in S$, then*

$$\Pr\left[M = \mathsf{Decrypt}\left(\mathsf{pk}, K, S, i, C\right) \middle| \begin{array}{rl} (\mathsf{pk}, \mathsf{msk}) & \leftarrow \mathsf{Setup}\left(1^\lambda, n\right) \\ C & \leftarrow \mathsf{Encrypt}\left(\mathsf{pk}, M, S\right) \\ K & \leftarrow \mathsf{KeyGen}\left(i, \mathsf{pk}, \mathsf{msk}\right) \end{array}\right] = 1.$$

Adaptive Security Game for Broadcast Encryption $\mathbf{Game}_{\mathcal{A},\mathrm{BE},n}(\lambda)$.

Setup. The challenger runs $\mathsf{Setup}\left(1^{\lambda}, n\right)$ to obtain a public key pk and master secret key msk, it hands out the public key to the adversary $\mathcal{A}$.

Key Query Phase. Adversary $\mathcal{A}$ adaptively issues private key queries for indices $i \in [n]$.

Challenge. The adversary then specifies a challenge set S^*, such that for all private keys queried we have that $i \notin S^*$ and two messages M_0 and M_1. The challenger picks $b \xleftarrow{\$} \{0,1\}$ sets $C \leftarrow \mathsf{Encrypt}\left(\mathsf{pk}, M_b, S^*\right)$ and gives this to $\mathcal{A}$.

Guess. The adversary $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$ for b and wins the game if $b = b'$.

Figure 2-2. Adaptive security game for broadcast encryption.

Adaptive security: *For all PPT algorithms $\mathcal{A}$ we have that*

$$\left|\Pr\left[b = b' | (b, b') \leftarrow \boldsymbol{Game}_{\mathcal{A},BE,n}\left(\lambda\right)\right] - \frac{1}{2}\right| \leq \epsilon\left(\lambda\right),$$

with $\epsilon(\cdot)$ a negligible function and $\boldsymbol{Game}_{\mathcal{A},BE,n}$ the security game as described in Figure 2-2.

2.1.4 Proof-of-publication ledgers

Our work makes use of a public append-only ledger that can produce a publicly-verifiable *proof of publication*. This concept was formalized by Goyal *et al.* [106], Choudhuri *et al.* [60], and Kaptchuk *et al.* [130], but related ideas have also been previously used by Liu *et al.* to realize time-lock encryption [144]. Plausible candidates for such ledgers have been the subject of great interest, due to the deployment of blockchains and other consensus networks [152]. Significant work has been done to formalize the notion of a public, append-only ledger [18, 19, 61, 106] and study its applications to cryptographic protocols [11, 36, 60]. This work uses a simplified ledger interface formalized in [60] that abstracts away details such as timing information

and temporary inconsistent views that are modeled in [18]. However, this simplified view captures the *eventual* functionality of the complex models, and is therefore equivalent for our purposes.

The ledger ideal functionality is provided in Figure 2-3. This functionality allows users to post arbitrary information to the ledger; this data is associated with a particular index on the ledger, with which any user can retrieve the original data as well as a proof of publication. For security, our functionality encodes a notion we refer to as *ledger unforgeability*, which requires that there exists an algorithm to verify a proof that a message has been posted to the ledger, and that adversaries cannot forge this proof.

Functionality $\mathcal{L}^{\mathsf{Verify}}$

GetCounter.: Upon receiving $(\mathsf{GetCounter})$ from any party, return ℓ.

Post.: Upon receiving (Post, x), the trusted party increments ℓ by 1, computes the proof of publication π_{publish} on $(\ell||x)$ such that $\mathsf{Verify}((\ell\|x), \pi_{\mathsf{publish}}) = 1$. Add the entry $(\ell, x, \pi_{\mathsf{publish}})$ to the entry table T. Respond with $(\ell, x, \pi_{\mathsf{publish}})$

GetVal.: Upon receiving (GetVal, ℓ), check if there is an entry $(\ell, x, \pi_{\mathsf{publish}})$ in the entry table T. If not, return $\perp$. Otherwise, return $(\ell, x, \pi_{\mathsf{publish}})$.

Figure 2-3. Ideal functionality for a proof-of-publication ledger, from [60].

2.1.5 Authenticated Communication

We use a variant of Canetti's ideal functionality for authenticated communication, $\mathcal{F}_{AUTH}$, to abstract the notion of message authentication [53]. This is presented in Figure 2-4. Since we restrict our analysis to static corruption, we simplify this functionality to remove the adaptive corruption interface.[2]

[2]Note that this ideal functionality only handles a single message transfer, but to achieve multiple messages, we rely on universal composition and use multiple instances of the functionality.

Functionality $\mathcal{F}_{AUTH}$

Sending: Upon receiving an input $(\mathsf{Send}, \mathsf{sid}, P_j, m)$ from P_i, if it is *not* the first instance of $(\mathsf{Send}, \cdot, \cdot, \cdot)$, the ideal functionality does nothing. Otherwise, it sends $(\mathsf{Sent}, \mathsf{sid}, P_i, P_j, m)$ to the adversary. If the adversary responds with (OK), then the ideal functionality sends $(\mathsf{Sent}, \mathsf{sid}, P_i, P_j, m)$ and halts. If the adversary does not approve, the ideal functionality drops the message and halts.

Figure 2-4. The message authentication ideal functionality $\mathcal{F}_{AUTH}$ supporting static corruption, adapted from [53].

2.1.6 Simulation Extractable Non-Interactive Zero Knowledge

In our protocols we require non-interactive zero knowledge proofs of knowledge that are simulation extractable. To preserve space, we refer the reader to the definitions of Sahai [172] and De Santis *et al.* [67]. Rather than rely on UC functionalities, we employ a NIZK directly in our protocols.

Definition 4 (Simulation Extractable Non-Interactive Zero Knowledge) A non-interactive
zero-knowledge proof of knowledge for a relation $\mathcal{R}$ is a set of algorithms $(\mathsf{ZKSetup}, \mathsf{ZKProve}, \mathsf{ZKVerify}, \mathsf{ZKSimulate})$ defined as follows:

$\mathsf{ZKSetup}(1^\lambda)$ returns the common reference string and simulation trapdoor $(\mathsf{CRS}_{ZK}, \tau)$.

$\mathsf{ZKProve}(\mathsf{CRS}_{ZK}, x, \omega)$ takes in the common reference string CRS_{ZK}, a statement x and a witness ω and outputs a proof π.

$\mathsf{ZKVerify}(\mathsf{CRS}_{ZK}, x, \pi)$ takes in the common reference string CRS_{ZK}, a statement x and a proof π, and outputs either 1 or 0.

$\mathsf{ZKSimulate}(\mathsf{CRS}_{ZK}, \tau, x)$ takes in the common reference string CRS_{ZK}, a statement x and the simulation trapdoor τ and outputs a proof π.

We say that a non-interactive zero-knowledge argument of knowledge is simulation extractable if it satisfies the following properties:

Completeness: If a prover has a valid witness, then they can always convince the verifier. More formally, for all relations $\mathcal{R}$ and all x, ω, if $\mathcal{R}(x, \omega) = 1$, then

$$\Pr\left[\mathsf{ZKVerify}(\mathsf{CRS}_{ZK}, x, \pi) = 1 \,\middle|\, (\mathsf{CRS}_{ZK}, \tau) \leftarrow \mathsf{ZKSetup}(1^\lambda), \pi \leftarrow \mathsf{ZKProve}(\mathsf{CRS}_{ZK}, x, \omega)\right] = 1$$

Perfect Zero knowledge: A scheme has zero-knowledge if a proof leaks no information beyond that truth of the statement x. We formalize this by saying that an adversary with oracle access to a prover cannot tell if that prover runs the honest algorithm ZKProve or uses the trapdoor and ZKSimulate.

$$\Pr\left[\mathcal{A}^{\mathsf{ZKProve}(\mathsf{CRS}_{ZK}, \cdot, \cdot)}(\mathsf{CRS}_{ZK}) = 1 | (\mathsf{CRS}_{ZK}, \cdot) \leftarrow \mathsf{ZKSetup}(1^\lambda)\right] \overset{s}{\approx}$$
$$\Pr\left[\mathcal{A}^{\mathsf{ZKSimulate}(\mathsf{CRS}_{ZK}, \tau, \cdot)}(\mathsf{CRS}_{ZK}) = 1 | (\mathsf{CRS}_{ZK}, \tau) \leftarrow \mathsf{ZKSetup}(1^\lambda)\right]$$

Simulation Extractability: There exists an extractor Extract such that

$$\Pr\left[\begin{array}{c|c} \mathcal{R}(x, \omega) = 1 & \begin{array}{c} (\mathsf{CRS}_{ZK}, \tau) \leftarrow \mathsf{ZKSetup}(1^\lambda), \\ (x, \pi) \leftarrow \mathcal{A}^{\mathsf{ZKSimulate}(\mathsf{CRS}_{ZK}, \tau, \cdot)}(\mathsf{CRS}_{ZK}), \\ \omega \leftarrow \mathsf{Extract}(\mathsf{CRS}_{ZK}, \tau, x, \pi) \end{array} \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

It has been shown that realizing this primitive for languages outside BBP requires the common reference string model [97, 98, 160]. We present the common reference string ideal functionality from [54] in Figure 2-5.

2.1.7 Lossy Encryption

Lossy encryption [25] is an encryption application of lossy trapdoor functions, which were introduced by Peikert and Waters [165]. Intuitively, lossy encryption is a public key encryption scheme that has an algorithm to generate *lossy keys* that are computationally indistinguishable

Functionality $\mathcal{F}_{CRS}^{\mathcal{D}}$

$\mathcal{F}_{CRS}^{\mathcal{D}}$ proceeds as follows, when parameterized by a distribution $\mathcal{D}$:

Common Reference String: When activated for the first time on input $(\mathsf{CRS}, \mathsf{sid})$, choose a value $d \xleftarrow{\$} \mathcal{D}$ and send back to the activating party. In each other activation return the value d to the activating party.

Figure 2-5. Ideal functionality for generating a Common Reference String, from [54].

from normal keys. When encrypting with these lossy keys, the resulting ciphertext contains no information about the plaintext.

Definition 5 A lossy public-key encryption scheme is a tuple of algorithms $(\mathsf{KeyGen}, \mathsf{KeyGen}_{\mathsf{loss}}, \mathsf{Enc}, \mathsf{Dec})$ defined as

$\mathsf{KeyGen}(1^\lambda)$ generates an injective keypair $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{KeyGen}_{\mathsf{loss}}(1^\lambda)$ generates a lossy keypair $(\mathsf{pk}, \cdot)$

$\mathsf{Enc}(\mathsf{pk}, m)$ takes in a public key pk and a plaintext message m and outputs a ciphertext c

$\mathsf{Dec}(\mathsf{sk}, c)$ takes in a secret key sk and a ciphertext c and either outputs $\perp$ or the message m

We require that the above algorithms satisfies the following properties:

Correctness on real keys: For all messages m, it should hold that

$$\Pr\left[m = \mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) \,\middle|\, (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)\right] = 1$$

Lossiness on lossy keys: for all $(\mathsf{pk}, \cdot) \leftarrow \mathsf{KeyGen}_{\mathsf{loss}}(1^\lambda)$ and m_0, m_1 such that $|m_0| = |m_1|$,

$$\mathsf{Enc}(\mathsf{pk}, tag, m_0) \overset{s}{\approx} \mathsf{Enc}(\mathsf{pk}, tag, m_1)$$

Indistinguishability of keys: Finally, over all random coins, it should hold that

$$\mathsf{KeyGen}(1^\lambda) \overset{c}{\approx} \mathsf{KeyGen}_{\mathsf{loss}}(1^\lambda)$$

2.1.8 Multi-sender Non-interactive Secure Computation

When instantiating our prospective protocol for arbitrary predicates in Section 5.5.1,we will require the use of Non-interactive Secure Computation (NISC) [7, 126]. In NISC, a receiver can post an encryption embedding a secret x_1 such that senders with secret x_2 can reveal $f(x_1, x_2)$ to the receiver by sending only a single message. Realizing such a scheme (see [126]) is feasible in the CRS model [53, 55] from two-round, UC-secure malicious oblivious transfer [64, 157], Yao's garbled circuits [121, 199], and generic non-interactive zero knowledge (see Section 2.1.6). The resulting protocols, however, are very inefficient and require non-blackbox use of the underlying cryptographic primitives. While this is sufficient for our purposes, we note that depending on specific functionality required in an instantiation of ARLEAS, it may be possible to use more efficient constructions (*i.e.* depending on the size of the predicate circuit, etc.) Because the notation for NISC protocols varies, we fix it for this work below. We omit the ideal functionality of multi-sender NISC from [7], due to space constraints. Because we require non-blackbox use of the primitive, we will use it directly rather than as a hybrid.

Definition 6 (Multi-sender Non-interactive Secure Computation) *Given a garbling scheme for a functionality* $f : \{0,1\}^{\mathsf{input}_1} \times \{0,1\}^{\mathsf{input}_2} \to \{0,1\}^{\mathsf{output}}$*, i.e. a tuple of PPT algorithms* $\Pi_{NISC} := (\mathsf{GenCRS}, \mathsf{NISC}_1, \mathsf{NISC}_2, \mathsf{Evaluate})$ *such that*

$\mathsf{GenCRS}(1^\lambda, \mathsf{input}; r) \to (\mathsf{CRS}_{\mathsf{NISC}}, \tau_{\mathsf{NISC}})$*:* GenCRS *takes the security parameter* 1^n *and outputs a CRS, along with a simulation backdoor* τ_{NISC}*. When we explicitly need to specify the randomness, we will include it as* r *as here.*

$\mathsf{NISC}_1(\mathsf{CRS}_{\mathsf{NISC}}, x_1; r) \to (\mathsf{nisc}_1^{public}, \mathsf{nisc}_1^{private})$*:* NISC_1 *takes in the CRS and an input* $x \in \{0,1\}^{\mathsf{input}_1}$ *and outputs the first message* NISC_1*. When we explicitly need to specify the randomness, we will include it as* r *as here.*

$\mathsf{NISC}_2(\mathsf{CRS}_{\mathsf{NISC}}, f, x_2, \mathsf{nisc}_1^{public}; r) \rightarrow \mathsf{nisc}_2$*:* NISC_2 *takes in the CRS, a circuit* C*, an input* $x_2 \in \{0,1\}^{\mathsf{input}_2}$ *and the first garbled circuit message* nisc_1^{public}*. It outputs the second message* nisc_2*. When we explicitly need to specify the randomness, we will include it as* r *as here.*

$\mathsf{Evaluate}(\mathsf{CRS}_{\mathsf{NISC}}, \mathsf{nisc}_2, \mathsf{nisc}_1^{private})$*:* Evaluate *takes as input the second GC message* nisc_2 *along with the private information* $\mathsf{nisc}_1^{private}$ *and outputs* $y \in \{0,1\}^{\mathsf{output}}$ *or the error symbol* $\perp$

We give an ideal world security definition for a multi-sender NISC in Figure 2-6.

Multi-sender Non-interactive Secure Computation.

Receiver Posting: Upon receiving a message (Input_1, x_1) from P_1, store x_1. Initialize and empty table $\mathbb{T}$ and ignore future Input_1 messages.

Sender Input: Upon receiving a message (Input_2, x_2) from P_i, store (P_i, x_2) in table $\mathbb{T}$.

Outputs: Upon receiving a message $(\mathsf{Outputs})$ from P_1, send $(\{(P_i, f(x_1, x_2)\}_{(P_i,x_2)\in\mathbb{T}})$

Figure 2-6. Ideal functionality for multi-sender NISC, from [7].

2.1.9 Witness Encryption and Extractable Witness Encryption

Our *retrospective* constructions require extractable witness encryption (EWE) [46], a variant of witness encryption in which the existence of a distinguisher can be used to construct an extractor for the necessary witness [95]. While EWE is a strong assumption, in later sections of this work we show that it is a minimal requirement for the existence of retrospective ARLEAS.

Definition 7 (Extractable Witness Encryption) *An extractable witness encryption*

$\Pi_{EWE} = (\mathsf{Enc}, \mathsf{Dec})$ *for an NP language* $\mathcal{L}$ *associated with relation* R *consists of the following algorithms:*

$\mathsf{Enc}(1^\lambda, x, m)$*: On input instance* x *and message* $m \in \{0,1\}$*, it outputs a ciphertext* c*.*

$\mathsf{Dec}(c, w)$*: On input ciphertext* c *and witness* w*, it outputs* m'*.*

We require that the above primitive satisfy the following properties:

Correctness*: For every* $(x, w) \in R$*, for every* $m \in \{0,1\}$*,*

$$\Pr[m = \mathsf{Dec}(\mathsf{Enc}(1^\lambda, x, m), w)] = 1$$

Extractable Security*: For any PPT adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*, if:*

$$\Pr\left[\mathcal{A}_1(1^\lambda, c, \mathsf{state}) = b \;\middle|\; \begin{array}{c} b \leftarrow \{0,1\} \\ (m_0, m_1, \mathsf{state}) \leftarrow \mathcal{A}_0(1^\lambda, x) \\ c \leftarrow \mathsf{Enc}(1^\lambda, x, m_b) \end{array}\right] \geq \frac{1}{2} + \mathsf{negl}(\lambda)$$

then there exists a PPT extractor Ext *such that for all auxiliary inputs* aux*:*

$$\Pr\left[w \leftarrow \mathsf{Ext}_{\mathcal{A}}(1^\lambda, x, aux) \;\; s.t. \;\; : (x, w) \in R\right] \geq \mathsf{negl}(\lambda)$$

2.1.10 Programmable Global Random Oracle Model

The security proof for our retrospective construction makes use of the programmable global random oracle model, introduced in [51].

The ideal functionality $\mathcal{G}_{\mathsf{pRO}}$ is illustrated in Figure 2-7.

2.2 Assumptions

In this section we will introduce a few assumptions in composite order bilinear groups that we will be needing in a few of the proofs in Chapter 3.

Functionality $\mathcal{G}_{\mathsf{pRO}}$.

Initiate with an empty list $\mathsf{List}_{\mathcal{H}}$

Query: Upon receiving message $(\mathsf{HashQuery}, m)$ from party P, the ideal functionality proceeds as follows. Find h such that $(m, h) \in \mathsf{List}_{\mathcal{H}}$. If no such h exists, let $h \xleftarrow{\$} \{0,1\}^{\ell}$ and store (m, h) in $\mathsf{List}_{\mathcal{H}}$. Return $(\mathsf{HashConfirm}, h)$ to P.

Program: Upon receiving message $(\mathsf{ProgramRO}, m, h)$ from adversary $\mathcal{A}$, the ideal functionality proceeds as follows. If $\exists h' \in \{0,1\}^{\ell}$ such that $(m, h') \in \mathsf{List}_{\mathcal{H}}$ and $h \neq h'$, then abort. Otherwise, add (m, h) to $\mathsf{List}_{\mathcal{H}}$ and output $(\mathsf{ProgramConfirm})$ to $\mathcal{A}$

Figure 2-7. Ideal functionality for the global programmable random oracle, from [51].

2.2.1 Composite Order Bilinear Groups

Composite order bilinear groups were first introduced by Boneh *et al.* [42] These groups are defined by taking a group generator, which is an algorithm $\mathcal{G}$ that takes as input a security parameter λ and outputs a description of a bilinear group $\mathbb{G}$. This bilinear group is of composite order, which throughout this paper would be an order composed of three distinct primes p_1, p_2, and p_3, *i.e.* $(N = p_1 p_2 p_3, \mathbb{G}, \mathbb{G}_T, e)$. $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of order N and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a map such that:

1. (Bilinear) $\forall g, h \in \mathbb{G}, a, b \in \mathbb{Z}_N, e\left(g^a, h^b\right) = e\left(g, h\right)^{ab}$;
2. (Non-degenerate) $\exists g \in \mathbb{G}$ such that $e\left(g, g\right)$ has order N in $\mathbb{G}_T$.

Furthermore, all operations in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the bilinear map e should be computable in polynomial time with respect to the security parameter λ. Also note that the subgroups $\mathbb{G}_{p_1}, \mathbb{G}_{p_2}$, and $\mathbb{G}_{p_3}$ of respective order p_1, p_2, and p_3 are orthogonal to each other, *i.e.* $\forall g_i \in \mathbb{G}_{p_i}, g_j \in \mathbb{G}_{p_j}$ and $i \neq j$, $e\left(g_i, g_j\right) = 1$.

We will use the same assumptions as introduced by Lewko and Waters [141]. They prove that these assumptions hold in the generic group model using the general framework of Katz,

Sahai, and Waters [131]. Because our first SME construction is based on the construction of the IBE by Lewko and Waters [141], we can re-use the same assumptions.

Assumption 1 *Given a group generator $\mathcal{G}$:*

$$(N = p_1 p_2 p_3, \mathbb{G}, \mathbb{G}_T, e) \xleftarrow{\$} \mathcal{G}(\lambda),$$
$$g \xleftarrow{\$} \mathbb{G}_{p_1}, X_3 \xleftarrow{\$} \mathbb{G}_{p_3},$$
$$D = (g, X_3),$$
$$T_1 \xleftarrow{\$} \mathbb{G}_{p_1 p_2}, T_2 \xleftarrow{\$} \mathbb{G}_{p_1}.$$

We say this assumption holds if there exists a negligible function ϵ in the security parameter λ for any adversary $\mathcal{A}$ such that:

$$|\Pr[\mathcal{A}(D, T_1) = 1] - \Pr[\mathcal{A}(D, T_2) = 1]| \leq \epsilon(\lambda).$$

Assumption 2 *Given a group generator $\mathcal{G}$:*

$$(N = p_1 p_2 p_3, \mathbb{G}, \mathbb{G}_T, e) \xleftarrow{\$} \mathcal{G}(\lambda),$$
$$g, X_1 \xleftarrow{\$} \mathbb{G}_{p_1}, X_2, Y_2 \xleftarrow{\$} \mathbb{G}_{p_2}, X_3, Y_3 \xleftarrow{\$} \mathbb{G}_{p_3},$$
$$D = (g, X_1 X_2, X_3, Y_2 Y_3),$$
$$T_1 \xleftarrow{\$} \mathbb{G}, T_2 \xleftarrow{\$} \mathbb{G}_{p_1 p_3}.$$

We say this assumption holds if there exists a negligible function ϵ in the security parameter λ for any adversary $\mathcal{A}$ such that:

$$|\Pr[\mathcal{A}(D, T_1) = 1] - \Pr[\mathcal{A}(D, T_2) = 1]| \leq \epsilon(\lambda).$$

Assumption 3 *Given a group generator* $\mathcal{G}$*:*

$$(N = p_1 p_2 p_3, \mathbb{G}, \mathbb{G}_T, e) \xleftarrow{\$} \mathcal{G}(\lambda), \alpha, s \xleftarrow{\$} \mathbb{Z}_N$$

$$g \xleftarrow{\$} \mathbb{G}_{p_1}, X_2, Y_2, Z_2 \xleftarrow{\$} \mathbb{G}_{p_2}, X_3 \xleftarrow{\$} \mathbb{G}_{p_3},$$

$$D = (g, g^{\alpha} X_2, X_3, g^{s} Y_2, Z_2),$$

$$T_1 = e(g, g)^{\alpha s}, T_2 \xleftarrow{\$} \mathbb{G}_T.$$

We say this assumption holds if there exists a negligible function ϵ *in the security parameter* λ *for any adversary* $\mathcal{A}$ *such that:*

$$|\Pr[\mathcal{A}(D, T_1) = 1] - \Pr[\mathcal{A}(D, T_2) = 1]| \leq \epsilon(\lambda).$$

Chapter 3

Efficient Set Membership Encryption and Applications

This chapter is based on joint work with Matthew Green and Abhishek Jain.

3.1 Introduction

Recent developments in secure multiparty computation (MPC) have led to the increasing usage and deployment of the technology. This deployment has illustrated the need for further improvements in the efficiency of MPC protocols. One broad area of potential improvement has to do with the fixed cost needed for Oblivious Transfer (OT) [169]. OT is a fundamental cryptographic protocol and is foundational to the construction of MPC protocols [101, 125, 125, 134, 169, 198]. In practice, OT is a significant contributor to the overhead of MPC; moreover, the number of OT invocations typically increases with the size of the function inputs. This has motivated efficiency optimizations such as OT extension [23, 124, 138, 176]. While such improvements reduce the *computational* complexity of multiple OT interactions, they still require interactive communication that grows linearly with the number of invocations.

Recently Cho *et al.* [59] proposed a new primitive called *laconic Oblivious Transfer* (ℓOT) to address this communication cost. In laconic OT a receiver first produces a succinct digest

of a vector of selector bits. The sender then uses this digest to encrypt a corresponding database of message pairs. The critical property in this scheme is that the receiver must be able to decrypt exactly one message from each pair, corresponding to its previous selections. The key efficiency requirement are as follows: the digest size must be independent of the database size, while the sender's running time and the receiver's decryption time (for each position) must be poly-logarithmic in the size of the receiver's input.

Laconic OT has many promising applications and the instantiation proposed by Cho *et al.* is elegant and relies on well-studied cryptographic hardness assumptions. Unfortunately, it is far from practical. While asymptotically efficient, the proposed scheme includes substantial concrete overhead that makes it unusable for real-world deployment. In particular, the construction makes extensive use of elliptic curve scalar multiplications that are embedded within chains of sequential garbled circuits, resulting in enormous concrete bandwidth costs. Some optimizations have been proposed to improve this and related protocols [62]; however, even the optimized realizations are challenging to implement – let alone deploy for real-world applications [62, 184].

The impracticality stems from a focus on asymptotic efficiency instead of concrete efficiency. A recent line of work [9, 108] recognized this and proposed much more concretely efficient constructions by allowing for asymptotically larger decryption times – *linear*, as opposed to poly-logarithmic. These works also realize a related notion of *laconic private set intersection* (ℓPSI), where a sender and a receiver can compute the intersection of their respective sets in a manner such that the communication complexity of the protocol is independent of the size of the receiver's set. The work of Goyal *et al.* [108] presents constructions based on various number-theoretic assumptions, while Alamati *et al.* [9] presents a construction based on the ϕ-hiding assumption.

The main drawback of these works is the requirement of large decryption time, in concrete

Table 3-I. Overview of our asymptotic and concrete efficiency in comparison with Cho *et al.* [59], Goyal *et al.* [108], and Alamati *et al.* [9] for database size $n = 2^{31}$. (DDH = Decisional Diffie-Hellman assumption, q-DBDHI = q-Decisional Bilinear Diffie-Hellman inversion assumption, SDP = Subgroup Decision Problem in composite order bilinear groups, sBDHE = selective Bilinear Diffie-Hellman Exponentiation assumption)

	crs size	Hash size	Send size	crs size	Hash size	Send size	Receive	Assumption
Cho *et al.* [59]	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	4.6kB	48 bytes	1.2PB[3]	-	DDH
Goyal *et al.* [108]	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	103.1GB	48 bytes	1.25kB	8.1 days	q-DBDHI
Goyal *et al.* [108] + §3.6.1	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(1)$	2.2MB	2.2MB	1.25kB	15.1s	q-DBDHI
Alamati *et al.* [9]	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	0.8MB	3.9MB	7.7MB	85.9s	ϕ-hiding
This work §3.4	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	221.4EB	48 bytes	1.34kB	27.7 minutes	SDP
This work §3.4 + §3.6.1	$\mathcal{O}(n)$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(1)$	103.1GB	2.2MB	1.34kB	38.7ms	SDP
This work §3.5	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	412.3GB	48 bytes	1.34kB	27.7 minutes	sBDHE
This work §3.5 + §3.6.1	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(1)$	8.9MB	2.2MB	1.34kB	38.7ms	sBDHE

terms, for moderate to large size databases. In this work, we take a new approach towards designing ℓOT schemes. As we discuss shortly, our approach yields schemes that achieve the same asymptotic complexity as the state-of-the-art, but achieves orders of magnitude improvements in decryption times. This brings the area of laconic cryptography to the realm of practice.

A new approach.. In this work we investigate an alternative approach to building ℓOT schemes. We begin with a simple observation: namely, that ℓOT has similarities to primitives that have been studied in the literature, most notably efficient constant-size *broadcast encryption* [41, 82]. In broadcast encryption (BE), an encryptor transmits a message to a subset of recipients such that only recipients in the set can decrypt the resulting ciphertext. While this functionality is clearly different from ℓOT, the two systems share a similar structure: each can be viewed as a form of "subset" encryption in which a compact ciphertext can be decrypted by some keys and not others. Of course, broadcast encryption on its own does not obviously imply ℓOT. This motivates the following question: *can efficient broadcast encryption constructions be used as a stepping stone to construct laconic OT?*

In this work we answer the previous question in the affirmative. Our first contribution

[3]This is an estimate based on circuit size for curve multiplications on secp192k1 given by Jayaraman *et al.* [128], for more details on this computation see §3.7.2.

is to observe that certain pairing-based broadcast encryption schemes can be transformed into a related primitive that we name *set membership encryption* (SME). SME is a form of functional encryption [43, 173] that combines properties of broadcast encryption with those of Identity-Based Encryption (IBE) [39]. In this paradigm, a master authority generates a set of secret keys for a collection of parties. The encryptor now specifies a *single* party to be the recipient. The novel ingredient in this primitive is that the encryption algorithm *additionally* receives a succinct commitment that identifies a specific subset of possible recipients. A party can decrypt the resulting ciphertext if and only if they were identified as both the intended recipient *and* they are included within this commitment.

We show two constructions of set membership encryption. First, we begin with an efficient adaptively secure broadcast encryption scheme by Waters [190] that we then combine with the identity based encryption scheme of Lewko and Waters [141] to build adaptively secure set membership encryption. Second, we build a selectively secure version of set membership encryption based on the work by Boneh et al. [41]. The latter construction achieves significantly better parameters.

A key property of set membership encryption is that the scheme remains secure *even when all possible recipients collude*. This means that all parties' secret keys can be revealed to an adversary without compromising the security of the scheme, *i.e.* ciphertexts that are intended for a recipient that was not in the subset of recipients remain secure. This allows us to construct a laconic OT for a database of fixed size N via the simple expedient of generating $2N$ parties' public and secret keys in a trusted setup phase, and publishing the resulting key material as a structured reference string (SRS). The Receiver can then commit to its selector bits by encoding these as a commitment for the set membership encryption scheme.

Asymptotic vs Concrete Costs.. An important note regarding this approach is that the resulting constructions produce the same efficiency tradeoff as in the work of Goyal *et al.*

and Alamati *et al.* In comparison with the original work of Cho *et al.*, asymptotically, our construction reduces the bandwidth complexity of each ciphertext from $\mathcal{O}(\log n)$ to constant size, but requires an increase in the size of the structured reference string (SRS) to $\mathcal{O}(n)$ (rather than a constant) and a corresponding increase in the decryption complexity of the receiver to $\mathcal{O}(n)$ rather than $\mathsf{poly}(\log(n))$. Our first construction achieves similar parameters, but a larger structured reference string.

We believe that the longer size of the SRS might be acceptable for some applications since it can be re-used for multiple protocol executions, and was already introduced as useful by Goyal *et al.* We further demonstrate that other tradeoffs are possible, yielding lower decryption times. Specifically, by allowing for a larger digest (sublinear, as opposed to constant-sized), we can achieve sublinear sized SRS and sublinear decryption complexity. Nevertheless, the longer asymptotic decryption complexity of our construction compared to the original work of Cho *et al.* remains a limitation, and further improvements on this front remain an interesting avenue for future work.

Our constructions perform surprisingly well when we evaluate the concrete costs. Indeed, the asymptotic complexity discussed above obscures a significant concrete improvement over the work of Cho *et al.*, due to the fact that our construction removes the need for many sequential garbled circuit evaluations, *e.g.* while prior works' ciphertexts easily grow into *petabytes* of data, ours is only a few hundred bytes. The works of Goyal *et al.* and Alamati *et al.* also achieve these better communication complexities. Our key improvement over these works is in the decryption complexity. In particular, the decryption time of our scheme (specifically the variant that achieves sublinear decryption complexity) is orders of magnitude faster than prior schemes.

We refer the reader to Table 3-I and §3.7 for a detailed comparison of our results with prior work.

Extensions.. Finally, we consider two basic extensions to our schemes. First, we note that the basic definition of ℓOT as proposed by Cho *et al.* does not include *Receiver privacy*. In practice, this property is added to a basic protocol using a two-party secure computation protocol instantiated with a garbled circuit. While a similar approach can be used with our constructions as well, we show that in the interactive setting, our construction can achieve Receiver privacy without the use of garbled circuits. Finally, we investigate an extension that was originally proposed by Cho *et al.*, namely, *updatable ℓOT*. We introduce a more general definition of this primitive and show how to realize this definition using our SME constructions. We note that the constructions from Goyal *et al.* [108] and Alamati *et al.* [9] do not seem to imply such generalization.

Concurrent Work. In concurrent work, Aranha *et al.* [15] present a very similar laconic private set-intersection scheme based on pairings. They reduce their scheme to the security of the Strong Bilinear Decisional Diffie-Hellman Problem, which can also be proven secure in the generic group model. Although the laconic PSI scheme itself shows similarities, we believe our approach and new primitive introduce a fresh look upon this problem.

3.1.1 Applications

We observe that SME and the laconic primitives that can be derived from it have several potential applications that motivate its study. Moreover, understanding the nature of these applications is important, as it can help to determine the appropriate efficiency requirements for an SME scheme. In previous work, many different applications have been discussed as well, most of these can be achieved without much trouble from SME or derived primitives. We briefly discuss two direct applications below.

One-Time Programs.. Goldwasser, Kalai and Rothblum [103] proposed the notion of *one-time programs*. These programs employ a form of secure hardware token, with multiple

OT-like functionalities that "self-destruct" after use. In practice, the cost of this token functionality imposes a significant barrier to the deployment of such programs: since each token functionality holds one input label for a garbled circuit (and can be used only once), the input size (or number of program executions) is therefore bounded by the number of functionalities that can be included into a practical device. Laconic OT removes this restriction: by compressing a large selector database into a ℓOT digest, an evaluator can now evaluate a polynomial number of input wires (or program executions) using a constant number of token OT functionalities.

3.1.2 Technical Overview

In this section we will discuss how to build set membership encryption, but first, we will provide an overview of laconic Oblivious Transfer and discuss the early construction of Cho *et al.* [59]. In the original definition of laconic OT receiver privacy is missing, however, we have added receiver privacy to the definition of laconic OT as well.

Laconic Oblivious Transfer. In a traditional OT, a Sender with two messages interacts with a Receiver who possesses a selector bit. At the conclusion of the interaction, the Receiver learns one message (and nothing about the other message) and the Sender does not learn the Receiver's selection. Laconic OT generalizes this primitive to multiple interactions: the Receiver possesses a database $D \in \{0,1\}^n$ of selector bits and the Sender has n message pairs. To rule out the naive construction, laconic OT adds an efficiency restriction: the communication from Receiver to Sender must be compact, and ideally independent of the database size. An alternative view of laconic OT poses it as a type of encryption: the Receiver computes a hash of the selector bits, and the Sender uses this hash as a form of "public key" to encrypt messages for specific indices and positions. Most critically, in this formulation a single digest can be re-used to encrypt many distinct messages.

Security. Traditional OT provides privacy for both Sender and Receiver. The basic ℓOT security definition proposed by Cho *et al.* requires only privacy for the Sender's inputs. Cho *et al.* point out that Receiver privacy can be added by embedding the laconic encryption algorithm into a garbled circuit that can be evaluated by the Receiver using labels retrieved from the sender using a traditional OT protocol. However, given our very different approach, we can add receiver privacy from the start. Therefore, we have added it to the definition of laconic OT.

Efficiency. Cho *et al.* specify precise asymptotic efficiency requirements in their definition of laconic OT. Notably, they limit the size of the digest to be only polynomially-dependent on the security parameter and independent of the database size. Computationally, the digest must be computable in time $n \cdot \mathsf{poly}(\log n)$ and encryption and decryption time are bounded by $\mathsf{poly}(\log n)$. Unfortunately, due to the nature of our constructions we have to relax some of these bounds: specifically, we allow decryption to require $\mathcal{O}(n)$, which is an asymptotic setback, but we can still show concrete improvements of the decryption time.

The construction of Cho *et al.*. Cho *et al.* propose a construction in two parts. Given a security parameter λ, they propose a laconic OT scheme that takes as input a selector database of size 2λ and creates a digest of size λ. To achieve further compression, the second part of the construction uses this scheme in a binary Merkle tree to reduce a database of arbitrary size to a digest of size λ.

The key observation of the Cho *et al.* scheme is that the root of this tree can be used as an encryption key. This is done by constructing an efficient witness encryption scheme for a specific language involving hash functions, and then dividing the selector database into blocks of λ bits each. These blocks form the leaves of the tree and the root of the tree becomes the digest of the laconic OT scheme. Encryption proceeds by evaluating the witness encryption at each level of the tree. To make this process non-interactive, each level

of encryption is embedded into a garbled circuit, resulting in ciphertexts that comprise chains of $\log N$ garbled circuits. While this design is elegant and the overall complexity assumptions are mild, practical implementations must bear the cost of evaluating elliptic curve point multiplications within many garbled circuits in order to decrypt a single laconic ciphertext: in practice this results in a substantial concrete overhead.

Our Approach.. In broadcast encryption (BE) [82], an encryptor wishes to broadcast a message such that only a subset of receivers can receive the message. Intuitively one might be tempted to construct laconic OT from broadcast encryption via the following heuristic: each of 2 selector bits/position in the Receiver's database could be treated as a single recipient in a universe of $2n$ possible recipients. The Sender could then encrypt each of its messages to a subset of the recipients that would correspond to the appropriate selector bits in the Receiver's database. In this vision, the Receiver would be unable to decrypt messages in positions where its selector bit was not appropriately set.

Of course this intuition does not work, for several reasons. First: broadcast encryption is fundamentally the wrong primitive for this task: instead of encrypting to a specific recipient if and only if the recipient is in a chosen subset, broadcast encryption allows decryption by *any* recipient in the set. In practice this means that the recipient can open both messages at a given index. Moreover even if we ignore this fundamental issue, broadcast encryption has no notion of a succinct *digest* to encode the set of allowed recipients. Finally, the intuition above elides an important detail about the nature of the secret keys: even if keys are generated via a trusted setup procedure (or honestly by the Sender), not every broadcast encryption scheme will retain its security when all secret keys are known to an adversarial Receiver.

A key intuition of this work is that while broadcast encryption is not the right primitive, these problems can be solved by adapting specific broadcast encryption schemes to construct a new protocol that we call *set membership encryption*. This new protocol requires several

features: it must incorporate a means to specify the recipient set via a compact commitment (or digest); it must allow the encryptor to encrypt to a specific recipient *as long as* they are in the recipient set; and it must provide a strong collusion resistance property. We now outline the key steps by which we construct this primitive.

Constructing a succinct digest. As a first step, we consider the problem of modifying broadcast encryption to incorporate a succinct digest of the recipient set. Our basic observation for this modification is that certain efficient broadcast encryption schemes [41, 141, 190] feature compact ciphertexts that are independent of the recipient size, and also admit homomorphic operations on the ciphertext. The nature of these protocols enables an efficient process for constructing a *digest* of the recipient set: this is done by introducing (1) a first "hashing" procedure Hash that takes as input a recipient set and outputs a succinct broadcast encryption ciphertext encrypting the identity element, and (2) a subsequent "encryption" operation Encrypt that takes the previous ciphertext and homomorphically embeds a new plaintext. Concretely, we observe that in many pairing-based BE schemes the recipient set is contained in the ciphertext as a product of certain group elements corresponding to the recipients. We can compute that product during the Hash algorithm and finish the creation of the ciphertext during the Encrypt algorithm.

Modifying the encryption functionality. Unlike broadcast encryption, set membership encryption requires two inputs to the encryption function. First: it takes as input a succinct digest of the recipient set that is produced using the approach described immediately above. Additionally it takes a specific receiver identity. The scheme must allow decryption by a specific recipient key *if and only if* the recipient was identified by the encryptor and is in the recipient set. This novel functionality combines elements of broadcast encryption with those of IBE.[4] Because of the mathematical properties of the pairing-based schemes, we can

[4]Notably, this new scheme must use the same key for both functionalities, which rules out simple combinations of two distinct schemes.

instantiate a second version of the same BE scheme, such that we can bind one recipient to the set of recipients by taking their product. Moreover, we ensure that two secret keys for the same receiver are bound as well, such that the resulting secret key can decrypt both coupled ciphertexts.

Collusion resistance. In the security game of a normal BE scheme, an adversary can receive any key they want as long as they do not appear in the challenge set, *i.e.* the set of receivers to which the challenge ciphertext is encrypted. However, we want the stronger property that given all possible secret keys, ciphertexts are still secure when they are encrypted to a recipient that was not in the recipient set. The way we achieve this is that while tying both the secret keys, we ensure that the adversary cannot use a set of keys to derive a different key. By strongly coupling the binding of both keys to the master secret key, we avoid that any malicious keys can be crafted from a combination of secret keys without knowledge of the master secret key.

Defining security for set membership encryption. We define security for SME via a game-based definition that says that an adversary cannot learn anything from a ciphertext, when that ciphertext is formulated by using the hashing algorithm on a set of receivers and the encryption algorithm for one specific receiver that is not contained in the set, even given decryption keys for all possible receivers. We will define a strong adaptively secure definition, followed by a weaker selectively secure definition where the challenger knows the challenge set of receivers and the challenge receiver before creating a common reference string.

From SME to laconic OT. Having defined set membership encryption, this leaves the main question of how to create a laconic OT scheme. Figure 3-1 illustrates this process. As mentioned previously, the idea is to define two receivers for each location in the database: one receiver corresponds to a 0 bit and the other to a 1 bit. Now, the database can be mapped to said receivers and the Hash algorithm can be used to create a digest $\hat{D}$ of the database.

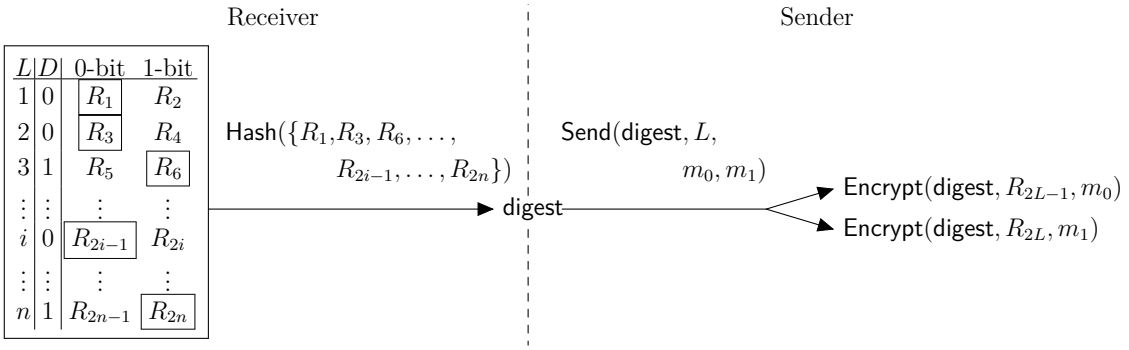

Figure 3-1. A schematic description of using SME to construct laconic OT based on an example database D. Note that for ease of presentation, we use simplified versions of the algorithms omitting any details such as a common reference string. Each R_i represents a receiver in the SME scheme.

Next, for every location, the sender can create two ciphertexts by encrypting the labels using both receivers at that location corresponding to 0 and 1. Given that the encoded database is used, only one of these receivers is encoded inside $\hat{D}$, therefore, due to the properties of the SME, the receiver can only decrypt one of the labels.

Adaptive Security. The adaptive definition for set membership encryption feels much more natural in this setting and therefore our goal is to build our protocols using an adaptively secure broadcast encryption scheme as the basic ingredient. These adaptive schemes are typically proven secure using the Dual System Encryption paradigm introduced by Waters [190]. We can build an adaptively secure scheme based on the work of Lewko and Waters [141] using composite order bilinear groups.

Boneh *et al.* [42] were the first to introduce a scheme using composite-order bilinear groups. Given that assumptions on composite order groups are relying on the fact that it is hard to factor the composite order, large primes must be chosen to create the composite order. This leads to larger groups than prime order bilinear groups. Luckily, Freeman [86] proposes conversions from composite to prime order groups for several schemes. Later, Lewko [140] provides a general conversion from composite order groups to prime order groups. The

assumptions that we use to prove our construction secure are borrowed from the work of Lewko and Waters [141], we describe these assumptions in §2.2.1. These assumptions are closely related to the subgroup decision assumption, which informally states that given an element $g \in \mathbb{G}$ there is no efficient algorithm to decide whether the element g has order p.

As in the work of Lewko and Waters [141], we use the composite order to introduce elements in different subgroups when creating semi-functional ciphertexts and semi-functional keys that are needed to use the Dual System Encryption technique of Waters [190]. We rely on the fact that subgroups are orthogonal to each other with respect to the pairing operation.

Constant Size Decryption Keys. Unfortunately, in the adaptively secure construction, decryption keys are linear in the number of possible receivers in the system. In the derived ℓOT, this leads to a common reference string of size $\mathcal{O}(n^2)$. Therefore, we show a version with constant sized decryption keys, but we can only show this to be selectively secure.

This construction is based on the broadcast encryption system introduced by Boneh *et al.* [41]. This BE scheme is also only selectively secure, which explains the fact that we can only hope to prove selective security for our set membership encryption scheme. To prove this scheme secure we introduce a new assumption which we call a selective bilinear Diffie-Hellman exponentiation assumption (sBDHE), which is an interactive variant of a general BDHE. We can prove this assumption to be secure in the generic bilinear group model [180], by using the generic proof template of Boneh *et al.* [40]

Improvements of our Laconic OT Construction. In the next part of this paper we will introduce a few improvements to our constructions, we will now give a brief technical overview of these improvements.

$\sqrt{n}$-Optimization. Although, our second construction already shows a concrete improvement over some of the previous work, the common reference string is still particularly large and the construction of Alamati *et al.* is still outperforming ours. By striking a new balance

between the size of the digest and the CRS, we can achieve much better efficiency. In order to achieve this, we only initiate the underlying SME with $2\sqrt{n}$ receivers, where n is the size of the database. Instead of hashing the database as a whole, we hash it in chunks of size $\sqrt{n}$. The digest consists of $\sqrt{n}$ sub-digests, which increases the asymptotic communication efficiency of the digest to $\mathcal{O}(\sqrt{n})$. However, as already shown in Table 3-I, we achieve much better concrete efficiency overall. This leads to a very practical construction of ℓOT.

3.2 Set Membership Encryption

In this section we introduce a new primitive called *set membership encryption* (SME). This primitive allows a first party, the hasher, to hash a subset of all receivers that can decrypt a ciphertext, but only a second party, the encryptor, adds a message to the ciphertext and defines a single recipient. Only when this recipient was included in the subset that was chosen by the hasher, the ciphertext can be decrypted. We give a game-based definition that specifies strong adaptive security as well as a second weaker selectively secure definition.

Definition 8 (set membership encryption) *A set membership encryption scheme (SME) consists of five randomized algorithms:*

$\mathsf{Setup}(1^\lambda, n) \to (\mathsf{pk}, \mathsf{msk})$. *This algorithm takes as input the security parameter λ and the maximum number of receivers n. It outputs a public key* pk *and a master secret key* msk.

$\mathsf{KeyGen}(k, \mathsf{pk}, \mathsf{msk}) \to K_k$. *This algorithm takes as input a receiver $k \in [n]$, a public key* pk, *and a master secret key* msk. *It outputs a private key K_k.*

$\mathsf{Hash}(\mathsf{pk}, S) \to \left(\hat{S}, \mathsf{st}\right)$. *This algorithm takes as input a public key* pk *and a subset $S \subseteq [n]$. It outputs a digest of the set S that is denoted $\hat{S}$ and some state* st.

$\mathsf{Encrypt}(\mathsf{pk}, i, M, \hat{S}) \to C$. *This algorithm takes as input a public key* pk*, a receiver* $i \in [n]$*, a message* M*, and a digest* $\hat{S}$*. It outputs a ciphertext* C*.*

$\mathsf{Decrypt}(\mathsf{pk}, K, S, i, C, \mathsf{st}) \to M$. *This algorithm takes as input a public key* pk*, a private key* K*, a subset* $S \subseteq [n]$*, a receiver* $i \in [n]$*, a ciphertext* C*, and state from* Hash st*. It outputs a plaintext message* M*.*

This scheme should have the following properties:

Correctness. *For all* $S \subseteq [n]$ *and all* $i \in S$*, we have*

$$\Pr\left[M = \mathsf{Decrypt}(\mathsf{pk}, K, S, i, C, \mathsf{st}) \;\middle|\; \begin{array}{ccc} (\mathsf{pk}, \mathsf{msk}) & \leftarrow & \mathsf{Setup}(1^\lambda, n) \\ (\hat{S}, \mathsf{st}) & \leftarrow & \mathsf{Hash}(\mathsf{pk}, S) \\ C & \leftarrow & \mathsf{Encrypt}(\mathsf{pk}, i, M, \hat{S}) \\ K & \leftarrow & \mathsf{KeyGen}(i, \mathsf{pk}, \mathsf{msk}) \end{array}\right] = 1.$$

Efficiency. *The size of the digest* $\hat{S}$ *is a fixed polynomial in* λ *independent of the size of the original set. Moreover, the algorithm* Hash *runs in time* $|D| \cdot \mathsf{poly}\left(\log |D|, \lambda\right)$*,* $\mathsf{Encrypt}$ *runs in time* $\mathsf{poly}\left(\log |D|, \lambda\right)$*, and* $\mathsf{Decrypt}$ *runs in time* $\mathsf{poly}\left(|D|, \lambda\right)$*.*

(Selective) Security. *For all PPT algorithms* $\mathcal{A}$ *we have that*

$$\left|\Pr\left[b = b' | (b, b') \leftarrow \boldsymbol{Game}_{\mathcal{A}, \mathsf{SME}, n}\left(\lambda\right)\right] - \frac{1}{2}\right| \leq \epsilon\left(\lambda\right),$$

with $\epsilon(\cdot)$ *a negligible function and* $\boldsymbol{Game}_{\mathcal{A}, \mathsf{SME}, n}$ *the security game as described in Figure 3-2. The same definition can be stated with the selective security game*

$$\boldsymbol{Game}_{\mathcal{A}, s\mathsf{SME}, n}(\lambda)$$

as described in Figure 3-3.

Privacy: *There exists a PPT simulator* $\mathcal{S}$ *such that for any set* S *of size at most* $n = \mathsf{poly}\left(\lambda\right)$ *for any polynomial function* $\mathsf{poly}\left(\cdot\right)$*, let* $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, n)$*,* $\forall i \in [n] : K_i = \mathsf{KeyGen}(i, \mathsf{pk}, \mathsf{msk})$*, and* $(\hat{S}, \mathsf{st}) \leftarrow \mathsf{Hash}(\mathsf{pk}, S)$*, it holds that*

$$\left(\mathsf{pk}, \{K_i\}_{i \in [n]}, \hat{S}\right) \overset{c}{\approx} \left(\mathsf{pk}, \{K_i\}_{i \in [n]}, \mathcal{S}\left(1^\lambda\right)\right).$$

Security game for set membership encryption $\mathbf{Game}_{\mathcal{A},\mathsf{SME},n}(\lambda)$.

Setup. The challenger runs $\mathsf{Setup}\left(1^\lambda, n\right)$ to obtain a public key pk and master secret key msk, it hands the public key to the adversary $\mathcal{A}$ as well as private keys $K_k \leftarrow \mathsf{KeyGen}(k, \mathsf{pk}, \mathsf{msk})$, for all $k \in [n]$.

Challenge. $\mathcal{A}$ picks a message M and sends this message together with a subset $S \subseteq [n]$ and a receiver $i \notin S$. The challenger computes a digest $(\hat{S}, \mathsf{st}) \leftarrow \mathsf{Hash}(\mathsf{pk}, S)$ and picks $b \xleftarrow{\$} \{0,1\}$ and sets $C \leftarrow \mathsf{Encrypt}\left(\mathsf{pk}, i, M, \hat{S}\right)$ if $b = 0$ and picks M' at random and sets $C \leftarrow \mathsf{Encrypt}\left(\mathsf{pk}, i, M', \hat{S}\right)$ otherwise. The challenger gives this digest, the state st, and the ciphertext to $\mathcal{A}$.

Guess. The adversary $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$ for b and wins the game if $b = b'$.

Figure 3-2. Security game for set membership encryption.

3.2.1 Laconic OT from Set Membership Encryption

We now prove that our newly introduced set membership encryption primitive also implies ℓOT, by constructing a ℓOT scheme based on a secure SME scheme.

Theorem 1 *Given a secure set membership encryption scheme*

$$\mathit{SME} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Hash}, \mathsf{Encrypt}, \mathsf{Decrypt})$$

there exists a secure laconic OT scheme $\ell\mathit{OT}$.

Proof. We build the following laconic OT scheme $\ell\mathsf{OT}$, with $\ell = |D|$

$\mathsf{crsGen}(1^\lambda, \ell)$: Set $n = 2\ell$, run $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, n)$, and run $K_i \leftarrow \mathsf{KeyGen}(i, \mathsf{pk}, \mathsf{msk})$, for all $i \in [n]$. Output

$$\mathsf{crs} = (\mathsf{pk}, \{K_i\}_{i\in[n]}).$$

$\mathsf{Hash}(\mathsf{crs}, D)$: Set $E = \left\{2i - \overline{D[i]} \middle| \forall i \in [\ell]\right\}$ and $(\mathsf{digest}, \mathsf{st}) \leftarrow \mathsf{Hash}(\mathsf{pk}, E)$. Output

$$\left(\mathsf{digest}, \hat{D} = (E, \mathsf{digest}, \mathsf{st})\right).$$

Selective security game for set membership encryption $\mathbf{Game}_{\mathcal{A},s\mathsf{SME},n}(\lambda)$.

Setup. The adversary $\mathcal{A}$ sends a set $S \subseteq [n]$ and a receiver $i \notin S$. The challenger runs $\mathsf{Setup}\left(1^\lambda, n\right)$ to obtain a public key pk and master secret key msk, it hands out the public key to the adversary $\mathcal{A}$ as well as private keys $K_k \leftarrow \mathsf{KeyGen}(k, \mathsf{pk}, \mathsf{msk})$, for all $k \in [n]$.

Challenge. $\mathcal{A}$ picks a message M and sends this message. The challenger computes a digest $(\hat{S}, \mathsf{st}) \leftarrow \mathsf{Hash}(\mathsf{pk}, S)$ and picks $b \xleftarrow{\$} \{0,1\}$ and sets $C \leftarrow \mathsf{Encrypt}\left(\mathsf{pk}, i, M, \hat{S}\right)$ if $b = 0$ and picks M' at random and sets $C \leftarrow \mathsf{Encrypt}\left(\mathsf{pk}, i, M', \hat{S}\right)$ otherwise. The challenger gives this ciphertext, the digest, and the state st to $\mathcal{A}$.

Guess. The adversary $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$ for b and wins the game if $b = b'$.

Figure 3-3. Selective security game for set membership encryption.

$\mathsf{Send}(\mathsf{crs}, \mathsf{digest}, L, m_0, m_1)$: create

$$c_0 = \mathsf{Encrypt}(\mathsf{pk}, 2L-1, m_0, \mathsf{digest})$$

and

$$c_1 = \mathsf{Encrypt}(\mathsf{pk}, 2L, m_1, \mathsf{digest}).$$

Output $c = (c_0, c_1)$.

$\mathsf{Receive}^{\hat{D}}(\mathsf{crs}, c, L)$: Parse c as (c_0, c_1). If $D[L] = 0$, set

$$m = \mathsf{Decrypt}(\mathsf{pk}, K_{2L-1}, E, 2L-1, c_0, \mathsf{st}).$$

If $D[L] = 1$, set

$$m = \mathsf{Decrypt}(\mathsf{pk}, K_{2L}, E, 2L, c_1, \mathsf{st}).$$

Correctness follows by inspection and because of the correctness of the underlying SME scheme.

To achieve receiver privacy, the simulator can internally use the simulator of the privacy property of the underlying SME scheme. Indistinguishability of both views follows immediately.

We construct the following simulator $\mathcal{S}(D, L, m_{D[L]})$:

- $\mathsf{digest} \leftarrow \mathsf{Hash}(\mathsf{crs}, D)$
- $c_0 = \mathsf{Encrypt}(\mathsf{pk}, 2L-1, m_{D[L]}, \mathsf{digest})$ and $c_1 = \mathsf{Encrypt}(\mathsf{pk}, 2L, m_{D[L]}, \mathsf{digest})$ and output $c = (c_0, c_1)$.

Assume we have a distinguisher $\mathcal{A}$ that can distinguish between the normal Send algorithm and this simulator, then we build an adversary $\mathcal{B}$ that can break the SME security game. $\mathcal{B}$ receives a public key pk and all private keys for every $i \in [n]$ from the challenger in the SME security game. It passes this information as the crs to $\mathcal{A}$. Next, $\mathcal{B}$ computes digest honestly using Hash, picks a message $m_{\overline{D[L]}}$ at random, and sends $m_{D[L]}$ and $m_{\overline{D[L]}}$ as well as E (from Hash), $2L - \overline{D[L]}$, and digest to the challenger in the SME security game. $\mathcal{B}$ receives a ciphertext c. Now, $\mathcal{B}$ sets $c_{\overline{D[L]}} = c$ and $c_{D[L]} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 2L - \overline{D[L]}, m_{D[L]}, \mathsf{digest})$. Now, $\mathcal{A}$ answers with either 0, *i.e.* we are using the real Send algorithm, in which case $\mathcal{B}$ answers with $D[L]$, or $\mathcal{A}$ answers with 1, *i.e.* we are using the simulator. In which case $\mathcal{B}$ sends $\overline{D[L]}$ to the challenger.

The efficiency of the laconic OT scheme follows directly from the efficiency of the set membership encryption scheme. □

Malicious Sender and Receiver.. Similar to previous work, the constructions that we give are only secure against semi-honest adversaries. Standard techniques can be used to upgrade to malicious adversaries, *e.g.* by the use of Non-Interactive Zero Knowledge proofs (NIZKs) [80]. Luckily, given the use of bilinear maps in our constructions it is possible to use quite optimal NIZKs such as the ones introduced by Groth et al. [114], and Groth [112]. More

recent work is exploring and further optimizing these types of NIZKs. [52, 58, 87, 113, 146] We suspect that these proofs can even be further optimized leading to a very practical construction, we leave these optimizations of the NIZKs for future work.

Other possibilities to improve the construction against malicious adversaries are the techniques of Ishai *et al.* [126], or using interactive zero-knowledge proofs, but this introduces more interactivity. Given the optimal setting for NIZKs, we rather prefer these former techniques.

3.3 New Broadcast Encryption Scheme

We introduce a new broadcast encryption scheme that is based on the IBE scheme that was introduced by Lewko and Waters [141]. To prove the IBE to be secure they use the Dual System Encryption that was introduced by Waters [190], where that paper contained a Broadcast encryption system, Lewko and Waters didn't give a construction. They made the interesting observation that their system was closely related to the IBE introduced by Boneh and Boyen [38] and their HIBE to the Boneh, Boyen, and Goh [40] HIBE. The latter also introduced a Broadcast encryption system. Therefore, this construction is only novel in combining the existing construction techniques as well as proving techniques from these different previous works. However, we will later need this exact Broadcast encryption scheme to create our Laconic OT.

Construction.

$\mathsf{Setup}\left(1^{\lambda}, n\right)$: Let $\mathbb{G}$ be a bilinear group of composite order $N = p_1 p_2 p_3$. Choose random elements $u_1, \ldots, u_n, g, \in \mathbb{G}_{p_1}$ and a random $\alpha \in \mathbb{Z}_N$. Output:

$$\mathsf{pk} = \left(u_1, \ldots, u_n, g, e\left(g, g\right)^{\alpha}\right).$$

and $\mathsf{msk} = g^{\alpha}$.

$\mathsf{Encrypt}\left(\mathsf{pk}, M, S\right)$:This function takes as input the public key pk, a message M, and the receiver set S. It outputs an encrypted messages C.

- Parse pk as $\left(u_1, \ldots, u_n, g, e\left(g,g\right)^{\alpha}\right)$.
- Pick $s \xleftarrow{\$} \mathbb{Z}_N$
- Output:

$$C = \left(C_0, C_1, C_2\right) = \left(M \cdot e\left(g,g\right)^{\alpha s}, \left(\prod_{i \in S} u_i\right)^s, g^s\right).$$

$\mathsf{KeyGen}\left(k, \mathsf{pk}, \mathsf{msk}\right)$: This function takes as input the index k of the receiver for which a key is generated, the public key pk, and the master secret key msk. It outputs a secret key K.

- Parse pk as $\left(u_1, \ldots, u_n, g, e\left(g,g\right)^{\alpha}\right)$.
- Parse msk as g^{α}.
- Pick $r \xleftarrow{\$} \mathbb{Z}_N$, $R_3, R_3^{(1)}, \ldots, R_3^{(n)} \xleftarrow{\$} \mathbb{G}_{p_3}$
- Output:

$$K = \left(K_1, K_2, \left\{K_{3,i}\right\}_{\substack{i \in [n] \\ i \neq k}}\right) = \left(g^r R_3, g^{\alpha} u_k^r R_3^{(k)}, \left\{u_i^r R_3^{(i)}\right\}_{\substack{i \in [n] \\ i \neq k}}\right)$$

$\mathsf{Decrypt}\left(\mathsf{pk}, K, S, k, C\right)$: This function takes as input the public key pk, a secret key K, the receiver set S, an index k, and the ciphertext C. It outputs a plaintext message M.

- Parse K as $\left(K_1, K_2, \left\{K_{3,i}\right\}_{\substack{i \in [n] \\ i \neq k}}\right)$.
- Parse C as $\left(C_0, C_1, C_2\right)$
- Compute:

$$e\left(g,g\right)^{\alpha s} = \frac{e\left(K_2, C_2\right) e\left(\prod_{\substack{i \in S \\ i \neq k}} K_{3,i}, C_2\right)}{e\left(K_1, C_1\right)}. \tag{3.1}$$

- Output:

$$M = \frac{C_0}{e\left(g,g\right)^{\alpha s}}.$$

Correctness. We show that Equation (3.1) indeed recovers $e(g,g)^{\alpha s}$, which then can be used to recover the plaintext:

$$\begin{aligned}\frac{e(K_2,C_2)\, e\left(\prod_{\substack{i\in S\\ i\neq k}} K_{3,i}, C_2\right)}{e(K_1,C_1)} &= \frac{e\left(g^{\alpha} u_k^{r_k} R_{3,k}^{(k)}, g^{t_b}\right) e\left(\prod_{\substack{i\in S\\ i\neq k}} u_i^{r_k} R_{3,k}^{(i)}, g^{s}\right)}{e\left(g^{r_k} R_{3,k}, \left(\prod_{i\in S} u_i\right)^{s}\right)} \\ &= \frac{e(g,g)^{\alpha s}\, e\left(\left(\prod_{i\in S} u_i\right)^{r_k}, g^{s}\right)}{e\left(g^{r_k}, \left(\prod_{i\in S} u_i\right)^{s}\right)} \\ &= e(g,g)^{\alpha s}\end{aligned}$$

Security. We will now prove the following theorem:

Theorem 2 *If Assumptions 1, 2, and 3 hold, then our BE system is adaptively secure.*

Proof. To prove this statement we will use the Dual System Encryption technique of Waters [190]. First, we will introduce a semi-functional ciphertext as well as a semi-functional key. A semi-functional ciphertext can still be decrypted by a normal key, but cannot be decrypted by a semi-functional key. However, the semi-functional key can still decrypt normal ciphertexts.

Using these semi-functional versions of the ciphertext and keys, we will use a set of hybrids that start with changing the challenge ciphertext into a semi-functional ciphertext. Next, one-by-one we change each key that gets queried by the adversary into a semi-functional key. As a last step we can replace the message by a random message, this final version does not contain any information about the ciphertext anymore. If all these hybrids are indeed indistinguishable to the adversary, we can use the hybrid lemma to conclude that our broadcast encryption construction is indeed secure.

Semi-functional Ciphertext. We let g_2 denote a generator of subgroup $\mathbb{G}_{p_2}$. First generate a normal ciphertext $C' = (C'_0, C'_1, C'_2)$. Sample random exponents $x, z_c \xleftarrow{\$} \mathbb{Z}_N$. Then $C = (C'_0, C'_1 g_2^{x z_c}, C'_2 g_2^{x})$.

Semi-functional Key. Create a normal key $K' = \left(K'_1, K'_2, \left\{K'_{3,i}\right\}_{\substack{i\in[n]\\ i\neq k}}\right)$. Sample random exponents $\gamma, \{z_{k,i}\}_{i\in[n]} \xleftarrow{\$} \mathbb{Z}_N$. Then $K = \left(K'_1 g_2^{\gamma}, K'_2 g_2^{\gamma z_{k,k}}, \left\{K'_{3,i} g_2^{\gamma z_{k,i}}\right\}_{\substack{i\in[n]\\ i\neq k}}\right)$.

Decrypting a semi-functional ciphertext with a semi-functional key adds an extra blinding factor of $e(g_2, g_2)^{x\gamma\left(\sum_{i\in[n]} z_{k,i} - z_c\right)}$. If $\sum_{i\in[n]} z_{k,i} = z_c$, decryption will still work, which means the key is *nominally* semi-functional.

Now we introduce the different hybrids:

> $\mathbf{Game}_{Real}$ is the real security game for broadcast encryption.
>
> $\mathbf{Game}_j$ is the same as the real security game except that the challenge ciphertext is semi-functional and the first j queried keys are also semi-functional. In $\mathbf{Game}_q$ the ciphertext and all keys are semi-functional, with q being the number of queries the attacker makes.
>
> $\mathbf{Game}_{Final}$ is the same as $\mathbf{Game}_q$ except that the ciphertext is a semi-functional ciphertext of a random message.

Note that $\mathbf{Game}_0$ is the same as the real security game except that the challenge ciphertext is a semi-functional ciphertext, while all secret keys remain normal secret keys.

We conclude the proof by using three lemmas. First, Lemma 1 proves that $\mathbf{Game}_{Real}$ is indistinguishable from $\mathbf{Game}_0$. Next, Lemma 2 proves that $\mathbf{Game}_{i-1}$ is indistinguishable from $\mathbf{Game}_i$, for any $i \in [q]$, with q the number of queries made by the adversary. Finally, Lemma 3 proves that $\mathbf{Game}_q$ is indistinguishable from $\mathbf{Game}_{Final}$, which concludes the proof because that last game does not contain any information about the message anymore. □

Lemma 1 *Suppose there exists an algorithm $\mathcal{A}$ such that $\boldsymbol{Game}_{Real}Adv_{\mathcal{A}} - \boldsymbol{Game}_0 Adv_{\mathcal{A}} = \epsilon$. Then there exists an algorithm $\mathcal{B}$ with advantage ϵ in breaking Assumption 1.*

Proof. Given an adversary $\mathcal{A}$ that can distinguish between $\mathbf{Game}_{Real}$ and $\mathbf{Game}_0$, we now show how to construct an adversary $\mathcal{B}$ that can break Assumption 1. $\mathcal{B}$ receives g, X_3, T from an instance of Assumption 1. It samples random exponents $\alpha, \{a_i\}_{i\in[n]} \xleftarrow{\$} \mathbb{Z}_N$ and sets $g = g, u_i = g^{a_i}$. It sends the public parameters $\{N, u_1, \ldots, u_n, g, e(g,g)^\alpha\}$ to $\mathcal{A}$. For each key queried by $\mathcal{A}$ for an index k, $\mathcal{B}$ samples random exponents r_k, t_k, and $\forall j \in [n] : w_{k,j} \xleftarrow{\$} \mathbb{Z}_N$ and sets:

$$K = \left(g^{r_k} X_3^{t_k}, g^\alpha u_k^{r_k} X_3^{w_{k,k}}, \left\{ u_i^{r_k} X_3^{w_{k,i}} \right\}_{\substack{i\in[n] \\ i\neq k}} \right).$$

$\mathcal{B}$ receives two messages M_0 and M_1 from $\mathcal{A}$ and a challenge set S^*. $\mathcal{B}$ samples $\beta \xleftarrow{\$} \{0,1\}$ and forms the ciphertext:

$$C = \left(M_\beta e(T,g)^\alpha, T^{\sum_{i\in S^*} a_i}, T \right).$$

This sets g^s to be equal to the $\mathbb{G}_{p_1}$ part of T. If $T \in \mathbb{G}_{p_1 p_2}$ then this is a semi-functional ciphertext with $z_c = \sum_{i\in S^*} a_i$. We note that the value of z_c modulo p_2 is uncorrelated with the values a_i modulo p_1, so this is still properly distributed. If $T \in \mathbb{G}_{p_1}$, this is a normal ciphertext. If the adversary $\mathcal{A}$ outputs 0, meaning it decides we are in $\mathbf{Game}_{Real}$, $\mathcal{B}$ also outputs 0, i.e. $T \in \mathbb{G}_{p_1}$. Otherwise, if the adversary $\mathcal{A}$ outputs 1, meaning it decides we are in $\mathbf{Game}_0$, $\mathcal{B}$ also outputs 1, i.e. $T \in \mathbb{G}_{p_1 p_2}$. Hence, if the adversary $\mathcal{A}$ has an advantage in distinguishing between both games, $\mathcal{B}$ also has an advantage in breaking Assumption 1. □

Lemma 2 *Suppose there exists an algorithm $\mathcal{A}$ such that $\boldsymbol{Game}_{j-1} Adv_{\mathcal{A}} - \boldsymbol{Game}_j Adv_{\mathcal{A}} = \epsilon$. Then there exists an algorithm $\mathcal{B}$ with advantage ϵ in breaking Assumption 2.*

Proof. Given an adversary $\mathcal{A}$ that can distinguish between $\mathbf{Game}_{j-1}$ and $\mathbf{Game}_j$, we now show how to construct an adversary $\mathcal{B}$ that can break Assumption 1. $\mathcal{B}$ receives $g, X_1X_2, X_3, Y_2Y_3, T$ from an instance of Assumption 2. It samples random $\alpha, \{a_i\}_{i\in[n]} \xleftarrow{\$} \mathbb{Z}_N$, sets $g = g, u_i = g^{a_i}$, and sends public parameters $\{N, u_1, \ldots, u_n, g, e(g,g)^\alpha\}$ to $\mathcal{A}$. When $\mathcal{A}$ requests a key for index k $\mathcal{B}$ responds with:

$k < j$: create a semi-functional key by sampling $r_k, z_{k,1}, \ldots, z_{k,n}, t_k \xleftarrow{\$} \mathbb{Z}_N$ and set:

$$K = \left(g^{r_k} (Y_2 Y_3)^{t_k}, g^{\alpha} u_k^{r_k} (Y_2 Y_3)^{z_{k,k}}, \{ u_i^{r_k} (Y_2 Y_3)^{z_{k,i}} \}_{\substack{i \in [n] \\ i \neq k}} \right).$$

This is a properly distributed semi-functional key with $g_2^{\gamma} = Y_2^{t_k}$. Again, we note that the values of t_k and $z_{k,i}$ modulo p_2 and modulo p_3 are uncorrelated.

$k > j$: create a normal key by sampling $r_k, w_{k,1}, \ldots, w_{k,n}, t_k \xleftarrow{\$} \mathbb{Z}_N$ and set:

$$K = \left(g^{r_k} X_3^{t_k}, g^{\alpha} u_k^{r_k} X_3^{w_{k,k}}, \left\{ u_i^{r_k} X_3^{w_{k,i}} \right\}_{\substack{i \in [n] \\ i \neq k}} \right).$$

This is identical to a normal construction of a key with $R_3 = X_3^{t_k}$ and $R_3^{(i)} = X_3^{w_{k,i}}$ for all i.

$k = j$: $\mathcal{B}$ sets $z_{k,i} = a_i$ and samples $w_{k,1}, \ldots, w_{k,n} \xleftarrow{\$} \mathbb{Z}_N$ and sets:

$$K = \left(T, g^{\alpha} T^{z_{k,k}} X_3^{w_{k,k}}, \left\{ T^{z_{k,i}} X_3^{w_{k,i}} \right\}_{\substack{i \in [n] \\ i \neq k}} \right).$$

Depending on value T this is either a normal key, i.e. if $T \in \mathbb{G}_{p_1 p_3}$, or a semi-functional key, i.e. when $T \in \mathbb{G}$

At some point $\mathcal{A}$ will send two messages M_0 and M_1, and a challenge set S^*. $\mathcal{B}$ samples $\beta \xleftarrow{\$} \{0, 1\}$ and forms the challenge ciphertext:

$$C = \left(M_{\beta} e\left(X_1 X_2, g\right)^{\alpha}, (X_1 X_2)^{\sum_{i \in S^*} a_i}, X_1 X_2 \right).$$

This sets $g^s = X_1$ and $z_c = \sum_{i \in S^*} a_i$. Again, because the values a_i modulo p_2 are uncorrelated to their counterparts modulo p_1, these values are properly distributed.

Note that $\mathcal{B}$ can only create a nominally semi-functional key for position j.

If $T \in \mathbb{G}_{p_1 p_3}$ then $\mathbf{Game}_{j-1}$ was properly simulated by $\mathcal{B}$, if $T \in \mathbb{G}$, however, this is a simulation of $\mathbf{Game}_j$. Therefore, if $\mathcal{A}$ responds with 0, i.e. it believes we are in $\mathbf{Game}_{j-1}$, $\mathcal{B}$ also responds with 0, i.e. $T \in G_{p_1 p_3}$. Similarly, if $\mathcal{A}$ responds with 1, i.e. it believes we are in

$\mathbf{Game}_j$, $\mathcal{B}$ also responds with 1, i.e. $T \in G$. We can conclude that if $\mathcal{A}$ has an advantage distinguishing between two games, $\mathcal{B}$ also has an advantage in breaking Assumption 2. $\square$

Lemma 3 *Suppose there exists an algorithm $\mathcal{A}$ such that $\boldsymbol{Game}_q Adv_{\mathcal{A}} - \boldsymbol{Game}_{Final} Adv_{\mathcal{A}} = \epsilon$. Then there exists an algorithm $\mathcal{B}$ with advantage ϵ in breaking Assumption 3.*

Proof. Given an adversary $\mathcal{A}$ that can distinguish between $\mathbf{Game}_q$ and $\mathbf{Game}_{Final}$, we now show how to construct an adversary $\mathcal{B}$ that can break Assumption 3. $\mathcal{B}$ receives $g, g^{\alpha} X_2, X_3, g^s Y_2, Z_2, T$ from an instance of Assumption 3. It samples random $\{a_i\}_{i \in [n]} \xleftarrow{\$} \mathbb{Z}_N$, sets $g = g, u_i = g^{a_i}, e(g,g)^{\alpha} = e(g^{\alpha} X_2, g)$ and sends public parameters

$$\{N, u_1, \ldots, u_n, g, e(g,g)^{\alpha}\}$$

to $\mathcal{A}$. When $\mathcal{A}$ requests a key for index k, $\mathcal{B}$ responds with a semi-functional key by sampling $c_{k,1}, \ldots, c_{k,n}, r_k, t_k, w_{k,1}, \ldots, w_{k,n}, \gamma_k \xleftarrow{\$} \mathbb{Z}_N$ and sets:

$$K = \left(g^{r_k} Z_2^{\gamma_k} X_3^{t_k}, g^{\alpha} X_2 u_k^{r_k} Z_2^{c_{k,k}} X_3^{w_{k,k}}, \left\{ u_i^{r_k} Z_2^{c_{k,i}} X_3^{w_{k,i}} \right\}_{\substack{i \in [n] \\ i \neq k}} \right).$$

$\mathcal{A}$ sends two messages M_0 and M_1 and a challenge set S^*. $\mathcal{B}$ samples $\beta \xleftarrow{\$} \{0,1\}$ and forms a challenge ciphertext:

$$C = \left(M_{\beta} T, (g^s Y_2)^{\sum_{i \in S^*} a_i}, g^s Y_2 \right).$$

This sets $z_c = \sum_{i \in S^*} a_i$, again note that this is properly distributed because of a_i modulo p_1 being uncorrelated to a_i modulo p_2.

If $T = e(g,g)^{\alpha s}$ then this is a properly distributed semi-functional ciphertext with message M_{β}. If T is random in $\mathbb{G}_T$, then this is a semi-functional ciphertext with a random message. Therefore, when $\mathcal{A}$ responds with 0, i.e. it believes we are in $\mathbf{Game}_q$, $\mathcal{B}$ also responds with 0, i.e. $T = e(g,g)^{\alpha s}$. Otherwise, when $\mathcal{A}$ responds with 1, i.e. it believes we are in $\mathbf{Game}_{Final}$, $\mathcal{B}$ also responds with 1, i.e. $T \xleftarrow{\$} \mathbb{G}_T$. We can conclude that if $\mathcal{A}$ has an advantage distinguishing between two games, $\mathcal{B}$ also has an advantage in breaking Assumption 3. $\square$

3.4 SME Construction

We start by giving an adaptively secure construction of the set membership encryption protocol. This is based on an efficient adaptively secure broadcast encryption scheme by Waters [190] that we then combine with the identity based encryption scheme of Lewko and Waters [141]. This construction works in composite order bilinear groups and is proven to be adaptively secure by using the Dual System Encryption framework of Waters [190].

$\mathsf{Setup}\left(1^{\lambda}, n\right)$: Let $\mathbb{G}$ be a bilinear group of composite order $N = p_1 p_2 p_3$. Choose random elements $u_1, \ldots, u_n, \tilde{u}_1, \ldots, \tilde{u}_n, g \in \mathbb{G}_{p_1}$ and a random $\alpha \in \mathbb{Z}_N$. Set $\mathsf{pk} = (u_1, \ldots, u_n, \tilde{u}_1, \ldots, \tilde{u}_n, g, e\left(g, g\right)^{\alpha})$, and $\mathsf{msk} = g^{\alpha}$. Output $(\mathsf{pk}, \mathsf{msk})$.

$\mathsf{KeyGen}(k, \mathsf{pk}, \mathsf{msk})$: This algorithm takes as input a receiver $k \in [n]$, a public key pk, and a master secret key msk. Parse pk as $(u_1, \ldots, u_n, \tilde{u}_1, \ldots, \tilde{u}_n, g, e\left(g, g\right)^{\alpha})$ and msk as g^{α}. Pick $r \xleftarrow{\$} \mathbb{Z}_N$, and $R_3, R_3^{(1)}, \ldots, R_3^{(n)} \xleftarrow{\$} \mathbb{G}_{p_3}$ and set

$$d_k = \left(g^r R_3, g^{\alpha} \left(\tilde{u}_k u_k\right)^r R_3^{(k)}, \left\{ u_j^r R_3^{(j)} \right\}_{\substack{j \in [n] \\ j \neq k}} \right).$$

$\mathsf{Hash}\left(\mathsf{pk}, S\right)$: This algorithm takes as input a public key pk and a subset $S \subseteq [n]$. It outputs a digest $\hat{S}$ and state st. Parse pk as $(u_1, \ldots, u_n, \tilde{u}_1, \ldots, \tilde{u}_n, g, e\left(g, g\right)^{\alpha})$. Pick uniformly random $z \xleftarrow{\$} \mathbb{G}_{p_1}$ and compute: $\hat{S} = z \prod_{j \in S} u_j$. Output $(\hat{S}, \mathsf{st} = z)$.

$\mathsf{Encrypt}\left(\mathsf{pk}, k, M, \hat{S}\right)$: This algorithm takes as input a public key pk, a receiver $k \in [n]$, a message M, and a digest $\hat{S}$. It outputs a ciphertext $C = (c_1, c_2, c_3)$. Parse pk as $(u_1, \ldots, u_n, \tilde{u}_1, \ldots, \tilde{u}_n, g, e\left(g, g\right)^{\alpha})$. Pick $t \xleftarrow{\$} \mathbb{Z}_N$ and set

$$\begin{aligned} C &= \left(M e\left(g, g\right)^{\alpha t}, \left(\tilde{u}_k \hat{S}\right)^t, g^t \right) \\ &= \left(M e\left(g, g\right)^{\alpha t}, \left(\tilde{u}_k z \prod_{j \in S} u_j \right)^t, g^t \right). \end{aligned}$$

$\mathsf{Decrypt}\left(\mathsf{pk}, d_k, S, k, C, \mathsf{st}\right)$: This algorithm takes as input a public key pk, a private key d_k, a subset $S \subseteq [n]$, a receiver $k \in [n]$, a ciphertext C, and state $\mathsf{st} = z$. Parse pk as $\left(u_1, \ldots, u_n, \tilde{u}_1, \ldots, \tilde{u}_n, g, e\left(g, g\right)^{\alpha}\right)$ and C as $\left(c_1, c_2, c_3\right)$. Parse

$$d_k = \left(\delta_1, \delta_2, \{\delta_{3,j}\}_{\substack{j \in [n] \\ j \neq k}}\right).$$

Compute

$$K = \frac{e\left(z, c_3\right) e\left(\delta_2, c_3\right) e\left(\prod_{\substack{j \in S \\ j \neq k}} \delta_{3,j}, c_3\right)}{e\left(\delta_1, c_2\right)}. \tag{3.2}$$

Output $M \leftarrow c_1 K^{-1}$.

Correctness. We show that Equation (3.2) indeed recovers K, which then can be used to recover the plaintext message M

$$\begin{aligned}
K &= \frac{e\left(\delta_2, c_3\right) e\left(\prod_{\substack{j \in S \\ j \neq k}} \delta_{3,j}, c_3\right)}{e\left(\delta_1, c_2\right)} \\
&= \frac{e\left(g^{\alpha}\left(\tilde{u}_k u_k\right)^r R_3^{(k)}, g^t\right) e\left(\prod_{\substack{j \in S \\ j \neq k}} u_j^r R_3^{(j)}, g^t\right)}{e\left(g^r R_3, \left(\tilde{u}_k \prod_{j \in S} u_j\right)^t\right)} \\
&= \frac{e\left(g, g\right)^{\alpha t} e\left(\left(\tilde{u}_k \prod_{j \in S} u_j\right)^r, g^t\right)}{e\left(g^r, \left(\tilde{u}_k \prod_{j \in S} u_j\right)^t\right)} \\
&= e\left(g, g\right)^{\alpha t}
\end{aligned}$$

Efficiency. In this construction it can be seen that the size of the public key is $2n + 1$ random elements in $\mathbb{G}$ and 1 element in the target group. The msk however is just 1 element in $\mathbb{G}$. The size of the keys are linear in the number of users of the protocol, *i.e.* $n + 1$ group elements.

Hash can be computed by doing $|S| - 1$ multiplications within $\mathbb{G}$, therefore, this algorithm runs in time $\mathcal{O}\left(|S|\right)$. The output of the algorithm is just 1 group element. On the other hand,

$\mathsf{Encrypt}$ runs in constant time, because it only computes 1 multiplication and 2 exponentiations in $\mathbb{G}$, and 1 exponentiation and 1 multiplication in $\mathbb{G}_T$. The output of this algorithm is 2 elements in $\mathbb{G}$ and 1 element in $\mathbb{G}_T$.

Finally decryption runs in $\mathcal{O}(|S|)$ because it needs to do $|S|$ curve multiplications in $\mathbb{G}$, 2 pairings, and 1 multiplication and 1 division in $\mathbb{G}_T$.

Privacy. To show privacy we need to create a simulator such that for any set S of size at most $n = \mathsf{poly}(\lambda)$ for any polynomial function $\mathsf{poly}(\cdot)$, let $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, n)$, $\forall i \in [n] : K_i = \mathsf{KeyGen}(i, \mathsf{pk}, \mathsf{msk})$, and $(\hat{S}, \mathsf{st}) \leftarrow \mathsf{Hash}(\mathsf{pk}, S)$, it holds that

$$\left(\mathsf{pk}, \{K_i\}_{i\in[n]}, \hat{S}\right) \overset{c}{\approx} \left(\mathsf{pk}, \{K_i\}_{i\in[n]}, \mathcal{S}\left(1^\lambda\right)\right).$$

A very straight forward simulator is just sampling a uniform random element in $\mathbb{G}$, because of the randomly chosen blinding factor z during hashing, it is clear that both outputs are perfectly indistinguishable.

Theorem 3 *If Assumptions 1, 2, and 3 hold, then our set membership encryption system is secure.*

Proof. We will use the Dual System Encryption technique that was introduced by Waters [190]. Therefore, we will introduce a semi-functional version of the ciphertext as well as a semi-functional key. A semi-functional ciphertext can still be decrypted by a normal key, but cannot be decrypted by a semi-functional key. Similarly, semi-functional keys can still decrypt normal ciphertexts.

Next, we will use a set of hybrids starting with the security game $\mathbf{Game}_{\mathcal{A},\mathsf{SME},n}$ where the challenger selects $b' = 0$ and encrypts the real message. Next, we will change the challenge ciphertext to be a semi-functional ciphertext. The following hybrids will replace all private keys one-by-one with a semi-functional version. In the last hybrid we can remove the real message and replace it with a random message. This last hybrid has no information about

the message anymore, therefore, statistically hides the message. One can follow the path of hybrids backwards to end up with the security game where the challenger picks $b' = 1$ and therefore encrypts a random message.

Semi-functional Ciphertext. We let g_2 denote a generator of subgroup $\mathbb{G}_{p_2}$. First generate a normal ciphertext $C' = (c'_1, c'_2, c'_3)$. Sample random exponents $x, z_c \xleftarrow{\$} \mathbb{Z}_N$. Then $C = (c'_1, c'_2 g_2^{x z_c}, c'_3 g_2^{x})$.

Semi-functional Key. Create a normal key $d'_i = \left(\delta'_1, \delta'_2, \left\{\delta'_{3,i}\right\}_{\substack{i\in[n]\\ i\neq k}}\right)$. Sample random exponents $\gamma, \{z_{k,i}\}_{i\in[n]} \xleftarrow{\$} \mathbb{Z}_N$. Then

$$d_i = \left(\delta'_1 g_2^{\gamma}, \delta'_2 g_2^{\gamma z_{k,k}}, \left\{\delta'_{3,i} g_2^{\gamma z_{k,i}}\right\}_{\substack{i\in[n]\\ i\neq k}}\right).$$

Decrypting a semi-functional ciphertext with a semi-functional key adds an extra blinding factor of $e\left(g_2, g_2\right)^{x\gamma\left(\sum_{i\in[n]} z_{k,i} - z_c\right)}$. If $\sum_{i\in[n]} z_{k,i} = z_c$, decryption will still work, which means the key is *nominally* semi-functional.

Next, we will describe the different hybrids starting with the security game $\mathbf{Game}_{\mathcal{A},\mathsf{SME},n}$ where the challenger selects $b' = 0$ and encrypts the real message, and ending with a game that doesn't contain any information about the message anymore.

$\mathbf{Game}_{Real}$ is the security game $\mathbf{Game}_{\mathcal{A},\mathsf{SME},n}$ where the challenger selects $b' = 0$ and encrypts the real message.

$\mathbf{Game}_{\tau}$ is the same as $\mathbf{Game}_{Real}$ except that the challenge ciphertext is semi-functional and the first τ queried keys are also semi-functional. In $\mathcal{H}_n$ the ciphertext and all keys are semi-functional.

$\mathbf{Game}_{Final}$ is the same as $\mathcal{H}_n$ except that the challenge ciphertext is a semi-functional ciphertext of a random message. Note that this hybrid does not contain any information about the plaintext message anymore.

We conclude the proof by using three lemmas. First, Lemma 4 proves that $\mathbf{Game}_{Real}$ is indistinguishable from $\mathbf{Game}_0$. Next, we show in Lemma 5 that $\mathbf{Game}_{\tau-1}$ is indistinguishable from $\mathbf{Game}_\tau$, for any $\tau \in [n]$. Finally, Lemma 6 proves that $\mathbf{Game}_\tau$ is indistinguishable from $\mathbf{Game}_{Final}$, which concludes the proof because that last hybrid does not contain any information about the message anymore. □

Lemma 4 *Suppose there exists an algorithm $\mathcal{A}$ such that*

$$\boldsymbol{Game}_{Real} Adv_{\mathcal{A}} - \boldsymbol{Game}_0 Adv_{\mathcal{A}} = \epsilon.$$

Then there exists an algorithm $\mathcal{B}$ with advantage ϵ in breaking Assumption 1.

Proof. Given an adversary $\mathcal{A}$ that can distinguish between $\mathbf{Game}_{Real}$ and $\mathbf{Game}_0$, we now show how to construct an adversary $\mathcal{B}$ that can break Assumption 1. $\mathcal{B}$ receives g, X_3, T from an instantiation of Assumption 1. It samples random exponents $\alpha, \kappa, \{a_i, c_i\}_{i\in[n]} \xleftarrow{\$} \mathbb{Z}_N$ and sets $g = g, u_i = g^{a_i}, \tilde{u}_i = g^{c_i}$. $\mathcal{B}$ sets $\mathsf{pk} = (u_1, \ldots, u_n, \tilde{u}_1, \ldots, \tilde{u}_n, g, e(g,g)^\alpha)$ and sends it to $\mathcal{A}$.

For each key query with index k, $\mathcal{B}$ responds by sampling random exponents r_k, t_k, and $w_{k,1}, \ldots, w_{k,n} \xleftarrow{\$} \mathbb{Z}_N$ and setting:

$$d_k = \left(g^{r_k} X_3^{t_k}, g^\alpha (\tilde{u}_k u_k)^{r_k} X_3^{w_{k,k}}, \left\{ u_i^{r_k} X_3^{w_{k,i}} \right\}_{\substack{i\in[n] \\ i\neq k}} \right).$$

$\mathcal{B}$ receives a set $S \subseteq [n]$ and an element $j \in [n]$, but $j \notin S$, and a message M from $\mathcal{A}$. $\mathcal{B}$ computes:

$$C = \left(M e(T,g)^\alpha, T^{c_j + \kappa + \sum_{i\in S} a_i}, T \right).$$

This sets g^t to be equal to the $\mathbb{G}_{p_1}$ part of T. If $T \in \mathbb{G}_{p_1 p_2}$ then this is a semi-functional ciphertext with $z_c = c_j + \kappa + \sum_{i\in S} a_i$. We note that the value of z_c modulo p_2 is uncorrelated with the values a_i and c_j modulo p_1, so this is still properly distributed. If $T \in \mathbb{G}_{p_1}$, this is a normal ciphertext. If $\mathcal{A}$ outputs 0, *i.e.* it thinks we simulated $\mathbf{Game}_{Real}$, $\mathcal{B}$ also outputs 0,

i.e. $T \in \mathbb{G}_{p_1}$. Similarly, if $\mathcal{A}$ outputs 1, *i.e.* it thinks we simulated $\mathbf{Game}_0$, $\mathcal{B}$ also outputs 1, *i.e.* $T \in \mathbb{G}_{p_1 p_2}$. Hence, if the adversary $\mathcal{A}$ has an advantage in distinguishing between both games, $\mathcal{B}$ also has an advantage in breaking Assumption 1. □

Lemma 5 *Suppose there exists an algorithm $\mathcal{A}$ such that*

$$\boldsymbol{Game}_{\tau-1} Adv_{\mathcal{A}} - \boldsymbol{Game}_{\tau} Adv_{\mathcal{A}} = \epsilon.$$

Then we build an algorithm $\mathcal{B}$ with advantage ϵ in breaking Assumption 2.

Proof. Given an adversary $\mathcal{A}$ that can distinguish between $\mathbf{Game}_{\tau-1}$ and $\mathbf{Game}_{\tau}$, we now show how to construct an adversary $\mathcal{B}$ that can break Assumption 2. $\mathcal{B}$ receives $g, X_1X_2, X_3, Y_2Y_3, T$ from an instantiation of Assumption 2. It samples random

$$\alpha, \kappa, \{a_i, c_i\}_{i\in[n]} \xleftarrow{\$} \mathbb{Z}_N,$$

sets $g = g, u_i = g^{a_i}, \tilde{u}_i = g^{c_i}$. $\mathcal{B}$ sends

$$\mathsf{pk} = (u_1, \ldots, u_n, \tilde{u}_1, \ldots, \tilde{u}_n, g, e(g,g)^{\alpha})$$

to $\mathcal{A}$.

When receiving a query for a key at position k, $\mathcal{B}$ does the following:

$k < \tau$: create a semi-functional key by sampling $r_k, z_{k,1}, \ldots, z_{k,n}, t_k \xleftarrow{\$} \mathbb{Z}_N$ and set:

$$d_k = \left(g^{r_k}(Y_2Y_3)^{t_k}, g^{\alpha}(\tilde{u}_k u_k)^{r_k}(Y_2Y_3)^{z_{k,k}}, \{u_i^{r_k}(Y_2Y_3)^{z_{k,i}}\}_{\substack{i\in[n] \\ i\neq k}} \right).$$

This is a properly distributed semi-functional key with $g_2^{\gamma} = Y_2^{t_k}$. Again, we note that the values of t_k and $z_{k,i}$ modulo p_2 and modulo p_3 are uncorrelated.

$k > \tau$: create a normal key by sampling $r_k, w_{k,1}, \ldots w_{k,n}, t_k \xleftarrow{\$} \mathbb{Z}_N$ and set:

$$d_k = \left(g^{r_k} X_3^{t_k}, g^{\alpha}(\tilde{u}_k u_k)^{r_k} X_3^{w_{k,k}}, \left\{ u_i^{r_k} X_3^{w_{k,i}} \right\}_{\substack{i\in[n] \\ i\neq k}} \right)$$

$k = \tau$: $\mathcal{B}$ sets $z_{k,i} = a_i$ and $z_{k,k} = c_k + a_k$ and samples $w_{k,1}, \ldots, w_{k,n} \xleftarrow{\$} \mathbb{Z}_N$ and sets:

$$d_k = \left(T, g^{\alpha} T^{z_{k,k}} X_3^{w_{k,k}}, \left\{ T^{z_{k,i}} X_3^{w_{k,i}} \right\}_{\substack{i \in [n] \\ i \neq k}} \right).$$

A few key points to note:

1. When changing a key to a semi-functional key, the adversary $\mathcal{B}$ itself could try and create a semi-functional ciphertext for that specific key, to try and distinguish. However, this is where the fact that this key is *nominally* semi-functional kicks in. The created ciphertext would still be decryptable by that specific key.
2. When the adversary $\mathcal{B}$ itself tries to create a semi-functional key for the semi-functional challenge ciphertext, it will again become a nominally semi-functional key, because again you would have to set $z_{k,k} = c_k$ and z_c would become $c_j + \kappa + \sum_{i \in S} a_i$.

$\mathcal{B}$ receives a set $S \subseteq [n]$ and an element $j \in [n]$, but $j \notin S$, and a message M from $\mathcal{A}$. $\mathcal{B}$ computes:

$$C = \left(M e \left(X_1 X_2, g \right)^{\alpha}, \left(X_1 X_2 \right)^{c_j + \kappa + \sum_{i \in S} a_i}, X_1 X_2 \right).$$

This sets $g^t = X_1$ and $z_c = c_j + \kappa + \sum_{i \in S} a_i$. Again, because the values a_i and c_j modulo p_2 are uncorrelated to their counterparts modulo p_1, these values are properly distributed.

If $T \in \mathbb{G}_{p_1 p_3}$ then $\mathbf{Game}_{\tau - 1}$ was properly simulated by $\mathcal{B}$, if $T \in \mathbb{G}$, however, this is a simulation of $\mathbf{Game}_{\tau}$. If $\mathcal{A}$ outputs 0, *i.e.* it thinks we are in $\mathbf{Game}_{\tau - 1}$, $\mathcal{B}$ also outputs 0, *i.e.* $T \in \mathbb{G}_{p_1 p_3}$. Similarly, if $\mathcal{A}$ outputs 1, *i.e.* it thinks we are in $\mathbf{Game}_{\tau}$, $\mathcal{B}$ also outputs 1, *i.e.* $T \in \mathbb{G}$. Hence, if the adversary $\mathcal{A}$ has an advantage in distinguishing between both hybrids, $\mathcal{B}$ also has an advantage in breaking Assumption 2. □

Lemma 6 *Suppose there exists an algorithm $\mathcal{A}$ such that*

$$\boldsymbol{Game}_n Adv_{\mathcal{A}} - \boldsymbol{Game}_{Final} Adv_{\mathcal{A}} = \epsilon.$$

Then we build an algorithm $\mathcal{B}$ with advantage ϵ in breaking Assumption 3.

Proof. Given an adversary $\mathcal{A}$ that can distinguish between the two games, $\mathbf{Game}_n$ and $\mathbf{Game}_{Final}$, we now show how to construct an adversary $\mathcal{B}$ that can break Assumption 3. $\mathcal{B}$ receives $g, g^{\alpha}X_2, X_3, g^{t\overline{D[L]}}Y_2, Z_2, T$ from an instantiation of Assumption 3. It samples random $\kappa, \{a_i, c_i\}_{i\in[n]} \xleftarrow{\$} \mathbb{Z}_N$, sets $g = g, u_i = g^{a_i}, \tilde{u}_i = g^{c_i} e(g,g)^{\alpha} = e(g^{\alpha}X_2, g)$. $\mathcal{B}$ sends

$$\mathsf{pk} = (u_1, \ldots, u_n, \tilde{u}_1, \ldots, \tilde{u}_n, g, e(g,g)^{\alpha})$$

to $\mathcal{A}$.

When receiving a query for a key at position k, $\mathcal{B}$ computes a semi-functional key by sampling

$$v_1, \ldots, v_n, r_k, t_k, w_{k,1}, \ldots, w_{k,n}, \gamma_k \xleftarrow{\$} \mathbb{Z}_N$$

and sets:

$$d_k = \left(g^{r_k} Z_2^{\gamma_k} X_3^{t_k}, g^{\alpha} X_2 (\tilde{u}_k u_k)^{r_k} Z_2^{v_{k,k}} X_3^{w_{k,k}}, \left\{ u_i^{r_k} Z_2^{v_{k,i}} X_3^{w_{k,i}} \right\}_{\substack{i\in[n] \\ i\neq k}} \right).$$

An important note to make is that $\mathcal{B}$ itself cannot create a real key, but only semi-functional keys, because the master secret key g^{α} is tied to an element of $\mathbb{G}_{p_2}$.

$\mathcal{B}$ receives a set $S \subseteq [n]$ and an element $j \in [n]$, but $j \notin S$, and a message M from $\mathcal{A}$. $\mathcal{B}$ computes:

$$C = \left(MT, \left(g^t Y_2\right)^{c_j + \kappa + \sum_{i\in S} a_i}, g^t Y_2 \right).$$

This sets $z_c = c_j + \kappa + \sum_{i\in S} a_i$. Same argument about the correlation of a_i and c_j.

If $T = e(g,g)^{\alpha t}$ then this is a properly distributed semi-functional ciphertext with message M. If T is random in $\mathbb{G}_T$, then this is a semi-functional ciphertext with a random message. If $\mathcal{A}$ outputs 0, *i.e.* it thinks we are in $\mathbf{Game}_n$, $\mathcal{B}$ also outputs 0, *i.e.* $T = e(g,g)^{\alpha t}$. Similarly, if $\mathcal{A}$ outputs 1, *i.e.* it thinks we are in $\mathbf{Game}_{Final}$, $\mathcal{B}$ also outputs 1, *i.e.* $T \xleftarrow{\$} \mathbb{G}_T$. Hence, if the adversary $\mathcal{A}$ has an advantage in distinguishing between both hybrids, $\mathcal{B}$ also has an advantage in breaking Assumption 3. □

3.5 SME with Constant Size Decryption Keys

Next, we introduce a construction that can reduce the decryption keys from $\mathcal{O}(n)$ to constant size, more specifically one group element. Unfortunately, by doing so we use the weaker selective security definition. This selectively secure scheme is based upon the broadcast encryption scheme by Boneh, Gentry, and Waters [41]. The original scheme was selectively CPA secure, therefore, we can only hope to distill a selectively secure SME from this construction.

$\mathsf{Setup}\left(1^{\lambda}, n\right)$: Let $\mathbb{G}$ be a bilinear group of prime order p. Choose random generator $g \in \mathbb{G}$ and random $\alpha, \{\beta_i\}_{i\in[n]} \in \mathbb{Z}_p$. Compute $g_i = g^{(\alpha^i)} \in \mathbb{G}$ for $i \in [2n] \setminus \{n+1\}$, and $g^{\beta_i} \in \mathbb{G}$ for $i \in [n]$. Also, choose $\gamma \in \mathbb{Z}_p$ and set $v = g^{\gamma} \in \mathbb{G}$. Output $\mathsf{pk} = \left(g, \{g_i\}_{i\in[2n]\setminus\{n+1\}}, \left\{g^{\beta_i}\right\}_{i\in[n]}, v\right)$, and $\mathsf{msk} = \left(\gamma, \{\beta_i\}_{i\in[n]}\right)$

$\mathsf{KeyGen}\left(k, \mathsf{pk}, \mathsf{msk}\right)$: This algorithm takes as input a receiver $k \in [n]$, a public key pk, and a master secret key msk. Parse pk as

$$\left(g, \{g_i\}_{i\in[2n]\setminus\{n+1\}}, \left\{g^{\beta_i}\right\}_{i\in[n]}, v\right)$$

and msk as $\left(\gamma, \{\beta_i\}_{i\in[n]}\right)$. Output $d_k = g_k^{\gamma} g_k^{\beta_k}$.

$\mathsf{Hash}\left(\mathsf{pk}, S \subseteq [n]\right)$: This algorithm takes as input a public key pk and a subset $S \subseteq [n]$. It outputs a digest $\hat{S}$ and a state st. Parse pk as $\left(g, \{g_i\}_{i\in[2n]\setminus\{n+1\}}, \left\{g^{\beta_i}\right\}_{i\in[n]}, v\right)$. Pick uniformly random $z \xleftarrow{\$} \mathbb{G}$ and compute $\hat{S} = zv \prod_{j\in S} g_{n+1-j}$.Output $(\hat{S}, \mathsf{st} = z)$

$\mathsf{Encrypt}\left(\mathsf{pk}, i, M, \hat{S}\right)$: This algorithm takes as input a public key pk, a receiver $i \in [n]$, a message M, and a digest $\hat{S}$. It outputs a ciphertext $C^* = (\mathsf{Hdr}, C)$. Parse pk as $(g, \{g_i\}_{i\in[2n]\setminus\{n+1\}}, \left\{g^{\beta_i}\right\}_{i\in[n]}, v)$. Pick $t \xleftarrow{\$} \mathbb{Z}_p$ and set

$$\mathsf{Hdr} = \left(g^t, \left(g^{\beta_i}\hat{S}\right)^t\right) = \left(g^t, \left(vg^{\beta_i} \prod_{j\in S} g_{n+1-j}\right)^t\right).$$

Compute $K = e(g, g_{n+1})^t$, note that you can compute $e(g, g_{n+1})$ as $e(g_1, g_n)$. Set $C = M \cdot K$ and output $C^* = (\mathsf{Hdr}, C)$.

$\mathsf{Decrypt}(\mathsf{pk}, d_i, S, i, C, \mathsf{st})$: This algorithm takes as input a public key pk, a private key d_i, a subset $S \subseteq [n]$, a receiver $i \in [n]$, a ciphertext C, and a state $\mathsf{st} = z$ Parse pk as $\left(g, \{g_i\}_{i\in[2n]\setminus\{n+1\}}, \left\{g^{\beta_i}\right\}_{i\in[n]}, v\right)$ and C as (Hdr, C). Parse $\mathsf{Hdr} = (c_1, c_2)$. Compute

$$K = \frac{e(g_i, c_2)}{e\left(d_i \prod_{\substack{j\in S \\ j\neq i}} g_{n+1-j+i}, c_1\right)}. \tag{3.3}$$

Output $M \leftarrow C \cdot K^{-1}$.

Correctness. We show that Equation (3.3) indeed recovers K, which then can be used to recover the plaintext message M

$$\begin{aligned}
\frac{e(g_i, c_2)}{e\left(d_i \prod_{\substack{j\in S \\ j\neq i}} g_{n+1-j+i}, c_1\right)} &= \frac{e\left(g^{(\alpha^i)}, \left(v g^{\beta_i} \prod_{j\in S} g_{n+1-j}\right)^t\right)}{e\left(\left(g^{(\alpha^i)}\right)^\gamma \left(g^{(\alpha^i)}\right)^{\beta_i} \prod_{\substack{j\in S \\ j\neq i}} g_{n+1-j+i}, g^t\right)} \\
&= \frac{e\left(g^{(\alpha^i)}, g_{n+1-i}\right)^t e\left(g^{(\alpha^i)}, v \prod_{\substack{j\in S \\ j\neq i}} g_{n+1-j}\right)^t e\left(g^{(\alpha^i)}, g^{\beta_i}\right)^t}{e\left(\left(g^{(\alpha^i)}\right)^\gamma \prod_{\substack{j\in S \\ j\neq i}} g_{n+1-j+i}, g^t\right) e\left(\left(g^{(\alpha^i)}\right)^{\beta_i}, g^t\right)} \\
&= e(g, g_{n+1})^t \frac{e\left(g, v^{(\alpha^i)} \prod_{\substack{j\in S \\ j\neq i}} g_{n+1-j+i}\right)^t}{e\left(\left(g^{(\alpha^i)}\right)^\gamma \prod_{\substack{j\in S \\ j\neq i}} g_{n+1-j+i}, g\right)^t} \\
&= e(g, g_{n+1})^t = K
\end{aligned}$$

Efficiency. In this construction it can be seen that the size of the public key is $3n + 1$ elements in $\mathbb{G}$. The msk is $n + 1$ elements in $\mathbb{Z}_p$. The size of the secret keys is just 1 group element, which is a big improvement in comparison with our first construction in Section 3.4.

Hash can be computed by doing $|S|$ multiplications within $\mathbb{G}$, therefore, this algorithm runs in time $\mathcal{O}(|S|)$. The output of the algorithm is just 1 group element. On the other hand,

$\mathsf{Encrypt}$ runs in constant time, because it only computes 1 multiplication and 2 exponentiations in $\mathbb{G}$, and 1 pairing, 1 exponentiation, and 1 multiplication in $\mathbb{G}_T$. The output of this algorithm is 2 elements in $\mathbb{G}$ and 1 element in $\mathbb{G}_T$.

Finally decryption runs in $\mathcal{O}(|S|)$ because it needs to do $|S|-1$ multiplications, 2 pairings, and 1 multiplication and 1 division in $\mathbb{G}_T$.

Privacy. To show privacy we need to create a simulator such that for any set S of size at most $n = \mathsf{poly}(\lambda)$ for any polynomial function $\mathsf{poly}(\cdot)$, let $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, n)$, $\forall i \in [n] : K_i = \mathsf{KeyGen}(i, \mathsf{pk}, \mathsf{msk})$, and $(\hat{S}, \mathsf{st}) \leftarrow \mathsf{Hash}(\mathsf{pk}, S)$, it holds that

$$\left(\mathsf{pk}, \{K_i\}_{i\in[n]}, \hat{S}\right) \overset{c}{\approx} \left(\mathsf{pk}, \{K_i\}_{i\in[n]}, \mathcal{S}\left(1^\lambda\right)\right).$$

A very straight forward simulator is just sampling a uniform random element in $\mathbb{G}$, because of the randomly chosen blinding factor z during hashing, it is clear that both outputs are perfectly indistinguishable.

Security. We introduce a selective BDHE assumption, note that this can be stated as a GBDHE, except that some of the parameters can be chosen adversarial in advance. We can prove this assumption to be generically secure (*i.e.* in the generic bilinear group model [180]), by using the generic proof template of Boneh, Boyen, and Goh [40].

Assumption 4 *After given $S \subset [n]$ and $i \in [n]$, with $i \notin S$ by an adversary, a challenger outputs*

$$\left(g, h = g^t, \left\{g_i = g^{(\alpha^i)}\right\}_{i\in[2n]\setminus\{n+1\}}, \left\{g^{\beta_i}\right\}_{i\in[n]}, v = g^\gamma, \left\{g_i^\gamma g_i^{\beta_i}\right\}_{i\in[n]}, \left(vg^{\beta_i}\prod_{j\in S} g_{n+1-j}\right)^t\right),$$

and a value T that is either $e(h, g_{n+1})$ or a random element in $\mathbb{G}_T$. The assumption states that the adversary has negligible advantage in distinguishing between the two possibilities of T.

Lemma 7 *Given $S \subset [n]$ and $i \in [n]$, the above assumption is a selective BDHE assumption, ie: for the following polynomials, f is independent of (P, Q), if $i \notin S$.*

$$P = \left(1, t, \left\{\alpha^k\right\}_{k\in[2n]\setminus\{n+1\}}, \{\beta_k\}_{k\in[n]}, \gamma, \left\{\gamma\alpha^k + \beta_k\alpha^k\right\}_{k\in[n]}, \kappa, \gamma t + \beta_i t + \kappa t + t\sum_{j\in S}\alpha^{n+1-j}\right)$$
$$Q = (1)$$
$$f = t\alpha^{n+1}$$

Proof. To show that f is independent of (P, Q) we have to show that f cannot be constructed in the following way: $f = \sum_i \sum_j p_i p_j + \sum_k q_k$, where p_i, p_j are polynomials in P and q_k polynomials in Q.

To create the term $t\alpha^{n+1}$, we need to look at the term $\gamma t + \beta_i t + \kappa t + t\sum_{j\in S}\alpha^{n+1-j}$, because none of the other terms contain α^{n+1}, or any α^k in combination with t. To achieve the term we multiply this with α^k for $k \in S$. This results in $\gamma t\alpha^k + \beta_i t\alpha^k + \kappa t + t\sum_{j\in S}\alpha^{n+1-j+k}$. Now, to cancel out the term $\beta_i t\alpha^k$, we note that $k \neq i$ because $i \notin S$, therefore we cannot hope to use any of the $\gamma\alpha^k + \beta_k\alpha^k$. The only option to try and cancel out that term is by using $\gamma t + \beta_i t + \kappa t + t\sum_{j\in S}\alpha^{n+1-j}$ again if $k = n + 1 - j$ for some $j \in S$ and multiply it with β_i. However, now we introduced the term $\beta_i^2 t$, the only way this term can be created is in the same way we did, which will lead us in circles. This concludes the proof. □

Theorem 4 *If Assumption 4 holds, then our SME construction above is a selectively secure SME.*

Proof. Given an adversary $\mathcal{A}$ for our SME construction, we build an adversary $\mathcal{B}$ to break Assumption 4. First $\mathcal{B}$ receives a set $S \subset [n]$ and an index $i \in [n]$, but $i \notin S$. It forwards these elements to the challenger of the assumption. $\mathcal{B}$ receives

$$\left(g, h = g^t, \left\{g_i = g^{(\alpha^i)}\right\}_{i\in[2n]\setminus\{n+1\}}, \left\{g^{\beta_i}\right\}_{i\in[n]}, v = g^\gamma, \left\{g_i^\gamma g_i^{\beta_i}\right\}_{i\in[n]}, z, \left(zvg^{\beta_i}\prod_{j\in S} g_{n+1-j}\right)^t\right)$$

from the challenger. It sends

$$pk = \left(g, \{g_i\}_{i\in[2n]\setminus\{n+1\}}, \left\{g^{\beta_i}\right\}_{i\in[n]}, v\right)$$

to the adversary $\mathcal{A}$. $\mathcal{A}$ will respond with a message M. Now, $\mathcal{B}$ can respond with a correctly constructed digest $\hat{S} = zv\prod_{j\in S} g_{n+1-j}$, a state z,

$$\mathsf{Hdr} = \left(h, \left(zvg^{\beta_i}\prod_{j\in S} g_{n+1-j}\right)^t\right),$$

and $C = M \cdot T$ which is a correctly formed ciphertext that encrypts M when $T = e\left(h, g_{n+1}\right)$, and an encryption of a random message when T is random. If $\mathcal{A}$ says this is an encryption of the message M then $\mathcal{B}$ responds to the challenger that $T = e\left(h, g_{n+1}\right)$. On the other hand, if $\mathcal{A}$ says this is an encryption of a random message, then $\mathcal{B}$ responds to the challenger by saying T is random. If $\mathcal{A}$ has a non-negligible advantage in breaking this SME, then $\mathcal{B}$ has a non-negligible advantage in breaking Assumption 4. □

3.6 Extensions and Applications

In this section we introduce several extensions and improvements upon our construction of laconic OT, as well as a specific application for our new primitive SME. First we show how to decrease the CRS size by increasing the size of the digest. Next, we present how to get updatable laconic OT for our schemes, which is an extension of normal laconic OT that was introduced by Cho *et al.* [59] and that is important for the applications presented in their paper.

3.6.1 Optimization of the Laconic OT Construction

Given the fact that the CRS in our laconic OT constructionsis still quadratic and linear in the size of the database, respectively, we introduce the following optimization by striking a new balance between the sizes of the different components. We decrease the size of the CRS

by increasing the size of the digest. By doing this we break the efficiency definition of laconic OT that states that the digest can only depend on the security parameter, however, as we show in Section 3.7.2, we get a much more practical result by doing so.

We start with the laconic OT construction based on an SME scheme as shown in Section 3.2.1, remember that we set $E = \left\{2i - \overline{D[i]} \middle| \forall i \in [|D|]\right\}$,with D being the original database. Now we initiate the SME with size $2\sqrt{|D|}$, instead of $n = |E|$. We distribute all positions $L \in D$ to their respective bucket of size $2\sqrt{|D|}$. We can do this by computing $y = \left\lfloor \frac{i}{\sqrt{|D|}} \right\rfloor$, next we compute $\forall y, E_y = \{x - y\sqrt{n} \mid x \in E \cap]y\sqrt{n}, (y+1)\sqrt{n}]\}$ .The size of the CRS clearly decreases to $\mathcal{O}\left(\sqrt{|D|}\right)$, however, the digest will grow to size $\mathcal{O}\left(\sqrt{|D|}\right)$.

3.6.2 Updatable Laconic OT

For the applications in the original paper by Cho *et al.* [59] we need a slightly different version of laconic OT which the researchers called *updatable laconic OT*. They introduced two more algorithms SendWrite and ReceiveWrite, and required some form of sender privacy where the receiver would not learn the original digest. This definition was very tailored to their specific construction and application. Instead we propose a new version of updatable laconic OT by introducing an algorithm called UpdateDigest, an algorithm that the sender can run on their own, followed by sending the necessary information to the receiver, sender privacy can then be achieved by performing this operation in a garbled circuit.

Definition 9 (updatable laconic OT) *An updatable laconic OT scheme exists of the algorithms* (crsGen, Hash, Send, Receive) *as defined in Definition 1, additionally the algorithm* UpdateDigest *is added with the following syntax.*

UpdateDigest(crs, digest, L, b) $\rightarrow$ digest*. *It takes as input a common reference string* crs*, a digest* digest*, a location* $L \in \mathbb{N}$*, a bit* $b \in \{0, 1\}$ *to be written. It outputs a new digest* digest*.

On top of the properties of the normal laconic OT scheme we require the following properties:

Correctness with regards to writes: *For any database of size* $\ell = \mathsf{poly}(\lambda)$ *for any polynomial function* $\mathsf{poly}(\cdot)$*, any memory location* L *in*$[\ell]$*, any bit* $b \in \{0,1\}$ *the following holds. Let* D^* *be identical to* D *except that* $D^*[L] = b$*,*

$$\Pr\left[\mathsf{digest}^* = \mathsf{digest}' \;\middle|\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{crsGen}(1^\lambda, \ell) \\ (\mathsf{digest}, \hat{D}) \leftarrow \mathsf{Hash}(\mathsf{crs}, D) \\ (\mathsf{digest}^*, \hat{D}^*) \leftarrow \mathsf{Hash}(\mathsf{crs}, D^*) \\ \mathsf{digest}' \leftarrow \mathsf{UpdateDigest}(\mathsf{crs}, \mathsf{digest}, L, b) \end{array}\right] = 1,$$

where the probability is taken over the randomness of crsGen *and* UpdateDigest.

Note that the definition is quite similar to the definition in the original paper with respect to correctness, as described above, we leave sender privacy to the addition of a garbled circuit. Moreover, this algorithm outputs the updated digest in a normal representation while Cho *et al.* represent it by using a label corresponding to every bit, however, this is very tailored to the applications and setting they were working in, we believe there is major benefit in defining this more generally the way we do.

Updatable Laconic OT Construction Based on SME of Section 3.4. First, we show the construction of this new algorithm UpdateDigest in the context of the laconic OT construction that is derived from the SME scheme in Section 3.4. Correctness of this algorithm follows by simple inspection and as mentioned earlier sender privacy follows from the properties of the garbled circuit when performing this algorithm in a garbled circuit.

We define UpdateDigest as follows:

$\mathsf{UpdateDigest}(\mathsf{crs}, \mathsf{digest}, L, b)$: Output $\frac{\mathsf{digest} \cdot u_{2L-\bar{b}}}{u_{2L-b}}$.

Updatable Laconic OT Construction Based on SME of Section 3.5. Similarly, we show the construction of this new algorithm UpdateDigest in the context of the laconic OT construction that is derived from the SME scheme in Section 3.5.

We define $\mathsf{UpdateDigest}$ as follows:

$\mathsf{UpdateDigest}(\mathsf{crs}, \mathsf{digest}, L, b)$: Output $\frac{\mathsf{digest} \cdot g_{n+1-2L+\bar{b}}}{g_{n+1-2L+b}}$.

Note on Achieving Cho *et al.*'s Applications. When running the above instantiations of updatable laconic OT inside a garbled circuit we have all the same components as presented in Cho *et al.*, therefore, we can use our version of ℓOT in their applications. However, in terms of efficiency, we have to look at the increased CRS size in our construction in comparison with the construction of Cho *et al.* We note that in the application, the CRS is hard coded inside a garbled circuit, leading to a large garbled circuit, but given our much more efficient overall construction, this is still better than using Cho *et al.*'s construction. The other difference we have to take into account is receiver time, but given that this algorithm is running outside of garbled circuits, we refer the reader to our comparison in Section 3.7 to note that the overall efficiency of our scheme will easily outweigh Cho *et al.*'s construction. One downside remains the trusted setup that our construction needs, which Cho *et al.* does not.

3.7 Evaluation and Comparison

In this section we will evaluate our laconic OT scheme as derived from the SME constructions in Section 3.4 and Section 3.5, how to derive a laconic OT scheme can be found in Section 3.2.1. More specifically we will look at the concrete efficiency and compare this with previous work. We recall that we compare the derived laconic OT schemes instead of another primitive for ease of presentation and such that we can compare with the initial work of Cho *et al.* [59].

A difficulty in comparing with previous work is the plethora of papers that build upon the initial work, improving the construction step by step, but usually focusing on the underlying techniques and not necessarily laconic OT itself. Understanding how these all fit together is no easy task. Therefore, when we talk about Cho *et al.*, we actually mean a combination

of different papers. The basic construction is taken from the work by Cho *et al.*, but quite immediately after that paper, some improvements were made by Dottling and Garg [70]. The last improvement on which we base ourselves was made by Cong *et al.* [62], which in turn based their improvements on Goyal and Vusirikala [107].

Furthermore, we compare with the work from Goyal *et al.* [108] that introduces constructions for One-Way Functions with Encryption (OWFE). It is easy to see that OWFE implies laconic OT when the one-way function has the necessary compression to act as the hash function in the laconic OT scheme. Goyal *et al.* have such construction and we compare with their most efficient construction based on the q-DBDHI assumption.

Finally, we also compare with the work of Alamati *et al.* [9] that shows a construction for laconic OT from the ϕ-hiding assumption based on another OWFE construction by Goyal *et al.* [108]. Although, this construction seems to have great asymptotic efficiency, because of the ϕ-hiding assumption the construction happens in a rather large RSA group leading to concrete inefficiencies.

3.7.1 Asymptotic Efficiency

First we compare the asymptotic efficiency between previous work and this work. We compare the laconic OT scheme that is constructed from the SME constructions in Section 3.4, Section 3.5, and with the optimization described in Section 3.6.1. In Table 3-II we show the computational and communication efficiency of the different algorithms. In terms of computation time, we make a $\log n$ improvement for both Hash and Send in comparison with Cho *et al.*, but we go from polylogarithmic to linear decryption time.

In terms of communication complexity, the common reference string in our scheme is much larger compared to Cho *et al.* Nevertheless, the size of the encryption generated by Send decreases from logarithmic to constant size.

Table 3-II. Comparison of asymptotic computation and communication efficiency.

	crsGen	Hash	Send	Receive	crs size	Hash size	Send size
Cho *et al.* [59]	$\mathcal{O}(1)$	$n\mathsf{poly}(\log n)$	$\mathsf{poly}(\log n)$	$\mathsf{poly}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
Goyal *et al.* [108]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Goyal *et al.* [108] + §3.6.1	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(1)$
Alamati *et al.* [9]	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
This work §3.4	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
This work §3.4 + §3.6.1	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(n)$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(1)$
This work §3.5	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
This work §3.5 + §3.6.1	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(\sqrt{n})$	$\mathcal{O}(1)$

All our asymptotic efficiencies are similar to the ones in Goyal *et al.*, but Alamati *et al.* achieve better asymptotic efficiency by reducing the common reference string to constant size, hence, achieving overall better efficiency than Cho *et al.* with the exception of receiving time.

3.7.2 Concrete Efficiency

Now we will look into concrete efficiency of our scheme in comparison with previous work.

To the best of our knowledge, Cong *et al.* [62] are the only ones to make an estimate of the Send size in Cho *et al.* [59], they estimate that size to be 11TB, this is based on *one* curve multiplication at every level in their tree based construction.[5] However, when looking closely to the circuits that get garbled at every stage of the tree, they contain $\sim 2\lambda$ curve multiplications, leading to a much bigger circuit. Because in the work of Cong *et al.* they use a database of size 2^{31}, we will be using the same number throughout our comparisons.

Computing Efficiency for Cho *et al.*. To compute the concrete efficiency of Cho *et al.* we will use secp192k1 [170], because this seems to be the only curve for which someone actually computed the number of gates a circuit would contain when doing a curve multiplication [128]. Because this curve only has 96 bits of security, we do not hope to achieve any better security, and will use this security parameter for all other computations as well.

Moreover, we take the 30% optimization of Cong *et al.* into account as well, leading to a

[5]Remember that the construction in Cho *et al.* [59] contains a path from root to leaf on a tree, while at each level a garbled circuit is computed.

size of 1.2 petabytes. We compute the size using the following formula:

$$\text{size } = ((2\lambda \times d + 1) \times 19\,200\,000\,000 \times 160 \text{ bits}) \times 0.7,$$

where λ is the security parameter, d is the number of levels in the tree, the number 19.2 billion corresponds to the amount of non-XOR gates for doing a curve multiplication [128], and finally, by using half gates [200] for each non-XOR gate [135] we need two ciphertexts for each non-XOR gate which we will assume are 80 bits each. We also use the 0.7 factor to account for the 30% optimization by Cong *et al.* Also note that d is $\mathcal{O}(\log n)$, but not exactly $\log n$ because each leaf can contain 2λ bits, we compute d accordingly.

Computing Efficiency for Goyal *et al.*. We compute the concrete efficiency of Goyal *et al.* over the BLS12-381 curve, similar to how we compute our efficiency. Given the fact that both constructions are very similar we see very similar results. Only in receiver time, we see that our construction really shines in comparison with Goyal *et al.* We give some more details about computing this concrete receiver time below.

Computing Efficiency for Alamati *et al.*. To compute the concrete efficiency for Alamati *et al.* we have to compute a few of the parameters first. We note that the authors mention a security parameter λ, but next take an RSA composite number N with λ bits. If we want to achieve about 128 bit security, we need to take $\lambda = 2,048$. Next, for their PPRF trick that reduces the crs to constant size, we need to compute the value ξ. They write that this value needs to be $\mathcal{O}\left((\log 2^{\kappa})^2\right)$, where κ is the size of the different primes they use as exponents and $5\kappa \leq \lambda$. Therefore, κ needs to be around 409 bits, which leads to $\xi \approx (409 \cdot \log 2)^2 \approx 15,129$. Finally, we work in the group $\mathbb{Z}_{N^{\xi+1}}$, i.e. a group with numbers of size up to $2,048 \cdot 15,130 = 30,986,240$. Although, this is all still practically doable as the numbers in Table 3-III show, it is not ideal.

Moreover, receiver time is still linear in the database size and it is not possible to use the same $\sqrt{}$-optimization as described in Section 3.6.1 because this would grow the digest size to

Table 3-III. Comparison of concrete efficiency, with $n = 2^{31}$. Cho *et al.* is estimated over elliptic curve secp192k1 with $\lambda = 96$, Goyal *et al.* and our work are estimated on the BLS12-381 curve with security parameter roughly 120 bits, and Alamati *et al.* is estimated using an RSA group of 2048 bits to achieve around 128 bit security. (kB = 1000 bytes, MB = 1 million bytes, GB = 1 billion bytes, PB = 1 quadrillion bytes, EB = 1 quintillion bytes)

	λ	crs size	Hash size	Send size	Receive
Cho *et al.* [59]	96 bits	4.6kB	48 bytes	1.2PB	-
Goyal *et al.* [108]	$\sim$ 120 bits	103.1GB	48 bytes	1.25kB	8.1 days
Goyal *et al.* [108] + §3.6.1	$\sim$ 120 bits	2.2MB	2.2MB	1.25kB	15.1s
Alamati *et al.* [9]	$\sim$ 128 bits	0.8MB	3.9MB	7.7MB	85.9s
This work §3.4	$\sim$ 120 bits	221.4EB	48 bytes	1.34kB	27.7 minutes
This work §3.4 + §3.6.1	$\sim$ 120 bits	103.1GB	2.2MB	1.34kB	38.7ms
This work §3.5	$\sim$ 120 bits	412.3GB	48 bytes	1.34kB	27.7 minutes
This work §3.5 + §3.6.1	$\sim$ 120 bits	8.9MB	2.2MB	1.34kB	38.7ms

around 180GB, which is clearly undesirable in a laconic OT scheme.

Computing Concrete Receiver Computation Time. Finally we show how we achieve concretely better receiver time in comparison with all previous work. First, we note that computing the receiver time in the work by Cho *et al.* is nearly impossible, but given the Send size of 1.2PB it would already take 11 days just to transfer this amount of data over a 10Gbps line, let alone handle this amount of data on a reasonably sized computer.

Therefore, we focus our efforts on comparing receiver time with the work of Goyal *et al.* and Alamati *et al.* To get a fair estimate of the receiving time in all constructions we benchmarked all operations on an Apple M1 Max. For Goyal *et al.* and our work we use: 775ns for a multiplication, 325μs for an exponentiation, 1393μs for a pairing, and 4.5μs for a multiplication in the target group. In Goyal *et al.*, we ignored the symbolic evaluation of a degree-n polynomial. For Alamati *et al.* we benchmarked a multiplication in the respective group to take 40ns, we ignored the single exponentiation.

In Table 3-III, we show the full comparison of concrete efficiency.

3.8 Related Work

Laconic Oblivious Transfer (OT) was first introduced by Cho *et al.* [59] in comparison with regular OT, laconic OT requires the receiver's outgoing message to be small, more specifically, it shouldn't grow with the size of the database. They prove their system to be secure based on the Decisional Diffie-Hellman (DDH) assumption in the common reference string (CRS) model. To achieve this they use somewhere statistically binding (SSB) hash functions [122] combined with hash proof systems. This specific technique has been refined in several subsequent papers changing names to Chameleon Hashing [70], Hash Encryption [71], or Hash Garbling [47]. Most recently this technique was further optimized in the context of Registration Based Encryption (RBE) [62, 90, 92, 107]. Even with all these improvements the construction remains merely theoretical, although the asymptotic efficiency seems quite optimal, implementing the scheme would require giant garbled circuits impossible to create and evaluate on any normal sized computer.

Goyal *et al.* [108] introduced a construction for One-Way Functions with Encryption (OWFE) from which you can easily build laconic OT in the special case where the one-way function also has a compression property. This is the case for their particular construction, both the common reference string and the receiver time is linear in the database size, similar to our constructions. However, receiver time in our scheme is orders of magnitude faster. In 2021, Alamati *et al.* [9] improved one of the schemes from Goyal *et al.*, achieving nearly optimal asymptotic efficiency, but receiver time is still linear. Moreover, it is not possible to apply the square root-optimization to their scheme, because the digest size would grow to several gigabytes. The scheme is also based on the ϕ-hiding assumption, which is not desirable.

The application that was presented in the original paper on laconic OT [59], has been further developed in work by Garg *et al.* [91] Other laconic primitives such as laconic function

evaluation were achieved by Döttling *et al.* [72], Quach *et al.* [168], and Agrawal and Roşie [8]. In the work of Döttling *et al.* [73], they introduce the slightly stricter definition of private laconic OT.

We create the new primitive Set Membership Encryption inspired by Broadcast Encryption, a primitive that was first introduced by Fiat and Naor [82]. We are interested in the instantiations that have a short ciphertext as introduced by Boneh *et al.* [41], which has very good efficiency, but we can only prove a derivative of this scheme to be selectively secure. Therefore, we look to adaptively secure broadcast encryption schemes that are proven secure using the Dual System Encryption technique of Waters [190], but we distill a broadcast encryption system of the follow-up work by Lewko and Waters [141] in a composite order bilinear group, unfortunately, the private keys grow to $\mathcal{O}(n)$.

3.9 Conclusion

We introduced a new primitive which we call set membership encryption, where one party can define a set of receivers and a second party can encrypt to one specific receiver if and only if that receiver was part of the original set of receivers. We show how to build laconic OT from this primitive. Next, we show two constructions of this new primitive, the first one has linear sized decryption keys, but can be proven adaptively secure, while the second construction has constant sized decryption keys, but is only selectively secure.

Finally, we evaluate the efficiency of the laconic OT schemes that can be derived from said set membership encryption constructions. We compare with previous work on laconic OT and show that ours is several orders of magnitude more efficient.

Future Work. Improving the size of the public parameters is important future work to further increase the efficiency of this primitive. Given work on improving the efficiency of broadcast encryption we could hope to similarly improve this new primitive. Studying

if this could happen under the stronger adaptive security should be part of that future work. Another interesting question is if we can create a private or anonymous version of set membership encryption similar to what has been studied for private/anonymous broadcast encryption.

We give a few pointers of what can be investigated in future work:

Using different broadcast encryption schemes Given the extensive literature on broadcast encryption, it seems likely that other schemes might have the same type of property, where two keys for a different set of receivers can somehow be bound together. This could lead to even more efficient laconic OT constructions from bilinear maps or other assumptions.

Decreasing CRS size Given the nature of how we use broadcast encryption, it seems inherent that the CRS is always going to be linear in the size of the database. However, we can hope to create constructions that can generate both public key and private keys on the fly. Note that for the private keys, this would mean that the master secret key or some derivation probably needs to be part of the CRS, in which case it is important that the binding between both parts of the key is happening inside the msk.

Decreasing decryption time Although the linear decryption time follows directly from the BE schemes that we use, there is a lot of research on anonymous BE schemes. These schemes do not take the receiver set as input during decryption, therefore, we could hope that they will not be linear in the size of that set of receivers and therefore, not linear in the size of the database during decryption.

Chapter 4

Efficient Proofs of Software Exploitability for Real-world Processors

This chapter is based on joint work with Matthew Green, Mathias Hall-Andersen, Eric Hennenfent, Gabriel Kaptchuk, and Benjamin Perez, published at PoPETs 2023 [111]

4.1 Introduction

The proliferation of complex and critical software systems has given rise to the bug bounty paradigm, in which independent vulnerability research teams uncover and disclose ways to exploit deployed software in exchange for financial rewards. This process has resulted in the disclosure of several high-profile exploits in recent years [166], and hundreds of millions of dollars are awarded in bounties annually.

While bug bounty programs are invaluable to improving the security of software, they are plagued by issues of trust. Because vulnerability researchers and bug bounty program managers are not part of the same organization—and likely have no prior relationship—each side must trust that the other will fulfill their obligations honestly. Specifically, bug bounty program managers must trust that vulnerability research teams are not overselling their

capabilities and have discovered a serious exploit. On the other hand, vulnerability research teams worry that those managing bug bounty programs will adaptively change the reward after disclosure of the exploit, claiming that the exploit does not meet some criteria.

Currently, vulnerability researchers and bug bounty program managers bridge this trust gap by having the vulnerability research team "prove" its knowledge of an exploit using a video recording. Concretely, the bug bounty program will challenge the vulnerability research team to perform an operation that should be impossible (*e.g.*, launching the calculator application) and visually record the program execution. These proofs lack soundness, as video can easily be manipulated and cannot prove that the runtime environment matches the one specified by the bug bounty program. As such, the state of the art still leaves significant trust gaps within the bug bounty ecosystem.

In this work, we design a toolchain that bridges this trust gap using cryptographically sound proofs of exploit. These proofs give a computational guarantee that the vulnerability research team can exploit the system within the specified runtime environment, and they cannot be manipulated or forged. To ensure that these proofs do not disclose anything else to the bug bounty program team, we employ zero-knowledge (ZK) [99, 100] proofs, a class of proof systems that reveals nothing to the verifier beyond the veracity of the statement. Access to ZK proofs of exploit would allow vulnerability researchers and bug bounty programs to negotiate rewards without requiring significant leaps of faith.

Designing efficient ZK proofs of exploit requires both overcoming significant engineering challenges and non-trivial theoretical contributions. While prior work [115, 116] has contemplated similar applications, their systems are limited to proving the existence of *potential vulnerabilities or bugs* in publicly available source code—falling short of meeting the needs of the vulnerability research market. In our work, we precisely model real processor architectures and runtime environments within the ZK protocol, allowing our proofs to reason directly

about compiled binaries. Therefore, the proofs that our toolchain produces guarantee that the exploits will work on hardware. This level of fidelity is essential for allowing vulnerability research teams to precisely articulate and demonstrate their capabilities.

Envisioned Workflows. In order to illustrate the value of our techniques, consider three concrete ways that cryptographically sound proofs of exploit could be used:

(1) A vulnerability research (VR) team responds to a public bug bounty by submitting their ZK proof of exploit. Once the sponsor has verified the proof, a reward amount is determined and put into escrow until the VR team submits the exploit.

(2) A VR team discovers a bug in a piece of software for which there is no bug bounty program. If the developers choose not to award a bounty after initial discussions, the VR team could post the ZK proof of exploit to a public website, informing users that their existing systems are at risk. Critically, this does not *reveal* the exploit to malicious actors who might want to use the exploit to attack live systems. We note that this would also put pressure on developers to issue a bounty and patch their software, as responsible users will likely transition away from their products.

(3) A VR team discovers a bug in a piece of legacy software which is no longer maintained, or is running on devices that cannot perform firmware updates. The VR team can post the proof of vulnerability to a public website, creating a highly trustworthy warning against using the legacy software. If using the legacy software is unavoidable, we note that users could crowdsource funds to hire the VR team to design and issue a patch.

We note that these are only potential examples, and proofs of exploit may be valuable in other workflows.

4.1.1 Contributions

In this work, we design the first end-to-end modular toolchain that facilitates the creation of ZK proofs of program exploitability.[6] The toolchain takes in two inputs: (1) a public compiled binary,[7] and (2) the prover's private input that exploits a vulnerability in that program. Given these inputs, it then produces a non-interactive zero-knowledge proof (NIZK) of correct execution. This is conducted by evaluating the binary as a RAM program using a Boolean processor circuit. While previous work has explored the evaluation of RAM machines using custom-built processors, our system employs real-world processor architectures; to make our system efficient, we introduce several novel processor-agnostic techniques that reduce the size of the resulting circuit. Specifically, we reduce the size of the circuit from $O(t \log(t))$ to $O(t)$, where t is the number of processor cycles executed during program execution.

To evaluate the effectiveness of our toolchain, we produced ZK proofs of exploit for MSP430 binaries. First, we design a custom circuit implementation of the MSP430 processor that is optimized for ZK; this requires modeling system calls (syscalls) and complex addressing modes while minimizing the number of non-linear gates. Second, we provide the first public, generic implementation of the Katz, Kolesnikov, and Wang (KKW) "MPC-in-the-head" ZK protocol [132] and incorporate several significant improvements. Specifically, we show that the MPC-in-the-head with preprocessing paradigm that they propose can be modified to allow for optimized ring switching between Boolean and arithmetic representations, resulting in significantly more efficient proofs. Finally, we demonstrate the effectiveness of our approach by producing proofs of exploit for the Microcorruption CTF [3], a set of hacking challenges that run on an MSP430 processor and cover many common exploitation techniques such

[6] Although prior work has explored the possibility of proving the existence of *bugs* in source code, our work addresses a fundamentally harder problem of demonstrating that a bug can be exploited into a full exploit. We carefully contrast these two approaches in Section 4.3.

[7] Our toolchain can naturally also operate from program source, which is compiled using a standard compiler.

as buffer overflow, command injection, and ROP gadgets. The Microcorruption challenges also require bypassing mitigations such as address space layout randomization (ASLR), data execution protection (DEP), and stack canaries. Our toolchain can produce NIZK proofs about MSP430 programs at 216 instructions per second and 119 KB per instruction.[8]

Limitations. Our approach allows proofs about exploits that can be represented as a predicate over the processor states over a program's execution. This means that there are some classes of exploits about which we cannot provide proofs, like exploits that rely on microarchitectural bugs such as Spectre and Meltdown. Similarly, Row Hammer-style exploits cannot be expressed as such a predicate, as they require modeling physical properties of RAM. Accurately modeling these systems is challenging, independent of zero-knowledge proving; as such, these exploits are beyond the scope of this work. We note, however, that only the most sophisticated actors could successfully launch such an attack, and there are no documented cases of such exploits being used in the wild.

We note that our proofs do not attempt to conceal the running time of the exploit; the number of processor ticks required is included as a public part of the statement. This is a standard relaxation in prior work [29, 30, 32, 115], and given the trade-off of less efficient proofs, it is easy to "pad-out" the running time to conceal the trace length. Additionally, we note that any low-entropy probabilistic protections (*e.g.* ASLR) will always be vulnerable to computationally powerful adversaries, both for adversaries attacking live systems, *e.g.* using brute force, and for a prover generating a proof of exploit, *e.g.* grinding on random seed selection. This means that the meaning of a proof of exploit that overcomes low-entropy probabilistic defenses are nuanced: (1) when a proof is generated interactively and the processor randomness is sampled by the verifier, the proof implies that the prover has an exploit strategy that works on average, but may not always work; (2) when the proof is generated non-interactively, *i.e.* a computationally powerful prover may (invisibly) expend

[8]For hardware specifications, see Section 4.8

significant resources generating an accepting proof, the proof implies that there exists processor randomness such that the prover possesses a working exploit strategy.

Finally, we note that while our solution demonstrates the proofs of exploit are already practical, there remains more effort—both research and engineering—for the solution to be simple and easy to use. For example, vulnerability researchers must select the statement that they wish to prove carefully. Choosing the wrong statement could result in a proof that verifies but is semantically meaningless.

Ethical Concerns. Software exploits can be used to cause harm to people and organizations and there exist online markets where exploits are sold for nefarious purposes. As such, the techniques that we develop might also be used by individuals intent on causing harm. We note, however, that our techniques do not meaningfully increase the capabilities of these communities; allowing hackers prove—with cryptographic soundness error—that they know an exploit only serves to make exploit markets more trustworthy and more easily monitored. Critically, our techniques do not make it easier for attackers to discover or exploit vulnerabilities or meaningfully increase a hacker's power to conduct blackmail.

4.2 Technical Overview

4.2.1 Background: Zero-Knowledge and Ben-Sasson et al.'s RAM Reduction

Zero-knowledge proofs of knowledge (ZK) [99, 100] allow a prover to convince a verifier that they hold a witness demonstrating that some public statement is a member of an NP language without revealing anything beyond the membership itself. ZK techniques are now concretely efficient [10, 22, 34, 35, 50, 57, 66, 96, 116, 127, 132, 192, 196, 197] and power a number of practical applications [31, 148, 185, 201]. For formal definitions of ZK proofs of knowledge, see [163].

Most research on ZK focuses on the case in which the statement is provided in a format amenable to efficient proving systems (*e.g.*, a circuit or algebraic relation). Therefore, most proof techniques now *require* that the relations have such a representation. This requirement can be unnatural and cumbersome, forcing implementers to translate a relation from its "natural" representation to the representation supported by the prover. This process frequently involves error-prone manual effort or the use of an immature circuit compiler [28, 145, 151, 189].

RAM Reduction. Ben-Sasson et al. [29, 30, 32] proposed an efficient circuit-based approach for proving the correct execution of RAM programs which has also been used by more recent works [85, 115, 117]. They represent the execution of the RAM program with two different traces. The first is the *execution-ordered trace*, wherein each step represents a single iteration of a processor circuit, including instruction bytes, a register file, and the alleged contents of memory being accessed. The second is the *memory-ordered trace*, containing the set of memory reads and writes sorted by address, with ties broken by the operation that was executed first. Proving that these traces represent an honest execution of the RAM program consists of the following:

1. **Execution Trace Consistency.** For each step in the execution trace, the proof must demonstrate that the input and output states represent a valid transition. This is done using a circuit that represents the processor. The input to each evaluation of this circuit is a fixed number of values drawn from RAM, a register file, and other auxiliary data that may be useful in verifying correct execution. This circuit will output 1 if the circuit produces the same output as the real RAM program.

2. **Memory Trace Consistency.** Each step in the trace involves reading and writing some values from RAM. Naïvely ensuring that these reads and writes are consistent with the previously executed instructions would require verifying the entire contents of RAM in each step. Instead, they maintain an address-ordered list called the memory trace, consisting

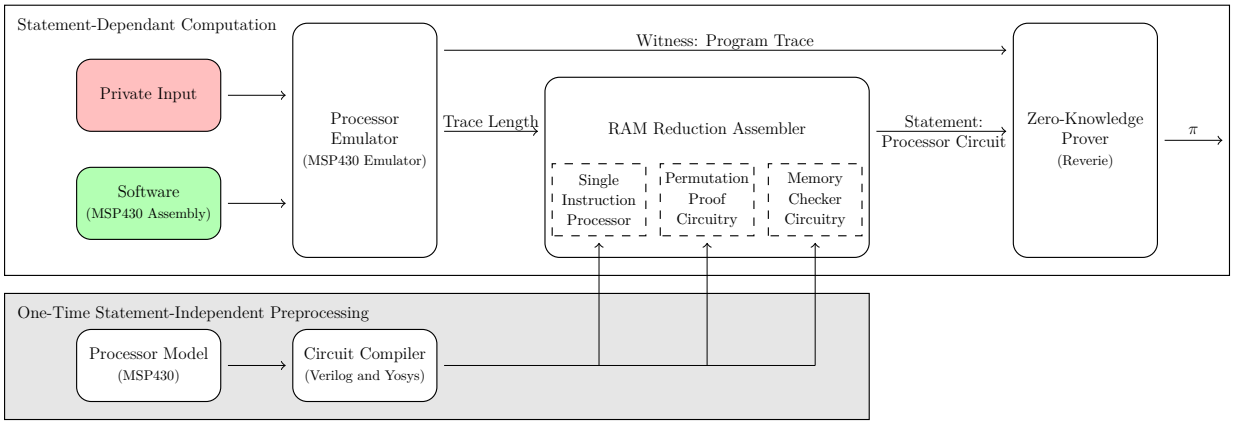

Figure 4-1. A high level overview of our toolchain for producing efficient zero-knowledge proofs for RAM programs on real processors. (1) The process starts with a one-time preprocessing phase which compiles the processor model into building blocks which are later assembled into a complete circuit. The circuit compiler (which we instantiate using Verilog and Yosys) generates the circuit for evaluating a single instruction, and the circuitry required to perform the permutation proof and check memory correctness. (2) When the prover wishes to create a proof, they feed the software, represented as assembly in the appropriate ISA, and any private program inputs into the processor emulator. The processor emulator runs the program to its conclusion and outputs the execution trace. (3) Based on the length of the trace, the RAM Reduction Assembler takes the preprocessed circuit components and creates the completed circuit. (4) The program trace, produced by the processor emulator, and the completed circuit, produced by the RAM reduction assembler, into any zero-knowledge prover to produce the final proof. We include the instantiations we use for our proofs of vulnerability in parenthesis.

of tuples of the form (step, operation, address, value), where step is a unique index in the execution trace, operation can either be read or write, and address is a location in memory [30]. The memory consistency circuit ensures that each read operation contains the same value as the most recent write operation to that address.

3. **Permutation Check.** The two proofs above ensure that the execution trace is consistent with the processor circuit and that the operations in the memory trace are valid. However, we must still demonstrate that these traces are consistent with one another; that is, the values provided to the execution trace consistency circuit correspond to the elements verified using the memory trace consistency circuit. To ensure this consistency, we employ

a permutation check that proves a one-to-one mapping between each read/write in the execution trace and some entry in the memory trace.

4.2.2 Formalizing Exploits

In order to produce cryptographically sound proofs of exploitability, we must have a formal NP language of which we can show a binary is a member. In our work, we are able to prove any exploit that is an arbitrary boolean predicate over the execution trace. Specifically, we can show that repeatedly applying the processor circuit to the processor state (for some public number of iterations) resulted in a processor state (or series of process states) that should have been impossible under honest execution. As such, we begin by designing circuit representations of real-world processors that are ZK friendly.

MSP430. In this work, we demonstrate the concrete feasibility of producing proofs of exploit for unaltered MSP430 binaries. MSP430 is a family of microprocessors commonly used in low-power environments. The version of the MSP430 ISA on which we focus has 27 instructions, including 12 double operand instructions (*e.g.* `MOV`, `ADD`, `AND`, `SUB`), 7 single operand instructions (*e.g.* `PUSH`, `CALL`), and 8 jump instructions (*e.g.* `JEQ`, `JNE`) [123, 147].

There are several significant obstacles to designing a circuit that implements the MSP430 instruction set architecture (ISA). MSP430 goes beyond a classic load/store architecture by incorporating 13 addressing modes. We augment our processor circuit using a set of memory hints in each step that provide the processor with the required information to complete the cycle's operation. The contents of the memory hints are interpreted based on the current instruction and are verified using the memory checker.

Given that the MSP430 is a small embedded processor it does not have an equivalent to system calls (syscalls) that are common in modern processors supported by full operating systems. Nevertheless, in some applications, including the Microcorruption CTFs, a library

can introduce the equivalent of certain system calls. We take a similar approach as in the creators of the Microcorruption CTFs to add syscalls to the MSP430 ISA. We will give more details about this modeling in Section 4.4.2.

Processor Predicates. There are many predicates over the execution trace that are highly relevant to demonstrating exploitability. For example, one simple predicate would be that the final program counter (PC) in the trace is some particular challenge value; if an attacker can set the PC arbitrarily, they likely can execute arbitrary code. We also consider more complex predicates, like showing that a syscall was executed during the trace that should have been impossible (*e.g.* turning on the device's microphone). Predicates about syscalls can also be used to show privilege escalation, by showing that the `GETEUID` syscall returned the value 0. Selecting the right predicate—or set of predicates—is an exploit-specific task that can be done by either the vulnerability researcher (once they have found an exploit) or the bug bounty program when setting their bounties.

To support such predicates, we add syscall support to our processor circuit, making it the first ZK processor to include syscalls. When the program encounters a syscall, the processor freezes the registers and enables the finite state machine. The processor executes the syscall for an arbitrary number of steps until some exit condition is met (*e.g.*, for the `GETS` syscall, until the processor reads a maximum number of characters or encounters a null byte). The processor then unfreezes and continues execution. This allows syscalls to be unrolled on the fly without requiring significant, special-purpose circuitry.

4.2.3 Producing Efficient ZK Proofs of Exploit

With a formalization of exploits in hand, we develop a toolchain to produce proofs of exploit. An overview of our toolchain can be found in Figure 4-1, including a processor emulator, the RAM reduction assembler, and the ZK prover. The remaining task is to develop the

necessary cryptographic optimizations such that the proofs of exploit that our toolchain produces are *efficient*.

Notation. We use $[b]$ for a share of a bit b, similarly we will use $[\![x]\!]$ for an arithmetic share of an element x in the respective arithmetic ring.

Reverie. Our second main technical contribution in this work is Reverie, the first publicly available,[9] general use implementation of the KKW MPC-in-the-head ZK protocol [132]. Reverie is written in Rust and incorporates many optimizations to make it more efficient, including bit slicing, memory efficient representations of the circuit, and proof streaming. The prover can compute the root of a Merkle tree with 256 leaves in just 8 seconds, significantly faster than prior NIZK implementations (see Table 4-II in Section 4.8).

Reverie also improves on KKW's initial protocol by including efficient ring switching based on edaBits [78]. To switch an element between rings, the prover generates shares of random elements in the two relevant rings during preprocessing. The prover then masks the value, reconstructs it in the clear, ring switches the public element, and removes the secret-shared mask.

For example, consider ring switching a value $v \in \mathbb{F}_{2^{32}}$ into an equivalent binary decomposition $(v_1, v_2, \ldots, v_{32}) \in \mathbb{F}_2^{32}$. The prover begins by generating random sharings of the values $r \in \mathbb{F}_{2^{32}}$ and $(r_1, r_2, \ldots, r_{32}) \in \mathbb{F}_2^{32}$ for the simulated players during the preprocessing, subject to the constraint $r = \sum_{i=1}^{32} r_i 2^i$. During online execution, the simulated parties publicly reconstruct the value $v + r$ and then decompose the public value into its binary representation $(v_1 + r_1), \ldots, (v_{32} + r_{32})$. The simulated parties then subtract their local shares of $r_1, \ldots, r_{32}$, resulting in a valid secret sharing of the values $(v_1, v_2, \ldots, v_{32}) \in \mathbb{F}_2^{32}$. This ring switching protocol is very efficient because generating verifiable, structured correlated randomness during preprocessing is very communication and computation efficient when using the KKW

[9] https://github.com/trailofbits/reverie

ZK protocol.

Efficient Permutation Proof. The RAM reduction outlined in Section 4.2.1 uses a routing network to implement the permutation proof between the execution trace and the memory trace. The routing network has asymptotic complexity $O(t \log(t))$, where t is the trace length, and large constants. A more efficient permutation proof, first explored by [45, 156], shows that two secret lists $\{A_i\}_{i\in[\ell]}$ and $\{B_i\}_{i\in[\ell]}$ are permutations by sampling a random challenge $x \xleftarrow{\$} \mathbb{Z}_q$ and testing if

$$\prod_{i=1}^{\ell}(A_i - x) \stackrel{?}{=} \prod_{i=1}^{\ell}(B_i - x).$$

To ensure that this test has negligible soundness error, it must be performed in a large field. However, our MSP430 processor operates over $\mathbb{F}_2$. Thus, the ring switching technique introduced above is vital to facilitating this permutation proof. Without access to an efficient ring switching technique, the test would have to be carried out in a small field with large soundness error, or the processor would need to operate over a large field, which would introduce high computational overhead. Concretely, the permutation proof costs just 380 AND gates and 2 multiplications for each element in the list.

Evaluation. We evaluate our toolchain by producing ZK proofs of exploitability for the Microcorruption Capture The Flag (CTF) exercises. Microcorruption CTF is a series of popular embedded device (MSP430) exploitation exercises that are freely available online. These exercises serve as a common entry point for individuals wishing to learn binary exploitation. Each challenge is named after a world city (see Table 4-I), and the exercises cover many common exploit techniques, such as heap and buffer overflows. Additionally, the processor implements important mitigation strategies, such as stack canaries, DEP, and ASLR. Thus, producing proofs of exploits for the Microcorruption CTF exercises demonstrates a wide variety of exploitation techniques, demonstrating the practicality of our approach.

The prover begins by initializing the processor emulator to a fresh state and loads the

public binary. The prover then emulates the binary when run on the private input, which produces an execution trace containing the processor state for each step and a memory trace containing the memory operations for each step. This emulation process stops once the desired processor state is reached (*e.g.*, the processor makes a restricted syscall). The prover then assembles the unrolled circuit from the pre-compiled library of components based on the length of the traces. The assembled circuit is provided as the statement to the ZK prover, and the traces are provided as a witness. Note that the only requirement we make of the ZK prover is that it is capable of performing ring switching.

Concretely, in one second, our implementation can produce a NIZK of correct processor execution of 216 MSP430 instructions requiring 119 KB of communication per instruction.

4.3 Related Work

Modeling RAM programs in ZK. TinyRAM [30] and BubbleRAM [115] are two custom ISAs developed to maximize performance with existing ZK schemes. They both use a load/store architecture with fewer than 30 instructions and ensure that decoding each instruction is inexpensive within a ZK prover. Among such works are vRAM [202] which constructs verifiable computation with a universal trusted setup for the TinyRAM ISA. The aims of our work differs from those in the verifiable computation literature in a number of important ways: (1) the proof size is linear (in particular the verifier complexity is linear). (2) we aim for concretely efficient prover complexity by using only symmetric key operations as opposed, *e.g.* to pairings in vRAM. (3) our techniques do not rely on a trusted setup (universal or otherwise) (4) we target real-world architecture. Despite proving a much more complicated architecture the proving speed (emulated CPU cycles/second) in this work (for MSP430) is ≈ 5 times greater than vRAM (for TinyRAM). While Ben Sasson et al. [32] later modified TinyRAM to have a von Neumann architecture, BubbleRAM remains a Harvard

architecture processor, which prevents it from reasoning about exploits that inject malicious code onto the stack or heap. As we discuss in the next subsection, the use of these custom ISAs limits the capabilities of a prover. For example, provers compile source code to the custom ISA, and source code is not available for many pieces of security critical software.

Proofs of Exploitability. In discussing prior work, we emphasize the difference between a vulnerability and an exploit. An exploit is maliciously crafted program input that produces unintended program behavior—or may even allow an attacker to affect the state of the computer beyond the program itself. A vulnerability, on the other hand, is a software weakness that could *potentially* be used in designing an exploit, for example an out-of-bounds memory write or a use-after-free bug. Vulnerabilities do not depend on architecture-specific constructs like the stack, heap, or mitigations such as ASLR, DEP, and pointer authentication codes (PAC). An exploit, however, is intrinsically linked to processor semantics. Therefore, it is not sufficient to reason only about source code when demonstrating the existence of an exploit.

Prior work on using ZK proofs for vulnerability disclosure [115, 116] has focused on manually annotating `C` code with assertions that a prover must demonstrate they can violate. This is accomplished by compiling the annotated code either directly to a circuit or to a custom ZK processor (*e.g.*, TinyRAM [30] or BubbleRAM/BubbleCache [115, 117]). While this approach is capable of proving many interesting vulnerabilities with extremely high efficiency, it has several drawbacks.

First, annotation of complex, real-world programs is time-consuming and error-prone. Source annotations cannot express many of the most commonly exploited classes of bugs [149, 183], and even the bugs theoretically detectable with annotations are difficult for programmers to find. Even if all these limitations could be overcome, this approach inherently requires access to source code, which is often not available.

Second, bugs in source do not always translate to exploits on a real processor. The example used by Heath and Kolesnikov [116] focuses on proving the existence of an out-of-bounds memory access—an operation many compilers will automatically prevent.

Finally, while bugs in source are common, successful exploits are rare. Fuzzing campaigns often find a large number of software bugs, but rarely convert these bugs into meaningful exploits. Research teams are unlikely to disclose a simple out-of-bounds read in ZK, as most such bugs do not lead to meaningful system compromise. Real bug bounties and vulnerability research consists of demonstrating how to leverage a vulnerability into an exploit (*e.g.*, privilege escalation, arbitrary code execution, or reading protected memory). Proving these capabilities cannot be done with source alone and are intrinsically linked to the compiled binary and architecture. For example, Heath et al. [117] claim that they can prove the existence of vulnerabilities in `sed` and `gzip` despite using a Harvard architecture. While it is true that they can prove vulnerabilities on such an architecture, they would not be able to demonstrate that the vulnerability is exploitable if the exploit involved executing malicious code off the stack, since the machine would not be able to fetch instructions stored in RAM.

4.4 Modeling Real-World Processors

In this section we discuss the technical details of modeling our target real-world processor, MSP430. First we discuss the necessary modeling to cover the basic MSP430 processor semantics and then discuss additions to the processor semantics that are helpful when modeling exploits.

4.4.1 Modeling MSP430 Processor Semantics

The MSP430 is a ubiquitous microcontroller [159], making it the perfect target for proofs of exploit. The MPS430 architecture contains 27 instructions, 13 addressing modes, and

16 registers with 16-bit words. We design a circuit which models the state transition associated with each of these instructions. We note, however, that MSP430 is not a load/store architecture—unlike the processor designed for ZK proofs—which increases the complexity of modeling memory.

Modeling Memory. Prior work on ZK processors use load/store architectures to cleanly separate memory accesses and logical operations. This allows the RAM reduction to treat non-memory operations as no-ops when performing the memory consistency check and permutation proof. However, many real-world processors, such as the MSP430, use a variety of addressing modes that prevent such a clean distinction from being made. For example, consider the instruction add `add @r5, 2(r6)`, which adds the contents of memory at the address `r5` to the contents of memory at address `r6+2` and stores the result at address `r6+2`. Not only does this instruction both access memory and use the processor's ALU, but it actually performs two reads and a write.

Our processor model handles such instructions by augmenting each instruction in the program trace to include three *memory hints*, which are used by the decoded instruction and verified with the memory checker. The hints are separated into two *read hints*, `src` and `dst`, and a single *write hint*. The hints each contain the relevant information for the implicit load/store operations encoded into some instructions (*e.g.* the address and value of memory to read/write). Specifically, the memory hints have the following structure:

- 1-bit On/Off indicator
- 16-bit Memory Address
- 19-bit Timestamp
- 1-bit Read/write indicator
- 1-bit Byte Mode indicator
- 1-bit Byte Mode Offset

- 16-bit Value

MSP430 supports byte operations on memory, so each memory hint indicates if it is in byte mode and the index of the byte on which the instruction is operating, if applicable.

Because MSP430 is a Von-Neumann architecture, fetching instructions constitutes a memory read. Each MSP430 instruction consists of a one-word opcode and up to two immediates, each of which requires its own read hint. Thus, the memory trace will contain six entries for each entry in the program trace. Checking these memory operations for consistency is straightforward, requiring only 194 AND gates per entry, so the memory checker requires 1,164 AND gates/cycle.

4.4.2 Interacting with the Program

In order to facilitate proofs of exploit, we choose to extend the base MSP430 ISA with cleanly modeled methods that allows the prover to interact with the program. Specifically, we are concerned with loading the program into the runtime, getting user inputs, and providing the program with entropy. While there are many potential ways to add these capabilities to the base ISA, we choose to add *system calls* that support these capabilities. This choice is inspired by the Microcorruption CTF challenges, which modeled system calls similarly in their version of the MSP430 ISA; by mirroring the choices made by the designers of the Microcorruption CTF challenges, we are able to "natively" support solutions for the challenges by directly mapping their syscalls onto our syscalls.

Modeling System Calls. System calls are an integral component of real-world software, providing the program access to key resources, including randomness, memory management, and user input. Many successful exploit strategies—and the techniques used to prevent such exploits—depend on the low-level details of syscall operations. For example, many processors implement memory protections such as ASLR by using system entropy to randomize the

address space layout. Prior work on ZK processors ignores syscalls and does not provide the processor with randomness.

We provide a general approach to handling syscalls initiated via software interrupts. Our approach does not rely on adding new instructions or storing information in registers or memory, as this would change the low-level processor behavior we aim to preserve. Instead, we augment each trace entry with a 48-bit value that encodes a finite state machine representing the current syscall status. This finite state machine is fed to a co-processor which is only triggered once a software interrupt is called. When a syscall is triggered, the following sequence of events occurs:

1. The processor freezes the register file, turns on the syscall flag, and loads the arguments and opcode into the syscall register.
2. Execution continues, but the processor operates on the syscall register instead of the register file.
3. Once the exit condition has been met, the syscall flag is turned off and normal execution resumes.

To better demonstrate this approach, we give the full details for our implementation of the `LOAD`, `GETS`, and `RAND` syscalls.

Getting user input. Before program execution begins, the prover uses the `LOAD` syscall to pre-load their input into a special memory bank that is read-only once program execution begins. Pre-loading input is important for reasoning about exploits that circumvent ASLR and stack canaries, since knowing or influencing the random values used in such mitigations would make significant parts of the exploit trivial.

When the processor starts execution, the PC is set to the first instruction in the input binary, but the syscall co-processor is turned on and set to `LOAD`. The first instruction of

the trace declares how many bytes of input will be loaded, and this value is placed in the syscall register. The processor will then continue to execute `LOAD` instructions, each time decrementing the syscall register until it reaches zero. At this point the syscall flag is turned off and program execution begins. At each step of the program, the processor checks that the prover cannot call `LOAD` after execution begins.

Once the input has been pre-loaded, the processor accesses it via the `GETS` syscall. `GETS` takes two arguments off the stack: the address to which the input will be written, and the maximum allowed length of the input in bytes. The syscall will exit once a null byte is encountered in the input or the maximum number of bytes is written.

When our MSP430 model encounters a call to `GETS`, the register file is frozen by turning on the syscall flag, and the target address and length are loaded into the syscall register. Subsequent clock cycles will use the memory hints in the trace to load user input byte-by-byte into memory, incrementing the address and decrementing the length variable in the syscall register. At each step, the input is checked for a null byte and the length variable is verified not to be zero. If either is zero, the syscall flag is turned off and normal processor execution resumes. Using this approach, the processor can emulate syscall operations — including the unrolling of variable length loops within the syscall logic — without altering the binary or memory state.

Processor Entropy. Our target version of MSP430 uses the `RAND` syscall to generate random values. In general, generation of high-entropy random values can be done using Fiat-Shamir. However, sometimes applications may use low-entropy random values, which cannot be generated using Fiat-Shamir while providing strong soundness guarantees, as the prover could grind to ensure that the randomness has the desired value. For example, 16-bit random values are used when calculating ASLR offsets and stack canaries. This limitation is inherent in the architecture itself — defenses that rely on low-entropy randomness will

always be vulnerable to computationally powerful adversaries.

To provide some meaningful soundness in the case where low entropy defenses are used, we design our processor to naturally extend to *interactive* proofs in which the verifier can supply randomness directly. First, the prover commits to all inputs that will be fed into the program by loading these values into a special memory bank prior to program execution, as specified in the previous section. Then, the verifier supplies a random seed value `seed` from which all randomness for the `RAND` syscall will be generated.

Specifically, the processor executes a special `GETRANDSEED` syscall to load the verifier supplied randomness `seed` into an auxiliary RAND register. The `GETRANDSEED` syscall can only be called once and only after the initial `LOAD` syscall has finished executing. The processor circuit will fail if the prover attempts to call `GETRANDSEED` again.

Once the prover has completed the `LOAD` phase, they execute the following steps in the clear:

1. Show the verifier that the PC is set to the program entry point, the syscall flag is turned on, and the syscall opcode is set to `GETRANDSEED`
2. Acquire the randomness `seed` from the verifier
3. Load the randomness into a public auxiliary RAND register
4. Turn the syscall flag off

Since the syscall flag is turned off once `GETRANDSEED` is finished, program execution must proceed normally from the binary entry point. During each processor cycle, the prover will evaluate $\mathsf{PRF}(\texttt{seed}, \texttt{step})$, where PRF is a pseudorandom function, and `step` is a counter indicating the number of processor cycles that have been executed. The first 16 bits of the output are then fed into the processor as the potential output of the `RAND` syscall. We emphasize that returning only 16 bits of randomness is inherent to the architecture. By

making the prover commit to all their inputs to the program before learning the `seed`, they must commit to an exploit strategy that can work for any value of randomness generated. We repeat that the meaning of a proof of exploit that circumvents low-entropy protections is nuanced; we refer the reader back to Section 4.1.1 for a discussion.

Users are provided with the option to disable processor randomness, since many applications do not need this feature. Additionally, note that running these proofs interactively is only necessary when there are low-entropy defense mechanisms that the prover must overcome, like ASLR.

4.5 Formalizing Exploits

Our aim is to provide vulnerability researchers with the necessary tools to precisely demonstrate exploits in real software without revealing underlying techniques. Therefore, we focus on creating a system that allows the prover to show that it knows some inputs such that running a public binary on those inputs on a real machine would result in a concrete exploit. This proof requires two components: demonstrating a given trace is valid, and demonstrating the trace triggered an exploit. The first component is handled using the previously discussed RAM reduction. We now discuss how exploits are shown during execution.

Many exploits can be detected by determining whether the attacker has arbitrary PC control. In this setting, the verifier challenges the prover to demonstrate they were able to produce a valid program trace concluding with the PC set to the challenge address. A similar protocol is used in the context of exploits that gain the ability to arbitrarily read or write memory.

A variety of exploits conclude with the execution of a syscall that should not have been accessible to the attacker. In an embedded systems context, this may manifest itself as turning on a microphone, turning off a security camera, or unlocking a door. This particular

notion of exploit is relatively straightforward to formalize in a ZK context. The prover simply needs to demonstrate that at some point during a valid program execution, a known malicious syscall was executed. This can be checked at the processor level by checking at each step whether the syscall flag is on and then examining the syscall opcode as specified in Section 4.4. All of these checks can be fed to a large OR statement at the conclusion of the proof to demonstrate whether a malicious syscall was executed. As we discuss in Section 4.8, this is how we formalize the Microcorruption exploits, all of which conclude in a call to the special `UNLOCK` interrupt.

Proving privilege escalation exploits — exploits which allow the prover to execute commands with root privileges on the machine — is more complicated. Generally, this would involve calling the `GETEUID` syscall and demonstrating the output is 0, using a similar approach as above. However, this would require modeling a runtime environment complex enough to have a notion of user privileges. We leave modeling a complex runtime environment as important future work.

Generally speaking, our approach facilitates proofs about exploits that can be represented as a Boolean expression on each processor state across the entire program execution. All of the above techniques are examples of this broader paradigm (*e.g.*, there exists a step of execution such that the instruction loaded by the processor is a malicious syscall). While this approach is sufficiently general to cover most common exploits, it has some fundamental limitations. In particular, our proof of exploit toolchain is incapable of reasoning about exploits that rely on microarchitectural bugs such as Spectre and Meltdown. Similarly, a Row Hammer type attack would also be out of scope since unintended physical properties of RAM cannot be simulated within a ZK context. Fortunately, most real-world exploits do not rely on microarchitectural bugs, so we do not view this as a major limitation.

Barriers to Easy Use. Although our toolchain allows provers to produce proofs for any

predicate over the processor states, the process of selecting the *right* predicate may be non-trivial—especially for vulnerability researchers without zero-knowledge expertise. Indeed, in our envisioned workflow (Section 4.1), we imagine that a sponsor might post a bug bounty to which vulnerability researchers could respond. One approach would be to have the bug bounty itself formalize the statement to prove in zero-knowledge; this approach is implicitly used in the Microcorruption CTF exercises, as the `UNLOCK` syscall is part of the challenge description. In more complex systems, there may be a huge number of potential processor states that would be considered problematic, such that enumerating all the processor states would be impractical. In such cases, the burden of selecting the correct statement—and demonstrating the statement's importance—would fall to the vulnerability researcher. Making these processes easier is important future work.

4.6 Circuit Compiler

The ZK proof system that we target accepts statements as either Boolean or arithmetic circuits. There are several tools created specifically for ZK statement generation such as Frigate [150], libsnark [2], and Circom [4], but they mostly target arithmetic circuits, which are not performant when handling real-world processor models. Frigate synthesizes code written in a subset of C. However, we found that it did not give us the granularity necessary to optimize circuits for MSP430.

Instead, we chose to write our processor circuit in Verilog, a widely used hardware description language (HDL) with mature open-source tooling. In particular, we used Yosys [194] to synthesize our core circuit components and Icarus Verilog for simulation and testing. Using Verilog allowed us to divide our RAM reduction into a collection of discrete Boolean modules, including the single-step MSP430 processor circuit and the memory consistency checker. We use Yosys to synthesize these components to a BLIF [1] file that encodes the hierarchical

arrangement of the components and their logic gates. Finally, we assemble these components into a flat, non-hierarchical encoding of the RAM reduction in the Bristol Fashion [16] using a circuit flattening library.

We designed our in-house flattener to take advantage of the fact that our circuit is highly structured, so we can aggressively cache flattened versions of the components and avoid repeating work. Using this approach of flattening components once and stapling them together, our flattening library can assemble the full RAM reduction for traces with 7k steps in 6 minutes using 20GB of RAM—an improvement of 99% in running time and 88% in RAM usage over using Yosys for flattening.

As described in Section 4.7.1, our permutation proof is prohibitively complex to be evaluated via a Boolean circuit, so we elected to specify it via an arithmetic circuit on $\mathbb{Z}_{2^{64}}$. Yosys and Verilog are only designed to operate on Boolean circuits, which presents a problem because using a HDL like Verilog is substantially easier than working at the level of individual gates when designing complex circuits.

We, however, use blackbox modules—a feature of Yosys designed to connect circuits to unknown hardware—to create models for the arithmetic logic gates in Verilog, which we then used to specify our permutation proof circuit. While we still had to ultimately specify the circuit at the gate level, working in Verilog broke up the circuit into hierarchical modules and assigned names to wires, greatly reducing debugging time.

After synthesizing the permutation circuit to a BLIF file, we pass it to our circuit compositor—a modified version of the circuit flattener that can accept a flattened Boolean circuit and a flattened arithmetic circuit and generate a specification for connecting the outputs of the Boolean circuit to the inputs of the arithmetic circuit using specialized BooleanToArithmetic gates. The 3-tuple of circuits consisting of the Boolean circuit, the connection circuit, and the arithmetic circuit is then passed to Reverie, which evaluates it as

the complete ZK statement.

4.7 Cryptographic Optimizations

Choice of Proof System. To instantiate our toolchain and optimize our proof system, we must first select a proof system. A number of considerations are relevant when selecting a suitable proof system for our particular application, most notably: (1) Prover/Verifier Complexity: Many widely deployed ZK proof systems are based on succinct non-interactive arguments of knowledge (SNARKs) (*e.g.* [112, 162]), which produce compact proof size at the expense of high prover runtime, complicated knowledge assumptions, and a trusted setup phase. While these tradeoffs are practical for space-limited applications, *e.g.* decentralized ledgers, the overhead of this approach would limit the complexity of RAM programs and exploits about which we could reason. Therefore we prioritize reducing concrete *prover time* rather than bandwidth. In order to somewhat offset the larger proof size we ensure that proofs can be verified in a streaming manner, meaning the verifier can process the proof as he is downloading it (without storing it). (2) Interactive vs Non-interactive: While interactive (private-coin) proofs systems can enable more efficient/flexible proofs, we opt for non-interactive proofs to enable a wider variety of use cases, as discussed in the introduction. This includes posting the proof for public verification and inclusion in long-term bug tracking logs. Non-interactivity can also be valuable when the prover may no longer be online or moving proofs across air-gaps (security researchers might be wary about allowing arbitrary people to open connections to the server holding the sensitive zero-day exploit).

These considerations lead us to believe that the KKW proof system [132] is well-suited for our application. Throughout this section we denote the party that executes the preprocessing in KKW as P_0 and use n to denote the number of parties in the MPC. We note that several improvements to the initial KKW system have been proposed recently, *e.g.* [22, 66], that

Input. Secret lists $A = \{A_i\}_{i\in[\ell]}$ and $B = \{B_i\}_{i\in[\ell]}$.
Public Input. A random challenge $x \in \mathbb{Z}_q$.
Circuit. Compute and compare

$$\prod_{i=1}^{\ell}(A_i - x) \stackrel{?}{=} \prod_{i=1}^{\ell}(B_i - x)$$

Output. 1 if the above check is valid, 0 otherwise.

Figure 4-2. Unknown Permutation Proof Circuit ($C_{\textbf{shuffle}}$). The circuit checks if two secret lists are permutations of each other.

could be integrated into our approach in future work.

4.7.1 Memory Permutation Proof (over $\mathbb{Z}_q$)

An unknown permutation proof is a zero-knowledge proof of knowledge that shows that the prover has two lists that are a permutation of each other, *i.e.* list $A = \{A_i\}_{i\in[\ell]}$ and $B = \{B_i\}_{i\in[\ell]}$ such that $\pi(A) = B$ for some permutation π. As the verifier does not know the lists nor the permutation, the proof is done with respect to a commitment to each list. We require an unknown permutation proof that will be efficient within MPC-in-the-head.

We implement the unknown permutation proof using the circuit defined in Figure 4-2 over a large ring, based on techniques first introduced by Bootle et al. [45], and first explored by Neff [156]. This stand-alone circuit receives two secret shared lists and a public randomly selected challenge x. Within the circuit, we view A and B as the set of roots of two polynomials, evaluate them at x and check equality, *i.e.* asserting $\prod_i(A_i - x) = \prod_i(B_i - x)$. Intuitively, perfect completeness follows on the commutativity of multiplication, while statistical soundness relies on the Swartz-Zippel lemma stating that two polynomials with distinct roots share an evaluation at a random point only with small probability[10].

For soundness, the random challenge x must be selected after the prover has committed

[10]When the size of the field dominates the degree of the polynomials. Note we do not need the soundness error to be negligible, but only to be dominated by $^{n-1}/_{n}$ from KKW.

to the secret shared lists, however the subsequent computation depends on the challenge. We accommodate this by introducing an additional round (5 rounds total)[11] in which the verifier samples x, after the prover has committed to the inputs/witness, but before committing to the views of every party.

Theorem 5 (Unknown Permutation Proof) *Given two lists A and B with ℓ elements in $\mathbb{Z}_q$ and an instance of the KKW protocol with n participants and m preprocessing repetitions. Using the above circuit and the challenge input inside a KKW protocol is an honest-verifier ZKPoK to prove knowledge of two lists A and B such that there exists a permutation π such that $\pi(A) = B$ with soundness/knowledge error* $\max\left\{\frac{1}{m}, \frac{1}{n} + \frac{\ell}{q-1} - \frac{\ell}{q-1}\frac{1}{n}\right\}$.

Proof.

Perfect completeness follows from the completeness of the KKW protocol as well as from the correctness of the circuit, which can be easily verified by inspection. Therefore, we will focus on proving honest-verifier zero-knowledge and soundness.

To prove that this protocol achieves perfect zero-knowledge, we can take the simulator $\mathcal{S}_{\mathbf{KKW}}$ that was used in KKW. The only change we have to make is that the simulator also chooses the challenge $x \in \mathbb{Z}_q$ uniformly at random. The same hybrid argument can be used as in the original proof. Given that the original simulator was indistinguishable from a real execution, we can conclude that this simulator is also indistinguishable from a real execution.

Similarly, to prove witness extraction, we can use the witness extractor from KKW. Note that after a full run of the protocol we have all messages as if we ran a normal KKW protocol for the circuit $C_{\mathbf{shuffle}}$, with a public input x, *i.e.* we don't have to extract x because it is part of the transcript. Hence, we can use the witness extractor as described in KKW to extract A and B, such that $\pi(A) = B$, for some permutation $\pi(\cdot)$.

[11] We reason that this additional round does not affect the knowledge error of the Fiat-Shamir transform, compared to the original 3 rounds. Note that, in general, soundness of the Fiat-Shamir transform decreases exponentially in the number of rounds.

The soundness error induced by the shuffle proof is $\frac{\ell}{q-1}$, which follow directly from the Schwartz-Zippel lemma. To see this, note that the x is selected at random and the number of points that are shared by the two polynomials is bounded by their degree ℓ. The soundness of the MPC-in-the-head protocol is $\max\left\{\frac{1}{m}, \frac{1}{n}\right\}$, as we are only considering the non-amplified version of KKW. To violate soundness, the prover must either succeed in the cheating during the preprocessing or the online phase. During the preprocessing, the probability is $\frac{1}{m}$. During the online phase, either the prover must cheat or produce an invalid shuffle proof. The probability of this happening is $\frac{1}{n} + \frac{\ell}{q-1} - \frac{\ell}{q-1}\frac{1}{n}$. Therefore, the overall soundness error is $\max\left\{\frac{1}{m}, \frac{1}{n} + \frac{\ell}{q-1} - \frac{\ell}{q-1}\frac{1}{n}\right\}$. □

Table 4-I. Benchmarks for proofs of exploits (at 128 bits of security) for a representative subset of the Microcorruption exercises. The selected exercises cover the most important exploit categories, including buffer overflow, code injection, and bypassing memory protection. These exercises are ordered by the difficulty of the exercise, as estimated by the Microcorruption creators.

Exercise Name	Processor Cycles	Prover (sec)	Verifier (sec)	Size (mb)	Exploit Type
New Orleans	2392	22	7	295	Password embedded in binary
Hanoi	6199	25	18	322	Buffer overflow
Cusco	5178	21	15	269	Buffer overflow
Montevideo	6676	28	20	358	Code injection via `strcpy` bug
Johannesburg	6311	26	19	332	Stack cookie bypass
Santa Cruz	12835	754	39	680	Code injection via `strcpy` bug
Addis Ababa	5360	23	17	296	Format string vulnerability
Novosibirsk	19833	89	63	1100	Format string vulnerability
Vladivostok	50823	454	152	6048	ASLR bypass

4.7.2 Ring Switching

One drawback of the permutation proof described in the previous section is that it relies on a large field/ring for soundness which leads to inefficient proofs of Boolean circuits. Unfortunately, real-world processors are most efficiently realized as Boolean circuits that pay a high cost for multiplication gates. The permutation proof can be implemented in a Boolean circuit by simulating a larger ring, however the $\log^2(q)$ overhead introduced by simulating the ring multiplication negates the improvements over the routing network used

in the work of Ben-Sasson et al. To avoid simulating arithmetic in a large ring, while still enabling application logic (CPU specification) to be proved using a Boolean circuit we rely on ring-switching techniques: enabling us to switch/pack a collection of Booleans into an element in a ring of sufficiently large order. This technique introduces an overhead of 3 AND-gates for every bit that needs translating. In our case, where we will switch to $\mathbb{Z}_{2^{64}}$, this means 192 AND-gates for every element in both lists. We base our ring-switching technique on the use of *edaBits* as introduced by Escudero et al. [78], which in turn was based on *daBits* by Rotaru and Wood [171]. We will apply the preprocessing optimization of KKW to achieve these results.

Preprocessing. Let ξ be the number of bits required to represent values of the larger field. During the preprocessing phase, we generate secret shares for the MPC players of the correlated random values r and $r_0, \ldots, r_{\xi-1}$, where r is a value in the larger ring and $r_0, \ldots, r_{\xi-1}$ are Boolean values, subject to the constraint $r = \sum_{i=0}^{\xi-1} r_i 2^i$, in the larger field. Thus, the players receive Boolean sharings $[r_0], \ldots, [r_{\xi-1}]$ and an arithmetic sharing $[\![r]\!]$. Note that none of the participants have any of the values $r, r_0, \ldots, r_{\xi-1}$ in the clear, they only possess a share of these values. Generation of this correlated randomness can be done using the same techniques used for Beaver triple generation in KKW: the dealer (P_0) generates and "sends" the shares to the respective players.

Online. The translation of $([x_0], \ldots, [x_\xi])$ into $[\![x]\!]$ with

$$x = \sum_{i=0}^{\xi} x_i 2^i$$

is done in the following way:

(1) In the Boolean circuit compute the $\mathbb{Z}_q$ addition of $r + x$ using a full adder, *i.e.* compute:

$$[(x+r)_0], \ldots, [(x+r)_{\xi-1}] = \\ ([x_0], \ldots, [x_{\xi-1}]) +_{\mathbb{Z}_q} ([r_0], \ldots, [r_\xi]))$$

(2) Reconstruct the masked bits $(x+r)_0, \ldots (x+r)_{\xi-1} \in \mathbb{Z}_2$, lift the bits to the ring $\mathbb{Z}_q$ and and convert the decomposition into $x + r \in \mathbb{Z}_q$ by publically computing the linear combination: $x' = x + r = \sum_{i=0}^{\xi-1} 2^i (x+r)_i \in \mathbb{Z}_q$

(3) In the arithmetic circuit subtract the randomness r from x' the input coming from the Boolean circuit, *i.e.* $x = x' - r$.

Note that only (1) has non-linear (over $\mathbb{Z}_2$) operations.

Theorem 6 (Ring Switching) *Given a Boolean circuit $C_{\mathbf{bool}}$ and an arithmetic circuit $C_{\mathbf{arith}}$ that need to be run consecutively, a definition of which output wires from $C_{\mathbf{bool}}$ are going into $C_{\mathbf{arith}}$, and an instance of the KKW protocol with n participants and m preprocessing repetitions. The above protocol is an honest-verifier ZKPoK with soundness/knowledge error* $\max\left\{\frac{1}{m}, \frac{1}{n}\right\}$.

Proof.

Completeness follows immediately from the completeness of the KKW protocol as well as the basic arithmetic used for transforming output from the boolean circuit to input to the arithmetic circuit.

To show perfect zero-knowledge we build the following simulator:

- Use the simulator $\mathcal{S}_{\mathbf{KKW}}$ on $C_{\mathbf{bool}}$,
- Actually do the transformation as it is done in the real protocol.
- Use the simulator $\mathcal{S}_{\mathbf{KKW}}$ on $C_{\mathbf{arith}}$,

Because $\mathcal{S}_{\mathbf{KKW}}$ generates a proof transcript that is indistinguishable form a real proof, and the second step is done exactly like it is done in the real protocol, we can conclude that this new simulator also produces a proof transcript that is indistinguishable from a real execution.

Witness extraction can be shown by first extracting the witness of the second circuit, and then using that witness for extracting the witness of the first circuit, which is also the witness for the complete circuit.

Soundness error is the maximum between both circuits of the soundness error as computed in KKW. To achieve better soundness, we can choose the number of executions according to the circuit with the worst soundness error. □

4.8 Implementation and Evaluation

Reverie. Our prover 'Reverie' [5] is an optimized implementation of the KKW [132] proof system in the Rust programming language. Reverie is generic and can be instantiated over any commutative ring. Reverie optimizes KKW for our particular application as follows:

- **Streaming.** Rather than compute the correlated randomness for the entire circuit before evaluation, Reverie interleaves the preprocessing with the online execution: in effect player P_0 is implemented as a coroutine. This avoids storing all the preprocessed material in memory.

- **Bit Slicing.** Every online player in KKW executes the same simple operation during the evaluation of addition and multiplication gates, hence bit-slicing 'across the players' allows executing every player in parallel, *e.g.* for the ring $\mathcal{R} = \mathbb{F}_2$ and $n = 64$ the values of two wires can be added using a single XOR of 64-bit integers.

- **Shadowing.** The model of execution in 'Reverie' is a straight-line RAM program: there is an array of cells and a program consists of a list of `Input`/`Add`/`Mul`/`Output` instructions reading/writing to cells. A circuit is a straight-line program in single assignment form (*i.e.* every cell is only written to once). Since the execution of a CPU is very local, this allows us to reclaim memory by overwriting cells, in practice reclaiming $> 95\%$ over naïvely

Table 4-II. Comparative Measurements for NIZKs computing 511 iterations of SHA256 (Merkle tree with 256 leaves). Measurements for prior work from [196] on an Amazon EC2 c5.9xlarge with 70GB of RAM and Intel Xeon platinum 8124m CPU with 18 3GHz virtual cores. Because these proof systems and implementations were unable to exploit parallelism, all benchmarks were run on a single thread. Reverie was benchmarked on a Digital Ocean virtual machine with 32 virtual cores and 256GB of memory. We note that our choice of protocol and our implementation is able to take advantage of the parallelism offer by the multiple cores, which is part of the reason Reverie is able to dramatically out-perform prior work.

Proof System	Gen (sec)	Prove (sec)	Ver (sec)	Size (KB)
Aurora [35]	-	3,199	15.2	174.3
Bulletproofs [50]	-	2,555	98	2
libSTARK [33]	-	2,022	0.044 s	395
Hyrax [188]	-	1,041	9.9	185
Ligero [10]	-	400	4	1,500
libSNARK [32]	1027	360	0.002	.013
Libra [196]	210	201	0.71	51
Reverie (This Work)	-	8	7.67	113,848

loading the circuit.

- **Parallel.** KKW requires many repetitions for soundness, these are executed in parallel.

All of these optimizations contribute to Reverie's exceptionally fast performance. Reverie is able to prove 511 iterations of SHA256 in 8 seconds. We compare this to the benchmarks reported in prior work from [196] in Table 4-II. We note that these are not strictly apples-to-apples comparisons as we were unable to control for the benchmarking environment for prior work. However, we note that Reverie does strikingly well. Libsnark requires 1,387 seconds, Bulletproofs requires 2,555 seconds, and Ligero requires 400 seconds (see Table 4-II). The proofs generated by Reverie are larger than the other three, but since it supports streaming all that is required is a network connection between prover and verifier with modest bandwidth.

Proofs of Exploitability: Microcorruption. We chose to use the Microcorruption CTF as a benchmark set for our ZK proof of vulnerability system. The CTF challenges involve hacking

Table 4-III. Breakdown of processor circuit components

Component	Non-linear Gates Per Instruction
Memory checker	1,164
Permutation proof	2,280
Processor	7,247
Decoder	568
ALU	549
Hint verifier	237
Operand fetching	2,176
Register file	2,880

a smart lock controlled by an MSP430 using common exploitation techniques such as buffer overflows, code injection, and bypassing memory protections. While the challenges contain a wide variety of bugs, ultimately they all conclude with a call to the `UNLOCK` system call. For example, the Addis Ababa challenge can be solved by using a format string vulnerability to overwrite a segment of memory that contains information about whether the correct password was entered or not, leading to a successful call to the `UNLOCK` system call.[12].

Therefore our ZK proofs of vulnerability check both that the witness trace is valid, and that at least one step of execution was a call to the `UNLOCK` system call. An advantage of this approach is that all ZK performance metrics are linear in the trace size, regardless of exploit technique.

Performance. We present benchmarks for a representative set of the Microcorruption exercises in Table 4-I. This set of benchmarks covers many of the most important exploit types, including buffer overflow, code injection, and bypassing memory protection. Each of these benchmarks was computed on a Digital Ocean virtual machine with 32 virtual cores and 256GB of memory. We found that our implementation produces a proof for 216 MSP430 instructions every second. Overall, each instruction requires 10,691 AND gates to execute. In

[12]For more details about the Microcorruption challenges, we point the reader to https://microcorruption.com or the reference manual [147]

Table 4-III, we give a breakdown of the gate count for each component of the RAM reduction, along with the major components of the processor. Although the resulting proofs produced are large and may take a non-trivial time to create, we note that these resources and time are insignificant compared to the effort it takes to develop the exploit and the time that the parties would spend negotiating disclosure.

Chapter 5

Abuse Resistant Law Enforcement Access Systems

This chapter is based on joint work with Matthew Green and Gabriel Kaptchuk, published as: Abuse Resistant Law Enforcement Access Systems. In: Canteaut, A., Standaert, FX. (eds) Advances in Cryptology – EUROCRYPT 2021. Lecture Notes in Computer Science, vol 12698. Springer, Cham. at Eurocrypt 2021 [110]

5.1 Introduction

Communication systems are increasingly deploying end-to-end (E2E) encryption as a means to secure physical device storage and communications traffic. E2E encryption systems differ from traditional link encryption mechanisms in that keys are not available to service providers, but are instead held by endpoints: typically end-user devices such as phones or computers. This approach ensures that plaintext data cannot be accessed by providers and manufacturers, or by attackers who may compromise their systems. Widely-deployed examples include messaging protocols [14, 181, 193], telephony [13], and device encryption [12, 104], with some systems deployed to billions of users.

The adoption of E2E encryption in commercial services has provoked a backlash from the law enforcement and national security communities around the world, based on concerns

that encryption will hamper agencies' investigative and surveillance capabilities [20, 83, 191]. The U.S. Federal Bureau of Investigation has mounted a high-profile policy campaign called "Going Dark" around these issues [79], and similar public outreach has been conducted by agencies in other countries [139]. These campaigns have resulted in legislative proposals in the United States [109, 167, 178] that seek to discourage the deployment of "warrant-proof" end-to-end encryption, as well as adopted legislation in Australia that requires providers to guarantee access to plaintext in commercial communication systems [187].

The various legislative proposals surrounding encryption have ignited a debate between technologists and policymakers. Technical experts have expressed concerns that these proposals, if implemented, will undermine the security offered by encryption systems [6, 154, 182], either by requiring unsafe changes or prohibiting the use of E2E encryption altogether. Law enforcement officials have, in turn, exhorted researchers to develop new solutions that resolve these challenges [20]. However, even the basic technical requirements of such a system remain unspecified, complicating both the technical and policy debates.

Existing Proposals for Law Enforcement Access. A number of recent and historical technical proposals have been advanced to resolve the technical questions raised by the encryption policy debate [24, 27, 68, 139, 174, 186, 195]. With some exceptions, the bulk of these proposals are variations on the classical *key escrow* [69] paradigm. In key escrow systems, one or more trusted authorities retain key material that can be used to decrypt targeted communications or devices.

Technologists and policymakers have criticized key escrow systems [6, 77, 155], citing concerns that, without additional protection measures, these systems could be abused to covertly conduct mass surveillance of citizens. Such abuses could result from a misbehaving operator or a compromised escrow keystore. Two recent policy working group reports [77, 155] provide evidence that, at least for the case of communications services, these concerns are

shared by members of the policy and national security communities.[13] Reflecting this consensus, recent high-profile technical proposals have limited their consideration only to the special case of *device encryption*, where physical countermeasures (*e.g.,* physical possession of a device, tamper-resistant hardware) can mitigate the risk of mass surveillance [27, 174]. Unfortunately, expanding the same countermeasures to messaging or telephony software seems challenging.

Abuse of Surveillance Mechanisms. Escrow-based access proposals suffer from three primary security limitations. First, key escrow systems require the storage of valuable key material that can decrypt most communications in the system. This material must be accessible to satisfy law enforcement request, but must simultaneously be defended against sophisticated, nation-state supported attackers. Second, in the event that key material is surreptitiously exfiltrated from a keystore, it may be difficult or impossible to detect its subsequent misuse. This is because escrow systems designed to allow lawful access to encrypted data typically store *decryption keys*, which can be misused without producing any detectable artifact.[14] Finally, these access systems require a human operator to interface between the digital escrow technology and the non-digital legal system, which raises the possibility of misbehavior by operators. These limitations must be addressed before any law enforcement access system can be realistically considered, as they are not merely theoretical: wiretapping and surveillance systems have proven to be targets for both nation-state attacks and operator abuse [49, 105, 153].

Overcoming these challenges is further complicated by law enforcement's desire to access data that was encrypted before an investigation is initiated. For example, several recent

[13]The Carnegie Institution report [77] concludes that "In the case of data in motion, for example, our group could identify no approach to increasing law enforcement access that seemed reasonably promising to adequately balance all of the various concerns".

[14]This contrasts with the theft of *e.g.,* digital certificates or signing keys, where abuse may produce artifacts such as fraudulent certificates [158] or malware artifacts that can be detected through Internet-wide surveillance.

investigations requested the unlocking of suspects' phones or message traffic in the wake of a crime or terrorist attack [142]. Satisfying these requests would require *retrospectively* changing the nature of the encryption scheme used: ciphertext must be strongly protected before an investigation begins, but they must become accessible to law enforcement after an investigation begins. Satisfying these contradictory requirements is extraordinarily challenging without storing key material that can access all past ciphertexts, since a ciphertext may be created *before* it is known if there will be a relevant investigation in the future.

Law enforcement access systems that do not fail open in the face of lost key material or malicious operators have been considered in the past, *e.g.,* [24, 37, 195]. Bellare and Rivest [24] proposed a mechanism to build *probabilistic* law enforcement access, in order to mitigate the risk of mass surveillance. Wright and Varia [195] proposed cryptographic puzzles as a means to increase the financial cost of abuse. While these might be theoretically elegant solutions, such techniques have practical limitations that may hinder their adoption: law enforcement is unlikely to tolerate arbitrary barriers or prohibitive costs that might impede legitimate investigations. Moreover, these proposals do little to enable detection of key theft or to prevent more subtle forms of misuse.

Towards Abuse Resistant Law Enforcement Access. In this work, we explore if it is technically possible to limit abuse while giving law enforcement the capabilities they are truly seeking: quickly decrypting relevant ciphertexts during legally compliant investigations. To do this, we provide a new cryptographic definition for an *abuse resistant law enforcement access system*. This definition focuses on abuse resistance by weaving *accountability* features throughout the access process. More concretely, our goal is to construct systems that realize the following three main features:

– **Global Surveillance Policies.** To prohibit abuse by authorized parties, access systems must enforce specific and *fine-grained* global policies that restrict the types of surveillance

that may take place. These policies could, for example, encompass limitations on the number of messages decrypted, the total number of targets, and the types of data accessed. They can be agreed upon in advance and made publicly available. This approach ensures that global limits can be developed that meet law enforcement needs, while also protecting the population against unlimited surveillance.

– **Detection of Abuse.** We require that any unauthorized use of escrow key material can be detected, either by the public or by authorized auditing parties. Achieving this goal ensures that even fully-adversarial use of escrow key material (*e.g.,* following an undetected key exfiltration) can be detected, and the system's security can be renewed through rekeying.

– **Operability.** At the same time, escrow systems must remain *operable*, in the sense that honest law enforcement parties should be able to access messages sent through a compliant system. We aim to guarantee this feature by ensuring that it is easy to verify that a message has been correctly prepared.

We stress that the notion of abuse-resistance is different from *impossible to abuse*. Under our definitions abuse may still happen, but the features described above will allow the abuse to be quickly identified and system security renewed. The most critical aspect of our work is that we seek to enforce these features *through the use of cryptography*, rather than relying on correct implementation of key escrow hardware or software, or proper behavior by authorities.

Prospective vs. retrospective surveillance. We will divide the access systems we discuss into two separate categories: *prospective* and *retrospective*. When using a *prospective* system, law enforcement may only access information encrypted sent or received from suspects *after* those suspects have been explicitly selected as targets for surveillance: this is analogous to "placing an alligator clip on a wire" in an analog wiretap. A *retrospective* access system, as described

above, allows investigators to decrypt past communications, even those from suspects who were not the target of surveillance when encryption took place. Retrospective access clearly offers legitimate investigators more capabilities, but may also present a greater risk of abuse. Indeed, achieving accountable access in the challenging setting of retrospective key escrow, where encryption may take place *prior* to any use of escrow decryption keys, is one of the most technically challenging aspects of this work.

Our contributions. More concretely, in this work we make the following contributions.

– **Formalizing security notions for abuse resistant law enforcement access systems.** We first provide a high-level discussion of the properties required to prevent abuse in a key escrow system, with a primary focus on the general data-in-motion setting: *i.e.,* we do not assume that targets possess trusted hardware. Based on this discussion, we formalize the roles and protocol interface of an Abuse-Resistant Law Enforcement Access System (ARLEAS): a message transmission framework that possesses law enforcement access capability with strong accountability guarantees. Finally, we provide an ideal functionality $\mathcal{F}_{\text{ARLEAS}}$ in Canetti's Universal Composability framework [53].

– **A prospective ARLEAS construction from lossy encryption or non-interactive secure computation.** We show how to realize ARLEAS that is restricted to the case of *prospective* access: this restricts the use of ARLEAS such that law enforcement must identify surveillance parameters before a target communication occurs. We first show that this can be constructed efficiently using lossy encryption and efficient simulation sound NIZKs, with the limitation that warrants must explicitly specify the *identities* of users being targeted for surveillance. We then show a generalization of this construction such that warrants can be *arbitrary predicates* to be evaluated over each message metadata; our construction of this generalization relies on non-interactive secure computation [126].

– **A retrospective ARLEAS construction from proof-of-publication ledgers and extractable witness encryption.** We show how to realize ARLEAS that admits *retrospective* access, while still maintaining the auditability and detectability requirements of the system. The novel idea behind our construction is to use secure *proof-of-publication ledgers* to condition cryptographic escrow operations. The cryptographic applications of proof-of-publication ledgers have recently been explored (under slightly different names) in several works [60, 106, 130, 175]. Such ledgers may be realized using recent advances in consensus networking, a subject that is part of a significant amount of research.

– **Evaluating the difficulty of retrospective systems.** Finally, we investigate the *minimal* assumptions for realizing retrospective access in an accountable law enforcement access system. As a concrete result, we present a lower-bound proof that any protocol realizing retrospective ARLEAS implies the existence of an extractable witness encryption scheme for some language $\mathcal{L}$ which is related to the ledger functionality and policy functions of the system. While this proof does not imply that all retrospective ARLEAS realizations require extractable witness encryption for general languages (*i.e.,* it may be possible to construct languages that have trivial EWE realizations), it serves as a guidepost to illustrate the barriers that researchers may face in seeking to build accountable law enforcement access systems.

5.1.1 Towards Abuse Resistance

In this work we consider the problem of constructing secure message transmission protocols with abuse resistant law enforcement access, which can be seen as an extension of secure message transmission as formalized in the UC framework by Canetti [53, 56]. Before discussing our technical contributions, we present the parties that interact with such a system and discuss several of the security properties we require.

The ARLEAS Setting. An ARLEAS system is comprised of three types of parties:

1. **Users:** Users employ a secure message transmission protocol to exchange messages with other users. From the perspective of these users, this system acts like a normal messaging service, with the additional ability to view public audit log information about the use of warrants on information sent through the system.

2. **Law Enforcement:** Law enforcement parties are responsible for initiating surveillance and accessing encrypted messages. This involves determining the scope of a surveillance request, obtaining a digital warrant, publishing transparency information, and then accessing the resulting data.

3. **Judiciary:** The final class of parties act as a check on law enforcement, determining whether a surveillance request meets the necessary legal requirements. In our system, any surveillance request must be approved by a judge before it is activated on the system. In our model we assume a single judge per system, though in practice this functionality can be distributed.

At setup time an ARLEAS system is parameterized by three functions, which we refer to as the global policy function, $p(\cdot)$, the warrant transparency function, $t(\cdot)$, and the warrant scope check function, $\theta(\cdot)$.[15] The purpose of these functions will become clear as we discuss operation and desired properties below. Finally, our proposals assume the existence of a verifiable, public broadcast channel, such as an append-only ledger. While this ledger may be operated by a centralized party, in practice we expect that such systems would be highly-distributed, *e.g.* using blockchain or consensus network techniques.

ARLEAS Operation. To initiate a surveillance request, law enforcement must first identify a specific class of messages (*e.g.* by metadata or sender/receiver); it then requests a surveillance

[15]We later introduce a fourth parameterizing function, but omit it here for the clarity of exposition.

warrant w from a judge. The judge reviews the request and authorizes or rejects the request. If the judge produces an authorized warrant, law enforcement must take a final step to *activate* the warrant in order to initiate surveillance. This activation process is a novel element of an abuse resistant access scheme, and it is what allows for the detection of misbehavior. To enforce this, we require that activation of a warrant w results in the publication of some information that is viewable by all parties in the system. This information consists of two parts: (1) a proof that the warrant is *permissible* in accordance with the global policy function, *i.e.* $p(w) = 1$, and (2) some transparency data associated with the warrant. The amount and nature of the transparency data to be published is determined by the warrant transparency function $t(w)$. Once the warrant has been activated, and the relevant information has been made public, law enforcement will be able to access any message that is within the scope of the warrant, as defined by the warrant scope check function $\theta(w)$.

ARLEAS Properties. For a law enforcement access system to be considered *abuse resistant*, it must satisfy the following intuitive properties:

- **Messages Secure without a Warrant.** A system must provide strong cryptographic security against attackers who are not authorized to receive messages, and law enforcement can only access the message if a judge has issued an applicable warrant.
- **Global Surveillance Policies.** Even in cases where all escrow authorities (*i.e.* law enforcement and judges) collude, surveillance requests must always obey a set of global limits defined by the *global policy function* which was chosen during system setup.
- **Abuse Detectability.** To enable detection of abuse or theft of key material, we require that whenever a warrant w is activated, law enforcement must publish $t(w)$ to all parties in the system, where $t(\cdot)$ is a *warrant transparency function* defined at system setup. This publication must occur even in cases where all escrow authorities collude.

– **Target Anonymity.** To preserve the integrity of investigations, users should learn no information about the contents of a warrant beyond what is revealed by the transparency function.

– **Escrow Verifiability.** Escrow authorities must be assured that the access system will decrypt messages within the scope of an activated, valid warrant. Because senders can always behave dishonestly (*e.g.* using alternative encryption mechanisms or encode messages using steganography), this guarantee cannot be enforced for all possible sender behavior. Instead, we mandate a weaker property that we call *escrow verifiability*: this ensures that recipients and/or service providers can filter messages that will decrypt differently for receivers and law enforcement. This ensures that *compliant messages* will be accessible by escrow authorities under appropriate circumstances.

In Section 5.3 we formalize this intuition and present a concrete security definition for an ARLEAS.

5.1.2 Technical Overview

We now present an overview of the key technical contributions of this work. We will consider this in the context of secure message transmission systems, which can be generalized to the setting of encrypted storage. Our overview will begin with intuition for building prospective ARLEAS, and then we will proceed to retrospective ARLEAS.

Accountability From Ledgers. For an ARLEAS the most difficult properties to satisfy are accountability and detectability. Existing solutions attempt to achieve this property by combining auditors and key escrow custodians; in order to retrieve key material that facilitates decryption, law enforcement must engage with an auditor. This solution, however, does not account for dishonest authorities, and is therefore vulnerable to covert key exfiltration and collusion. In our construction, we turn to public ledgers — a primitive that can be

realized using highly-decentralized and auditable systems — as a way to reduce these trust assumptions.

Ledgers have the property that any party can access their content. Importantly, they also have the property that any parties can be convinced that other parties have access to these contents. Thus, if auditing information is posted on a ledger, all parties are convinced that that information is truly public. We note that using ledgers in this way is fundamentally different from prior work addressing encrypted communications; our ledger is a public functionality that does not need to have any escrow secrets. As such, if it is corrupted, there is no private state that can be exploited by an attacker.

Warm up: Prospective ARLEAS. To build to our main construction, we first consider the simpler problem of constructing a *prospective* access system, one that is capable of accessing messages that are sent subsequent to a warrant being activated. For our practical construction, we make a further simplifying assumption that law enforcement will target *specific parties* for surveillance. We then extend this paradigm to allow law enforcement to target messages using arbitrary predicates.

A key aspect of this construction is that we consider a relatively flexible setting where parties have network access, and can receive periodic communications from escrow system operators prior to transmitting messages. We employ a public ledger for transmission of these messages, which provides an immutable record as well as a consistent view of these communications. The goal in our approach is to ensure that escrow updates embed information about the specifics of surveillance warrants that are active, while ensuring that even corrupted escrow parties cannot abuse the system.

Escrow lossy encryption. The basic intuition of our approach is to construct a "dual-trapdoor" public-key encryption system [48] that senders can use to encrypt messages to specific parties. This scheme is designed with two ciphertexts c_1 and c_2, such that c_1 can be

decrypted by the intended recipient using a normal secret key, while c_2 can be decrypted by law enforcement only if the recipient is under active surveillance. A feature of this scheme is that for all recipients not the target of surveillance, c_2 should contain no information about the plaintext.

Lossy encryption [165] is a natural tool to use to encrypt c_2, as an injective key can be used to encrypt c_2 when the recipient is under surveillance (preserving the plaintext) and a lossy key can be used to encrypt c_2 when the recipient is not under surveillance (destroying any information about the plaintext). A naive solution would have law enforcement generate either an injective key or a lossy key information for each user, as only law enforcement knows which users are under surveillance. Instead, we realize a more efficient construction using a tag-based variant of lossy encryption [25, 118, 119, 165] that we call *none-but-N* lossy-tag-based encryption, or LTE. In this scheme, key generation creates public parameters mpk with respect to the set of user identifiers (tags) $\mathcal{T}$ that are under active surveillance, along with a secret decryption key. The public parameters are proportional in size to the number of users under surveillance. When encrypting a message, a sender encrypts under both the recipient's public key and tag, along with mpk.

An LTE scheme must satisfy three main security properties. First, if the parameter generation process is run honestly with some set of user identifiers $\mathcal{T}$ and any (even biased) random coins r, then even an adversarial law enforcement should not be able to retrieve the message m if the receiver is not in $\mathcal{T}$. Second, to ensure that law enforcement access is possible, we require that adversarial encryptors cannot produce a ciphertext that appears correctly formatted but does not admit decryption. Finally, we require that mpk must at least computationally hide the set of users that are being targeted for surveillance, *i.e.* no efficient adversary with mpk should be able to recover any information about $\mathcal{T}$, beyond the size of its description. In Section 2.1.7 we discuss candidate constructions for LTE schemes based on the lossy encryption scheme in [25].

Building prospective ARLEAS for identities from LTE. Given an appropriate lossy tag-based encryption scheme, the remainder of the ARLEAS construction proceeds as follows. The global parameters of the scheme are created at setup: these include a public verification key for the judge presiding over the system, as well as a global transparency function t and policy function p agreed on by system participants.

When a law enforcement agency wishes to add a user to those being surveilled, it creates a new warrant w embedding the tag of that user and adds it to the set of active warrants $\mathcal{W}$. Law enforcement then runs the LTE parameter generation algorithm on the set of tags $\mathcal{T}$ embedded in the warrants in $\mathcal{W}$, obtaining mpk and the corresponding secret key. Law enforcement next contacts the judge to obtain a signature over w, and proceeds to generate a NIZK π that the following statements hold: (1) the prover posses a signature from the judge for each warrant in $\mathcal{W}$, (2) the parameter mpk was correctly generated with respect to the user-set $\mathcal{T}$ specified in the warrants in $\mathcal{W}$, (3) each warrant $w \in \mathcal{W}$ is permissible according to the global policy function p, and (4) the transparency information $\mathsf{info} \leftarrow \{t(w), \forall w \in \mathcal{W}\}$ was calculated honestly. Finally, it transmits $(mpk, \mathsf{info}, \pi)$ to a global ledger. Each participant in the system must ensure that this message was correctly published, and verify the proof π. If this proof verifies correctly, the participants will accept the new parameter mpk and use this value for all subsequent encryptions.

A critical security property of this system is that, even if law enforcement and judges collude (*e.g.,* if both parties become catastrophically compromised), users retain the assurance that issued warrant in violation of the global policy p cannot be used. Moreover, even in this event, the publication of a transparency record info ensures that every warrant activated in the system produces a detectable artifact that can be used to identify abuse.[16]

[16]The flexible nature of the transparency function t ensures that these records can contain both publicly-visible records (*e.g.,* a quantized description of the user set size, as well as private information that can be encrypted to auditors.

Prospective ARLEAS for Arbitrary Predicates. The solution presented above is inherently identity based, which restricts the types of warrants that a judge is able to issue. We now describe a version of this system that facilitates law enforcement access to messages if they possess a valid warrant embedding an *arbitrary predicate* such that this predicate, evaluated over the message metadata, is satisfied. Unlike the identity-based solution, generating key material for each possible situation cannot work; while the number of identities in the system may be bounded, there are an exponential number of predicate functions that law enforcement might want to embed into warrants. Instead, we rely on non-interactive secure computation (NISC) [126], a reusable, non-interactive version of two-party computation. NISC for an arbitrary function f allows a receiver to post an encryption of some secret x_1 such that all players can reveal $f(x_1, x_2)$ to the receiver with only one message, without revealing anything about x_2 beyond the output of the function. We leverage such a scheme to have senders reveal the message plaintext to law enforcement if and only if law enforcement's input to the NISC scheme contains a valid, pertinent warrant.

As before, law enforcement computes the transparency information for their warrant $\mathsf{info} \leftarrow t(w)$ along with the first message of the NISC scheme, embedding the warrant. Both of these are posted onto the ledger, along with a non-interactive zero-knowledge proof of correctness and compliance with the policy function. Whenever a sender sends a message m, they generate c_1 as normal and then generate c_2 which, using the NISC scheme, allows law enforcement to compute

$$f(w, (m, \mathsf{meta})) = m \wedge \theta(\mathsf{meta}, w),$$

where $\theta(\cdot, \cdot)$ evaluates if the warrant applies to this particular message (we will discuss $\theta(\cdot, \cdot)$ in more detail in Section 5.3). Notice that if $\theta(\mathsf{meta}, w) = 0$, then the output of the NISC evaluation is uncorrelated with the message. However, if $\theta(\mathsf{meta}, w) = 1$, meaning law enforcement has been issued a valid warrant, then the message is recovered. As before, users

are assured that all activated warrants are acceptable according to the policy function and detectable artifacts must be generated before any messages can be decrypted.

From Prospective to Retrospective. The major limitation of the ARLEAS construction above is that it is fundamentally restricted to the case of *prospective* access. Abuse resistance derives from the fact that "activation" of a warrant results in a distribution of fresh encryption parameters to users, and each of these updates renders only a subset of communications accessible to law enforcement. A second drawback of the prospective protocol is that it requires routine communication between escrow authorities and the users of the system, which may not be possible in all settings.

Updating these ideas to provide *retrospective* access provides a stark illustration of the challenges that occur in this setting. In the retrospective setting, the space of targeted communications is unrestricted at the time that encryption takes place. By the time this information is known, both sender and recipient may have completed their interaction and gone offline. Using some traditional, key-based solution to this problem implies the existence of powerful master decryption keys that can access *every* ciphertext sent by users of the system. Unfortunately, granting such power to any party (or set of parties) in our system is untenable; if this key material is compromised, any message can be decrypted without leaving a detectable artifact. The technical challenge in the retrospective setting is to find an alternative means to enable decryption, such that decryption is only possible on the conditions that (1) a relevant warrant has been issued that is compliant with the global policy function, (2) a detectable artifact has been made public. This mechanism must remain secure even when encryption occurs significantly before the warrant is contemplated.

Ledgers as a cryptographic primitive. A number of recent works [60, 61, 106, 130, 175] have proposed to use public ledgers as a means to *condition* cryptographic operations on published events. This paradigm was initially used by Choudhuri *et al.*[60] to achieve fairness

in MPC computations, while independently a variant was proposed by Goyal and Goyal [106] to construct one-time programs without the need for trusted hardware. Conceptually, these functionalities all allow decryption or program execution to occur only *after* certain information has been made public. This model assumes the existence of a secure global ledger $\mathcal{L}$ that is capable of producing a publicly-verifiable proof π that a value has been made public on the ledger. In principle, this ledger represents an alternative form of "trusted party" that participates in the system. However, unlike the trusted parties proposed in past escrow proposals [68], ledgers do not store any decryption secrets. Moreover, recent advances in consensus protocols, and particularly the deployment of proof-of-work and proof-of-stake cryptocurrency systems. *e.g.,* [44, 65, 93, 133], provide evidence that these ledgers can be operated safely at large scale.

Following the approach outlined by Choudhuri *et al.* [60], we make use of the ledger to *conditionally encrypt* messages such that decryption is only possible following the verifiable publication of the transparency function evaluated over a warrant on the global ledger. For some forms of general purpose ledgers that we seek to use in our system, this can be accomplished using extractable witness encryption (EWE) [46].[17] EWE schemes allow a sender to encrypt under a statement such that decryption is possible only if the decryptor knows of a witness ω that proves that the statement is in some language $\mathcal{L}$, where $\mathcal{L}$ parameterizes the scheme. While candidate schemes for witness encryption are known for specific languages (*e.g.* hash proof systems [63, 88]), EWE for general languages is unlikely to exist [89].

Building Retrospective ARLEAS from EWE. Our retrospective ARLEAS construction assumes the existence of a global ledger that produces verification proofs π_{publish} that a warrant has been published to a ledger. As mentioned before, we aim to condition law enforcement access on the issuance of a valid warrant and the publication of a detectable

[17] Using the weaker witness encryption primitive may be possible if the ledger produces *unique* proofs of publication.

artifact. In a sense, we want to use this published detectable artifact as a key to decrypt relevant ciphertexts. Thus, in this construction, a sender encrypts each message under a statement with a witness that shows evidence that these conditions have been met. This language reasons over (1) the warrant transparency function, (2) a function determining the relevance of the warrant to ciphertext, (3) the global policy function, (4) the judge's warrant approval mechanism, and (5) the ledger's proof of publication function.

On the Requirement of EWE. We justify the use of EWE in our construction by showing that the existence of a secure protocol realizing retrospective ARLEAS implies the existence of a secure EWE scheme for a related language that is deeply linked to the ARLEAS protocol. Intuitively, the witness for this language should serve as proof that the protocol has been correctly executed; law enforcement should be able to learn information about a message if and only if the accountability and detectability mechanisms have been run. For the concrete instantiation of retrospective ARLEAS, we give in Section 5.6, this would include getting a valid proof of publication from the ledger. If the protocol is realized with a different accountability mechanism, the witness encryption language will reason over that functionality. No matter the details of the accountability mechanism, we note that it should be difficult for law enforcement to locally simulate the mechanism. If it were computationally feasible, then law enforcement would be able to circumvent the accountability mechanism with ease.

5.1.3 Contextualizing ARLEAS In The Encryption Debate

This work is motivated by the active global debate on whether to mandate law enforcement access to encrypted communication systems via key escrow. Reduced to its essentials, this debate incorporates two broad sub-questions. First: can mandatory key escrow be deployed safely? Secondly, if the answer to the first question is positive: *should it be deployed?*

We do not seek to address the second question in this work. Many scholars in the policy

and technical communities have made significant efforts in tackling this issue [6, 20, 77, 155] and we do not believe that this work can make a substantial additional contribution. We stress, therefore, that our goal in this work is not to propose techniques for real-world deployment. Numerous practical questions and technical optimizations would need to be considered before ARLEAS could be deployed in practice.

Instead, the purpose of this work is to provide data to help policymakers address the first question. We have observed a growing consensus among stakeholders that key escrow systems should provide strong guarantees of information security as a precondition for deployment. Some stakeholders in the law-enforcement and national security communities grant that key escrow systems *should not be deployed* unless they can mitigate the risk of mass-surveillance via system abuse or compromise.[18] Unfortunately, there is no agreement on the definition of safety, and the technical community remains divided on whether traditional key escrow security measures (such as the use of secure hardware, threshold cryptography and policy safeguards) will be sufficient. We believe that the research community can help to provide answer these questions, and a failure to do so will increase the risk of unsound policy.

Our contribution in this paper is therefore to take a first step towards this goal. We attempt to formalize a notion of abuse-resilient key escrow, and to determine whether it can be realized using modern cryptographic techniques. Our work is focused on *feasibility*. With this perspective in mind, we believe that our work makes at least three necessary contributions to the current policy debate:

Surface the notion of cryptographic abuse-resistance. We raise the question of whether key escrow can be made *abuse resistant* using modern cryptographic technologies, and investigate what such a notion would imply. A key aspect of this discussion is the

[18] For evidence of this consensus, see *e.g.,* the 2018 National Academies of Sciences Report [155], which provides a framework for discussing such questions. See also a recent report by the Carnegie Endowment [77] which chooses to focus only on the problem of escrow for physical devices rather than data in motion, providing the following explanation: "it is much harder to identify a potential solution to the problems identified regarding data in motion in a way that achieves a good balance" (p. 10).

question of detectability: by making abuse and key exfiltration publicly detectable, we can test law enforcement's belief that backdoor secrets can remain secure, and renew security by efficiently re-keying the system.

Separate the problems of prospective and retrospective surveillance. By emphasizing the technical distinctions between prospective and retrospective surveillance, we are able to highlight the design space in which it is realistic to discuss law enforcement access mechanisms. In particular, our technical results in this work illustrate the cryptographic implausibility of retrospective ARLEAS: this may indicate that retrospective surveillance systems are innately susceptible to abuse.

Shift focus to public policy. In defining and providing constructions for prospective and retrospective ARLEAS, we formalize the notion of a global policy function and a transparency function (see Section 5.3). By making these functions explicit, we hope to highlight the difficult policy issues that must be solved before deploying any access mechanism. As noted by Feigenbaum and Weitzner [81], there are limits what cryptography can contribute to this debate; legal and policy experts must do a better job reducing the gray area between rules and principles so that technical requirements can be better specified.

Finally, we note that the existence of a cryptographic construction for ARLEAS may not be sufficient to satisfy lawn enforcement needs. The mathematics for cryptographically strong encryption systems is already public and widespread, and determined criminals may simply implement their own secure messaging systems [76]. Alternatively, they may use steganography or pre-encrypt their messages with strong encryption to prevent "real" plaintext from being recovered by law enforcement while still allowing contacts to read messages [120]. These practical problems will likely limit the power of any ARLEAS and must be considered carefully by policy makers before pushing for deployment.

5.2 Related work

The past decade has seen the start of academic work investigating the notion of accountability for government searches. Bates *et al.*[21] focus specifically on CALEA wiretaps and ensuring that auditors can ensure law enforcement compliance with court orders. In the direct aftermath of the Snowden leaks, Segal *et al.* [177] explored how governments could accountably execute searches without resorting to dragnet surveillance. Liu *et al.* [143] focus on making the number of searches more transparent, to allow democratic processes to balance social welfare and individual privacy. Kroll *et al.* [136, 137] investigate different accountability mechanisms for key escrow systems, but stop short of addressing end-to-end encryption systems and the collusion problems we address in this work. Kamara [129] investigates cryptographic means of restructuing the NSA's metadata program. Backes considered anonymous accountable access control [17], while Goldwasser and Park [102] investigate similar notions with the limitation that policies themselves may be secret, due to national security concerns. Frankle *et al.* [84] make use of ledgers to get accountability for search procedures, but their solution cannot be extended to the end-to-end encryption setting. Wright and Varia [195] give a construction that uses cryptographic puzzles to impose a high cost for law enforcement to decrypt messages. Servan-Schreiber and Wheeler [179] give a construction for accountability that randomly selects custodians that law enforcement must access to decrypt a message. Panwar *et al.* [161] attempt to integrate the accountability systems closely with ledgers, but do not use the ledgers to address access to encryption systems. Finally, Scafuro [175] proposes a closely related concept of "break-glass encryption" and give a construction that relies on trusted hardware.

5.3 Definitions

5.4 Lossy Tag Encryption

We will require a specific generalization of lossy encryption we call lossy-tag encryption (LTE). Intuitively, this is an encryption scheme with a single public key in which encryption takes as input a "tag" in addition to the public key and plaintext. Encrypting under a tag from a specific subset will produce in an injective ciphertext, while the remaining tags will produce a lossy ciphertext. This notion is closely related to numerous previous works, including lossy encryption [25], lossy trapdoor functions [165], identity-based lossy trapdoor functions [26] all-but-one functions [165], and all-but-n functions [118]. We define lossy-tag encryption formally as follows:

Definition 10 A lossy-tag encryption (LTE) scheme with respect to a tag space $\mathbb{T}$ consists of a tuple of algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ defined as follows:

- $\mathsf{KeyGen}(1^\lambda, \mathcal{T})$ takes in a set of tags $\mathcal{T} \subset \mathbb{T}$ of polynomial size in λ and outputs a public key mpk and a secret key msk.
- $\mathsf{Enc}(mpk, tag, m)$ encrypts the message m under the public key mpk and $tag \in \mathbb{T}$ to produce ciphertext c.
- $\mathsf{Dec}(\mathsf{msk}, tag, c)$ takes in the secret key msk, $tag \in \mathbb{T}$ and a ciphertext c and either outputs a message m or $\perp$.

We require that the above algorithms satisfy the follow properties

- **Correctness on injective tags**:

$$\Pr\left[m = \mathsf{Dec}(\mathsf{msk}, tag, \mathsf{Enc}(mpk, tag, m)) \;\middle|\; \begin{array}{c} tag \in \mathcal{T} \\ (mpk, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{T}) \end{array}\right] = 1$$

– **Lossiness on lossy tags**: for all messages m_0, m_1 and all sets $\mathcal{T}$, if $tag \notin \mathcal{T}$ and $(mpk, \mathsf{msk}) \in \mathsf{KeyGen}(1^\lambda, \mathcal{T})$, then

$$\mathsf{Enc}(mpk, tag, m_0) \overset{s}{\approx} \mathsf{Enc}(mpk, tag, m_1)$$

– **Indistinguishability of tag sets**: for all sets $\mathcal{T}_0 \neq \mathcal{T}_1$ such that $|\mathcal{T}_0| = |\mathcal{T}_1|$,

$$\mathsf{KeyGen}(1^\lambda, \mathcal{T}_0) \overset{c}{\approx} \mathsf{KeyGen}(1^\lambda, \mathcal{T}_1)$$

A stronger version of this definition could remove the requirement that $|\mathcal{T}_0| = |\mathcal{T}_1|$ for indistinguishability of tag sets. For our constructions, we do not concern ourselves with this leakage.

Realizing lossy-tag encryption: When the size of $\mathbb{T}$ is polynomial in the security parameter, it is trivial to realize lossy-tag encryption from standard lossy encryption [25] simply by generating one lossy keypair to represent each "tag". However, even for small sets $\mathbb{T}$ this may produce unreasonably large public keys. In this work, we present a direct instantiation of a lossy-tag encryption scheme based on DDH, such that the public parameters mpk that are linear in $|\mathcal{T}|$.

Let $\mathbb{G}$ be a cyclic group of order p. Define $\mathsf{pk} = (g, h, \tilde{g}, \tilde{h}) \in \mathbb{G}^4$, and

$$\mathsf{DoubleEncrypt}(\mathsf{pk}, m; r_1, r_2)$$

to output $(g^{r_1} h^{r_2}, \tilde{g}^{r_1} \tilde{h}^{r_2} \cdot m)$. As noted in [25], if pk is a DDH tuple then this encryption is injective, but if pk is a random tuple then the encryption is statistically lossy. We now present a construction Π_{LTE} for lossy-tag encryption as follows:

- $\mathsf{KeyGen}(params, \mathcal{T}) \to (mpk, \mathsf{msk})$. Sample $params = p, \mathcal{G}, g, h, \hat{g}, \hat{h}$ where $\mathcal{G}$ has order p and $g, h, \hat{g}, \hat{h}$ are generators of $\mathcal{G}$. Sample a random polynomial $A(x)$ in $\mathbb{Z}_p$ of degree $k = |\mathcal{T}|$ such that for each $s \in \mathcal{T}, A(x) = 0$. Then compute $B(x) = d_2 A(x)$ for some

constant $d_2 \neq \frac{d_0}{d_1}$. Let α_i be the i^{th} coefficient of $A(x)$ and β_i be the i^{th} coefficient of $B(x)$. Use rejection sampling to sample random $\eta_0 \dots \eta_k$ such then when η_i is interpreted as the i^{th} coefficient of a polynomial $E(x)$, $E(tag) \neq 0$ for all $tag \in \mathcal{T}$. Compute $mpk = ((g^{\eta_k}\hat{g}^{\alpha_k}, \dots, g^{\eta_0}\hat{g}^{\alpha_0}), (h^{\eta_k}\hat{h}^{\beta_k}, \dots, h^{\eta_0}\hat{h}^{\beta_0}))$. Compute $\mathsf{msk} = (\eta_0, \eta_1, \dots, \eta_k)$ and output mpk, msk.

- $\mathsf{Enc}(params, mpk, tag, m) \to c$.
  - Parse $((g^{\eta_k}\hat{g}^{\alpha_k}, \dots, g^{\eta_0}\hat{g}^{\alpha_0}), (h^{\eta_k}\hat{h}^{\beta_k}, \dots, h^{\eta_0}\hat{h}^{\beta_0})) \leftarrow mpk$
  - Compute the user public key
$$\mathsf{pk} = (g, h, \prod_{i=0}^{k}(g^{\eta_i}\hat{g}^{\alpha_i})^{tag^i}, \prod_{i=0}^{k}(h^{\eta_i}\hat{h}^{\beta_i})^{tag^i})$$
  - Sample $r_1, r_2 \leftarrow \mathbb{Z}_p$
  - Compute and return
$$c = (g^{r_1}h^{r_2}, \left(\prod_{i=0}^{k}(g^{\eta_i}\hat{g}^{\alpha_i})^{tag^i}\right)^{r_1} \cdot \left(\prod_{i=0}^{k}(h^{\eta_i}\hat{h}^{\beta_i})^{tag^i}\right)^{r_2} \cdot m)$$
- $\mathsf{Dec}(params, \mathsf{msk}, tag, c) \to m$
  - Parse $(\eta_0, \eta_1, \dots, \eta_k) \leftarrow \mathsf{msk}$
  - Parse $(c_1, c_2) \leftarrow c$
  - Compute $y = \sum_{i=0}^{k} \eta_0 (tag)^k$
  - Compute and return $m = \frac{c_2}{c_1^y}$.

Proof. We now prove that Π_{LTE} above realizes the the functionality of lossy-tag encryption. To do so, we recall the construction of a lossy encryption scheme in Section 4.1 of [25]. As mentioned above, they observe that ElGamal double encryption is injective when the public key has the structure (g, h, g^x, h^x), but is lossy when the public key has the structure (g, h, g^x, h^y), for $x \neq y$.

Correctness on injective tags. For injective tags, $A(x) = B(x) = 0$. Recall that the public key used during encryption is computed as $(g, h, \prod_{i=0}^{k}(g^{\eta_i}\hat{g}^{\alpha_i})^{tag^i}, \prod_{i=0}^{k}(h^{\eta_i}\hat{h}^{\beta_i})^{tag^i})$. Written another way, this is $(g, h, g^{E(tag)}\hat{g}^{A(tag)}, h^{E(tag)}\hat{h}^{B(tag)})$. Because $A(x) = B(x) = 0$, the public key is $(g, h, g^{E(tag)}, h^{E(tag)})$, where $E(tag)$ is non-zero. Note that this form is the same form as an injective key from [25], so the resulting ciphertext is injective with corresponding private key $E(tag)$.

Lossiness on lossy tags. For lossy tags, $A(x) \neq B(x) \neq 0$. This can be observed because $B(x) = kA(x)$, and there are at most $|\mathcal{T}|$ zeros of a degree $|\mathcal{T}|$ polynomial, and all all these zeros were set to be the injective tags. The public key used for encryption is, as before, $(g, h, g^{E(tag)}\hat{g}^{A(tag)}, h^{E(tag)}\hat{h}^{B(tag)})$. Without loss of generality, the public key can then be written as

$$(g, h, g^{E(tag)+d_0A(tag)}, h^{E(tag)+d_1B(tag)}).$$

Note that because $B(\cdot)$ was sampled such that $d_2A(x) = B(x)$, and $\frac{d_0}{d_1} \neq d_2$, then $E(tag) + d_0A(tag) \neq E(tag) + d_1B(tag)$. Thus this public key is structured exactly like the lossy key from [25], so the resulting ciphertext is lossy.

Indistinguishability of tag sets. Due to the key indistinguishability of [25], it is clear that a lossy key and an injective key, when computed during encryption, are statistically indistinguishable. All that remains to argue is that the public parameters leak no information about the tag set besides its size (note that the size of mpk trivially leaks $|\mathcal{T}|$). Notice that it is sufficient to show that this property holds when two sets differ in only a single tag, as a straightforward hybrid argument in which a single tag is swapped in each hybrid can generalize the result. Next, notice that each element in mpk is formed like a Pendersen commitment [164] to α_i or β_i. Thus, it is clear to see that if there exists an adversary that can distinguish between sets, it can be used to break the hiding property of Pendersen commitments.

5.4.1 Defining ARLEAS

We now formally define the notion of an Abuse-Resistant Law Enforcement Access System (ARLEAS). An ARLEAS is a form of message transmission scheme that supports accountable access by law enforcement officials. To emphasize the core functionality, we base our security definitions on the UC Secure Message Transmission ($\mathcal{F}_{\mathsf{SMT}}$) notion originally introduced by Canetti [53]. Indeed, our systems can be viewed as an extension of a multi-message SMT functionality [56], with added escrow capability.

Parties and system parameters. An ARLEAS is an interactive message transmission protocol run between several parties and network components:

- User P_i: Users are the primary consumer of the end-to-end encrypted service or application. These parties, which may be numerous, interact with the system by sending messages to other users.
- Judge P_J: The judge is responsible for determining the validity of a search and issuing search warrants to law enforcement. The judge interacts with the system by receiving warrant requests and choosing to deny or approve the request.
- LawEnforcement P_{LE}: Law enforcement is responsible for conducting searches pursuant to valid warrants authorized by a judge. Law enforcement interacts with the system by requesting warrants from the judge and collecting the plaintext messages relevant to their investigations.

A concrete ARLEAS system also assumes the existence of a communication network that parties can use to transmit encrypted messages to other users. To support law enforcement access, it must be possible for law enforcement to "tap" this network and receive encrypted communications between targeted users. For the purposes of this exposition, we will assume that law enforcement agents have access to any communications transmitted over the network

(*i.e.,* the network operates as a transparent channel.) In practice, a service provider would handle the transmissions of ciphertexts. This service provider would also be responsible for storing ciphertext and metadata, and providing this information to law enforcement. Our simplified model captures the worst case network security assumption, where the service provider cooperates with all law enforcement requests. Service providers would also be responsible for checking that messages sent by users are compliant with the law enforcement access protocol. We move this responsibility to the receiver for simplicity. We discuss the role of service providers more in Section 5.8.

An ARLEAS system is additionally parameterized by four functions, which are selected during a trusted setup phase:

- $t(w)$: the deterministic *transparency function* takes as input a warrant w and outputs specific information about the warrant that can be published to the general public.
- $p(w)$: the deterministic *global policy* function takes as input a warrant w and outputs 1 if this warrant is allowed by the system.
- $\theta(w, \mathsf{meta})$: the deterministic *warrant scope check* takes as input a warrant w and per-message metadata meta. It outputs 1 if meta is in scope of w for surveillance.
- $v(\mathsf{meta}, aux)$: The deterministic *metadata verification functionality* takes as input metadata associated with some message meta and some auxiliary information aux and determines if the metadata is correct. This auxiliary information could contain the ciphertext, global timing information, or some authenticated side channel information.

We discuss concrete instantiations of these functions in Section 5.8.

ARLEAS scheme. An ARLEAS scheme comprises a set of six possibly interactive protocols. We provide a complete API specification for these protocols in later sections:

- Setup. On input a security parameter, this trusted setup routine generates all necessary parameters and keys needed to run the full system.
- SendMessage. On input a message m, metadata meta, and a recipient identity, this protocol sends an encrypted message from one party to another.
- RequestWarrant. On input a description of the warrant request, this procedure allows law enforcement to produce a valid warrant.
- ActivateWarrant. Given a warrant w, this protocol allows law enforcement and a judge to confirm and activate a warrant.
- VerifyWarrantStatus. Given a warrant w, this protocol is used to verify that a warrant is valid and active.
- AccessMessage. in the retrospective case, this protocol is used by law enforcement to open a message.

UC ideal functionality. To define the properties of an ARLEAS system, we present a formal UC ideal functionality $\mathcal{F}_{\text{ARLEAS}}$ in Figure 5-1. Recalling that ARLEAS can be instantiated in one of two modes, supporting only prospective or retrospective surveillance, we present a single definition that supports a parameter, $\mathsf{mode} \in \{\mathsf{pro}, \mathsf{ret}\}$.

Ideal World. For any ideal-world adversary $\mathcal{S}$ with auxiliary input $z \in \{0,1\}^*$, input vector x, and security parameter λ, we denote the output of the ideal world experiment by $\mathbf{Ideal}_{\mathcal{S},\mathcal{F}^{v,t,p,\theta,\mathsf{mode}}_{\text{ARLEAS}}}(1^\lambda, x, z)$.

Real World. The real world protocol starts with the adversary $\mathcal{A}$ selecting a subset of the parties to compromise $\mathcal{P}^{\mathcal{A}} \subset \mathcal{P}$, where $\mathcal{P}^{\mathcal{A}} \subset \{\{P_i\}, \{P_{\text{LE}}\}, \{P_{\text{LE}}, P_J\}\}$, where we denote sender with P_i and receiver with P_j. We limit the subsets of parties that can be compromised to these cases, because any other combination is trivial to simulate or can be deducted from

Functionality $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{mode}}$

The ideal functionality is parameterized by $\mathsf{mode} \in \{\mathsf{pro}, \mathsf{ret}\}$, a metadata verification function $v : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$, the transparency function $t(\cdot)$, the global policy function $p(\cdot)$, and the warrant scope check functionality $\theta(\cdot,\cdot)$. The three latter functions are as defined above. We denote the session identifier as sid to separate different runs of the same protocol. We have several parties:

- $P_1, \ldots, P_n$: participants in the system
- P_J: the generator of a warrant
- P_{LE}: Law enforcement that can read the message given a valid warrant

Send Message: Upon receiving a message $(\mathsf{SendMessage}, \mathsf{sid}, P_j, m, \mathsf{meta}, \mathsf{valid})$ where $\mathsf{valid} \in \{0,1\}$ from party P_i, it sends $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta})$ to the adversary. If (sid, c) is received from the adversary,

- If $\mathsf{valid} = 0$ or $v(\mathsf{meta}, aux) = 0$, send $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m)$ to P_i and send $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, 0)$ to P_{LE}.
- If $\mathsf{valid} = 1$, $v(\mathsf{meta}, aux) = 1$, and there is no entry w in the active warrant table W_{active} send $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m)$ to P_i and P_j, and send $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c)$ to P_{LE}.
- If $\mathsf{valid} = 1$, $v(\mathsf{meta}, aux) = 1$, and there is an entry w in the active warrant table W_{active} send $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m)$ to P_i, P_j, and P_{LE}.

Finally, store $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m)$ in the message table M.

Request Warrant: Upon receiving a message $(\mathsf{RequestWarrant}, \mathsf{sid}, w)$ from P_{LE}, the ideal functionality first checks if $p(w) = 1$, responding with $\perp$ and aborting if not. Otherwise, the ideal functionality sends $(\mathsf{ApproveWarrant}, w)$ to P_J. If P_J responds with $(\mathsf{Disapprove})$, the trusted functionality sends $\perp$ to P_{LE}. If P_J responds with $(\mathsf{Approve})$, the trusted functionality sends $(\mathsf{Approve})$ to P_{LE}, and stores the entry w in the issued warrant table W_{issued}.

Activate Warrant: Upon receiving a message $(\mathsf{ActivateWarrant}, \mathsf{sid}, w)$ from P_{LE}, the ideal functionality checks to see if $w \in W_{\text{issued}}$, responding with $\perp$ and aborting if not. If $w \in W_{\text{issued}}$, the trusted functionality adds the entry w to the active warrant table W_{active}, computes $t(w)$, and sends $(\mathsf{NotifyWarrant}, t(w))$ to all parties and the adversary.

Verify Warrant Status: Upon receiving message $(\mathsf{VerifyWarrantStatus}, \mathsf{sid}, c, \mathsf{meta}, w)$ from P_{LE}, if $\mathsf{mode} = \mathsf{pro}$, the ideal functionality responds with $\perp$ and aborts. Otherwise, if $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m) \in M$ and $w \in W_{\text{active}}$ such that $\theta(w, \mathsf{meta}) = 1$, the ideal functionality returns 1. Finally, if $\theta(w, \mathsf{meta}) = 0$ or $w \notin W_{\text{active}}$, it returns 0.

Access message: Upon receiving message $(\mathsf{AccessData}, \mathsf{sid}, c, \mathsf{meta}, w)$ from P_{LE}, if $\mathsf{mode} = \mathsf{pro}$, the ideal functionality responds with $\perp$ and aborts. Otherwise, if $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m) \in M$ and $w \in W_{\text{active}}$ such that $\theta(w, \mathsf{meta}) = 1$, the ideal functionality returns m. Finally, if $\theta(w, \mathsf{meta}) = 0$ or $w \notin W_{\text{active}}$, it returns 0.

Figure 5-1. Ideal functionality for an Abuse Resistant Law Enforcement Access System.

Protocol $\mathbf{Real}_{\mathcal{A},\Pi}(1^\lambda, x, z)$

$\mathbf{Real}_{\mathcal{A},\Pi}(1^\lambda, x, z)$ is parameterized by the protocol $\Pi = (\mathsf{Setup}, \mathsf{SendMessage}, \mathsf{RequestWarrant}, \mathsf{ActivateWarrant}, \mathsf{VerifyWarrantStatus}, \mathsf{AccessMessage})$ and a variable $\mathsf{mode} \in \{\mathsf{pro}, \mathsf{ret}\}$.

1. When $\mathbf{Real}_{\mathcal{A},\Pi}(1^\lambda, x, z)$ is initialized, then all parties engage in the interactive protocol $\Pi.\mathsf{Setup}$
2. When P_i is activated with $(\mathsf{SendMessage}, \mathsf{sid}, P_j, m, 1)$, parties P_i, P_j, and P_{LE} engage in the interactive protocol $\Pi.\mathsf{SendMessage}$. P_{LE} learns some metadata meta about the message.
3. When P_i is activated with $(\mathsf{SendMessage}, \mathsf{sid}, P_j, m, 0)$, parties P_i, and P_{LE} engage in the interactive protocol $\Pi.\mathsf{SendMessage}$ (with P_j not getting output). P_{LE} learns some metadata meta about the message.
4. When P_{LE} is activated with $(\mathsf{RequestWarrant}, \mathsf{sid}, \hat{w})$, parties P_{LE} and P_J engage in the interactive protocol $\Pi.\mathsf{RequestWarrant}$.
5. When P_{LE} is activated with $(\mathsf{ActivateWarrant}, \mathsf{sid}, w)$, all parties engage in the interactive protocol $\Pi.\mathsf{ActivateWarrant}$.
6. When P_{LE} is activated with $(\mathsf{VerifyWarrantStatus}, \mathsf{sid}, c, \mathsf{meta}, w)$, if $\mathsf{mode} = \mathsf{pro}$, P_{LE} returns $\perp$. Otherwise, P_{LE} calls the non-interactive functionality $\Pi.\mathsf{VerifyWarrantStatus}(c, \mathsf{meta}, w)$
7. When P_{LE} is activated with $(\mathsf{AccessData}, \mathsf{sid}, c, \mathsf{meta}, w)$, if $\mathsf{mode} = \mathsf{pro}$, P_{LE} returns $\perp$. Otherwise, P_{LE} calls the non-interactive functionality $\Pi.\mathsf{AccessMessage}(c, \mathsf{meta}, w)$

Figure 5-2. The real world experiment for a protocol implementing $\mathcal{F}_{\mathsf{ARLEAS}}^{v,t,p,\theta,\mathsf{mode}}$

the other cases. For example, if both P_i and P_j would be corrupted, there is nothing stopping them from not using the system. Moreover, we also don't consider the case where P_J is the only corrupted party, this case is a more specific then when both P_{LE} and P_J are corrupted and P_J on its own doesn't have any additional information to achieve anything different. All parties engage in a real protocol execution Π, the adversary $\mathcal{A}$ sends all messages on behalf of the corrupted parties and can choose any polynomial time strategy.

In a real world protocol we assume that communication between a sender P_i and receiver P_j happens over a transparent channel, meaning all other parties are able to receive all

communication. We make this choice to simplify the protocol and security proofs. In the real world, this can be modeled with a service provider relaying messages between P_i and P_j that always complies with law enforcement requests and hands over encrypted messages when presented with a valid warrant. Note that this makes our modeling the worst case scenario, and therefore captures more selective service providers. Additionally, in practice, this service provider would validate if messages are well-formed to make sure P_i and P_j follow the real protocol.

For any adversary $\mathcal{A}$ with auxiliary input $z \in \{0,1\}^*$, input vector x, and security parameter λ, we denote the output of Π by $\mathbf{Real}_{\mathcal{A},\Pi}(1^\lambda, x, z)$.

Definition 11 *A protocol Π is said to be a secure ARLEAS protocol computing $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{mode}}$ if for every PPT real-world adversary $\mathcal{A}$, there exists an ideal-world PPT adversary $\mathcal{S}$ corrupting the same parties such that for every input x and auxiliary input z it holds that*

$$\mathbf{Ideal}_{\mathcal{S},\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{mode}}}(1^\lambda, x, z) \overset{c}{\approx} \mathbf{Real}_{\mathcal{A},\Pi}(1^\lambda, x, z)$$

5.5 Prospective Solution

In this section we describe a prospective ARLEAS scheme, the first of which work with warrants that specify the target's identity and the second of which supports arbitrary predicates. Recall that the key feature of the prospective case is that warrants must be activated *before* targets perform encryption. A key implication of this setting is that new cryptographic material can be generated and distributed to users each time law enforcement updates the set of active warrants. The technical challenge, therefore, is to ensure that this material is distributed in such a way that the surveillance it permits is *accountable*, without revealing to targets any confidential information about which messages are being accessed.

The need for accountability restricts us from using many natural cryptographic tools. For example, Identity Based Encryption (IBE) systems provide a natural form of key escrow.

Unfortunately, in a standard IBE scheme this key escrow is absolute: the master authority can decrypt any ciphertext in the system. To enable limited surveillance, we require a system in which only a subset of communications will be targeted at any time epoch, and no additional information about *non-targeted* plaintexts will be revealed to the authorities. The first scheme relies on the Π_{LTE} scheme presented in Section 2.1.7 and ensures that messages sent to recipients under surveillance contain a copy of the ciphertext that can be decrypted by law enforcement, while messages send to recipients not under surveillance contain only a lossy ciphertext.

For generality, our main second construction supports targeting by allowing warrants to specify an arbitrary predicate over the *metadata* of a transmitted messages. In practice, we realize this functionality through the use of public ledgers and non-interactive secure computation techniques.

5.5.1 UC-Realizing $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\textsf{pro}}$ for Identity-Based Predicates

Our first construction only permits warrants that specify the identity of individuals to surveil and leverages the Π_{LTE} scheme presented in Section 2.1.7. With this encryption scheme, encrypting to some public keys create information theoretic lossy ciphertexts, while using other public keys results in injective ciphertexts. We use this scheme to allow law enforcement to decrypt messages exactly when they have activated a warrant corresponding to the recipient's identity. When sending a new message, a user encrypts directly to the recipient as normal and creates a second ciphertext using the Π_{LTE} scheme. The key material used for the second ciphertext comes public parameters generated by law enforcement that are posted onto the public ledger.

Our construction makes use of a CCA secure encryption system Π_{Enc}, a SUF-CMA secure signature scheme Π_{Sign}, a lossy-tag encryption scheme Π_{LTE} (presented in Section 2.1.7). We now present a protocol $\pi_{\text{PRO}}^{v,t,p,\theta}$ in the $\mathcal{L}^{\textsf{Verify}}$, $\mathcal{F}_{CRS}^{\Pi_{\text{NIZK}},\textsf{ZKSetup}}$, $\mathcal{F}_{AUTH}$ hybrid model. Our scheme

consists of the following interactive protocols:

$\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{Setup}$:

- All users send (CRS) to $\mathcal{F}_{CRS}^{\Pi_{\text{NIZK}}.\mathsf{ZKSetup}}$ to retrieve the common reference string for the NIZK scheme.
- Each user P_j computes $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \Pi_{\text{Enc}}.\mathsf{KeyGen}(1^\lambda)$ and selects a unique tag tag_j and sends (pk_j, tag_j) to P_{LE} and to each other user P_i via $\mathcal{F}_{AUTH}$.
- The judge P_J computes $(\mathsf{pk}_{\text{sign}}, \mathsf{sk}_{\text{sign}}) \leftarrow \Pi_{\text{Sign}}.\mathsf{KeyGen}(1^\lambda)$ and send $\mathsf{pk}_{\text{sign}}$ to all other users via $\mathcal{F}_{AUTH}$.
- Law enforcement P_{LE} runs $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{ActivateWarrant}$ with as input an empty set $\emptyset$ as the valid warrants.

$\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{SendMessage}$:

- The sender P_i computes the ciphertext $(c_1, c_2, \pi, \mathsf{meta})$ as follows, and sends it to P_j and P_{LE} via $\mathcal{F}_{AUTH}$:
  - Send ($\mathsf{GetCounter}$) to $\mathcal{L}^{\mathsf{Verify}}$ and receive the current counter ℓ. Then query $\mathcal{L}^{\mathsf{Verify}}$ on (GetVal, ℓ) to receive the latest posting $(\ell, x, \pi_{\mathsf{publish}})$. Parse x as $(mpk, \pi, \mathsf{info})$. If $\Pi_{\text{NIZK}}.\mathsf{ZKVerify}(mpk, \mathsf{info}, \pi) = 0$ or

    $$\mathcal{L}^{\mathsf{Verify}}.\mathsf{Verify}(\ell \| (mpk, \pi, \mathsf{info}), \pi_{\mathsf{publish}}) = 0$$

    return $\perp$ and halt.
  - $c_1 \leftarrow \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}_j, m; r_1)$, where $r_1 \xleftarrow{\$} \{0,1\}^\lambda$
  - $c_2 \leftarrow \Pi_{\text{LTE}}.\mathsf{Enc}(mpk, tag_j, m; r_2)$ where $r_2 \xleftarrow{\$} \{0,1\}^\lambda$
  - Use $\Pi_{\text{NIZK}}.\mathsf{ZKProve}$ to compute π such that

    $$\pi \leftarrow NIZK\left\{(m, r_1, r_2) : \begin{array}{c} c_1 = \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}_j, m; r_1) \wedge \\ c_2 = \Pi_{\text{LTE}}.\mathsf{Enc}(mpk, tag_j, m; r_2) \end{array}\right\}$$

– Create $\mathsf{meta} \leftarrow tag_j$

- Upon receiving c from P_i, P_j calls $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$ on c. If the output is 1, then recover the message as $m \leftarrow \Pi_{\text{Enc}}.\mathsf{Dec}(\mathsf{sk}_j, c_2)$
- Upon receiving c from P_i, P_{LE} calls $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$ on c. If the output is 1, then recover the message as $m \leftarrow \Pi_{\text{LTE}}.\mathsf{Dec}(\mathsf{msk}, tag_j)$

$\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$:

- Any party parses $(c_1, c_2, \pi, \mathsf{meta}) \leftarrow c$ and verifies that π is correct and computes $v(\mathsf{meta}, aux)$, aborting if the output is 0. Otherwise, output 1.

$\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{RequestWarrant}$:

- P_{LE} sends $(\mathsf{RequestWarrant}, tag)$ to P_J via $\mathcal{F}_{AUTH}$. P_J then either decides to send $(\mathsf{Disapprove})$ to P_{LE} and halt or executes the following:
  – Verify that $p(tag) = 1$. If not send $(\mathsf{Disapprove})$ to P_{LE} and abort.
  – $\sigma \leftarrow \Pi_{\text{Sign}}.\mathsf{Sign}(\mathsf{sk}_{\text{sign}}, tag)$
  – Send the signed warrant (tag, σ) to P_{LE} via $\mathcal{F}_{AUTH}$.

$\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{ActivateWarrant}$:

- P_{LE} adds the new warrant w to the set of valid warrants $\mathcal{W}$. It then extracts the set of tags $\mathcal{T} \leftarrow \{tag | (tag, \sigma) \in \mathcal{W}\}$
- Compute $(mpk, \mathsf{msk}) \leftarrow \Pi_{\text{LTE}}.\mathsf{KeyGen}(1^\lambda, \mathcal{T}; r)$ for $r \xleftarrow{\$} \{0,1\}^\lambda$
- Compute $\mathsf{info} \leftarrow \{t(tag, \sigma) | (tag, \sigma) \in \mathcal{W}\}$

- Use $\Pi_{\text{NIZK}}.\mathsf{ZKProve}$ to compute π such that

$$\pi \leftarrow NIZK\{(\mathcal{W}, \mathcal{T}, \mathsf{msk}, r) : \mathsf{info} = \{t(tag, \sigma) | (tag, \sigma) \in \mathcal{W}\} \wedge$$
$$\mathcal{T} \leftarrow \{tag | (tag, \sigma) \in \mathcal{W}\} \wedge$$
$$(mpk, \mathsf{msk}) \in \Pi_{\text{LTE}}.\mathsf{KeyGen}(1^\lambda, \mathcal{T}; r) \wedge$$
$$\forall (tag, \sigma) \in \mathcal{W}, \Pi_{\text{Sign}}.\mathsf{Verify}(\mathsf{pk}_{\text{sign}}, tag, \sigma) = p(tag) = 1\}$$

- Send $(\mathsf{Post}, (mpk, \pi, \mathsf{info}))$ to $\mathcal{L}^{\mathsf{Verify}}$ and receive $(\ell, x, \pi_{\mathsf{publish}})$.

Theorem 7 *Assuming a CCA secure public key encryption scheme Π_{Enc}, a SUF-CMA secure signature scheme Π_{Sign}, a NIZK scheme Π_{NIZK}, and a lossy-tag encryption scheme Π_{LTE}, $\pi_{PRO}^{v,t,p,\theta}$ UC-realizes $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,pro}$ initialized in prospective mode in the $\mathcal{L}^{\mathsf{Verify}}$, $\mathcal{F}_{CRS}^{\Pi_{NIZK}.\mathsf{ZKSetup}}$, $\mathcal{F}_{AUTH}-$hybrid model for* meta *that contains receiver identity $\theta(w, \mathsf{meta}) = (w == \mathsf{meta})$.*

Proof. We prove that the above construction securely realizes $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ by showing that there does not exist a distinguisher $\mathcal{Z}$ that can distinguish between an interaction with the ideal functionality and a simulator $\mathcal{S}$ and the real protocol $\pi_{\text{PRO}}^{v,t,p,\theta}$. We define the interaction with the real protocol as follows: The experiment is initialized with N users $P_1, \ldots P_N$, law enforcement P_{LE} and a judge P_J. The adversary $\mathcal{A}$ chooses a subset of these users to corrupt. Then, users run $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{Setup}$, with $\mathcal{A}$ controlling the actions of the corrupted parties. Honest users then, according to their arbitrary strategy, run $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{SendMessage}$ to exchange messages with other users. Law enforcement interacts with the judge to get warrants via $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{RequestWarrant}$ and uses $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{ActivateWarrant}$ to start surveilling a user. Honest parties follow an arbitrary strategy, but follow the protocol and corrupted parties are controlled by the adversary.

We start our proof by first considering the case where a single user P_i is corrupted.

P_i is corrupted.

We begin by showing that $\pi_{\text{PRO}}^{v,t,p,\theta}$ UC-realizes the $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ when a user P_i is compromised. We construct the simulator $\mathcal{S}$ as follows:

1. $\mathcal{S}$ generates the common reference string for the NIZK scheme directly, and stores the trapdoor τ. $\mathcal{S}$ runs $(\mathsf{pk}_{\text{sign}}, \mathsf{sk}_{\text{sign}}) \leftarrow \Pi_{\text{Sign}}.\mathsf{KeyGen}(1^\lambda)$ as the judge would in the real protocol, and outputs $\mathsf{pk}_{\text{sign}}$ to the adversary $\mathcal{A}$. $\mathcal{S}$ initializes an instance of the ideal functionality in prospective mode. Finally, $\mathcal{S}$ runs $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{ActivateWarrant}$ with an empty active warrants set.

2. $\mathcal{S}$ computes $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \Pi_{\text{Enc}}.\mathsf{KeyGen}(1^\lambda)$ and samples a unique $tag_j \leftarrow \mathbb{Z}_p$ for all honest P_j and sends (pk_j, tag_j) to $\mathcal{A}$. Additionally, $\mathcal{S}$ waits to receive pk_i, tag_i from P_i.

3. When $\mathcal{S}$ receives $(c_1, c_2, \pi, \mathsf{meta})$ from $\mathcal{A}$ intended for uncorrupted P_j, $\mathcal{S}$ begins by verifying π and checking that meta is correct. If these checks pass, set $b \leftarrow 1$, and $b \leftarrow 0$ otherwise. $\mathcal{S}$ computes $m \leftarrow \Pi_{\text{Enc}}.\mathsf{Dec}(\mathsf{sk}_j, c_1)$. $\mathcal{S}$ then sends $(\mathsf{SendMessage}, P_j, m, b)$ to the ideal functionality.

4. Upon receiving $(\mathsf{Sent}, \mathsf{meta}, c, m)$ from the ideal functionality, $\mathcal{S}$ computes $(c_1, c_2, \pi, \mathsf{meta})$ using $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{SendMessage}$ and forwards it to P_i.

5. Upon receiving $(\mathsf{NotifyWarrant}, t(w))$ from $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$, $\mathcal{S}$ aggregates $t(w)$ from all

   $$(\mathsf{NotifyWarrant}, \cdot)$$

   messages seen so far into info. $\mathcal{S}$ then randomly chooses a set of tags $\mathcal{T}$ such that $|\mathcal{T}| = |\mathsf{info}|$ and computes $(mpk, \mathsf{msk}) \leftarrow \Pi_{\text{LTE}}.\mathsf{KeyGen}(1^\lambda, \mathcal{T})$. $\mathcal{S}$ then simulates the proof π and sends $(\mathsf{Post}, (mpk, \pi, \mathsf{info}))$ to $\mathcal{L}$.

We proceed with a hybrid argument. Let $\mathcal{H}_0$ denote the distribution of the view of $\mathcal{A}$ in the real world interaction.

$\mathcal{H}_1$: Let $\mathcal{H}_1$ be the same as $\mathcal{H}_0$, but instead of having the common reference string generated by the $\mathcal{F}_{CRS}^{\Pi_{\text{NIZK}}.\mathsf{ZKSetup}}$, the common reference string is generated using $(\mathsf{crs}, \tau) \leftarrow \Pi_{\text{NIZK}}.\mathsf{ZKSetup}(1^\lambda)$. Note that the common reference string is selected from exactly the same distribution, so the difference in the distribution of the view of $\mathcal{A}$ between $\mathcal{H}_0$ and $\mathcal{H}_1$ is 0.

$\mathcal{H}_2$: Let $\mathcal{H}_2$ be the same as $\mathcal{H}_1$, but when an honest P_{LE} sends $(\mathsf{Post}, (mpk, \pi, \mathsf{info}))$to $\mathcal{L}$, the proof π is instead simulated with τ. Because of the perfect zero-knowledge property of Π_{NIZK}, the adversary's view in $\mathcal{H}_2$ and $\mathcal{H}_1$ is statistically close.

$\mathcal{H}_3$: Let $\mathcal{H}_3$ be the same as $\mathcal{H}_2$, but when an honest P_{LE} sends $(\mathsf{Post}, (mpk, \pi, \mathsf{info}))$to $\mathcal{L}$, let mpk be generated with an random set of tags with the correct size. Because of the indistinguishability of tag sets property of Π_{LTE}, the difference in the distribution in the view of the $\mathcal{A}$ is negligible.

Because the view of $\mathcal{A}$ in $\mathcal{H}_3$ is distributed the same as in the ideal world with simulator $\mathcal{S}$, the proof is done. Notice that this proof extends directly to multiple corrupt users, as the views of the users are independent, except when they send messages to each other. However, such messages do not require simulation. Thus it suffices to simulate each one independently.

P_{LE} is corrupted. We now extend the previous proof to include corrupted P_{LE}. We extend $\mathcal{S}$ to simulate the view of P_{LE}. Note that step 5 described in $\mathcal{S}$ above is no longer applicable for corrupted users, as the notification mechanism from the actual protocol will look correct.

1. $\mathcal{S}$ generates the common reference string for the NIZK scheme directly, and stores the trapdoor τ. Then, $\mathcal{S}$ sets $\mathsf{ValidParameters}$ to true. $\mathcal{S}$ then runs $\Pi_{\text{Sign}}.\mathsf{KeyGen}(1^\lambda) \rightarrow (\mathsf{pk}_{\text{sign}}, \mathsf{sk}_{\text{sign}})$ as the judge would in the read protocol, and outputs $\mathsf{pk}_{\text{sign}}$ to the adversary $\mathcal{A}$. $\mathcal{S}$ initializes an instance of the ideal functionality in prospective mode. $\mathcal{S}$ computes $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \Pi_{\text{Enc}}.\mathsf{KeyGen}(1^\lambda)$ and samples a unique $tag_j \leftarrow \mathbb{Z}_p$ for all honest P_j and sends (pk_j, tag_j) to $\mathcal{A}$. When $\mathcal{S}$ detects $(mpk, \pi, \mathsf{info})$ posted on $\mathcal{L}$, it verifies the proof π, and sets $\mathsf{ValidParameters}$ to false if it does not verify or $|\mathsf{info}| \neq 0$. $\mathcal{S}$ then initializes

an empty warrant table W.

2. We split the case of receiving $(\mathsf{Sent}, \mathsf{meta}, c)$ from $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ intended for P_{LE} into three cases. All begin by $\mathcal{S}$ extracting the recipient P_j from meta:

   (a) If $\mathsf{ValidParameters}$ is false, $\mathcal{S}$ silently drops the message.

   (b) If $\mathsf{ValidParameters}$ is true and P_j is not corrupted, $\mathcal{S}$ and chooses a message m_0 and computes the ciphertext $(c_1, c_2, \pi, \mathsf{meta})$ by encrypting m_0 using

   $$\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{SendMessage}(\mathsf{pk}_j, tag_j, m_0).$$

   (c) If $\mathsf{ValidParameters}$ is true and P_j is corrupted, $\mathcal{S}$ will also receive $(\mathsf{Sent}, \mathsf{meta}, c, m)$ from the ideal functionality, intended for P_j. $\mathcal{S}$ then encrypts m using $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{SendMessage}(\mathsf{pk}_j, tag_j, m)$

   Finally, $\mathcal{S}$ sends the resulting ciphertext to $\mathcal{A}$.

3. Upon receiving $(\mathsf{Sent}, \mathsf{meta}, c, 0)$ from $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ intended for P_{LE}, $\mathcal{S}$ samples a random message and creates a ciphertext with $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{SendMessage}$. Then, it generates a false proof instead of the real proof.

4. Upon receiving $(\mathsf{Sent}, \mathsf{meta}, c, m)$ from $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ due to an active warrant,

   if $\mathsf{ValidParameters}$ is true, $\mathcal{S}$ calls the send message algorithm of the real protocol on

   $$\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{SendMessage}(\mathsf{pk}_j, tag_j, m)$$

   and sends the output to $\mathcal{A}$.

5. Upon receiving $(\mathsf{RequestWarrant}, tag_j)$ from the adversary, intended for P_J, $\mathcal{S}$ generates a warrant w for the ideal functionality that only targets P_j and sends $(\mathsf{RequestWarrant}, w)$ to $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$. If $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ answers with $(\mathsf{Approve})$, $\mathcal{S}$ uses the $\mathsf{sk}_{\text{sign}}$ for P_J to form and

sign (tag_j, σ) as in the real protocol and adds (w, False) to W. If $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ responds with $(\mathsf{Disapprove})$, $\mathcal{S}$ send $\perp$ to the adversary.

6. $\mathcal{S}$ monitors $\mathcal{L}$. Upon seeing a new post on the ledger, $\mathcal{S}$ performs the following steps

   (a) Retrieve the post by sending $(\mathsf{GetCounter})$ to $\mathcal{L}$ and receive the current counter ℓ. Then query $\mathcal{L}$ on (GetVal, ℓ) to receive the latest posting $(\ell, (mpk, \pi, \mathsf{info}), \pi_{\mathsf{publish}})$

   (b) Verify $\Pi_{\text{NIZK}}.\mathsf{ZKVerify}(mpk, \mathsf{info}, \pi) = 1$. If the proof does not verify, the $\mathcal{S}$ sets $\mathsf{ValidParameters}$ to false and halts.

   (c) Set $\mathsf{ValidParameters}$ to true and run $\mathsf{Extract}$ to recover $(\mathcal{W}, \mathcal{T}, \mathsf{msk}, r)$ from π. If extraction fails, the simulator halts with an error.

   (d) If there is an entry (w, False) in W for $w \in \mathcal{W}$ $\mathcal{S}$ sends $(\mathsf{ActivateWarrant}, w)$ to the ideal functionality and sets the entry to be (w, True).

We proceed with a hybrid argument. Let $\mathcal{H}_0$ denote the distribution of the view of $\mathcal{A}$ in the real world interaction.

$\mathcal{H}_\mathbf{1}$: Let $\mathcal{H}_1$ be the same as $\mathcal{H}_0$, but instead of having the common reference string generated by the $\mathcal{F}_{CRS}^{\Pi_{\text{NIZK}}.\mathsf{ZKSetup}}$, the common reference string is generated using $(\mathsf{crs}, \tau) \leftarrow \Pi_{\text{NIZK}}.\mathsf{ZKSetup}(1^\lambda)$. Note that the common reference string is selected from exactly the same distribution, so the the distribution in the view of $\mathcal{A}$ between $\mathcal{H}_0$ and $\mathcal{H}_1$ is statistically close.

$\mathcal{H}_\mathbf{2}$: Let $\mathcal{H}_2$ be the same as $\mathcal{H}_1$, except the proof of ciphertexts consistency in a ciphertext $(c_1, c_2, \pi, \mathsf{meta})$ bound for an honest user for which there is no active warrant is simulated. Due to the zero-knowledge property of Π_{NIZK}, the difference in the view of the adversary in $\mathcal{H}_2$ and $\mathcal{H}_1$ is negligible.

$\mathcal{H}_\mathbf{3}$: Let $\mathcal{H}_3$ be the same as $\mathcal{H}_2$, except when $\mathcal{S}$ receives a ciphertext $(c_1, c_2, \pi, \mathsf{meta})$bound for an honest user for which there is no active warrant, $\mathcal{S}$ samples a message m_0 that would

result in the same metadata and sets $c_1 \leftarrow \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}_j, m_0; r_1)$. By the CCA security of Π_{Enc} the advantage of $\mathcal{A}$ in distinguishing between $\mathcal{H}_3$ and $\mathcal{H}_2$ is negligible.

$\mathcal{H}_4$: Let $\mathcal{H}_4$ be the same as $\mathcal{H}_3$, except the second ciphertext element c_2 in a ciphertext $(c_1, c_2, \pi, \mathsf{meta})$ bound for an honest user for which there is no active warrant is computed as $\Pi_{\text{LTE}}.\mathsf{Enc}(mpk, tag_j, m_0; r_2)$. By the lossy property of Π_{LTE}, $\mathcal{H}_4$ and $\mathcal{H}_3$ are statistically indistinguishable.

$\mathcal{H}_5$: Let $\mathcal{H}_5$ be the same as $\mathcal{H}_4$, except the proof of ciphertexts consistency in a ciphertext $(c_1, c_2, \pi, \mathsf{meta})$ bound for an honest user for which there is no active warrant is computed honestly with respect to the plaintext message m_0. Again, by the zero-knowledge property of Π_{NIZK}, the difference in the view of the adversary in $\mathcal{H}_5$ and $\mathcal{H}_4$ is negligible.

$\mathcal{H}_6$: Let $\mathcal{H}_6$ be the same as $\mathcal{H}_5$, except when $\mathcal{A}$ detects $(mpk, \pi, \mathsf{info})$ being posted on the ledger, $\mathcal{S}$ attempts to run the extractor $\Pi_{\text{NIZK}}.\mathsf{Extract}$ and abort the experiment if it fails. However, because the extractor only fails with negligible probability, the difference in the view of the adversary between $\mathcal{H}_6$ and $\mathcal{H}_5$ is negligible.

$\mathcal{H}_6$ has the same distribution as $\mathcal{S}$, concluding the proof. In the real world, P_{LE} would be denied warrants at the same rate as in the ideal world, as an honest P_J handles warrant requests in the same way. One note is that law enforcement can "deactivate" warrants in way not possible in the ideal functionality. However, when warrants are deactivated, honestly encrypting the message still hides the plaintext from the adversary.

P_J and P_{LE} are corrupted. We now focus on the case when P_J and P_{LE} are both corrupted. Note that step 4 of the above simulator description is no longer relevant, as the warrant request is handled internally by $\mathcal{A}$. Our simulator requires on minor changes to steps 1 and 5, which we show below.

1. Do the setup as before, but waiting to receive $\mathsf{pk}_{\text{sign}}$ from $\mathcal{A}$

⋮

5. $\mathcal{S}$ monitors $\mathcal{L}$. Upon seeing a new post on the ledger, $\mathcal{S}$ performs the following steps

   (a) Retrieve the post by sending ($\mathsf{GetCounter}$) to $\mathcal{L}$ and receive the current counter ℓ. Then query $\mathcal{L}$ on (GetVal, ℓ) to receive the latest posting $(\ell, (mpk, \pi, \mathsf{info}), \pi_{\mathsf{publish}})$

   (b) Verify $\Pi_{\text{NIZK}}.\mathsf{ZKVerify}(mpk, \mathsf{info}, \pi) = 1$. If the proof does not verify, the $\mathcal{S}$ sets $\mathsf{ValidParameters}$ to false and halts.

   (c) Set $\mathsf{ValidParameters}$ to true and runs $(\mathcal{W}, \mathcal{T}, \mathsf{msk}, r) \leftarrow \mathsf{Extract}(\mathsf{CRS}_{ZK}, \tau, x, \pi)$. If extraction fails, the simulator halts with an error.

   (d) for each warrant $w \in \mathcal{W}$ for which there does not exist an entry (w, True) in W, the $\mathcal{S}$ executes the following steps

      i. $\mathcal{S}$ generates a warrant w for the ideal functionality that only targets P_j and sends
      $(\mathsf{RequestWarrant}, w)$ to $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$.

      ii. When $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ sends $(\mathsf{ApproveWarrant}, w)$ to the $\mathcal{S}$ intended to P_J, $\mathcal{S}$ responds $(\mathsf{Approve})$ on behalf of P_J

      iii. $\mathcal{S}$ sends $(\mathsf{ActivateWarrant}, w)$ to the ideal functionality

      iv. $\mathcal{S}$ adds (w, True) to W

In fact, the hybrid argument above holds directly in this case as well. First notice that there are no additional messages that require simulation. By giving the adversary control of P_J, we give it the ability to create valid warrants independently. However, there is no change in behavior expected until those warrants are activated. As such, those warrants can be requested from the ideal functionality right as they are being activated. □

5.5.2 UC-Realizing $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ for Arbitrary Predicates

The prospective solution we have presented is limited in the flexibility of warrants; unlike the ideal functionality, warrants were limited to specifying a single individual. It is clear that

it would be better to support *arbitrary* warrants whose applicability to some meta could be checked with the warrant scope functionality θ.

First, we note that directly extending the existing solution's intuitions is insufficient, as creating a one-to-one correspondence between tags and meta would clearly result in exponentially sized public parameters. We require a mechanism that evaluates the warrant scope predicate $\theta(w, \mathsf{meta})$ and outputs the message only if the result is 1. Clearly this can be accomplished using extractable witness encryption, and indeed we will use extractable witness encryption to accomplish a similar goal in Section 5.6. However, we are able to leverage the *prospective* nature of this case to realize prospective ARLEAS from non-interactive secure computation (NISC) [126]. Recall that a NISC scheme for some function f allows a receiver to post an encryption of some secret x_1 such that all players can reveal $f(x_1, x_2)$ to the receiver with only one message, without revealing anything about x_2 beyond the output of the function. For the following construction, we require an NISC scheme for the function I_k, defined as

$$I_k((w_1, w_2, \ldots, w_k), (m, \mathsf{meta})) = m \wedge (\theta(\mathsf{meta}, w_1) \vee \ldots \vee \theta(\mathsf{meta}, w_k)).$$

This function evaluates the warrant scope check functionality on the metadata over k different warrants. If any of them evaluate to true, the message is output. Otherwise, I_k outputs 0. Note that the number of warrants is an explicit parameter of the function and its circuit representation.

Law enforcement begins by posting the first message of the NISC scheme, embedding as input their k warrants, along with the transparency information and proof of correctness. As before, senders send a ciphertext (c_1, c_2, π). c_1 remains a normal public key ciphertext for the recipient. c_2 is modified (from the previous construction) to be the second message of the NISC scheme. This message must embed the inputs (m, meta). Most known realizations of NISC rely on garbled circuits, with the second message containing the garbling of the

intended function and hardcoding the sender's inputs. As such, we need to ensure that the sender computes the second message of the NISC scheme with respect to the correct functionality; this can be handled by requiring malicious security from the underlying NISC scheme or by including a correctness in the NIZK π. As we require a proof of consistency (*i.e.* c_1 *and* c_2 *embed the same message*), we already require non-blackbox use of the NISC scheme.

Upon receiving the resulting ciphertext, law enforcement can attempt to decrypt by evaluating the NISC ciphertext. By the security of the NISC scheme, law enforcement will only learn information about the plaintext if they have a relevant warrant and posted the required transparency information, accomplishing our goal.

We now proceed to give a formal description of this protocol.

$\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{Setup}$:

- All users send (CRS) to $\mathcal{F}_{CRS}^{\Pi_{\text{NIZK}}.\mathsf{ZKSetup}}$ to retrieve the common reference string for the NIZK scheme and all users send (CRS) to $\mathcal{F}_{CRS}^{\Pi_{\text{NISC}}.\mathsf{GenCRS}}$ to retrieve the common reference string for the NISC scheme $\mathsf{CRS}_{\mathsf{NISC}}$.
- Each user P_j computes $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \Pi_{\text{Enc}}.\mathsf{KeyGen}(1^\lambda)$ and sends pk_j to P_{LE} and to each P_i via $\mathcal{F}_{AUTH}$.
- The judge P_J computes $(\mathsf{pk}_{\text{sign}}, \mathsf{sk}_{\text{sign}}) \leftarrow \Pi_{\text{Sign}}.\mathsf{KeyGen}(1^\lambda)$ and send $\mathsf{pk}_{\text{sign}}$ to all other users via $\mathcal{F}_{AUTH}$.
- Law enforcement P_{LE} runs $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{ActivateWarrant}$ with an empty set $\emptyset$ as the valid warrants.

$\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{SendMessage}$:

- The sender P_i computes the ciphertext $(c_1, c_2, \pi, \mathsf{meta})$ as follows, and sends it to P_j and P_{LE} via $\mathcal{F}_{AUTH}$:

– Send (GetCounter) to $\mathcal{L}^{\mathsf{Verify}}$ and receive the current counter ℓ. Then query $\mathcal{L}^{\mathsf{Verify}}$ on (GetVal, ℓ) to receive the latest posting $(\ell, x, \pi_{\mathsf{publish}})$. Parse x as $(\mathsf{nisc}_1^{\text{public}}, \pi, \mathsf{info})$. If $\Pi_{\text{NIZK}}.\mathsf{ZKVerify}(\mathsf{nisc}_1^{\text{public}}, \mathsf{info}, \pi) = 0$ or

$$\mathcal{L}^{\mathsf{Verify}}.\mathsf{Verify}(\ell \| (\mathsf{nisc}_1^{\text{public}}, \pi, \mathsf{info}), \pi_{\mathsf{publish}}) = 0$$

return $\perp$ and halt.

– $c_1 \leftarrow \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}_j, m; r_1)$, where $r_1 \xleftarrow{\$} \{0,1\}^\lambda$

– Create meta

–

$$\mathsf{nisc}_2 \leftarrow \Pi_{\text{NISC}}.\mathsf{NISC}_2(\mathsf{CRS}_{\mathsf{NISC}}, I_{|\mathsf{info}|}, (m, \mathsf{meta}), \mathsf{nisc}_1^{\text{public}}; r_2),$$

where $r_2 \xleftarrow{\$} \{0,1\}^\lambda$

– $c_2 \leftarrow \mathsf{nisc}_2$

– Use $\Pi_{\text{NIZK}}.\mathsf{ZKProve}$ to compute π such that

$$\pi \leftarrow \mathit{NIZK}\left\{(m, r_1, r_2) : \begin{array}{c} c_1 = \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}_j, m; r_1) \wedge \\ c_2 = \Pi_{\text{NISC}}.\mathsf{NISC}_2(\mathsf{CRS}_{\mathsf{NISC}}, I_{|\mathsf{info}|}, (m, \mathsf{meta}), \mathsf{nisc}_1^{\text{public}}; r_2) \end{array}\right\}$$

- Upon receiving c from P_i, P_j calls $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$ on c. If the output is 1, then recover the message as $m \leftarrow \Pi_{\text{Enc}}.\mathsf{Dec}(\mathsf{sk}_j, c_2)$
- Upon receiving c from P_i, P_{LE} calls $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$ on c. If the output is 1, then recover the message as $m \leftarrow \Pi_{\text{NISC}}.\mathsf{Evaluate}(\mathsf{CRS}_{\mathsf{NISC}}, \mathsf{nisc}_2, \mathsf{nisc}_1^{\text{private}})$

$\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$:

- Any party parses $(c_1, c_2, \pi, \mathsf{meta}) \leftarrow c$ and verifies that π is correct and computes $v(\mathsf{meta}, aux)$, aborting if the output is 0. Otherwise, output 1.

$\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{RequestWarrant}$:

- P_{LE} sends $(\mathsf{RequestWarrant}, \hat{w})$ to $P_J$via $\mathcal{F}_{AUTH}$. P_J then either decides to send $(\mathsf{Disapprove})$ to P_{LE} and halt or executes the following:
  - Verify that $p(\hat{w}) = 1$. If not send $(\mathsf{Disapprove})$ to P_{LE} and abort.
  - $\sigma \leftarrow \Pi_{\text{Sign}}.\mathsf{Sign}(\mathsf{sk}_{\text{sign}}, \hat{w})$
  - Send the signed warrant $w = (\hat{w}, \sigma)$ to P_{LE} via $\mathcal{F}_{AUTH}$.

$\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{ActivateWarrant}$:

- P_{LE} adds the new warrant w to the set of valid warrants $\mathcal{W}$. Let $w^* = w_1 \| \ldots \| w_{|\mathcal{W}|}$ for $w_i = (\hat{w}_i, \sigma_i) \in \mathcal{W}$.
- $(\mathsf{nisc}_1^{\text{public}}, \mathsf{nisc}_1^{\text{private}}) \leftarrow \Pi_{\text{NISC}}.\mathsf{NISC_1}(\mathsf{CRS}_{\mathsf{NISC}}, w^*; r)$ and record $\mathsf{nisc}_1^{\text{private}}$
- Compute $\mathsf{info} \leftarrow \{t(w) | w \in \mathcal{W}\}$
- Use $\Pi_{\text{NIZK}}.\mathsf{ZKProve}$ to compute π such that

$$\pi \leftarrow NIZK \left\{ \begin{array}{r} \mathsf{info} = \{t(w) | w \in \mathcal{W}\} \wedge \\ (\mathcal{W}, \mathsf{nisc}_1^{\text{private}}, r) : (\mathsf{nisc}_1^{\text{public}}, \mathsf{nisc}_1^{\text{private}}) \leftarrow \Pi_{\text{NISC}}.\mathsf{NISC_1}(\mathsf{CRS}_{\mathsf{NISC}}, w^*; r) \wedge \\ \forall (\hat{w}, \sigma) \in \mathcal{W}, \Pi_{\text{Sign}}.\mathsf{Verify}(\mathsf{pk}_{\text{sign}}, \hat{w}, \sigma) = p(\hat{w}) = 1 \end{array} \right\}$$

- Send $(\mathsf{Post}, (\mathsf{nisc}_1^{\text{public}}, \pi, \mathsf{info}))$ to $\mathcal{L}^{\mathsf{Verify}}$ and receive $(\ell, x, \pi_{\mathsf{publish}})$.

Theorem 8 *Assuming a CCA secure public key encryption scheme Π_{Enc}, a SUF-CMA secure signature scheme Π_{Sign}, a NIZK scheme Π_{NIZK}, and an NISC scheme Π_{NISC}, $\pi_{PRO}^{v,t,p,\theta}$ UC-realizes $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,pro}$ initialized in prospective mode in the $\mathcal{L}^{\mathsf{Verify}}$, $\mathcal{F}_{CRS}^{\Pi_{NIZK}.\mathsf{ZKSetup}}$, $\mathcal{F}_{CRS}^{\Pi_{NISC}.\mathsf{GenCRS}}$, $\mathcal{F}_{AUTH}-$hybrid model.*

Proof. As above, we now prove that our construction securely realizes $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ by showing that there does not exist a distinguisher $\mathcal{Z}$ that can distinguish between an interaction with the ideal functionality and a simulator $\mathcal{S}$ and the real protocol $\pi_{\text{PRO}}^{v,t,p,\theta}$. We define the interaction with the real protocol as above.

We start our proof by first considering the case where a single user P_i is corrupted.

P_i is corrupted.

We begin by showing that $\pi_{\text{PRO}}^{v,t,p,\theta}$ UC-realizes the $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ when a user P_i is compromised. We construct the simulator $\mathcal{S}$ as follows:

1. $\mathcal{S}$ generates the common reference string for the NIZK scheme and $(\mathsf{CRS}_{\mathsf{NISC}}, \tau_{\mathsf{NISC}}) \leftarrow \Pi_{\text{NISC}}.\mathsf{GenCRS}(1^\lambda)$directly, and stores the trapdoors $(\tau, \tau_{\mathsf{NISC}})$. $\mathcal{S}$ runs $(\mathsf{pk}_{\text{sign}}, \mathsf{sk}_{\text{sign}}) \leftarrow \Pi_{\text{Sign}}.\mathsf{KeyGen}(1^\lambda)$as the judge would in the real protocol, and outputs $\mathsf{pk}_{\text{sign}}$ to the adversary $\mathcal{A}$. $\mathcal{S}$ initializes an instance of the ideal functionality in prospective mode. Finally, $\mathcal{S}$ runs $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{ActivateWarrant}$ with an empty active warrants set.

2. $\mathcal{S}$ computes $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \Pi_{\text{Enc}}.\mathsf{KeyGen}(1^\lambda)$ for all honest P_j and sends pk_j to $\mathcal{A}$. Additionally, $\mathcal{S}$ waits to receive pk_i from P_i.

3. When $\mathcal{S}$ receives $(c_1, c_2, \pi, \mathsf{meta})$ from $\mathcal{A}$ intended for uncorrupted P_j, $\mathcal{S}$ begins by verifying π and checking that meta is correct. If these checks pass, set $b \leftarrow 1$, and $b \leftarrow 0$ otherwise. $\mathcal{S}$ computes $m \leftarrow \Pi_{\text{Enc}}.\mathsf{Dec}(\mathsf{sk}_j, c_1)$. $\mathcal{S}$ then sends $(\mathsf{SendMessage}, P_j, m, b)$ to the ideal functionality.

4. Upon receiving $(\mathsf{Sent}, \mathsf{meta}, c, m)$ from the ideal functionality, $\mathcal{S}$ computes $(c_1, c_2, \pi, \mathsf{meta})$ using $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{SendMessage}$ and forwards it to P_i.

5. Upon receiving $(\mathsf{NotifyWarrant}, t(w))$ from $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$, $\mathcal{S}$ aggregates $t(w)$ from all

$$(\mathsf{NotifyWarrant}, \cdot)$$

messages seen so far into info. $\mathcal{S}$ then chooses random inputs for the first round messages of the NISC scheme to get $(\mathsf{nisc}_1^{\text{public}}, \cdot)$ and simulates the proof π and sends $(\mathsf{Post}, (\mathsf{nisc}_1^{\text{public}}, \pi, \mathsf{info}))$ to $\mathcal{L}$.

We proceed with a hybrid argument. Let $\mathcal{H}_0$ denote the distribution of the view of $\mathcal{A}$ in the real world interaction.

$\mathcal{H}_\mathbf{1}$: Let $\mathcal{H}_1$ be the same as $\mathcal{H}_0$, but instead of having the common reference string generated by the $\mathcal{F}_{CRS}^{\Pi_{\text{NIZK}}.\mathsf{ZKSetup}}$, the common reference string is generated using $(\mathsf{CRS}_{ZK}, \tau) \leftarrow \Pi_{\text{NIZK}}.\mathsf{ZKSetup}(1^\lambda)$ and $(\mathsf{CRS}_{\mathsf{NISC}}, \tau_{\mathsf{NISC}}) \leftarrow \Pi_{\text{NISC}}.\mathsf{GenCRS}(1^\lambda)$. Note that the common reference strings are selected from exactly the same distribution, so the difference in the distribution of the view of $\mathcal{A}$ between $\mathcal{H}_0$ and $\mathcal{H}_1$ is 0.

$\mathcal{H}_\mathbf{2}$: Let $\mathcal{H}_2$ be the same as $\mathcal{H}_1$, but when an honest P_{LE} sends $(\mathsf{Post}, (\mathsf{nisc}_1^{\text{public}}, \pi, \mathsf{info}))$to $\mathcal{L}$, the proof π is instead simulated with τ. Because of the perfect zero-knowledge property of Π_{NIZK}, the adversary's view in $\mathcal{H}_2$ and $\mathcal{H}_1$ is statistically close.

$\mathcal{H}_\mathbf{3}$: Let $\mathcal{H}_3$ be the same as $\mathcal{H}_2$, but when an honest P_{LE} sends $(\mathsf{Post}, (\mathsf{nisc}_1^{\text{public}}, \pi, \mathsf{info}))$to $\mathcal{L}$, let $\mathsf{nisc}_1^{\text{public}}$ be generated on random input of the right length. By the security of Π_{NISC}, the difference in the distribution in the view of the $\mathcal{A}$ is negligible.

Because the view of $\mathcal{A}$ in $\mathcal{H}_3$ is distributed the same as in the ideal world with simulator $\mathcal{S}$, the proof is done. Notice that this proof extends directly to multiple corrupt users, as the views of the users are independent, except when they send messages to each other. However, such messages do not require simulation. Thus it suffices to simulate each one independently.

P_{LE} is corrupted. We now extend the previous proof to include corrupted P_{LE}. We extend $\mathcal{S}$ to simulate the view of P_{LE}. Note that step 5 described in $\mathcal{S}$ above is no longer applicable for corrupted users, as the notification mechanism from the actual protocol will look correct.

1. $\mathcal{S}$ generates the common reference string for the NIZK scheme and $(\mathsf{CRS}_{\mathsf{NISC}}, \tau_{\mathsf{NISC}}) \leftarrow \Pi_{\text{NISC}}.\mathsf{GenCRS}(1^\lambda)$directly, and stores the trapdoors $(\tau, \tau_{\mathsf{NISC}})$. $\mathcal{S}$ runs $(\mathsf{pk}_{\text{sign}}, \mathsf{sk}_{\text{sign}}) \leftarrow \Pi_{\text{Sign}}.\mathsf{KeyGen}(1^\lambda)$as the judge would in the real protocol, and outputs $\mathsf{pk}_{\text{sign}}$ to the adversary $\mathcal{A}$. $\mathcal{S}$ initializes an instance of the ideal functionality in prospective mode. Finally, $\mathcal{S}$ runs $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{ActivateWarrant}$ with an empty active warrants set. $\mathcal{S}$ computes

$(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \Pi_{\text{Enc}}.\mathsf{KeyGen}(1^\lambda)$ for all honest P_j and sends pk_j to $\mathcal{A}$. When $\mathcal{S}$ detects $(\mathsf{nisc}_1^{\text{public}}, \pi, \mathsf{info})$ posted on $\mathcal{L}$, it verifies the proof π, and sets ValidParameters to false if it does not verify or $|\mathsf{info}| \neq 0$. $\mathcal{S}$ then initializes an empty warrant table W.

2. We split the case of receiving $(\mathsf{Sent}, \mathsf{meta}, c)$ from $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ intended for P_{LE} into three cases. All begin by $\mathcal{S}$ extracting the recipient P_j from meta:

   (a) If ValidParameters is false, $\mathcal{S}$ silently drops the message.

   (b) If ValidParameters is true and P_j is not corrupted, $\mathcal{S}$ and chooses a message m_0 and computes the ciphertext $(c_1, c_2, \pi, \mathsf{meta})$ by encrypting m_0 using

   $$\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{SendMessage}(\mathsf{pk}_j, m_0).$$

   (c) If ValidParameters is true and P_j is corrupted, $\mathcal{S}$ will also receive $(\mathsf{Sent}, \mathsf{meta}, c, m)$ from the ideal functionality, intended for P_j. $\mathcal{S}$ then encrypts m using $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{SendMessage}(\mathsf{pk}_j, m)$

   Finally, $\mathcal{S}$ sends the resulting ciphertext to $\mathcal{A}$.

3. Upon receiving $(\mathsf{Sent}, \mathsf{meta}, c, 0)$ from $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ intended for P_{LE}, $\mathcal{S}$ samples a random message and creates a ciphertext with $\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{SendMessage}$. Then, it generates a false proof instead of the real proof.

4. Upon receiving $(\mathsf{Sent}, \mathsf{meta}, c, m)$ from $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ due to an active warrant, if ValidParameters is true, $\mathcal{S}$ calls the send message algorithm of the real protocol on

   $$\pi_{\text{PRO}}^{v,t,p,\theta}.\mathsf{SendMessage}(\mathsf{pk}_j, m)$$

   and sends the output to $\mathcal{A}$.

5. Upon receiving $(\mathsf{RequestWarrant}, \hat{w})$ from the adversary, intended for P_J, $\mathcal{S}$ sends $(\mathsf{RequestWarrant}, \hat{w})$ to $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$. If $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ answers with $(\mathsf{Approve})$, $\mathcal{S}$ uses the $\mathsf{sk}_{\text{sign}}$

for P_J to form and sign $w = (\hat{w}, \sigma)$ as in the real protocol and adds (w, False) to W. If $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ responds with $(\mathsf{Disapprove})$, $\mathcal{S}$ send $\perp$ to the adversary.

6. $\mathcal{S}$ monitors $\mathcal{L}$. Upon seeing a new post on the ledger, $\mathcal{S}$ performs the following steps

   (a) Retrieve the post by sending $(\mathsf{GetCounter})$ to $\mathcal{L}$ and receive the current counter ℓ. Then query $\mathcal{L}$ on (GetVal, ℓ) to receive the latest posting $(\ell, (\mathsf{nisc}_1^{\text{public}}, \pi, \mathsf{info}), \pi_{\mathsf{publish}})$

   (b) Verify $\Pi_{\text{NIZK}}.\mathsf{ZKVerify}(\mathsf{nisc}_1^{\text{public}}, \mathsf{info}, \pi) = 1$. If the proof does not verify, the $\mathcal{S}$ sets $\mathsf{ValidParameters}$ to false and halts.

   (c) Set $\mathsf{ValidParameters}$ to true and run $\mathsf{Extract}$ to recover $(\mathcal{W}, \mathsf{nisc}_1^{\text{private}}, r)$ from π. If extraction fails, the simulator halts with an error.

   (d) If there is an entry (w, False) in W for $w \in \mathcal{W}$ $\mathcal{S}$ sends $(\mathsf{ActivateWarrant}, w)$ to the ideal functionality and sets the entry to be (w, True).

We proceed with a hybrid argument. Let $\mathcal{H}_0$ denote the distribution of the view of $\mathcal{A}$ in the real world interaction.

$\mathcal{H}_1$:Let $\mathcal{H}_1$ be the same as $\mathcal{H}_0$, but instead of having the common reference string generated by the $\mathcal{F}_{CRS}^{\Pi_{\text{NIZK}}.\mathsf{ZKSetup}}$, the common reference string is generated using $(\mathsf{CRS}_{ZK}, \tau) \leftarrow \Pi_{\text{NIZK}}.\mathsf{ZKSetup}(1^\lambda)$ and $(\mathsf{CRS}_{\mathsf{NISC}}, \tau_{\mathsf{NISC}}) \leftarrow \Pi_{\text{NISC}}.\mathsf{GenCRS}(1^\lambda)$. Note that the common reference strings are selected from exactly the same distribution, so the the distribution in the view of $\mathcal{A}$ between $\mathcal{H}_0$ and $\mathcal{H}_1$ is statistically close.

$\mathcal{H}_2$: Let $\mathcal{H}_2$ be the same as $\mathcal{H}_1$, except when $\mathcal{S}$ receives a ciphertext $(c_1, c_2, \pi, \mathsf{meta})$such that no signed warrant w has been issued such that $\theta(w, \mathsf{meta}) = 1$, the the ciphertexts consistency proof π is simulated. Due to the zero-knowledge property of Π_{NIZK}, the difference in the view of the adversary in $\mathcal{H}_2$ and $\mathcal{H}_1$ is negligible.

$\mathcal{H}_3$: Let $\mathcal{H}_3$ be the same as $\mathcal{H}_2$, except when $\mathcal{S}$ receives a ciphertext $(c_1, c_2, \pi, \mathsf{meta})$such that no signed warrant w has been issued such that $\theta(w, \mathsf{meta}) = 1$, $\mathcal{S}$ samples a message m_0

that would result in the same metadata and sets $c_1 \leftarrow (\Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}_j, m_0; r_1)$. By the CCA security of Π_{Enc} the advantage of $\mathcal{A}$ in distinguishing between $\mathcal{H}_3$ and $\mathcal{H}_2$ is negligible.

$\mathcal{H}_4$: Let $\mathcal{H}_4$ be the same as $\mathcal{H}_3$, except when $\mathcal{S}$ receives a ciphertext $(c_1, c_2, \pi, \mathsf{meta})$such that no signed warrant w has been issued such that $\theta(w, \mathsf{meta}) = 1$, $\mathcal{S}$ computes c_2 as $\Pi_{\text{NISC}}.\mathsf{NISC}_2(\mathsf{CRS}_{\mathsf{NISC}}, C, (m_0, \mathsf{meta}), \mathsf{nisc}_1^{\text{public}}; r_2)$. By the security of $\Pi_{\text{NISC}}.\mathsf{NISC}_2$, $\mathcal{A}$ can only learn the correct output of C. Because there is no issued warrant $\theta(w, \mathsf{meta}) = 1$, this means the output of C is independent of m_0. Therefore, $\mathcal{H}_4$ and $\mathcal{H}_3$ are computationally indistinguishable.

$\mathcal{H}_5$: Let $\mathcal{H}_5$ be the same as $\mathcal{H}_4$,except when $\mathcal{S}$ receives a ciphertext $(c_1, c_2, \pi, \mathsf{meta})$such that no signed warrant w has been issued such that $\theta(w, \mathsf{meta}) = 1$, π is computed honestly with respect to the plaintext message m_0. Again, by the zero-knowledge property of Π_{NIZK}, the difference in the view of the adversary in $\mathcal{H}_5$ and $\mathcal{H}_4$ is negligible.

$\mathcal{H}_6$: Let $\mathcal{H}_6$ be the same as $\mathcal{H}_5$, except when $\mathcal{A}$ detects $(\mathsf{nisc}_1^{\text{public}}, \pi, \mathsf{info})$ being posted on the ledger, $\mathcal{S}$ attempts to run the extractor $\Pi_{\text{NIZK}}.\mathsf{Extract}$ and abort the experiment if it fails. However, because the extractor only fails with negligible probability, the difference in the view of the adversary between $\mathcal{H}_6$ and $\mathcal{H}_5$ is negligible.

$\mathcal{H}_6$ has the same distribution as $\mathcal{S}$, concluding the proof. In the real world, P_{LE} would be denied warrants at the same rate as in the ideal world, as an honest P_J handles warrant requests in the same way. One note is that law enforcement can "deactivate" warrants in way not possible in the ideal functionality. However, when warrants are deactivated, honestly encrypting the message still hides the plaintext from the adversary.

P_J and P_{LE} are corrupted. We now focus on the case when P_J and P_{LE} are both corrupted. Note that step 4 of the above simulator description is no longer relevant, as the warrant request is handled internally by $\mathcal{A}$. Our simulator requires on minor changes to steps 1 and 5, which we show below.

1. Do the setup as before, but waiting to receive $\mathsf{pk}_{\text{sign}}$ from $\mathcal{A}$

⋮

5. $\mathcal{S}$ monitors $\mathcal{L}$. Upon seeing a new post on the ledger, $\mathcal{S}$ performs the following steps

   (a) Retrieve the post by sending ($\mathsf{GetCounter}$) to $\mathcal{L}$ and receive the current counter ℓ. Then query $\mathcal{L}$ on (GetVal, ℓ) to receive the latest posting $(\ell, (\mathsf{nisc}_1^{\text{public}}, \pi, \mathsf{info}), \pi_{\mathsf{publish}})$

   (b) Verify $\Pi_{\text{NIZK}}.\mathsf{ZKVerify}(\mathsf{nisc}_1^{\text{public}}, \mathsf{info}, \pi) = 1$. If the proof does not verify, the $\mathcal{S}$ sets $\mathsf{ValidParameters}$ to false and halts.

   (c) Set $\mathsf{ValidParameters}$ to true and run $\mathsf{Extract}$ to recover $(\mathcal{W}, \mathsf{nisc}_1^{\text{private}}, r)$ from π. If extraction fails, the simulator halts with an error.

   (d) for each warrant $w \in \mathcal{W}$ for which there does not exist an entry (w, True) in W, the $\mathcal{S}$ executes the following steps

      i. $\mathcal{S}$ sends ($\mathsf{RequestWarrant}, w$) to $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$.

      ii. When $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{pro}}$ sends ($\mathsf{ApproveWarrant}, w$) to the $\mathcal{S}$ intended to P_J, $\mathcal{S}$ responds ($\mathsf{Approve}$) on behalf of P_J

      iii. $\mathcal{S}$ sends ($\mathsf{ActivateWarrant}, w$) to the ideal functionality

      iv. $\mathcal{S}$ adds (w, True) to W

In fact, the hybrid argument above holds directly in this case as well. First notice that there are no additional messages that require simulation. By giving the adversary control of P_J, we give it the ability to create valid warrants independently. However, there is no change in behavior expected until those warrants are activated. As such, those warrants can be requested from the ideal functionality right as they are being activated. □

5.6 Retrospective Solution

In the previous section we proposed a protocol to realize ARLEAS under the restriction that access would be prospective only. That protocol requires that law enforcement must activate a warrant and post the resulting parameters on the ledger before any targeted communication occurs. In this section we address the retrospective case. The key difference in this protocol is that law enforcement may activate a warrant at any stage of the protocol, even after a target communication has occurred.

In this setting we assume law enforcement has a way of getting messages that were sent in the past. As described before, we take the simplifying assumption that messages automatically get send to law enforcement. In practice, either a service provider can forward them, after checking the warrant. One can try to avoid surveillance by using expiring messages, but service providers can be forced to keep encrypted messages for a certain period of time. Or law enforcement can actively record messages in transit.

Our construction makes use of an extractable witness encryption scheme Π_{EWE} (see Definition 7) to encrypt the law enforcement ciphertext c_2. This scheme is parameterized by a language L_{EWE} that is defined with respect to the transparency function $t(\cdot)$, the policy function $p(\cdot)$, the targeting function $\theta(\cdot,\cdot)$, the warrant signing key $\mathsf{pk}_{\text{sign}}$, and the ledger verification function $\mathcal{L}.\mathsf{Verify}$, as follows:

$$L_{\text{EWE}} = \left\{ \mathsf{meta} \;\middle|\; \exists w, (t, \mathsf{info}, \pi_{\mathsf{publish}}) \text{ s.t. } \begin{array}{l} w = (\hat{w}, \sigma), \mathcal{L}.\mathsf{Verify}((\ell \| \mathsf{info}), \pi_{\mathsf{publish}}) = 1, \\ \mathsf{info} = t(w), \Pi_{\text{Sign}}.\mathsf{Verify}(\mathsf{pk}_{\text{sign}}, \hat{w}, \sigma) = 1, \\ p(\hat{w}) = 1, \theta(\hat{w}, \mathsf{meta}) = 1 \end{array} \right\}$$

Intuitively, these ciphertexts can only be decrypted by law enforcement once they have performed all the accountability tasks required by the ARLEAS.

Our construction also makes use of a simulation-extractable NIZK scheme Π_{NIZK} satisfying

Definition 4. We will prove statements in the following languages:

$$L^1_{\text{NIZK}} = \left\{ (c_1, c_2, \mathsf{pk}, \mathsf{meta}) \middle| \exists (r, r_1, r_2) \text{ s.t. } \begin{array}{l} c_1 = \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}, r; r_1) \wedge \\ c_2 = \Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r; r_2) \end{array} \right\}$$

$$L^2_{\text{NIZK}} = \left\{ (\mathsf{info}, \mathsf{pk}_{\text{sign}}) \middle| \exists (\hat{w}, \sigma) \text{ s.t. } \begin{array}{l} \Pi_{\text{Sign}}.\mathsf{Verify}(\mathsf{pk}_{\text{sign}}, \hat{w}, \sigma) = 1 \wedge \\ x \leftarrow t(w) \end{array} \right\}$$

We will describe our protocol in a hybrid model that makes use of several functionalities. These include $\mathcal{L}$, $\mathcal{F}^{\mathcal{D}}_{CRS}$, $\mathcal{G}_{\mathsf{pRO}}$ and $\mathcal{F}_{AUTH}$.

5.6.1 UC-Realizing $\mathcal{F}^{v,t,p,\theta,\mathsf{ret}}_{\mathbf{ARLEAS}}$

We now provide a description of the retrospective ARLEAS protocol $\pi^{v,t,p,\theta}_{\text{RET}}$.

$\pi^{v,t,p,\theta}_{\text{RET}}.\mathsf{Setup}$:

- All users send (CRS) to $\mathcal{F}^{\Pi_{\text{NIZK}}.\mathsf{ZKSetup}}_{CRS}$ to retrieve the common reference string for the NIZK scheme.
- P_J computes $(\mathsf{pk}_{\text{sign}}, \mathsf{sk}_{\text{sign}}) \leftarrow \Pi_{\text{Sign}}.\mathsf{KeyGen}(1^\lambda)$ and sends $\mathsf{pk}_{\text{sign}}$ to all other users via $\mathcal{F}_{AUTH}$.

$\pi^{v,t,p,\theta}_{\text{RET}}.\mathsf{SendMessage}$:

- The sender P_i computes the ciphertext $(c_1, c_2, c_3, \pi, \mathsf{meta})$ as follows, and sends it to P_j and P_{LE} via $\mathcal{F}_{AUTH}$:
  - Sample $r \leftarrow \{0,1\}^\lambda$
  - Query the random oracle to obtain the hashes:

    $(\mathsf{HashConfirm}, r_1) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“ENC”} \| r \| m))$,

    $(\mathsf{HashConfirm}, r_2) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“WE”} \| r \| m))$, and

    $(\mathsf{HashConfirm}, r_3) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“RP”} \| r))$
  - $c_1 \leftarrow \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}, r; r_1)$, $c_2 \leftarrow \Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r; r_2)$, and $c_3 \leftarrow m \oplus r_3$

– Use $\Pi_{\text{NIZK}}.\mathsf{ZKProve}$ to compute

$$\pi \leftarrow NIZK\{(r, r_1, r_2) : c_1 = \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}_j, r; r_1) \wedge c_2 = \Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r; r_2)\}$$

- Upon receiving (send, c), P_j performs the following steps:
  - Call $\pi_{\text{RET}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$ on c, aborting if the output is 0;
  - Compute $r' \leftarrow \Pi_{\text{Enc}}.\mathsf{Dec}(\mathsf{sk}_j, c_1)$
  - $(\mathsf{HashConfirm}, r_3) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“RP”} \| r'))$
  - Compute $m' \leftarrow c_3 \oplus r_3$
  - $(\mathsf{HashConfirm}, r_1) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“ENC”} \| r' \| m'))$
  - $(\mathsf{HashConfirm}, r_2) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“WE”} \| r' \| m'))$
  - Then to verify that the message has not been mauled, P_j recomputes $c_1' \leftarrow \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}_j, r'; r_1)$ and $c_2' \leftarrow \Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r'; r_2)$. If $c_1 \neq c_1'$ or $c_2 \neq c_2'$, return $\perp$. Otherwise, return m'.
- Upon receiving (send, c), P_{LE} calls $\pi_{\text{RET}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$ on c, aborting if the output is 0, and then calls $\pi_{\text{RET}}^{v,t,p,\theta}.\mathsf{AccessMessage}$ on c.

$\pi_{\text{RET}}^{v,t,p,\theta}.\mathsf{VerifyMessage}$:

- Any party parses $(c_1, c_2, c_3, \pi, \mathsf{meta}) \leftarrow c$ and verifies that π is correct and computes $v(\mathsf{meta}, aux)$, aborting if the output is 0. Otherwise, output 1.

$\pi_{\text{RET}}^{v,t,p,\theta}.\mathsf{RequestWarrant}$:

- P_{LE} sends $(\mathsf{RequestWarrant}, \hat{w})$ to P_J via $\mathcal{F}_{AUTH}$. P_J then either decides to send $(\mathsf{Disapprove})$ to P_{LE} and halt or executes the following:
  - Verify that $p(\hat{w}) = 1$. If not send $(\mathsf{Disapprove})$ to P_{LE} and abort.
  - $\sigma \leftarrow \Pi_{\text{Sign}}.\mathsf{Sign}(wsk, \hat{w})$

– Send the signed warrant $w = (\hat{w}, \sigma)$ to P_{LE} via $\mathcal{F}_{AUTH}$.

$\pi_{\text{RET}}^{v,t,p,\theta}$.ActivateWarrant:

- P_{LE} computes $\mathsf{info} \leftarrow t(w)$; uses $\Pi_{\text{NIZK}}.\mathsf{ZKProve}$ to compute

$$\pi \leftarrow \mathit{NIZK}\{(w) : w = (\hat{w}, \sigma), \Pi_{\text{Sign}}.\mathsf{Verify}(\mathsf{pk}_{\text{sign}}, \hat{w}, \sigma) = 1 \wedge \mathsf{info} \leftarrow t(w)\};$$

and sends $(\mathsf{Post}, (\mathsf{info}, \pi))$ to $\mathcal{L}^{\mathsf{Verify}}$. It receives and returns $(\ell, \mathsf{info}, \pi_{\mathsf{publish}})$.

$\pi_{\text{RET}}^{v,t,p,\theta}$.VerifyWarrantStatus:

- P_{LE} calls $\Pi_{\text{EWE}}.\mathsf{Dec}(c_2, \mathsf{meta}, (\hat{w}, \sigma), (\ell, \mathsf{info}, \pi_{\mathsf{publish}}))$. If the output is $\perp$, return 0. Otherwise, return 1.

$\pi_{\text{RET}}^{v,t,p,\theta}$.AccessMessage:

- P_{LE} computes $r' \leftarrow \Pi_{\text{EWE}}.\mathsf{Dec}(c_2, \mathsf{meta}, (\hat{w}, \sigma), (\ell, \mathsf{info}, \pi_{\mathsf{publish}}))$.
- $(\mathsf{HashConfirm}, r_3) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“RP”} \| r'))$
- Recovers $m' \leftarrow c_3 \oplus r_3$.
- $(\mathsf{HashConfirm}, r_1) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“ENC”} \| r' \| m'))$
- $(\mathsf{HashConfirm}, r_2) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“WE”} \| r' \| m'))$
- Recomputes $c_1' \leftarrow \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}_j, r'; r_1)$ and $c_2' \leftarrow \Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r'; r_2)$. If $c_1' = c_1$ and $c_2' = c_2$, P_{LE} returns m' and $\perp$ otherwise.

Theorem 9 *Assuming a CCA-secure public key encryption scheme Π_{Enc}, an extractable witness encryption scheme for L_{EWE}, a SUF-CMA secure signature scheme Π_{Sign}, and a simulation-extractable NIZK scheme Π_{NIZK}, $\pi_{RET}^{v,t,p,\theta}$ UC-realizes $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{ret}}$ in the $\mathcal{L}^{\mathsf{Verify}}$, $\mathcal{F}_{CRS}^{\Pi_{NIZK}.\mathsf{ZKSetup}}, \mathcal{G}_{\mathsf{pRO}}-$hybrid model.*

Proof. As in the prospective case in Section 5.5, we show that $\pi_{\text{RET}}^{v,t,p,\theta}$ UC-realizes $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\text{ret}}$ in a series of steps. First, we prove this for a single corrupt user, then extend that argument to multiple users. We then expand our analysis to consider corrupted law enforcement and corrupted judges.

P_i is corrupted. We begin with the simple case of a single corrupted user P_i controlled by an adversary $\mathcal{A}$. We construct a simulator $\mathcal{S}$ to mediate $\mathcal{A}$'s interaction with the ideal functionality as follows.

1. $\mathcal{S}$ begins by generating $(\mathsf{pk}_{\text{sign}}, \mathsf{sk}_{\text{sign}})$ as P_J would do. $\mathcal{S}$ then performs key generation for each honest party P_j. $\mathcal{S}$ then outputs all the public information to $\mathcal{A}$. Finally, $\mathcal{S}$ receives pk_i from $\mathcal{A}$.

2. When receiving a (CRS) request from $\mathcal{A}$, $\mathcal{S}$ generates $(\mathsf{CRS}_{ZK}, \tau) \leftarrow \Pi_{\text{NIZK}}.\mathsf{ZKSetup}(1^\lambda)$ and returns CRS_{ZK} to $\mathcal{A}$ and keeps τ.

3. Whenever $\mathcal{S}$ receives (send, c) from P_i intended for P_j, $\mathcal{S}$ parses $c = (c_1, c_2, c_3, \pi, \mathsf{meta})$ and verifies π. If it does not verify, set $b \leftarrow 1$, and set $b \leftarrow 0$ otherwise. Next, $(r, r_1, r_2) \leftarrow \Pi_{\text{NIZK}}.\mathsf{Extract}(\mathsf{CRS}_{ZK}, \tau, x, \pi)$, queries the random oracle $(\mathsf{HashConfirm}, r_3) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“RP”} \| r))$. It then computes $m \leftarrow c_3 \oplus r_3$ and verifies the structure of c_1, c_2 by re-computing the ciphertexts c_1, c_2 as in the real protocol. If some passes do not check, set $b \leftarrow 0$. Finally, it sends $(\mathsf{SendMessage}, \mathsf{sid}, P_j, m, b)$ to the ideal functionality.

4. When $\mathcal{S}$ receives $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m)$ from the ideal functionality destined for P_i, it identifies the public key for P_i and computes the ciphertext $c = (c_1, c_2, c_3, \pi, \mathsf{meta})$ as in the real protocol. It sends the resulting ciphertext to P_i.

5. When $\mathcal{S}$ receives $(\mathsf{NotifyWarrant}, t(w))$ from the ideal functionality, it simulates the proof π, and sends $(\mathsf{Post}, (t(w), \pi))$ to $\mathcal{L}$.

We prove that $\mathcal{A}$'s interaction with the real protocol and the ideal functionality, mediated by $\mathcal{S}$ are computationally indistinguishable by using the following hybrids.

Let $\mathcal{H}_0$ denote the distribution of the view of $\mathcal{A}$ in the real world interaction.

$\mathcal{H}_1$: Let $\mathcal{H}_1$ be the same as $\mathcal{H}_0$, but instead of having the common reference string generated by the $\mathcal{F}_{CRS}^{\Pi_{\text{NIZK}}.\mathsf{ZKSetup}}$, the common reference string is generated using $(\mathsf{crs}, \tau) \leftarrow \Pi_{\text{NIZK}}.\mathsf{ZKSetup}(1^\lambda)$. Note that the common reference string is selected from exactly the same distribution, therefore, the distribution of the view of $\mathcal{A}$ between $\mathcal{H}_0$ and $\mathcal{H}_1$ is the same.

$\mathcal{H}_2$: In this hybrid we will use the extractor $\Pi_{\text{NIZK}}.\mathsf{Extract}$ to get the randomness and create the ciphertext as in step 3. $\mathcal{H}_1$ and $\mathcal{H}_2$ are computationally indistinguishable as $\mathsf{Extract}$ will only fail with negligible probability.

$\mathcal{H}_3$: At this point we will simulate the proof π when publishing on the ledger. $\mathcal{H}_2$ and $\mathcal{H}_3$ are indistinguishable because of the zero knowledge property of Π_{NIZK}.

The view of the adversary in $\mathcal{H}_3$ is the same as its view when talking with $\mathcal{S}$ in the ideal world, which concludes the hybrid argument.

This argument extends to multiple parties as all ciphertexts can be properly simulated by either extracting from the NIZK or actually receiving the message in case the receiver is corrupted.

Subset of users and P_{LE} are corrupted. We now extend $\mathcal{S}$ from above to handle a corrupt P_{LE}. Note that step 5 of the above description is not longer relevant, as notifications will come directly from the ledger for the corrupt parties.

1. $\mathcal{S}$ runs the same setup as above. Additionally, $\mathcal{S}$ initializes an empty message equivocation table T.

2. When receiving a (CRS) request from $\mathcal{A}$, $\mathcal{S}$ generates $(\mathsf{CRS}_{ZK}, \tau) \leftarrow \Pi_{\text{NIZK}}.\mathsf{ZKSetup}(1^\lambda)$ and returns CRS_{ZK} to $\mathcal{A}$ and keeps τ.

3. We split the case of receiving $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta})$ from $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{ret}}$ intended for P_{LE} into two cases:

   (a) If P_j is not corrupted, $\mathcal{S}$ samples $r, r_1, r_2, r_3 \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$ (where $\mathsf{poly}(\cdot)$ is a polynomial function that upper-bounds the longest random string necessary) and computes the ciphertext components as follows

   - $c_1 \leftarrow \Pi_{\text{Enc}}.\mathsf{Enc}(pk_j, r; r_1)$
   - $c_2 \leftarrow \Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r; r_2)$
   - $c_3 \leftarrow r_3$
   - Use $\Pi_{\text{NIZK}}.\mathsf{ZKProve}$ to compute

$$\pi \leftarrow NIZK\left\{(r, r_1, r_2) \;\middle|\; \begin{array}{c} c_1 = \Pi_{\text{Enc}}.\mathsf{Enc}(\mathsf{pk}, r; r_1) \wedge \\ c_2 = \Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r; r_2) \end{array}\right\}$$

   $\mathcal{S}$ adds an entry $(r, r_1, r_2, r_3, c_1, c_2, c_3, \pi, \mathsf{meta})$ to T.

   (b) If P_i or P_j is corrupted, $\mathcal{S}$ will also receive $(\mathsf{Sent}, \mathsf{sid}, \mathsf{meta}, c, m)$ from the ideal functionality, intended for P_j. $\mathcal{S}$ then encrypts m using $\pi_{\text{RET}}^{v,t,p,\theta}.\mathsf{SendMessage}$, programming the random oracle honestly as needed.

   $\mathcal{S}$ then sends the resulting ciphertext to both the ideal functionality and $\mathcal{A}$.

4. Upon receiving $(\mathsf{Sent}, \mathsf{meta}, c, 0)$ from $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{ret}}$ intended for P_{LE}, $\mathcal{S}$ samples a random message and creates a ciphertext with $\pi_{\text{RET}}^{v,t,p,\theta}.\mathsf{SendMessage}$. Then, it generates a false proof instead of the real proof.

5. When $\mathcal{S}$ receives $(\mathsf{RequestWarrant}, \hat{w})$ from $\mathcal{A}$, intended for P_J, $\mathcal{S}$ sends $(\mathsf{RequestWarrant}, \hat{w})$ to the trusted party. If the trusted party responds with $\perp$, $\mathcal{S}$ sends $(\mathsf{Disapprove})$ to

$\mathcal{A}$. Otherwise, $\mathcal{S}$ generates a warrant $\hat{w}$ and signs it with $\sigma = \Pi_{\text{Sign}}.\mathsf{Sign}(\mathsf{sk}_{\text{sign}}, \hat{w})$ and sends $w = (\hat{w}, \sigma)$ to $\mathcal{A}$.

6. When $\mathcal{S}$ notices new posts on $\mathcal{L}$ it retrieves that information by sending $t \leftarrow (\mathsf{GetCounter})$ and $(t(w), \pi)) \leftarrow (\mathsf{GetVal}, t)$ to $\mathcal{L}$. $\mathcal{S}$ uses τ to extract w from π. Without loss of generality, let w be for user P_i. $\mathcal{S}$ sends $(\mathsf{ActivateWarrant}, w, P_j)$ to the ideal functionality. Next, $\mathcal{S}$ checks to see if there is an entry $(r, r_1, r_2, r_3, c_1, c_2, c_3, \pi, \mathsf{meta})$ for which $\theta(\hat{w}, \mathsf{meta}) = 1$ in T and sends $(\mathsf{AccessData}, (c_1, c_2, c_3, \pi), w)$ to the ideal functionality for all such records. If the ideal functionality responds with $\perp$, abort. Otherwise, the ideal functionality will return a message m. $\mathcal{S}$ then programs the random oracle by sending $(\mathsf{ProgramRO}, r, (r_3 \oplus m))$, $(\mathsf{ProgramRO}, (\text{“WE”}r \| m), r_2)$ and $(\mathsf{ProgramRO}, (\text{“ENC”} \| r \| m), r_1)$, and responds to the initial query.

To show that the simulation above is computationally indistinguishable from the real experiment in the view of $\mathcal{A}$, we proceed with a hybrid argument. Let $\mathcal{H}_0$ denote the distribution of the view of $\mathcal{A}$ in the real world interaction.

$\mathcal{H}_\mathbf{1}$: Let $\mathcal{H}_1$ be the same as $\mathcal{H}_0$, but instead of having the common reference string generated by the $\mathcal{F}_{CRS}^{\Pi_{\text{NIZK}}.\mathsf{ZKSetup}}$, the common reference string is generated using $(\mathsf{crs}, \tau) \leftarrow \Pi_{\text{NIZK}}.\mathsf{ZKSetup}(1^\lambda)$. Note that the common reference string is selected from exactly the same distribution, therefore, the distribution of the view of $\mathcal{A}$ between $\mathcal{H}_0$ and $\mathcal{H}_1$ is the same.

$\mathcal{H}_\mathbf{2}$: In this hybrid we change the NIZK proof for L^1_{NIZK} to be a simulation. Because of the zero knowledge property of Π_{NIZK}, $\mathcal{H}_2$ is statistically close to $\mathcal{H}_3$.

$\mathcal{H}_\mathbf{3}$: Choose r_1, r_2, and r_3 uniformly at random in $\{0,1\}^\lambda$ to replace the randomness used to compute c_1, c_2, and c_3 respectively. Also, send $(\mathsf{ProgramRO}, (\text{“ENC”} \| r \| m), r_1)$, $(\mathsf{ProgramRO}, (\text{“WE”} \| r \| m), r_2)$, and $(\mathsf{ProgramRO}, r, r_3)$ to $\mathcal{G}_{\mathsf{pRO}}$. Clearly the only way the view between $\mathcal{H}_1$ and $\mathcal{H}_2$ can differ is when $\mathcal{G}_{\mathsf{pRO}}$ was queried on these inputs before $\mathcal{S}$ programs them, in which case the protocol aborts, but this only happens with negligible

probability which we prove in Lemma 8.

$\mathcal{H}_4$: Change the simulated NIZK proof back to a real proof. Again, by the zero-knowledge property $\mathcal{H}_3$ and $\mathcal{H}_4$ are indistinguishable.

$\mathcal{H}_5$: Now, do everything as described in step 5 to recover the message m. Next, we change the ciphertext c_3 in step 3 to be r_3 and change the programming of $\mathcal{G}_{\mathsf{pRO}}$ to be $(\mathsf{ProgramRO}, r, m \oplus r_3)$ in step 5. Note that by security of the one-time pad $\mathcal{H}_4$ and $\mathcal{H}_5$ are indistinguishable.

Now, the view of the adversary in $\mathcal{H}_5$ is exactly the same as its view when talking with $\mathcal{S}$ in the ideal world, which concludes the hybrid argument.

□

Lemma 8 *For any adversary $\mathcal{A}$ against $\mathcal{H}_4$ in the security proof of $\pi_{RET}^{v,t,p,\theta}$ where P_{LE} is compromised, the probability that $\mathcal{A}$ queries or programs $\mathcal{G}_{\mathsf{pRO}}$ on r, "ENC"$\|r\|m$, or "WE"$\|r\|m$ without sending $(\mathsf{Post}, (t(w), \pi))$ to $\mathcal{L}$ is negligible, for $r \xleftarrow{\$} \{0,1\}^\lambda$ and m uniformly at random from the message space, both used inside $\mathcal{H}_4$.*

Proof. Assume such adversary $\mathcal{A}$ exists, then if $\mathcal{A}$ has queried or programmed $\mathcal{G}_{\mathsf{pRO}}$ on one of the inputs it must have had r. Either $\mathcal{A}$ has selected r by accident which can only happen with probability $2^{-\lambda}$, or it has extracted it from the ciphertext, which we will now show with a series of hybrids can only happen with negligible probability.

$\mathcal{H}'_0$: This looks exactly the same as $\mathcal{H}_4$. The encryption is created in the following way:

- sample $r \leftarrow \{0,1\}^\lambda$
- $(\mathsf{HashConfirm}, r_1) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“ENC”}\|r\|m))$,
- $(\mathsf{HashConfirm}, r_2) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“WE”}\|r\|m))$, and
- $(\mathsf{HashConfirm}, r_3) \leftarrow \mathcal{G}_{\mathsf{pRO}}(\mathsf{HashQuery}, (\text{“RP”}\|r))$

- Create the components or the encryption in the following way:
  - $c_1 \leftarrow \Pi_{\text{Enc}}.\mathsf{Enc}(pk_j, r; r_1)$
  - $c_2 \leftarrow \Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r; r_2)$
  - $c_3 \leftarrow m \oplus r_3$
  - Use $\Pi_{\text{NIZK}}.\mathsf{ZKSimulate}$ to simulate the proof

$\mathcal{H}'_{\mathbf{1}}$: Replace all three calls to $\mathcal{G}_{\mathsf{pRO}}$ with uniform random values r_1, r_2, r_3. By the properties of $\mathcal{G}_{\mathsf{pRO}}$ the view of $\mathcal{A}$ doesn't change between $\mathcal{H}'_0$ and $\mathcal{H}'_1$. Note that we only care about the view of $\mathcal{A}$ up until it calls $\mathcal{G}_{\mathsf{pRO}}$ on one of the three values, so we don't have to program $\mathcal{G}_{\mathsf{pRO}}$ accordingly.

$\mathcal{H}'_{\mathbf{2}}$: Now we replace c_1 by $\Pi_{\text{Enc}}.\mathsf{Enc}(pk_j, r'; r_1)$, which is possible because the proof is simulated. By the security of Π_{Enc} the view of the adversary doesn't change.

$\mathcal{H}'_{\mathbf{3}}$: Next, we replace c_2 with $\Pi_{\text{EWE}}.\mathsf{Enc}(\mathsf{meta}, r'; r_2)$. If there would be a distinguisher for $\mathcal{H}'_2$ and $\mathcal{H}'_3$ we can build a distinguisher for Π_{EWE}, by the extractable security of Π_{EWE} we can now extract a witness, but this is in contradiction with the ideal functionality $\mathcal{G}_{\mathsf{ledger}}$, as we assumed it was not called. Therefore, $\mathcal{H}'_2$ and $\mathcal{H}'_3$ must be indistinguishable.

We see that $\mathcal{H}'_3$ does not contain any reference to r, and $\mathcal{A}$ would not have changed its strategy as the difference between its view in $\mathcal{H}'_0$ and $\mathcal{H}'_3$ is computationally indistinguishable. □

Subset of users, P_J, and $P_{\mathbf{LE}}$ are corrupted. We further extend our analysis to cover both corrupt law enforcement and corrupt judges. Notice that step 4 of the previous simulator description is no longer relevant, as these requests are handled inside $\mathcal{A}$. The proof is exactly the same is in the previous case.

5.7 On the Need for Extractable Witness Encryption

The retrospective solution we present in Section 5.6 relies on extractable witness encryption. Intuitively, this strong assumption is required in our construction because a user must encrypt in a way that decryption is only possible under certain circumstances. Because the description of these circumstances can be phrased as an NP relation, witness encryption represents a "natural" primitive for realizing it. However, thus far we have not shown that the use of extractable witness encryption is *strictly* necessary. Given the strength (and implausibility [89]) of the primitive, it is important to justify its use. We do this by showing that any protocol Π_{A} that UC-realizes $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{ret}}$ implies the existence of extractable witness encryption for a related language. Notice that this does not mean the existence of a particular ARLEAS instantiation implies the existence of generic extractable witness encryption scheme, but rather a specific, non-trivial scheme.

Before proceeding to formally define this related language, we give some intuition about its form. We wish to argue that a protocol Π_{A} *acts like* an extractable witness encryption scheme in the specific case where an adversary has corrupted the escrow authorities P_{LE} and P_J (along with an arbitrary number of unrelated users). Recall that in order to learn any information about a message sent in Π_{A}, the following conditions must be met: specifically, law enforcement must correctly run the protocol for $\Pi_{\text{A}}.\mathsf{RequestWarrant}$ and $\Pi_{\text{A}}.\mathsf{ActivateWarrant}$ such that if $\Pi_{\text{A}}.\mathsf{VerifyWarrantStatus}$ were to be called, it would output 1.[19] For the protocol we presented in Section 5.6, this corresponds to obtaining a correct proof of publication from the ledger. Importantly, it must be impossible for law enforcement and judges to generate this information independently; if it were possible, it would be easy for these parties to circumvent the accountability mechanism.

We give a formal definition of this language $\mathcal{L}$ below. We denote the view of a user P_i as

[19] As specified in the ideal functionality, during verification it will be checked that a warrant was properly requested and activated.

$\mathcal{V}_{P_i}$, where this view is a collection of the views of running all algorithms that appear. We abuse notation slightly and denote the protocol transcript resulting from a sender P_S sending a message m to P_R as $\Pi_{\mathrm{A}}.\mathsf{SendMessage}(\cdot, P_S, P_R, m)$

$$\mathcal{L} = \left\{ (\mathsf{meta}, \mathsf{sid}) \middle| \exists \left(w, c, \begin{Bmatrix} \mathcal{V}_{P_{\mathrm{LE}}}, \mathcal{V}_{P_J}, \\ \mathcal{V}_{P_0}, \ldots, \mathcal{V}_{P_n} \end{Bmatrix} \right) \text{ s.t. } \begin{array}{c} c, \mathsf{meta} \leftarrow \Pi_{\mathrm{A}}.\mathsf{SendMessage}(\mathsf{sid}, P_S, P_R, m), \\ (\mathsf{Approve}) \leftarrow \Pi_{\mathrm{A}}.\mathsf{RequestWarrant}(\mathsf{sid}, w), \\ (\mathsf{NotifyWarrant}, t(w)) \leftarrow \Pi_{\mathrm{A}}.\mathsf{ActivateWarrant}(\mathsf{sid}, w), \\ 1 \leftarrow \Pi_{\mathrm{A}}.\mathsf{VerifyWarrantStatus}(\mathsf{sid}, w, \mathsf{meta}, c) \end{array} \right\}$$

In this language, the statement comprises some specified metadata and a valid instance of the protocol Π_{A} from the perspectives of the parties P_{LE}, P_J, and the users P_i without the sender and receiver. This setup specifies all the relevant components of the protocol (including the ledger functionality, in the case of the protocol presented in Section 5.6). The witness is a valid transcript starting with that setup, that includes the sending party sending a message with the appropriate metadata and concludes with a call to $\Pi_{\mathrm{A}}.\mathsf{VerifyWarrantStatus}$ that returns 1. Note that if $\mathsf{VerifyWarrantStatus}$ returns 1, then in the real protocol, $\mathsf{AccessMessage}$ would return the relevant plaintext. Unlike other common witness encryption languages, we note that all correctly sampled statements are trivially in the language and have multiple witnesses. Therefore, we need the strong notion of extractable witness encryption. As we will discuss, finding a witness for the statement remains a difficult task.

Consider the implications if it were computationally feasible for an adversary to generate a witness for an honestly sampled statement for $\mathcal{L}$. This would imply that an adversary corrupting P_{LE} and P_J interacting with the real protocol has a correct witness, which includes a call to $\mathsf{ActivateWarrant}$, this implies our accountability property. Such a protocol could never succeed in meeting our original goals; law enforcement would always be able to simulate the steps required for proper accountability. An accountability mechanism that can be locally simulated cannot guarantee that all parties can monitor the mechanism, undermining the purpose of the protocol.

To formalize this intuition, we begin by describing an extractable witness encryption scheme Π_{EWE} for language $\mathcal{L}$ given access to an ARLEAS protocol Π_{A}.

$\mathsf{Enc}(x, m)$ parses $(\mathsf{meta}, \mathsf{sid})$ from x and calls $\Pi_{\text{A}}.\mathsf{SendMessage}(\mathsf{sid}, m, P_S, P_R)$ such that it outputs meta, c. It then returns the views $\{\mathcal{V}_{P_{\text{LE}}}, \mathcal{V}_{P_J}, \mathcal{V}_{P_0}, \ldots, \mathcal{V}_{P_n}\}$ resulting from that run, excluding the private information associated with sending the message.

$\mathsf{Dec}(c, \omega)$ parses $c, w, \mathsf{meta}, \mathsf{sid}$ from c and ω, calls $m \leftarrow \Pi_{\text{A}}.\mathsf{AccessMessage}(\mathsf{sid}, w, \mathsf{meta}, c)$ and returns the result.

It is easy to see that this construction satisfies the correctness property of extractable witness encryption. Notice that a valid witness needs to contain inputs to VerifyWarrantStatus such that it outputs 1. Because VerifyWarrantStatus is defined to return 1 exactly when AccessMessage will return a message, the above decryption algorithm will return a message only with a valid witness.

We introduce the metadata in the statement in order to fix a witness to a particular statement. Note that our protocol generates an encryption as running part of the protocol, actually generating part of the witness. If metadata is not included in the statement, then *any* witness for a particular setup can be used to decrypt *any* ciphertext generated by the encryption oracle under the same statement. While this is not inherently problematic for extractable witness encryption, it no longer corresponds neatly to ARLEAS. Recall that warrants in ARLEAS specify the metadata for which they are relevant through the warrant scope check functionality $\theta(\cdot, \cdot)$ and this property must be enforced in the language. We now proceed to show that the above scheme Π_{EWE} satisfies extractable security if Π_{A} UC-realizes $\mathcal{F}_{\text{ARLEAS}}^{v,t,p,\theta,\mathsf{ret}}$.

Theorem 10 *Given a protocol Π_A that UC-realizes $\mathcal{F}_{ARLEAS}^{v,t,p,\theta,\mathsf{ret}}$, Π_{EWE} is a secure extractable witness encryption scheme for the language $\mathcal{L}$.*

Proof. Given an adversary $\mathcal{A}$ with non-negligible advantage in the extractable witness encryption game for language $\mathcal{L}$, either

1. We construct an extractor $\mathsf{Ext}_{\mathcal{A}}(1^{\lambda}, x, aux)$ by verifying if the adversary $\mathcal{A}$ ran

$$\Pi_{\mathrm{A}}.\mathsf{RequestWarrant}(\mathsf{sid}, w)$$

and $\Pi_{\mathrm{A}}.\mathsf{ActivateWarrant}(\mathsf{sid}, w)$ such that $\Pi_{\mathrm{A}}.\mathsf{VerifyWarrantStatus}(\mathsf{sid}, w, \mathsf{meta}, c) = 1$. If this was the case, the extractor would have all information to form a witness that it can output;

2. else, if such extractor does not exist, we construct a distinguisher $\mathcal{Z}$ that distinguishes between Π_{A} and ARLEAS ideal functionality. $\mathcal{Z}$ proceeds as follows

   (a) When $\mathcal{A}$ asks to sample a statement, $\mathcal{Z}$ instantiates Π_{A} with parties $\{P_{\mathrm{LE}}, P_J, P_0, \ldots, P_n, P_S, P_R\}$ on honest random coins. $\mathcal{Z}$ then generates some arbitrary metadata meta associated with a message that P_S could send in the future. and returns $\mathsf{meta}, \mathsf{sid}$ to $\mathcal{A}$.

   (b) When $\mathcal{A}$ sends the challenge plaintexts m_0, m_1 (such that $|m_0| = |m_1|$) on statement x, $\mathcal{Z}$ then flips a coin $b \xleftarrow{\$} \{0,1\}$, $\mathcal{Z}$ has P_S call

$$\Pi_{\mathrm{A}}.\mathsf{SendMessage}(\mathsf{sid}, m_b, P_S, P_R)$$

   such that it outputs c, meta. $\mathcal{Z}$ then returns the updated views of P_{LE}, P_J and the N other users to $\mathcal{A}$.

   (c) When $\mathcal{A}$ outputs the guess b' and halts, $\mathcal{Z}$ outputs $b' == b$, where 1 indicates the real world and 0 indicates the ideal world.

Note that in the ideal functionality, the joint views of law enforcement and the judge contain no information about the plaintext, because the ciphertext is chosen by the

ideal world adversary without access to the plaintext. As such, if the adversary is able to distinguish between messages with non-negligible probability, $\mathcal{Z}$ must be interacting with the real world protocol.

□

Implications For Practical Retrospective ARLEAS. The relationship between retrospective ARLEAS and extractable witness encryption is an indication of the difficulty of realizing retrospective ARLEAS in practice. In very specific cases, it may be possible to phrase certain existing encryption schemes as witness encryption schemes, for example some IBE schemes. General purpose extractable witness encryption, on the other hand, is considered implausible [89]. The extractable witness encryption language we have described above must reason over the ledger authentication language and the various functionalities that parameterize an retrospective ARLEAS system. As such, the difficulty of realizing a practical retrospective ARLEAS will hinge on the complexity of the ledger and the parameterizing functionalities. If they are centralized and simple, it may be possible to instantiate an retrospective ARLEAS using the protocol we provided in Section 5.6 and known encryption techniques. However, the security provided by a centralized ledger is not significant, as a compromised central authority could circumvent the accountability properties of the system. Thus, we believe that this result indicates that instantiating an retrospective ARLEAS with meaningful security is impractical with known techniques.

5.8 An ARLEAS' Parameterizing Functions in Practice

In the name of being generalizable, we have presented both our definition and protocols with a significant number of parameterized functions, making our construction very abstract. To better understand how an ARELAS might actually work in practice, we give possible choices for various parts of the system.

5.8.1 Service Providers

For simplicity, in the protocols and ideal functionalities we present in this work, there is no service provider. Users send ciphertexts *directly to law enforcement*. In practice, law enforcement does not directly operate a communication network, and thus a service provider is crucial. We do not give a formal treatment of the responsibilities of the service provider, as it is not central to our work, but assume that the service provider's role is similar to modern systems. This includes authenticating users, delivering messages, and verifying that messages are properly constructed. Moreover, we assume that the service provider is transparent to law enforcement, *i.e.* all requests for data will be approved.

5.8.2 Transparency Functionalities

Recall that when law enforcement wants to activate an issued warrant, an ARLEAS requires that they make some information, determined by the transparency function, about that warrant public. In practice, choosing a transparency function is a balance between robust accountability and operability (*i.e.* not tipping off individuals that they are under surveillance).

We briefly consider three possible transparency functions:

- **Counting Warrants.** A baseline transparency function would leak the *number* of warrants activated over time. Each time law enforcement activates a group of warrants, the transparency function counts these valid warrants and outputs this count. While the impact of this simple transparency function may seem limited, it provides a way for the government to detect key exfiltration. This can be done by observing when the number of warrants activated is greater than those known about.

- **Issuing Court.** Another possible transparency function would leak the identity of the court that issued the warrant. While this could potentially leak more information

about *where* an activated warrant is going to be used (*i.e.* if it was issued by a local court), it also provides significantly stronger protections and oversight. For instance, if the warrant signing keys of a particular court are compromised, a system with this transparency will quickly identify the problem and be able to re-key just that court.

- **Differential Private Analytics.** The above transparency functions do not leak any information about the *contents* of the warrant. As our final example transparency function, we consider a function that leaks differentially private [74, 75] information about activated warrants. The analytics could, for instance, include the racial identity of targeted individuals, so that civil liberties groups could monitor courts with a problematic history. Computing these differential private analytics could either be done with randomized response or by modifying the interface to the transparency function to take in *all* active warrants. Either way, the random coins used must be deterministically generated from the warrants, as a possibly adversarial law enforcement would be able to choose them otherwise. Additionally, special attention must be paid to the privacy budget if iterative analytics will be run over the same warrants.

5.8.3 Policy Functionalities

While the transparency function allows for *detection* of malicious behavior, the policy function *prevents* certain types of warrants altogether. As part of activating a warrant, law enforcement must prove (in zero-knowledge) that all the warrants being activated satisfy the policy function. Thus it will be impossible to *activate* warrants that are not compliant with the policy function, even if a malicious judge does issue such warrants.

We consider three possible choices of policy functions, which limit the space of possible valid warrants:

- **Warrants Must Specify Individual Targets.** In order to limit the risk of unfettered

surveillance, it may be prudent to require that warrants specify individuals, rather than an entire group. Note that this does not inherently limit the power of law enforcement, as a group can be monitored by simply issuing individual warrants for each member. However, it could prevent *unintentionally* issuing a warrant with an overly broad scope. To mitigate this risk, the system can be set up with a policy function that checks the structure of each warrant and ensures that it specifies only a single individual.

- **Warrants Must Have A Limited Time Scope.** It may be desirable to require all warrants to specify the length of time for which they are valid. Put another way, warrants should only apply to messages encrypted within a fixed time-frame. This could prevent unintended mission creep or reuse of old, stale warrants. As before, this does not affect the power of law enforcement (renewed warrants can be issued at will) but it does reduce the possibility for unintentionally using warrants after a case has been concluded.

- **Warrants Issued By Problematic Courts Must Be Subject To Additional Oversight.** Many law enforcement organizations in the United States have a problematic history of violating civil rights. In these cases, it is common to have Federal bodies oversee the actions of these law enforcement organizations. This oversight continues until the federal government is convinced that it is no longer warranted. This arrangement can be formalized by requiring warrants to bear the signature of the federal oversight body. Alternatively this role could be outsourced to a civil liberty group instead of a federal body.

5.8.4 Metadata and Warrant Scope Check Functionalities

Each message sent in an ARLEAS has associated metadata that is added by the sender. This metadata is information about the message that is *public* to the service provider and

law enforcement. In modern systems, some metadata may be added by the sender while other metadata may be added by the service provider relaying the message. We explicitly consider the following three kinds of metadata, but note that the space of potential metadata information could be significantly larger:

- **Sender/Receiver Identity.** A service provider is responsible for routing messages from a sender to a receiver. As such, it is important the service provider is able to know these identities so that it can fulfill this functionality. Warrants that targets specific users can use this information to determine if a message should be decryptable.

- **Timestamp.** Timestamp information is a common piece of information included with a message in modern systems. As mentioned above, including timestamp information in message metadata and in warrant scopes is an important way of ensuring that a warrant is only used to surveil the intended targets and not used surreptitiously after an investigation has finished.

- **Geolocation.** Geolocation information gives a message a physical origin and can be provided either by global positioning systems or by identifying the first piece of static networking equipment through which the message is transmitted (*e.g.* a cell tower). This information could be used by law enforcement to target groups moving through a specific location. For instance, law enforcement may have a strong justification that illicit activity is happening in a remote area, but are unable to identify *who* is visiting the area. Allowing warrants to reason over geolocation is a powerful investigative tool, but should be considered carefully as to not accidentally enable dragnet surveillance.

While we are not *overly* interested in designing a system secure against malicious senders (see Section 5.1.3), it would be preferable for a service provider to be able to drop messages that are flagrantly flaunting the system using the metadata verification function $v(\cdot, \cdot)$. This

function verified if metadata associated with a particular message is correctly. There are two main approaches for a metadata verification functionality. The first is a "common sense" approach: the service provider checks that the metadata supplied by the user is reasonable from the view of the service provider. A second approach would be to get *authenticated* metadata sources, where appropriate. For instance, existing GPS satellite messages could be modified to transmit a *cryptographically signed* version of their signal, and messages could include this in the metadata. Alternatively, a mobile device get an attestation from all connected cell towers proving their location. Neither approach is foolproof — colluding devices in different geolocations might be able to spoof a location — but they could raise the difficulty of circumventing the system.

Warrant Scope Check. In addition to being input to the metadata verification functionality, the metadata also is an input to the *warrant scope check* $\theta(\cdot, \cdot)$. This functionalities take in a warrant and some metadata, and decides if the associated message is within the scope of the supplied warrant. We do not specify how this check would work in practice, as it is tightly coupled to the format of warrants and metadata. A minimal implementation might be to split the warrant into a list of clauses or requirements and ensure the metadata satisfies each one.

Conclusions and general discussion

In this work we studied three types of imbalanced cryptographic protocols, namely, computational, communicational, and functional imbalanced cryptographic protocols. In all three categories we looked at some interesting protocols and their applications, more specifically, the possibilities of constructing these protocols in a practical setting. More than just looking into theoretical improvements, we have looked at concrete optimizations and implementations. In all cases practical efficiency was paramount.

In the category of computationally and communicationally imbalanced protocols, we looked at the area of laconic cryptography. The most famous example is laconic Oblivious Transfer, where the intent is to optimize both communication and computation over general Oblivious Transfer. We have looked at a novel approach to construct such laconic Oblivious Transfer protocol, leading to a more concretely efficient construction in comparison with previous work. We did a thorough analysis of our system to properly compare with related work.

Furthermore, we have looked at the most famous communicationally imbalanced protocol, the non-interactive zero knowledge proof. Instead of trying to optimize the primitive, we studied the very practical application of proving the existence of an exploit in a compiled binary for a real-world processor. To achieve this, we built an optimized version of the KKW protocol in combination with the techniques first outlined by Ben-Sasson *et al.*, as well as a complete toolchain to start with the binary and the exploit and create a full non-interactive

proof.

Finally, we looked into a functionally imbalanced protocol, by exploring backdoors in end-to-end encryption. In such protocols, law enforcement has a very different role in comparison with other parties in the system. They can only access the relevant messages if they fulfill certain requirements, whereas a normal recipient of the message should always be able to read the message. One of these requirements that law enforcement needs to fulfill is a form of abuse resistance. We split the definition into two parts, prospective and retrospective, and show that the former case can be somewhat practically achieved, but the retrospective case can only be achieved by using extractable witness encryption, moreover, we show that extractable witness encryption is necessary for achieving this primitive.

We believe imbalanced cryptographic protocols to be a very active field of research on all different aspects, we have merely touched upon some interesting topics, protocols, and applications in this broad category. We have showed that most of these protocols are becoming very practical, albeit with some limitations. Nevertheless, we think this topic can be pushed further towards practical implementations that can be valuable in very different settings.

References

[1] Berkeley logic interchange format (blif). 1992.

[2] libsnark: a c++ library for zksnark proofs, 2012-2020. https://github.com/scipr-lab/libsnark.

[3] Microcorruption: Embedded security ctf, 2013. https://microcorruption.com.

[4] Circom: a circuit compiler for zksnarks, 2018. https://github.com/iden3/circom.

[5] Reverie: An efficient and generalized implementation of the ikos-style kkw proof system, 2022. https://github.com/trailofbits/reverie.

[6] Harold Abelson, Ross Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Michael A. Specter, and Daniel J. Weitzner. Keys under doormats: mandating insecurity by requiring government access to all data and communications. *Journal of Cybersecurity*, 1(1):69–79, 11 2015. ISSN 2057-2085. doi: 10.1093/cybsec/tyv009. URL https://doi.org/10.1093/cybsec/tyv009.

[7] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404. Springer, Heidelberg, May 2014. doi: 10.1007/978-3-642-55220-5_22.

[8] Shweta Agrawal and Razvan Roşie. Adaptively secure laconic function evaluation for NC^1. 2021.

[9] Navid Alamati, Pedro Branco, Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Sihang Pu. Laconic private set intersection and applications. Cryptology ePrint Archive, Report 2021/728, 2021. https://eprint.iacr.org/2021/728.

[10] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017. doi: 10.1145/3133956.3134104.

[11] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 443–458. IEEE Computer Society Press, May 2014. doi: 10.1109/SP.2014.35.

[12] Apple. icloud security overview. Available at https://support.apple.com/en-us/HT202303, . URL https://support.apple.com/en-us/HT202303.

[13] Apple. Facetime. Available at https://apps.apple.com/us/app/facetime/id1110145091, . URL https://apps.apple.com/us/app/facetime/id1110145091.

[14] Apple. imessage. Available at https://support.apple.com/explore/messages, . URL https://support.apple.com/explore/messages.

[15] Diego Aranha, Chuanwei Lin, Claudio Orlandi, and Mark Simkin. Laconic private set-intersection from pairings. Cryptology ePrint Archive, Report 2022/529, 2022. https://eprint.iacr.org/2022/529.

[16] David Archer, Victor Arribas Abril, Steve Lu, Pieter Maene, Nele Mertens, Danilo Sijacic, and Nigel Smart. 'bristol fashion' mpc circuits. https://homes.esat.kuleuven.be/~nsmart/MPC/, 2022.

[17] Michael Backes, Jan Camenisch, and Dieter Sommer. Anonymous yet accountable access control. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, WPES '05, pages 40–46, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595932283. doi: 10.1145/1102199.1102208. URL https://doi.org/10.1145/1102199.1102208.

[18] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 324–356. Springer, Heidelberg, August 2017. doi: 10.1007/978-3-319-63688-7_11.

[19] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 913–930. ACM Press, October 2018. doi: 10.1145/3243734.3243848.

[20] William Barr. Attorney General William P. Barr Delivers Keynote Address at the International Conference on Cyber Security. Available at https://www.justice.gov/opa/speech/attorney-general-william-p-barr-delivers-keynote-address-international-conference-cyber, July 2019.

[21] Adam M. Bates, Kevin R. B. Butler, Micah Sherr, Clay Shields, Patrick Traynor, and Dan S. Wallach. Accountable wiretapping -or- I know they can hear you now. In *NDSS 2012*. The Internet Society, February 2012.

[22] Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 495–526. Springer, Heidelberg, May 2020. doi: 10.1007/978-3-030-45374-9_17.

[23] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC*, pages 479–488. ACM Press, May 1996. doi: 10.1145/237814.237996.

[24] Mihir Bellare and Ronald L. Rivest. Translucent cryptography - an alternative to key escrow, and its implementation via fractional oblivious transfer. *Journal of Cryptology*, 12(2):117–139, March 1999. doi: 10.1007/PL00003819.

[25] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 1–35. Springer, Heidelberg, April 2009. doi:

10.1007/978-3-642-01001-9_1.

[26] Mihir Bellare, Eike Kiltz, Chris Peikert, and Brent Waters. Identity-based (lossy) trapdoor functions and applications. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 228–245. Springer, Heidelberg, April 2012. doi: 10.1007/978-3-642-29011-4_15.

[27] Steven M. Bellovin, Matt Blaze, Dan Boneh, Susan Landau, and Ronald R. Rivest. Analysis of the CLEAR protocol per the National Academies' framework. Technical Report CUCS-003-18, Columbia University, May 2018. URL https://mice.cs.columbia.edu/getTechreport.php?techreportID=1637.

[28] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 257–266. ACM Press, October 2008. doi: 10.1145/1455770.1455804.

[29] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems: extended abstract. In Robert D. Kleinberg, editor, *ITCS 2013*, pages 401–414. ACM, January 2013. doi: 10.1145/2422436.2422481.

[30] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Heidelberg, August 2013. doi: 10.1007/978-3-642-40084-1_6.

[31] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014. doi: 10.1109/SP.2014.36.

[32] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 781–796. USENIX Association, August 2014.

[33] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. https://eprint.iacr.org/2018/046.

[34] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 701–732. Springer, Heidelberg, August 2019. doi: 10.1007/978-3-030-26954-8_23.

[35] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019. doi: 10.1007/978-3-030-17653-2_4.

[36] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 421–439. Springer, Heidelberg, August 2014. doi: 10.1007/978-3-662-44381-1_24.

[37] Matt Blaze. Oblivious key escrow. In Ross Anderson, editor, *Information Hiding*, pages

335–343, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-49589-5.

[38] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 223–238. Springer, Heidelberg, May 2004. doi: 10.1007/978-3-540-24676-3_14.

[39] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001. doi: 10.1007/3-540-44647-8_13.

[40] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440–456. Springer, Heidelberg, May 2005. doi: 10.1007/11426639_26.

[41] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 258–275. Springer, Heidelberg, August 2005. doi: 10.1007/11535218_16.

[42] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 325–341. Springer, Heidelberg, February 2005. doi: 10.1007/978-3-540-30576-7_18.

[43] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011. doi: 10.1007/978-3-642-19571-6_16.

[44] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018. doi: 10.1007/978-3-319-96884-1_25.

[45] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune K. Jakobsen, and Mary Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 595–626. Springer, Heidelberg, December 2018. doi: 10.1007/978-3-030-03326-2_20.

[46] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 52–73. Springer, Heidelberg, February 2014. doi: 10.1007/978-3-642-54242-8_3.

[47] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous IBE, leakage resilience and circular security from new assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 535–564. Springer, Heidelberg, April / May 2018. doi: 10.1007/978-3-319-78381-9_20.

[48] Emmanuel Bresson, Dario Catalano, and David Pointcheval. A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications. In Chi-Sung Laih, editor, *ASIACRYPT 2003*, volume 2894 of *LNCS*, pages 37–54. Springer, Heidelberg, November / December 2003. doi: 10.1007/978-3-540-40061-5_3.

[49] Cassell Bryan-Low. Vodafone, Ericsson Get Hung Up In Greece's Phone-Tap Scandal. *The Wall Street Journal*, June 2006. URL https://www.wsj.com/articles/SB115085571895085969.

[50] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg

Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018. doi: 10.1109/SP.2018.00020.

[51] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 280–312. Springer, Heidelberg, April / May 2018. doi: 10.1007/978-3-319-78381-9_11.

[52] Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019. doi: 10.1145/3319535.3339820.

[53] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. doi: 10.1109/SFCS.2001.959888.

[54] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001. doi: 10.1007/3-540-44647-8_2.

[55] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002. doi: 10.1145/509907.509980.

[56] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 565–582. Springer, Heidelberg, August 2003. doi: 10.1007/978-3-540-45146-4_33.

[57] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1825–1842. ACM Press, October / November 2017. doi: 10.1145/3133956.3133997.

[58] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020. doi: 10.1007/978-3-030-45721-1_26.

[59] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 33–65. Springer, Heidelberg, August 2017. doi: 10.1007/978-3-319-63715-0_2.

[60] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 719–728. ACM Press, October / November 2017. doi: 10.1145/3133956.3134092.

[61] Arka Rai Choudhuri, Vipul Goyal, and Abhishek Jain. Founding secure computation on blockchains. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume

11477 of *LNCS*, pages 351–380. Springer, Heidelberg, May 2019. doi: 10.1007/978-3-030-17656-3_13.

[62] Kelong Cong, Karim Eldefrawy, and Nigel P. Smart. Optimizing registration based encryption. Cryptology ePrint Archive, Report 2021/499, 2021. https://ia.cr/2021/499.

[63] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, Heidelberg, April / May 2002. doi: 10.1007/3-540-46035-7_4.

[64] Ivan Damgård, Jesper Buus Nielsen, and Claudio Orlandi. Essentially optimal universally composable oblivious transfer. In Pil Joong Lee and Jung Hee Cheon, editors, *ICISC 08*, volume 5461 of *LNCS*, pages 318–335. Springer, Heidelberg, December 2009.

[65] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Heidelberg, April / May 2018. doi: 10.1007/978-3-319-78375-8_3.

[66] Cyprien de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy. Limbo: Efficient zero-knowledge MPCitH-based arguments. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 3022–3036. ACM Press, November 2021. doi: 10.1145/3460120.3484595.

[67] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Heidelberg, August 2001. doi: 10.1007/3-540-44647-8_33.

[68] Dorothy E. Denning. The US key escrow encryption technology. *Computer Communications*, 17(7):453–457, 1994. doi: 10.1016/0140-3664(94)90099-X. URL https://doi.org/10.1016/0140-3664(94)90099-X.

[69] Dorothy E Denning and Dennis K Branstad. A taxonomy for key escrow encryption systems. *Communications of the ACM*, 39(3):34–40, 1996.

[70] Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 537–569. Springer, Heidelberg, August 2017. doi: 10.1007/978-3-319-63688-7_18.

[71] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Daniel Masny. New constructions of identity-based and key-dependent message secure encryption schemes. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 3–31. Springer, Heidelberg, March 2018. doi: 10.1007/978-3-319-76578-5_1.

[72] Nico Döttling, Sanjam Garg, Vipul Goyal, and Giulio Malavolta. Laconic conditional disclosure of secrets and applications. In David Zuckerman, editor, *60th FOCS*, pages 661–685. IEEE Computer Society Press, November 2019. doi: 10.1109/FOCS.2019.00046.

[73] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2019. doi: 10.1007/978-3-030-26954-8_1.

[74] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[75] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 265–284. Springer, Heidelberg, March 2006. doi: 10.1007/11681878_14.

[76] EncroChat. Encrochat network. http://encrochat.network/.

[77] Encryption Working Group. Moving the Encryption Policy Conversation Forward. Technical report, Carnegie Endowment for International Peace, 2019. URL https://carnegieendowment.org/files/EWG__Encryption_Policy.pdf.

[78] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 823–852. Springer, Heidelberg, August 2020. doi: 10.1007/978-3-030-56880-1_29.

[79] Federal Bureau of Investigation. Going Dark. Available at https://www.fbi.gov/services/operational-technology/going-dark.

[80] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st FOCS*, pages 308–317. IEEE Computer Society Press, October 1990. doi: 10.1109/FSCS.1990.89549.

[81] Joan Feigenbaum and Daniel J Weitzner. On the incommensurability of laws and technical mechanisms: Or, what cryptography can't do. In *Cambridge International Workshop on Security Protocols*, pages 266–279. Springer, 2018.

[82] Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 480–491. Springer, Heidelberg, August 1994. doi: 10.1007/3-540-48329-2_40.

[83] Lorenzo Franceschi-Bicchierai. FBI Director: Encryption Will Lead to a 'Very Dark Place'. *Mashable*, October 2014. URL https://mashable.com/2014/10/16/fbi-director-encryption-going-dark-speech/.

[84] Jonathan Frankle, Sunoo Park, Daniel Shaar, Shafi Goldwasser, and Daniel J. Weitzner. Practical accountability of secret processes. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 657–674. USENIX Association, August 2018.

[85] Nicholas Franzese, Jonathan Katz, Steve Lu, Rafail Ostrovsky, Xiao Wang, and Chenkai Weng. Constant-overhead zero-knowledge for RAM programs. Cryptology ePrint Archive, Report 2021/979, 2021. https://eprint.iacr.org/2021/979.

[86] David Mandell Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 44–61. Springer, Heidelberg, May / June 2010. doi: 10.1007/978-3-642-13190-5_3.

[87] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

[88] Sanjam Garg, Rafail Ostrovsky, Ivan Visconti, and Akshay Wadia. Resettable statistical zero knowledge. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 494–511.

Springer, Heidelberg, March 2012. doi: 10.1007/978-3-642-28914-9_28.

[89] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 518–535. Springer, Heidelberg, August 2014. doi: 10.1007/978-3-662-44371-2_29.

[90] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 689–718. Springer, Heidelberg, November 2018. doi: 10.1007/978-3-030-03807-6_25.

[91] Sanjam Garg, Rafail Ostrovsky, and Akshayaram Srinivasan. Adaptive garbled RAM from laconic oblivious transfer. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 515–544. Springer, Heidelberg, August 2018. doi: 10.1007/978-3-319-96878-0_18.

[92] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 63–93. Springer, Heidelberg, April 2019. doi: 10.1007/978-3-030-17259-6_3.

[93] Peter Gazi, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *2019 IEEE Symposium on Security and Privacy*, pages 139–156. IEEE Computer Society Press, May 2019. doi: 10.1109/SP.2019.00040.

[94] Craig Gentry and Brent Waters. Adaptive security in broadcast encryption systems (with short ciphertexts). In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 171–188. Springer, Heidelberg, April 2009. doi: 10.1007/978-3-642-01001-9_10.

[95] Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 426–443. Springer, Heidelberg, August 2014. doi: 10.1007/978-3-662-44371-2_24.

[96] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 1069–1083. USENIX Association, August 2016.

[97] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1):169–192, February 1996. ISSN 0097-5397. doi: 10.1137/S009753979 1220688. URL https://doi.org/10.1137/S0097539791220688.

[98] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, December 1994. doi: 10.1007/BF00195207.

[99] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th FOCS*, pages 174–187. IEEE Computer Society Press, October 1986. doi: 10.1109/SFCS.1986.47.

[100] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 171–185. Springer, Heidelberg, August 1987. doi: 10.1007/3-540-47721-7_11.

[101] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987. doi: 10.1145/28395.28420.

[102] Shafi Goldwasser and Sunoo Park. Public accountability vs. secret laws: Can they coexist? a cryptographic proposal. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*, WPES '17, pages 99–110, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450351751. doi: 10.1145/3139550.3139565. URL https://doi.org/10.1145/3139550.3139565.

[103] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 39–56. Springer, Heidelberg, August 2008. doi: 10.1007/978-3-540-85174-5_3.

[104] Google. Encrypt your data - pixel phone help. Available at https://support.google.com/pixelphone/answer/2844831?hl=en. URL https://support.google.com/pixelphone/answer/2844831?hl=en.

[105] Siobhan Gorman. NSA Officers Spy on Love Interests. *The Wall Street Journal*, August 2013. URL https://blogs.wsj.com/washwire/2013/08/23/nsa-officers-sometimes-spy-on-love-interests/.

[106] Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 529–561. Springer, Heidelberg, November 2017. doi: 10.1007/978-3-319-70500-2_18.

[107] Rishab Goyal and Satyanarayana Vusirikala. Verifiable registration-based encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 621–651. Springer, Heidelberg, August 2020. doi: 10.1007/978-3-030-56784-2_21.

[108] Rishab Goyal, Satyanarayana Vusirikala, and Brent Waters. New constructions of hinting PRGs, OWFs with encryption, and more. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 527–558. Springer, Heidelberg, August 2020. doi: 10.1007/978-3-030-56784-2_18.

[109] Sen. Lindsey Graham. Eliminating abusive and rampant neglect of interactive technologies act of 2020. https://www.congress.gov/bill/116th-congress/senate-bill/3398/text, March 2020.

[110] Matthew Green, Gabriel Kaptchuk, and Gijs Van Laer. Abuse resistant law enforcement access systems. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 553–583. Springer, Heidelberg, October 2021. doi: 10.1007/978-3-030-77883-5_19.

[111] Matthew Green, Mathias Hall-Andersen, Eric Hennenfent, Gabriel Kaptchuk, Benjamin Perez, and Gijs Van Laer. Efficient proofs of software exploitability for real-world processors. Cryptology ePrint Archive, Paper 2022/1223, 2022. URL https://eprint.iacr.org/2022/1223.

[112] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016. doi: 10.1007/978-3-662-49896-5_11.

[113] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, Heidelberg, August 2017. doi: 10.1007/978-3-319-63715-0_20.

[114] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 97–111. Springer, Heidelberg, August 2006. doi: 10.1007/11818175_6.

[115] David Heath and Vladimir Kolesnikov. A 2.1 KHz zero-knowledge processor with BubbleRAM. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2055–2074. ACM Press, November 2020. doi: 10.1145/3372297.3417283.

[116] David Heath and Vladimir Kolesnikov. Stacked garbling for disjunctive zero-knowledge proofs. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 569–598. Springer, Heidelberg, May 2020. doi: 10.1007/978-3-030-45727-3_19.

[117] David Heath, Yibin Yang, David Devecsery, and Vladimir Kolesnikov. Zero knowledge for everything and everyone: Fast ZK processor with cached ORAM for ANSI C programs. In *2021 IEEE Symposium on Security and Privacy*, pages 1538–1556. IEEE Computer Society Press, May 2021. doi: 10.1109/SP40001.2021.00089.

[118] Brett Hemenway, Benoît Libert, Rafail Ostrovsky, and Damien Vergnaud. Lossy encryption: Constructions from general assumptions and efficient selective opening chosen ciphertext security. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 70–88. Springer, Heidelberg, December 2011. doi: 10.1007/978-3-642-25385-0_4.

[119] Dennis Hofheinz. All-but-many lossy trapdoor functions. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 209–227. Springer, Heidelberg, April 2012. doi: 10.1007/978-3-642-29011-4_14.

[120] Thibaut Horel, Sunoo Park, Silas Richelson, and Vinod Vaikuntanathan. How to subvert backdoored encryption: Security against adversaries that decrypt all ciphertexts. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 42:1–42:20. LIPIcs, January 2019. doi: 10.4230/LIPIcs.ITCS.2019.42.

[121] Omer Horvitz and Jonathan Katz. Universally-composable two-party computation in two rounds. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 111–129. Springer, Heidelberg, August 2007. doi: 10.1007/978-3-540-74143-5_7.

[122] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015. doi: 10.1145/2688073.2688105.

[123] Texas Instruments. Msp430x1xx family user guide. https://www.ti.com/lit/ug/slau049f/slau049f.pdf, 2006.

[124] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003. doi: 10.1007/978-3-540-45146-4_9.

[125] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, August 2008. doi: 10.1007/978-3-540-85174-5_32.

[126] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 406–425. Springer, Heidelberg, May 2011. doi: 10.1007/978-3-642-20465-4_23.

[127] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 955–966. ACM Press, November 2013. doi: 10.1145/2508859.2516662.

[128] Bargav Jayaraman, Hannah Li, and David Evans. Decentralized certificate authorities. *CoRR*, abs/1706.03370, 2017. URL http://arxiv.org/abs/1706.03370.

[129] Seny Kamara. Restructuring the NSA metadata program. In Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith, editors, *FC 2014 Workshops*, volume 8438 of *LNCS*, pages 235–247. Springer, Heidelberg, March 2014. doi: 10.1007/978-3-662-44774-1_19.

[130] Gabriel Kaptchuk, Matthew Green, and Ian Miers. Giving state to the stateless: Augmenting trustworthy computation with ledgers. In *NDSS 2019*. The Internet Society, February 2019.

[131] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. *Journal of Cryptology*, 26(2):191–224, April 2013. doi: 10.1007/s00145-012-9119-4.

[132] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 525–537. ACM Press, October 2018. doi: 10.1145/3243734.3243805.

[133] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, August 2017. doi: 10.1007/978-3-319-63688-7_12.

[134] Joe Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988. doi: 10.1145/62212.62215.

[135] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008. doi: 10.1007/978-3-540-70583-3_40.

[136] J Kroll, E Felten, and Dan Boneh. Secure protocols for accountable warrant execution. 2014.

[137] Joshua A Kroll, Joe Zimmerman, David J Wu, Valeria Nikolaenko, Edward W Felten, and Dan Boneh. Accountable cryptographic access control.

[138] Enrique Larraia. Extending oblivious transfer efficiently - or - how to get active security with constant cryptographic overhead. In Diego F. Aranha and Alfred Menezes, editors, *LATINCRYPT 2014*, volume 8895 of *LNCS*, pages 368–386. Springer, Heidelberg, September 2015. doi: 10.1007/978-3-319-16295-9_20.

[139] Ian Levy and Crispin Robinson. Principles for a more informed exceptional access debate. *Lawfare*, Thursday 2018. URL https://www.lawfareblog.com/principles-more-informe

d-exceptional-access-debate.

[140] Allison B. Lewko. Tools for simulating features of composite order bilinear groups in the prime order setting. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 318–335. Springer, Heidelberg, April 2012. doi: 10.1007/978-3-642-29011-4_20.

[141] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 455–479. Springer, Heidelberg, February 2010. doi: 10.1007/978-3-642-11799-2_27.

[142] Eric Lichtblau and Joseph Goldstein. Apple Faces U.S. Demand to Unlock 9 More iPhones. *The New York Times*, February 2016. URL https://www.nytimes.com/2016/02/24/technology/justice-department-wants-apple-to-unlock-nine-more-iphones.html.

[143] J. Liu, M. D. Ryan, and L. Chen. Balancing societal security and individual privacy: Accountable escrow system. In *2014 IEEE 27th Computer Security Foundations Symposium*, pages 427–440, July 2014. doi: 10.1109/CSF.2014.37.

[144] Jia Liu, Tibor Jager, Saqib A. Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, 86(11):2549–2586, Nov 2018. ISSN 1573-7586. doi: 10.1007/s10623-018-0461-x. URL https://doi.org/10.1007/s10623-018-0461-x.

[145] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. In Matt Blaze, editor, *USENIX Security 2004*, pages 287–302. USENIX Association, August 2004.

[146] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019. doi: 10.1145/3319535.3339817.

[147] Microcorruption. Lockitall lockit pro user guide. https://microcorruption.com/public/manual.pdf, 2013.

[148] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed E-cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411. IEEE Computer Society Press, May 2013. doi: 10.1109/SP.2013.34.

[149] Matt Miller. Trends, challenges, and strategic shifts in the software vulnerability mitigation landscape. https://github.com/Microsoft/MSRC-Security-Research/blob/master/presentations/2019_02_BlueHatIL/2019_01%20-%20BlueHatIL%20-%20Trends%2C%20challenge%2C%20and%20shifts%20in%20software%20vulnerability%20mitigation.pdf, Feb 2019.

[150] B. Mood, D. Gupta, H. Carter, K. Butler, and P. Traynor. Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 112–127, 2016. doi: 10.1109/EuroSP.2016.20.

[151] Benjamin Mood, Debayan Gupta, Henry Carter, Kevin Butler, and Patrick Traynor. Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 112–127. IEEE, 2016.

[152] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. 2008. URL http:

//www.bitcoin.org/bitcoin.pdf.

[153] Ellen Nakashima. Chinese hackers who hacked Google gained access to sensitive data, U.S. officials say. *The Washington Post*, May 2013. URL https://www.washingtonpost.com/world/national-security/chinese-hackers-who-breached-google-gained-access-to-sensitive-data-us-officials-say/2013/05/20/51330428-be34-11e2-89c9-3be8095fe767_story.html.

[154] National Academies of Sciences, Engineering, and Medicine. *Exploring Encryption and Potential Mechanisms for Authorized Government Access to Plaintext*. The National Academies Press, 2016.

[155] National Academies of Sciences, Engineering, and Medicine. *Decrypting the Encryption Debate: A Framework for Decision Makers*. The National Academies Press, Washington, DC, 2018. URL https://www.nap.edu/catalog/25010/decrypting-the-encryption-debate-a-framework-for-decision-makers.

[156] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 116–125. ACM Press, November 2001. doi: 10.1145/501983.502000.

[157] Jesper Buus Nielsen and Claudio Orlandi. LEGO for two-party secure computation. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 368–386. Springer, Heidelberg, March 2009. doi: 10.1007/978-3-642-00457-5_22.

[158] Johnathan Nightingale. Fraudulent *.google.com Certificate, August 2011.

[159] Emmanuel Odunlade. Top 10 popular microcontrollers among makers. https://www.electronics-lab.com/top-10-popular-microcontrollers-among-makers/, Jun 2020.

[160] Yair Oren. On the cunning power of cheating verifiers: Some observations about zero knowledge proofs (extended abstract). In *28th FOCS*, pages 462–471. IEEE Computer Society Press, October 1987. doi: 10.1109/SFCS.1987.43.

[161] Gaurav Panwar, Roopa Vishwanathan, Satyajayant Misra, and Austin Bos. SAMPL: Scalable auditability of monitoring processes using public ledgers. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2249–2266. ACM Press, November 2019. doi: 10.1145/3319535.3354219.

[162] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013. doi: 10.1109/SP.2013.47.

[163] Rafael Pass and abhi shelat. A Course In Cryptography. https://www.cs.cornell.edu/courses/cs4830/2010fa/lecnotes.pdf, January 2010.

[164] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992. doi: 10.1007/3-540-46766-1_9.

[165] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 187–196. ACM Press, May 2008. doi: 10.1145/1374376.1374406.

[166] Ryan Pickren. Hacking the apple webcam (again). https://www.ryanpickren.com/safari

-uxss, 2021.

[167] Cody M. Poplin. Burr-feinstein encryption legislation officially released. *Lawfare*, April 2016. URL https://www.lawfareblog.com/burr-feinstein-encryption-legislation-officially-released.

[168] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th FOCS*, pages 859–870. IEEE Computer Society Press, October 2018. doi: 10.1109/FOCS.2018.00086.

[169] Michael O. Rabin. How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187, 2005. https://eprint.iacr.org/2005/187.

[170] Certicom Research. Sec 2: Recommended elliptic curve domain parameters, January 2010. URL https://www.secg.org/sec2-v2.pdf. [Online; Accessed on October 27, 2021].

[171] Dragos Rotaru and Tim Wood. MArBled circuits: Mixing arithmetic and Boolean circuits with active security. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *INDOCRYPT 2019*, volume 11898 of *LNCS*, pages 227–249. Springer, Heidelberg, December 2019. doi: 10.1007/978-3-030-35423-7_12.

[172] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999. doi: 10.1109/SFFCS.1999.814628.

[173] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005. doi: 10.1007/11426639_27.

[174] Stefan Savage. Lawful device access without mass surveillance risk: A technical design discussion. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1761–1774, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356930. doi: 10.1145/3243734.3243758. URL https://doi.org/10.1145/3243734.3243758.

[175] Alessandra Scafuro. Break-glass encryption. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 34–62. Springer, Heidelberg, April 2019. doi: 10.1007/978-3-030-17259-6_2.

[176] Peter Scholl. Extending oblivious transfer with low communication via key-homomorphic PRFs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 554–583. Springer, Heidelberg, March 2018. doi: 10.1007/978-3-319-76578-5_19.

[177] Aaron Segal, Bryan Ford, and Joan Feigenbaum. Catching bandits and only bandits: Privacy-preserving intersection warrants for lawful surveillance. In *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14)*, San Diego, CA, August 2014. USENIX Association. URL https://www.usenix.org/conference/foci14/workshop-program/presentation/segal.

[178] Sen. Marsha Blackburn Sen. Lindsey Graham, Sen. Tom Cotton. Lawful access to 5 encrypted data act. https://www.judiciary.senate.gov/press/rep/releases/graham-cotton-blackburn-introduce-balanced-solution-to-bolster-national-security-end-use-of-warrant-proof-encryption-that-shields-criminal-activity, June 2020.

[179] Sacha Servan-Schreiber and Archer Wheeler. Judge, jury & encryptioner: Exceptional access with a fixed social cost, 2019.

[180] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997. doi: 10.1007/3-540-69053-0_18.

[181] Signal. Signal secure messaging system. URL https://signal.org/.

[182] Manish Sing. Over two dozen encryption experts call on India to rethink changes to its intermediary liability rules. *TechCrunch*, February 2020. URL https://techcrunch.com/2020/01/09/over-two-dozen-encryption-experts-call-on-india-to-rethink-changes-to-its-intermediary-liability-rules/.

[183] Yannis Smaragdakis. Sound analysis: Can we tell the truth about programs? https://blog.sigplan.org/2019/09/18/sound-analysis-can-we-tell-the-truth-about-programs/, Sep 2019.

[184] Nigel Smart. Twitter thread: How many AND gates would there be in a combinatorial circuit for an elliptic curve point multiplication?, November 2020. URL https://twitter.com/SmartCryptology/status/1327280495978278914. [Online; @SmartCryptology].

[185] swisspost evoting. E-voting system 2019. https://gitlab.com/swisspost-evoting/e-voting-system-2019, 2019.

[186] Matt Tait. An approach to James Comey's technical challenge. *Lawfare*, April 2016. URL https://www.lawfareblog.com/approach-james-comeys-technical-challenge.

[187] Jamie Tarabay. Australian Government Passes Contentious Encryption Law. *The New York Times*, December 2018. URL https://www.nytimes.com/2018/12/06/world/australia/encryption-bill-nauru.html.

[188] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018. doi: 10.1109/SP.2018.00060.

[189] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. Emp-toolkit: Efficient multiparty computation toolkit. Available At https://github.com/emp-toolkit, 2016.

[190] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 619–636. Springer, Heidelberg, August 2009. doi: 10.1007/978-3-642-03356-8_36.

[191] Nicholas Watt, Rowena Mason, and Ian Traynor. David Cameron pledges anti-terror law for internet after Paris attacks. *The Guardian*, January 2015. URL https://www.theguardian.com/uk-news/2015/jan/12/david-cameron-pledges-anti-terror-law-internet-paris-attacks-nick-clegg.

[192] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. Cryptology ePrint Archive, Report 2020/925, 2020. https://eprint.iacr.org/2020/925.

[193] WhatsApp. WhatsApp Encryption Overview. Available at https://scontent.whatsapp.net/v/t61/68135620_760356657751682_6212997528851833559_n.pdf/WhatsApp-Security-Whitepaper.pdf, December 2017.

[194] Clifford Wolf. Yosys open synthesis suite. http://www.clifford.at/yosys/.

[195] Charles Wright and Mayank Varia. Crypto crumple zones: Enabling limited access without mass surveillance. In *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 288–306, April 2018. doi: 10.1109/EuroSP.2018.00028.

[196] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 733–764. Springer, Heidelberg, August 2019. doi: 10.1007/978-3-030-26954-8_24.

[197] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. QuickSilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2986–3001. ACM Press, November 2021. doi: 10.1145/3460120.3484556.

[198] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982. doi: 10.1109/SFCS.1982.38.

[199] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986. doi: 10.1109/SFCS.1986.25.

[200] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015. doi: 10.1007/978-3-662-46803-6_8.

[201] Greg Zaverucha. The picnic signature algorithm. Technical report, 2020. https://github.com/microsoft/Picnic/raw/master/spec/spec-v3.0.pdf.

[202] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vRAM: Faster verifiable RAM with program-independent preprocessing. In *2018 IEEE Symposium on Security and Privacy*, pages 908–925. IEEE Computer Society Press, May 2018. doi: 10.1109/SP.2018.00013.