On the Theory and Design of Post-Quantum Authenticated Key-Exchange, Encryption and Signatures

by

Cyprien P. R. Delpech de Saint Guilhem

Department of Computer Science
Faculty of Engineering
University of Bristol

A dissertation submitted to the University of Bristol in accordance with the requirements for award of the degree of Doctor of Philosophy in the Faculty of Engineering.

September 2020

Word count: 58,000

Abstract

This thesis presents contributions to three areas of cryptography under the common concern for security against quantum adversaries. It discusses authentication in key-exchange protocols, isogeny-based schemes and protocols, and signature schemes based on the MPC-in-the-head paradigm.

In the first part, nuances of authentication in key-exchange protocols are explored and a transformation to add authentication to existing non-authenticated protocols is proposed. In Chapter 3, the Bellare-Rogaway model of security in a modern formulation, augmented to accommodate new definitions, is presented. In Chapter 4, new definitions for nuances of authentication which had remained undefined until this work are introduced, relations between them are studied and a folklore composition theorem is proven. In Chapter 5, the new definitions are applied to the study of existing protocols and a new secure transformation is presented.

In the second part, constructions for secure public-key encryption schemes and oblivious transfer protocols based on isogenies of supersingular elliptic curves are presented. In Chapter 6, a new one-way function family is defined and an IND-CCA-secure encryption scheme is constructed. In Chapter 7, a new framework of semi-commutative structures, which captures both isogeny-based and exponentiation-based protocols, is defined and different instantiations corresponding to the literature are discussed. In Chapter 8, two oblivious transfer protocols built from these structures are then presented together with their proof of security in the framework of universally composable security.

In the third and final part, improvements to signature schemes built from the MPC-in-the-head proof paradigm are explored. In Chapter 9, the impact of using the AES block cipher within Picnic signatures is studied. In Chapter 10, new verification techniques for this paradigm are used to both study the use of the Legendre PRF and re-visit the efficiency of using the AES block cipher.

Acknowledgments

Looking back on the last four years, my heart goes out in thanks to the many minds that have shaped me as a researcher and as a person.

To Nigel and Bogdan, my advisors, who have both, each in their own way, taught me invaluable skills and the confidence to always broaden my horizons. While there is still much for me to learn, their example is one that I will carry with me always.

To Emmanuela, for keeping me on the right track and reminding me of what is important.

To my other co-authors, Christophe, Marc, Lauren, Javier, Péter and Ward for their enthusiasm in our work together and for the shared experiences of success and failure. To my colleagues at Bristol and Leuven for their generosity with their time and ideas. To Tim and Dragos particularly, for all the changes that we lived through together.

À ma famille pour son amour et ses encouragements, donnés sans réserve et depuis le tout début, surtout pendant les moments les plus difficiles. To my Kiwi family for their love and support strong enough to be felt halfway around the world.

To you Ashlee, for your love, your understanding and your unwavering support; for always being at my side through all of it.

Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's *Regulations and Code of Practice for Research Degree Programmes* and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

Signed:	*Cyprien P. R. Delpech de Saint Guilhem*
Date:	September 24, 2020

Contents

List of Acronyms

AES	advanced encryption standard.
AKE	authenticated key-exchange.
BR	Bellare–Rogaway.
CDH	computational Diffie–Hellman.
DH	Diffie–Hellman.
DLP	discrete logarithm problem.
HVZK	honest-verifier zero-knowledge.
KCI	key-compromise impersonation.
KDF	key-derivation function.
KE	key-exchange.
MAC	message authentication code.
MPC	multiparty computation.
MPCitH	MPC-in-the-head.
OT	oblivious transfer.
OWF	one-way function.
PKE	public-key encryption.
PRF	pseudo-random function.
PRG	pseudo-random generator.
QROM	quantum random oracle model.
ROM	random oracle model.
SIDH	supersingular isogeny Diffie–Hellman.
SPN	substitution-permutation network.
UKE	un-authenticated key-exchange.
UKS	unknown key-share.
ZK	zero-knowledge.

List of Figures

List of Tables

Chapter 1

Introduction

Variety is the spice of life.

This idiom provides an important, albeit simple, direction for academic research, especially in the field of cryptology. At a time when new cryptanalytic methods can emerge at any moment and destabilise established security guarantees, it is both desirable and necessary to have a wide variety of cryptographic tools available to address such threats.

After the seminal works of Diffie and Hellman in 1976 introducing public-key cryptography [60] and Rivest, Shamir and Adleman in 1978 introducing RSA encryption [113], a majority of schemes and protocols designed in this space relied on the families of discrete logarithm and factoring problems for their security. The publication of Shor's algorithm in 1994 [116], a quantum polynomial-time algorithm to solve the factoring and discrete logarithm problems, has meant that these two families can no longer be relied on for security in the long term.

Cryptographic research in the last two to three decades has focused heavily on finding alternative quantum-resistant methods and computational problems under the name of *post-quantum cryptography.* This thesis is part of this research effort and, staying true to the adage, presents a variety of contributions in this direction. After this Introduction and some general preliminary notions defined in Chapter 2, Part I discusses notions of authentication for key-exchange protocols and presents a transformation designed to add authentication using post-quantum tools. Part II then presents constructions of schemes and protocols based on the framework of isogenies of supersingular elliptic curves. Finally, Part III studies improvements in the paradigm of MPC-in-the-head for the design of efficient signature schemes. The rest of this Introduction discusses each of these parts in turn and presents the author's list of publications on which this thesis is based.

A note on narrative voice. I chose to write the content of this thesis in the first person plural to reflect the collaborative nature of the work. My personal contribution is highlighted at the beginning of each chapter.

1.1 Authentication and transformation for key-exchange

Key-exchange is one of the fundamental primitives of public-key cryptography, alongside encryption and signatures. Originating in the seminal work of Diffie and Hellman [60], its aim is to provide a secure method of obtaining shared secret keys without the need of an already-secure communication channel. To add authentication guarantees to these secret keys, one then needs to consider authenticated key-exchange (AKE) protocols.

The original definition of a distributed model of security for AKE protocols was given by Bellare and Rogaway in 1993 [15]. Such a distributed model is required as AKE protocols are intended for execution environments where many machines aim to communicate with each other without previously established secure channels. Shortly after, this original definition was adapted to the public-key setting [23]. This formulation was modernised later on [93] and, more recently, incorporated into a more general framework for game-based security notions [32]. These works are representatives of a long series of extensions to the original model [16, 23, 14, 35, 32, 97] which is traditionally referred to as the Bellare–Rogaway (BR) model. Other works have studied models with stronger adversaries against AKE protocols [35, 96, 50] but Part I of this document will be almost entirely restricted to the original adversarial model.

1.1.1 Security definitions for AKE protocols

Conceptually, security definitions for AKE protocols focus on two notions: authentication and (key) secrecy. Together, these notions ensure that, after the execution of the protocol, the final session-key is known only by the "right" parties. This simplistic presentation gives way to a complex landscape of interrelated notions upon closer examination: both secrecy and authentication each come with their own intricacies.

In the case of authentication, it can be either one-way or mutual, where either one or both parties authenticate during the protocol; it can either concern the entities involved or the session-key derived at the end; and it can also be implicit in the way the session terminates, or fully explicit. To illustrate this last nuance: we often visualize authentication as a challenge-response mechanism, e.g. where one party signs a random challenge and the other verifies the signature. This is very much an explicit authentication guarantee. Some protocols, for efficiency's sake, instead aim to imbed the authentication directly into the derived session-key, without such a challenge-response mechanism. In this case, the parties do not obtain an explicit guarantee that their partner is able to compute the correct response. However, if they observe that their session-key is used at a later time, then they can conclude that their partner was able to compute the response and therefore they can authenticate and trust the key that they hold. This is what we call implicit authentication, i.e. one that is achieved only implicitly at the time of the protocol. In addition, there is the notion of key-confirmation for which parties expect a message that demonstrates that their partner has derived the session-key in question. This notion also brings to light the fact that one party always terminates before the other, thus inducing an asymmetry which can be reflected in the guarantees provided by the protocol.

For secrecy, nuances are fewer but nonetheless important. The notion of forward secrecy was defined as early as 1997 [104] and captures the guarantee that a session-key will remain secret even if the parties' long-term secret material is compromised after the session has taken place. This notion was then also further nuanced and refined [92, 50].

The definition of Bellare and Rogaway was originally presented in two parts: a first required that parties mutually authenticate each other during the protocol, and a second required that they also derived an unpredictably random secret key. This presented a clear separation between (entity) authentication and key secrecy. Over time, however, definitions for AKE protocols have shifted away from this separation and towards guarantees only concerning the unpredictability of session-keys [92, 33]. Authentication was then captured only as part of the definitions of secrecy; this implies that attacks on authentication always have to be transformed into attacks on secrecy in order to be considered within a given model. Other security notions were formalized on their own, such as forward secrecy mentioned above or also key-confirmation [69], and some attempts were made to systematize the study of security models [41, 124, 49], but this happened outside of a unified framework and focused more on protocol design and less on the study of the definitions themselves.

Unfortunately, the lack of a comprehensive study of security notions, and the amalgamation of authentication together with secrecy has led to some missing formal definitions as well as unclear relation between different notions. Similarly, large classes of attacks against AKE protocols, such as unknown key-share or key-compromise impersonation attacks, have not been wholly captured by some security definitions but have nonetheless influenced the design of many protocols in attempts to avoid them. In this work we present security definitions intended to capture the nuances and complexity of authentication notions that are missing in the current literature.

In Chapter 3, we first present a formulation of the BR model based on the framework of [32] but modified and augmented with certain elements. In Chapter 4 we then define and study several new authentication notions as well as their relationship with one another. Finally in Section 5.1 we study three important KE and AKE algorithms in the light of our new authentication notions.

Post-quantum aspects. On their own, security definitions for authentication can easily be adapted to a post-quantum model simply by requiring that they hold even against polynomial-time quantum adversaries. This is the approach that we take in this thesis.

1.1.2 Generic transformations to achieve AKE security

Due to the different security guarantees that they aim to provide, AKE protocols can be difficult to construct from elementary cryptographic primitives. This approach also leads to protocols that have to be analyzed and proved in a monolithic way, with no possibility of modularity. These drawbacks led to the definition of generic transformations from simpler KE protocols to more complex AKE ones.

The fist such generic compiler was proposed by Bellare, Canetti and Krawczyk in 1998 [13] and later refined by Canetti and Krawczyk in 2001 [35] to protect against adversaries of different strengths. Both of these works share a common design principle: they first consider protocols in a model where authenticated links are guaranteed (Auth model) and then compile these to a model with unauthenticated links (UAuth model) with the use of authenticator tools. These authenticators function on a per-message basis to guarantee that a message sent securely in the Auth model is sent with the same security in the UAuth model. This approach guarantees ease of analysis and design, but at a great cost to efficiency. For example, an authenticator from [13] requires three messages in the UAuth model for each message in the Auth model. As a passive KE protocol requires at least two messages to be exchanged, this results in an AKE protocol with at least five rounds of communication (or more if the messages of the authenticators cannot be sent in parallel).

Other design principles are also possible. Katz and Yung consider the setting of group key-exchange and boost KE security to AKE security by having parties first exchange nonces and then augment all further messages with signatures containing these nonces [88]. When groups of only two parties are considered, this approach requires four messages to be exchanged. In [105], Morrissey, Smart and Warinschi study the TLS protocol as the successive application of two generic transformations from passive KE security to active AKE; this is however not efficient, particularly because of the complex nature and purpose of the TLS protocol itself. Another work by Jager et al. proposes to first execute the KE protocol and then obtain authentication with the exchange of nonces and the use of signatures [81]; this however results in eight rounds of communication (if not optimized). The later work of Li et al. also proposes to first execute the KE protocol and then exchange either signatures or ciphertexts and MACs to obtain authentication [98]; this again results in four or more rounds of communication.

The many transformations mentioned above all rely on a combination of digital signature schemes, public-key encryption (PKE) schemes and message authentication codes. Two other works have constructed transformation using key encapsulation mechanisms (KEMs) [28, 77] but both of these rely on identity-based and pairing-based primitives and cannot be generalised to all schemes.

These transformations all ensure that the authentication part of the resulting scheme can be analyzed and proved independently of the key-secrecy part of the initial passive KE protocol but they fall short of achieving optimal efficiency. In Section 5.2, we present a transformation that achieves a three-round AKE protocol by combining the authentication flows with those of the initial KE protocol.

Post-quantum aspects. When attempting to construct post-quantum AKE protocols, the primitives that are used matter more since they need to be post-quantum too. For the underlying passive KE protocols, the original ones based on Diffie–Hellman key exchange [60] can be replaced with post-quantum ones such as ones based on lattices [26, 4] or supersingular isogenies [66, 36]. At the time of writing our work of Section 5.2, there was not much confidence in post-quantum signature schemes. The lattice-based ones that had been proposed [75, 79, 78] were eventually broken [73, 106], and schemes based on other assumptions were still too inefficient to be practical. That state of the art at the time motivated our design choice of using only public-key encryption which demonstrated better promise in the post-quantum literature [100, 99].

1.2 Cryptography from supersingular isogenies

Among the several mathematical structures proposed to replace factoring and discrete logartihm hardness assumptions, isogenies of supersingular elliptic curves have emerged recently as a contender with strong potential. Jao and De Feo presented the idea to use such isogenies for public-key cryptography for the first time in 2011 [82]; some schemes based on isogenies of *ordinary* elliptic curves existed already [119], but they were impractically slow and it was shown that they were vulnerable to subexponential quantum attacks [40, pre-print in 2010]. In constrast, Jao and De Feo's proposed setting can be parameterized to offer much faster computations and no efficient quantum algorithm is yet known to attack the corresponding computational problems [21].

In addition to hash functions [37] which in fact pre-date Jao and De Feo's work, supersingular isogenies and the proposed problems have been used to build a vareity of primitives since their introduction, such as key-exchange and encryption [66, 6] (in the form of KEM-DEM constructions), signature and identification [66, 71] and also authenticated key-exchange [128].

In this new and growing body of works, two main categories of computational problems have emerged. The first is a more generic one, where the adversary is given less information about the curves involved in the challenges. Such a problem would, for example, consist in computing an isogeny between two supersingular curves E and E', defined over $\mathbb{F}_{p^2}$, without being given any further information; this is very similar conceptually to the discrete logarithm problem which consists in computing a such that $h = g^a$ for g and h given without further information other than the order of the group. The second category of problems arose from the fact that the first is not very practical to build protocols with. In their definition of the supersingular isogeny Diffie–Hellman (SIDH) key-exchange protocol, Jao and De Feo could not use problems of the first category to prove the security of the scheme because the setting prevents arbitrary commutativity of isogenies; here the concept of the protocol requires in fact that the operations of both parties are commuted versions of each other. To recover some commutativity, they designed the protocol to reveal images of certain points on the curves under the secret isogenies of the parties. As more information was then revealed, this lead to SIDH variants of the first category of problems.

1.2.1 Public-key encryption from generic isogeny problems

This additional information revealed as part of the protocols has been exploited for the first time by Petit in 2017 [110] and this attack has been further improved since [94]. While these attacks do not yet apply to the parameter space of the problems used by protocol of the SIDH type, they nonetheless undermine somewhat the hardness assumptions.

In anticipation of further progress in this direction, several schemes have found it desirable to avoid these SIDH variants of isogeny problems. Until the work presented in this thesis in Chapter 6 [57], this had only been achieved for signature schemes [120, 71] for generic supersingular isogenies over $\mathbb{F}_{p^2}$. A special case is the CSIDH work of Castryck et al. [36], building on the work of Couveignes [48] and Rostovtsev and Stolbunov [114], which presents a key-exchange protocol which assumes the hardness of the generic isogeny problems, but restricted to elliptic curves defined over $\mathbb{F}_p$ instead of $\mathbb{F}_{p^2}$. This restricted setting has the drawback of being vulnerable to subexponential quantum algorithms.

The work presented in Chapter 6 aims to continue this line of research on security based on generic problems. Indeed, we present a PKE scheme whose key-recovery security reduces directly to the most generic computational isogeny problem. The standard OW-CPA and IND-CCA security of the schemes do rely on the SIDH variants but these are more generic ones than previously used in the literature. Should further progress be made in the cryptanalysis of these problems, our PKE scheme furthermore offers strong flexibility in its parameters to adapt and remain secure.

1.2.2 Oblivious transfer from semi-commutative masking

One reason for the slower adoption of isogeny-based primitives, in terms of the number of schemes being proposed, is that some of the subtleties of this setting can be counter-intuitive (and even lead to dangerous security weaknesses when misunderstood [70]).

In Chapter 7, we consider a new approach to the design of isogeny-based protocols by defining the generic framework of invertible semi-commutative masking structures. These masking structures aim to generalize the exponentiation-only paradigm of protocol design used in certain applications such as key-exchange. This is more restrictive than other paradigms, such as the discrete-logarithm setting, due to the absence of other operations "in the exponent". The semi-commutative nature of these structures then captures the absence of full commutativity for supersingular isogenies as observed and remedied by Jao and De Feo [82] with the introducion of SIDH variants of the computational problems.

After defining them in generic terms, we then first instantiate these structures from group actions, which is a generalization not only of the traditional discrete-logarithm setting but also of Couveignes [48] and Castryck et al.'s CSIDH [36] setting of isogenies of $\mathbb{F}_p$-restricted supersingular elliptic curves. We then present an instantiation from isogenies of un-restricted curves over $\mathbb{F}_{p^2}$ which captures the SIDH setting. In addition, we define generic computational problems for the masking structures and discuss how their instantiation into the different settings result in problems which correspond to the ones currently used in the literature.

We then illustrate the use of this framework in Chapter 8 with the design of two oblivious transfer (OT) protocols. Originally proposed by Rabin in 1981 [112], oblivious transfer is a primitive that has since been used as a building block for many efficient secure multiparty protocols [107, 89, 126] (i.e. protocols where multiple party jointly execute the computation of a function on their respective secret inputs). Protocols for OT built from post-quantum primitives have already been proposed for the LWE, LPN and McEliece assumptions [108, 8, 29] but it took longer for isogeny-based protocols to emerge. Our work is in fact part of a group of concurrent and independent proposals to construct isogeny-based OT that appeared around the same time [125, 30, 9].

The OT protocols that we propose both achieve universal composability security against passive adversaries with static corruptions in the random oracle model. We show, furthermore, that our first protocol satisfies the security requirements of two-round OT to make use of a recent transformation which provides security against active adversaries [63].

1.3 Digital signatures from MPC-in-the-head

A common design approach for signature schemes is the use of the Fiat–Shamir transformation paradigm [68] which turns an interactive proof of knowledge protocol with public random coins into a non-interactive one with the use of random oracles. Such proofs can then be bound to a particular message by including it in the call to the random oracle that generates the random coins used in the protocol. By using a zero-knowledge protocol and proving knowledge of a witness (secret-key) to a public relation (public-key) in this way, one obtains a signature scheme secure both against key-recovery and forgery attacks.

To then achieve post-quantum security, both the zero-knowledge proof system and the particular relation that is proved must be resistant against quantum adversaries. Not originally presented as an advance in the post-quantum direction, the work of Ishai, Kushilevitz, Ostrovski and Sahai (IKOS) was the first to propose using multiparty computation (MPC) protocols and their security properties to create zero-knowledge proofs [80].

In what later became known as MPC-in-the-head (MPCitH), the prover executes a simulated MPC protocol between several parties evaluating an NP relation $R(x, w)$. In this computation, x is the public statement (known to the verifier) and w is the private statement, secret-shared among the simulated MPC parties. After simulating the MPC verification of $R(x, w)$, the prover commits to what each simulated party observed during the computation. The verifier of the proof system then asks for several of these views to be opened in order to verify that the MPC computation took place honestly, and that the prover did not make any party cheat to change the output of the protocol.

By opening only a subset of all the views of the simulated parties, the prover does not reveal any information about the secret witness shared among the parties. By asking to observe the honest behaviour of the opened parties, the verifier can obtain guarantees that the execution was not tampered with. Thus the post-quantum security properties of the MPC protocol, i.e. simulatability and resistance against corruptions, directly translate to the post-quantum zero-knowledge and soundness properties of the proof protocol.

To achieve the post-quantum security of the relation proved by the protocol, a natural option is to choose a relation based on symmetric-key primitives. This was first proposed by Giacomelli, Madsen and Orlandi in work which improved on the system of IKOS and demonstrated proofs of knowledge of pre-image for the SHA-1 and SHA-256 circuits [74]. Improving further on this work, Chase et al. proposed using the LowMC block cipher, specifically designed for better performance within MPC protocols, in the first construction of a signature scheme based on the MPCitH paradigm [39]. The core of the post-quantum security of this proposal is that the LowMC cipher, being a symmetric-key construction, is not vulnerable to the performant quantum algorithms which can solve factorization or discrete logarithm problems. In this case, the signer (or prover) demonstrates knowledge of a secret key k such that $\texttt{LowMC}_k(x) = y$ for a fixed plaintext-ciphertext pair (x, y). Further security comes from the fact that only one such pair is revealed as part of the public key of the signature scheme, and that the adversary cannot request more.

The work of Chase et al. became the basis of the Picnic signature scheme submitted to the NIST's Post-Quantum standardization project [38]. It was then further improved, both in the zero-knowledge proof component [86], and in the efficiency of the block cipher [62, 83, 84], until the current Picnic3 proposal.

Crucially, all these iterations focus on the LowMC block cipher which was only recently introduced in 2015 [3]. As the potential weaknesses of this primitive may not yet all be known, it is important to consider other options. In Part III of this thesis, we present work which proposes the use of different primitives within the MPCitH paradigm and the possible improvements that can be achieved.

In Chapter 9, we study the use of the AES block cipher [51] as a replacement for LowMC in Picnic-like signatures. We find that by using the same structure of proof system, adapted for arithmetic circuits over $\mathbb{F}_{2^8}$ for efficiency with the AES algorithm, the sizes of signatures would increase by a factor of between 2.48 and 3.09 (depending on the security level) compared to Picnic signatures. We also study the use of the original Rijndael algorithm, the submission from which AES was standardized, as it offers more flexibility in its design.

In Chapter 10 we explore a recent advance in MPCitH technique proposed by Baum and Nof which consists in replacing the MPC protocol that computes the circuit by one which verifies certain intermediary values that the prover was allowed to inject into the MPC. This creates efficiency gains since such verification can be checked probabilistically and in batches, thus not requiring the full execution of the block cipher circuit within the MPC protocol. We first present in Section 10.1 joint work with W. Beullens which proposes to use the Legendre PRF as the primitive behind the NP relation. In Section 10.2 we then return to the AES block cipher and present unpublished work which achieves signature sizes with an estimated increase factor of only between 1.07 and 1.23 compared to average Picnic3 signatures.

1.4 Publications of the author

The content of this thesis is selected from my work published over the course of my degree. The publications below are numbered continuously but listed in chronological order within each list. Author order was decided alphabetically for each publication and is not indicative of relative contribution; my personal contributions are detailed at the beginning of the chapters where a publication is first selected from.

In peer-reviewed venues:

1. [59] *C. Delpech de Saint Guilhem*, N. P. Smart, and B. Warinschi. Generic Forward-Secure Key Agreement without Signatures. Published in the proceed-

ings of *Information Security - 20th International Conference, ISC 2017*, Ho Chi Minh City, Vietnam. Contents of this work appear in Section 5.2.

2. [55] *C. Delpech de Saint Guilhem*, L. De Meyer, E. Orsini, and N. P. Smart. BBQ: Using AES in Picnic Signatures. Published in the proceedings of *Selected Areas in Cryptography - 26th International Conference, SAC 2019*, Waterloo, ON, Canada. Contents of this work appear in Chapter 9.

3. [19] W. Beullens, and *C. Delpech de Saint Guilhem*. LegRoast: Efficient Post-quantum Signatures from the Legendre PRF. Published in the proceedings of *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, Paris, France (held virtually due to 2020 pandemic). Contents of this work appear in Section 10.1.

4. [56] *C. Delpech de Saint Guilhem*, M. Fischlin, and B. Warinschi. Authentication in Key-Exchange: Definitions, Relations and Composition. Published in the proceedings of the *33rd IEEE Computer Security Foundations Symposium, CSF 2020*, Boston, MA, USA (held virtually due to 2020 pandemic). Contents of this work appear in Chapters 3 and 4 and in Section 5.1.

5. [58] *C. Delpech de Saint Guilhem*, E. Orsini, C. Petit, and N. P. Smart. Semi-Commutative Masking: A Framework for Isogeny-based Protocols, with an application to Fully Secure Two Round Isogeny-based OT. Accepted for publication in the proceedings of the *19th International Conference on Cryptology and Network Security* to be presented at the virtual edition of the conference. Contents of this work appear in Chapters 7 and 8.

In online pre-print archives:

6. [57] *C. Delpech de Saint Guilhem*, P. Kutas, C. Petit, and J. Silva. SÉTA: Supersingular encryption from torsion attacks. Cryptology ePrint Archive, Report 2019/1291. Contents of this work appear in Chapter 6. This work is now being merged with follow-up work from P. Kutas and C. Petit and others in preparation for submission.

Chapter 2

Preliminaries

In this short chapter we establish notation and state standard definitions of schemes and security notions which we will use throughout this thesis. Due to their different content, we delay literature discussions to the beginning of each part.

2.1 General definitions

We use λ to denote the computational security parameter and let 1^λ be its unary representation. We say that an algorithm executed on an input of length λ runs in $\mathsf{poly}(\lambda)$-time if it terminates within a number of steps bounded by a polynomial in λ. We denote by $\{0,1\}^n$ the set of all bit-strings of length n and by $\{0,1\}^*$ the set of all bit-strings of any finite length.

Definition 2.1 (Negligible function)**.** A function $f : \mathbb{N} \to \mathbb{R}$ is *negligible (in λ)* if, for every polynomial p, there exists $\lambda_0 \in \mathbb{N}$ such that, for all $\lambda \geq \lambda_0$,

$$f(\lambda) \leq \frac{1}{p(\lambda)}.$$

We let $\mathsf{negl}(\lambda)$ denote an arbitrary negligible function.

We give here the definition of two notions of mathematics: power-smooth numbers and the minimum entropy of a random variable.

Definition 2.2 (B-power-smooth number)**.** Let $N \in \mathbb{N}$ be an integer and let $N = \prod_i \ell_i^{e_i}$ be its unique prime factorization. Let $B \in \mathbb{N}$ be an integer bound. N is a *B-power-smooth number* if and only if, for all i,

$$\ell_i^{e_i} \leq B.$$

Definition 2.3 (Min-entropy)**.** Let X be a random variable. We say that X has *min-entropy* $H_\infty(X) = b$ if and only if

$$\max_x \Pr[X = x] \leq 2^{-b}.$$

2.2 Cryptography definitions

We now recall some standard definitions of primitives and schemes and their security notions. A comprehensive overview of this material can be found in Katz and Lindell's *Introduction to Modern Cryptography*, with some modifications to suit our notation in later chapters [87].

2.2.1 Pseudo-random functions and the random oracle model

A pseudo-random function (PRF) is one of the most important tools in cryptography. It circumvents the difficulty of generating sequences of truly random bits by producing an output which is computationally indistinguishable from random.

Definition 2.4 (Pseudo-random function)**.** Let $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be an efficiently computable, length-preserving keyed function. F is a *(quantum) pseudo-random function* if, for all (quantum) probabilistic $\mathsf{poly}(\lambda)$-time distinguishers $\mathcal{D}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\left| \Pr_{k \leftarrow\!\!\$ \{0,1\}^\lambda}[\mathcal{D}^{F_k(\cdot)} = 1] - \Pr_{f \leftarrow\!\!\$ \mathcal{F}}[\mathcal{D}^{f(\cdot)} = 1] \right| \leq \mathsf{negl}(\lambda)\,,$$

where $\mathcal{F}$ denotes the set of all functions mapping $\{0,1\}^n \to \{0,1\}^n$.

A pseudo-random generator (PRG) is then a strictly length-expanding PRF.

The *random oracle model (ROM)* is a proof technique used in cryptography to idealize random functions. When a hash function or a PRF, such as those used as a key-derivation function (KDF), is required to have a uniform output distribution, it is replaced by a random oracle constructed as a list populated with random output values each time a new input value is queried. A security proof can then be given in the ROM, as opposed to the *standard model*, which guarantees that the security of the scheme relies only on the random-looking ability of the function replaced by the oracle.

2.2.2 Symmetric cryptography

After the PRF, another important tool in cryptography is the one-way function (OWF). Rather than looking random, it's objective is to be hard to invert. In other words, it should be computationally unfeasible for a (quantum) adversary to recover the input corresponding to a given output. A OWF can also possess a *trapdoor* which enables the efficient recovery of inputs.

Definition 2.5 (Trapdoor one-way functions)**.** Let $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^n$ be an efficiently computable and keyed function. F is a *(quantum) one-way function* if, for all (quantum) probabilistic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\Pr_{k \leftarrow\$ \{0,1\}^\lambda}[x \leftarrow \mathcal{A}(F_k(\cdot), y); x \leftarrow\$ \{0,1\}^n, y \leftarrow F_k(x)] \leq \mathsf{negl}(\lambda)\,.$$

F is a *trapdoor* OWF if, for each $k \in \{0,1\}^\lambda$ there exists a efficient representation of F_k^{-1} such that $F_k^{-1}(y)$ is efficiently computable for all $y \in \{0,1\}^n$.

Unforgeable message authentication codes

A message authentication code (MAC) is a symmetric-key primitive that allows parties sharing a secret key k to authenticate and verify messages, thereby detecting eventual tampering with their content. Formally, a MAC is a triple of $\mathsf{poly}(\lambda)$-time algorithms $\mathtt{MAC} = (\mathtt{KGen}, \mathtt{Tag}, \mathtt{Vfy})$ such that, given a message and a key, $\mathtt{Tag}$ produces a tag, and such that, given a message, a tag and a key, $\mathtt{Vfy}$ verifies that the tag corresponds to the message.

Definition 2.6 (Message authentication code)**.** A *message authentication code* is a triple of probabilistic $\mathsf{poly}(\lambda)$-time algorithms $\mathtt{MAC} = (\mathtt{KGen}, \mathtt{Tag}, \mathtt{Vfy})$ such that:

1. The *key-generation* algorithm $\mathtt{KGen}$ takes as input the security parameter 1^λ and outputs a key k.
2. The *tag-generation* algorithm $\mathtt{Tag}$ takes as input a key k and a message $m \in \{0,1\}^*$ and outputs a tag t. We write this as $t \leftarrow \mathtt{Tag}_k(m)$.
3. The deterministic *verification* algorithm $\mathtt{Vfy}$ takes as input a key k, a message m and a tag t. It outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning $\mathsf{invalid}$. We write this as $b \leftarrow \mathtt{Vfy}_k(m,t)$.

1. A key k is generated by running $\texttt{KGen}(1^\lambda)$.
2. The adversary $\mathcal{A}$ is given input 1^λ and oracle access to $\texttt{Tag}_\mathsf{k}(\cdot)$. The adversary eventually outputs (m, t). Let $\mathcal{Q}$ denote the set of pairs of messages queried to the $\texttt{Tag}$ oracle together with their responses.
3. $\mathcal{A}$ succeeds if and only if (1) $\texttt{Vfy}_k(m, t) = 1$ and (2) $(m, t) \notin \mathcal{Q}$. In that case the output of the experiment is defined to be 1.

Figure 2.1: The sEUF-CMA security experiment $\mathbf{Exp}^{\mathsf{sEUF\text{-}CMA}}_{\mathcal{A},\texttt{MAC}}$.

It is required that, for every key k output by $\texttt{KGen}(1^\lambda)$ and every $m \in \{0,1\}^*$, it holds that $\texttt{Vfy}_k(m, \texttt{Tag}_k(m)) = 1$.

The security experiment for strong unforgeability against chosen-message attacks, denoted sEUF-CMA and presented in Figure 2.1, generates a random key and gives the adversary access to a $\texttt{Tag}$ oracle whilst recording pairs of queried messages and the corresponding tags. The goal of the adversary is to output a message and a corresponding valid tag such that this tag was never produced by the $\texttt{Tag}$ oracle for this message.

We denote $\mathcal{A}$'s advantage in the sEUF-CMA security game as

$$\mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathcal{A},\texttt{MAC}} = \Pr\left[\mathbf{Exp}^{\mathsf{sEUF\text{-}CMA}}_{\mathcal{A},\texttt{MAC}} = 1\right]$$

Definition 2.7 (Strong unforgeability)**.** A message authentication code $\texttt{MAC} = (\texttt{KGen}, \texttt{Tag}, \texttt{Vfy})$ is *strongly (quantum) unforgeable under an adaptive chosen-message attack*, or *strongly (quantum) secure*, if for all (quantum) probabilistic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathcal{A},\texttt{MAC}} \le \mathsf{negl}(\lambda)\,.$$

2.2.3 Asymmetric cryptography

As only Part I discusses key-exchange, we reserve these definitions for Chapter 3. Here we state standard definitions for PKE and signature schemes.

Public-key encryption schemes

A PKE scheme is a set of $\mathsf{poly}(\lambda)$-time algorithms used for the encryption of plaintexts into cyphertexts and the reverse decryption.

Definition 2.8 (Public-key encryption scheme)**.** A *public-key encryption scheme* is a tuple of probabilistic $\mathsf{poly}(\lambda)$-time algorithms $E = (\mathtt{Setup}, \mathtt{KGen}, \mathtt{Enc}, \mathtt{Dec})$ such that:

1. The *setup* algorithm $\mathtt{Setup}$ takes as input the security parameter 1^λ and outputs public parameters params required by the encryption scheme. We assume for convenience that 1^λ is implicit in params.

2. The *key-generation* algorithm $\mathtt{KGen}$ takes as input the parameters params and outputs a key pair $(\mathsf{pk}, \mathsf{sk})$. We assume for convenience that params is implicit in either pk or sk.

3. The *encryption* algorithm $\mathtt{Enc}$ takes as input a public key pk and a message m from the message space $\mathcal{M}$ (specified by params). It outputs a ciphertext c, and we write this as $c \leftarrow \mathtt{Enc}_{\mathsf{pk}}(m)$.

4. The deterministic *decryption* algorithm $\mathtt{Dec}$ takes as input a private key sk and a ciphertext c, and outputs a message m or a special symbol $\perp$ denoting failure. We write this as $m \leftarrow \mathsf{Dec}_{\mathsf{sk}}(c)$.

It is required that, except possibly with negligible probability over $(\mathsf{pk}, \mathsf{sk})$ output by $\mathtt{KGen}(\mathsf{params})$, we have $\mathtt{Dec}_{\mathsf{sk}}(\mathtt{Enc}_{\mathsf{pk}}(m)) = m$ for any valid message m.

The standard (single-user) active security notion for PKE schemes is that of *indistinguishability under chosen ciphertext attack*, denoted $\mathsf{IND\text{-}CCA}$. The corresponding security experiment gives an arbitrary adversary a randomly generated public-key and, upon query of a left-right oracle, denoted $\mathcal{O}_{\mathsf{LR}}$, with two messages of identical lengths, returns the encryption of one of the two. Given access to a decryption oracle, the adversary's goal is to guess which of the two messages the oracle encrypts. The adversary may query either oracle several times, with the only restriction that it may not query the decryption oracle on any ciphertext output by the left-right oracle.

In Chapter 5, we make use of the multi-user security notion described in [12]. For n participants, the n-$\mathsf{IND\text{-}CCA}$ security experiment is very similar to the single-user setting. The difference is that the adversary is provided with n different public keys and may query the left-right oracle on any one of these keys. Whether it is the right or left message which is encrypted is still selected at random, but this choice remains consistent between queries to all $\mathcal{O}_{\mathsf{LR}}$ oracles.

Formally, for an arbitrary public-key encryption scheme $E = (\mathtt{Setup}, \mathtt{KGen}, \mathtt{Enc}, \mathtt{Dec})$ and (quantum) $\mathsf{poly}(\lambda)$-time adversary $\mathcal{A}$, we assume that the challenger simulates a set $\mathcal{U}$ of n participants to the adversary. For a bit $b \in \{0, 1\}$ we then define the

1. $\texttt{Setup}(1^\lambda)$ is run to obtain params.
2. For each $U \in \mathcal{U}$, $\texttt{KGen}(\mathsf{params})$ is run to obtain keys $(\mathsf{pk}_U, \mathsf{sk}_U)$.
3. The adversary is given $\{\mathsf{pk}_U\}_{U \in \mathcal{U}}$ and access to several oracles: $\mathcal{O}_{\mathsf{LRpk}_U}(\cdot,\cdot)$ and $\mathsf{Dec}_{\mathsf{sk}_U}(\cdot)$ for each $U \in \mathcal{U}$. The $\mathcal{O}_{\mathsf{LR}}$ oracles take as input two valid messages m_0 and m_1 and returns $\texttt{Enc}_{\mathsf{pk}_U}(m_b)$.
4. $\mathcal{A}$ may query the decryption oracles whenever it wishes, but may never submit a ciphertext output by $\mathcal{O}_{\mathsf{LRpk}_U}$ to the corresponding decryption oracle $\texttt{Dec}_{\mathsf{sk}_U}$.
5. Finally, $\mathcal{A}$ outputs a guess bit b'. We define the output of the experiment to be that guess b'.

Figure 2.2: The n-IND-CCA-b security experiment $\mathbf{Exp}_{\mathcal{A},E}^{n\text{-}\mathsf{IND}\text{-}\mathsf{CCA}\text{-}b}$.

n-IND-CCA-b experiment in Figure 2.2. We denote $\mathcal{A}$'s advantage in the n-IND-CCA game as

$$\mathbf{Adv}_{\mathcal{A},E}^{n\text{-}\mathsf{IND}\text{-}\mathsf{CCA}} = \left|\Pr\left[\mathbf{Exp}_{\mathcal{A},E}^{n\text{-}\mathsf{IND}\text{-}\mathsf{CCA}\text{-}0} = 1\right] - \Pr\left[\mathbf{Exp}_{\mathcal{A},E}^{n\text{-}\mathsf{IND}\text{-}\mathsf{CCA}\text{-}1} = 1\right]\right|.$$

Definition 2.9 (n-IND-CCA security)**.** A public-key encryption scheme given by $E = (\texttt{Setup}, \texttt{KGen}, \texttt{Enc}, \texttt{Dec})$ is said to have *(quantum) polynomially-secure indistinguishable encryptions under a chosen-ciphertext attack* (or is *(quantum) n-CCA-secure*) if for all (quantum) probabilistic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{A},E}^{n\text{-}\mathsf{IND}\text{-}\mathsf{CCA}} \leq \mathsf{negl}(\lambda).$$

The formal definition for the usual IND-CCA security against one key follows from setting $n = 1$ in the experiment and definition above. Security in the multi-user setting and the single user setting are related by the following theorem [12].

Theorem 2.10. *Let $E = (\texttt{Setup}, \texttt{KGen}, \texttt{Enc}, \texttt{Dec})$ be a public-key encryption scheme and n be an integer of $\mathsf{poly}(\lambda)$ size. Then*

$$\mathbf{Adv}_{\mathcal{A},E}^{n\text{-}\mathit{IND}\text{-}\mathit{CCA}} \leq n \cdot \mathbf{Adv}_{\mathcal{A},E}^{1\text{-}\mathit{IND}\text{-}\mathit{CCA}}.$$

A weaker notion for PKE schemes is that of one-way security against chosen-message attacks (OW-CPA). This differs from single-user IND-CCA security in two

1. $\mathtt{Setup}(1^\lambda)$ is run to obtain params.
2. $\mathtt{KGen}(\mathsf{params})$ is run to obtain $(\mathsf{pk}, \mathsf{sk})$.
3. A random message $m \leftarrow_\$ \mathcal{M}$ is sampled and its ciphertext $c \leftarrow \mathtt{Enc}_{\mathsf{pk}}(m)$ is computed.
4. The adversary is given pk and c but no access to any oracle.
5. $\mathcal{A}$ eventually outputs a guess message m'. We define the output of the experiment to be 1 if $m' = m$ and 0 otherwise.

Figure 2.3: The OW-CPA security experiment $\mathbf{Exp}^{\mathsf{OW\text{-}CPA}}_{\mathcal{A},E}$.

aspects; first, the adversary does not have access to a decryption oracle and, second, the adversary must recover the entire plaintext behind a challenge ciphertext, not only distinguish one bit of information. We present the corresponding experiment in Figure 2.3. We denote $\mathcal{A}$'s advantage in the OW-CPA game as

$$\mathbf{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathcal{A},E} = \Pr\left[\mathbf{Exp}^{\mathsf{OW\text{-}CPA}}_{\mathcal{A},E} = 1\right].$$

Definition 2.11 (OW-CPA security)**.** A public-key encryption scheme given by $E = (\mathtt{Setup}, \mathtt{KGen}, \mathtt{Enc}, \mathtt{Dec})$ has *(quantum) one-way security against chosen-plaintext attacks* (or is *(quantum) OW-CPA-secure* if for all (quantum) probabilistic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathcal{A},E} \leq \mathsf{negl}(\lambda).$$

Digital signature schemes

A (digital) signature scheme is a set of $\mathsf{poly}(\lambda)$-time algorithms used to produce signatures on a given message and to verify such signatures against a given public key.

Definition 2.12 (Signature scheme)**.** A *(digital) signature scheme* is a tuple of probabilistic $\mathsf{poly}(\lambda)$-time algorithms $S = (\mathtt{Setup}, \mathtt{KGen}, \mathtt{Sig}, \mathtt{Vfy})$ such that:

1. The *setup* algorithm $\mathtt{Setup}$ takes as input the security parameter 1^λ and outputs public parameters params required by the signature scheme. We assume that 1^λ is implicit in params.

2. The *key-generation* algorithm $\mathtt{KGen}$ takes as input the parameters params and outputs a key pair $(\mathsf{pk}, \mathsf{sk})$. We assume that params is implicit in either pk or sk.

3. The *signature* algorithm $\mathtt{Sig}$ takes as input a secret key sk and a message m from the message space $\mathcal{M}$ (specified by params). It outputs a signature σ which we write as $\sigma \leftarrow \mathtt{Sig}_{\mathsf{sk}}(m)$.

4. The deterministic *verification* algorithm $\mathtt{Vfy}$ takes as input a public key pk, a message m and a signature σ and outputs a bit b indicating whether the signature is valid or not.

It is required that, except possibly with negligible probability over $(\mathsf{pk}, \mathsf{sk})$ output by $\mathtt{KGen}(\mathsf{params})$, it holds that $\mathtt{Vfy}_{\mathsf{pk}}(\mathtt{Sig}_{\mathsf{sk}}(m)) = 1$ for any valid message m.

Similarly to message authentication codes, the role of digital signature schemes is to provide authentication and integrity guarantees to messages. Existential unforgeability under adaptive chosen-message attacks (EUF-CMA) is therefore also the standard security notion here. The experiment is similar to that of Figure 2.1 and we summarize it within the following definition.

Definition 2.13 (EUF-CMA security)**.** A signature scheme $S = (\mathtt{Setup}, \mathtt{KGen}, \mathtt{Sig}, \mathtt{Vfy})$ is said to be *(quantum) existentially unforgeable under an adaptive chosen-message attack* if, for any (quantum) probabilistic $\mathsf{poly}(\lambda)$-time algorithm $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that:

$$\mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A},S} = \Pr\left[\begin{array}{c|c} \mathtt{Vfy}(\mathsf{pk}, m^*, \sigma^*) = 1 & \mathsf{params} \leftarrow \mathtt{Setup}(1^\lambda) \\ \wedge\, m^* \notin \mathcal{Q} & (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{KGen}(1^\lambda) \\ & (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathtt{Sig}_{\mathsf{sk}}(\cdot)}(\mathsf{pk}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where $\mathcal{A}^{\mathtt{Sig}_{\mathsf{sk}}(\cdot)}$ denotes $\mathcal{A}$'s access to a signing oracle with private key sk and $\mathcal{Q}$ denotes the set of messages m that were queried to $\mathtt{Sig}_{\mathsf{sk}}(\cdot)$ by $\mathcal{A}$.

Existential unforgeability under key-only attacks (EUF-KO) is a weaker notion of security for signature schemes but is a useful one to use as a stepping stone to proving EUF-CMA security. Here the adversary is given only the public key of the scheme and no access to a signing oracle.

Definition 2.14 (EUF-KO security)**.** A signature scheme $S = (\mathtt{Setup}, \mathtt{KGen}, \mathtt{Sig}, \mathtt{Vfy})$ is said to be *(quantum) existentially unforgeable under a key-only attack* if, for any (quantum) probabilistic $\mathsf{poly}(\lambda)$-time algorithm $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that:

$$\mathbf{Adv}_{\mathcal{A},S}^{\mathsf{EUF\text{-}KO}} = \Pr\left[\begin{array}{c|c} \mathtt{Vfy}(\mathsf{pk}, m^*, \sigma^*) = 1 & \mathsf{params} \leftarrow \mathtt{Setup}(1^\lambda) \\ \wedge\, m^* \notin \mathcal{Q} & (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{KGen}(1^\lambda) \\ & (m^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{pk}) \end{array}\right] \leq \mathsf{negl}(\lambda)\,.$$

Part I

Authenticated key-exchange

Chapter 3

Refining the BR model

In this chapter, we present Bellare and Rogaway's orginal model [15] for AKE and its adaptation to the public-key setting by Blake-Wilson, Johnson and Menezes [23]. We base our description on Kudla's BJM and mBJM models [93] together with the framework of game-based security definitions developed by Brzuska et al. [32] and formulate the BR definition of authentication in a predicate-based notation. In preparation for Chapters 4 and 5, we also augment the previous formulations of Kudla and Brzuska et al. with appropriate elements to capture forward secrecy, the definition of key-confirmation of Fischlin et al. [69] and our new authentication framework.

We first present how parties and protocols are modeled and then describe how an adversary can interact with instances of parties running sessions of a protocol to attempt to break a given security guarantee. We then discuss different ways in which the model can keep track of which parties exchanged with one another under the direction of the adversary before finally presenting an analogue of the original BR security definition.

We note that by taking all adversaries mentioned in the definitions of this chapter to be quantum algorithms, the security model presented here is inherently post-quantum.

Except where specified, the material in this chapter is the work of M. Fischlin[1], B. Warinschi[2] and this author; it was published in the proceedings of the *33rd IEEE Computer Security Foundations Symposium, CSF 2020*, and presented by this author at the virtual edition of the conference [56]. The additions to previous definitions and reformulations were the results of joint work between all three authors.

[1]Technische Universität Darmstadt, Germany.
[2]University of Bristol, U.K., and later Dfinity, Switzerland.

3.1 Modeling AKE protocols and their participants

We first define formally the notions of participant identities and of AKE protocols. For the former, we propose a new addition to existing models: a flexible mechanism to separate participants who are expected to authenticate themselves during an execution of the protocol—for example this is the case of servers, in opposition to clients, in the setting of Internet browsers. This addition allows for a flexible modeling of the spectrum between no authentication and mutual authentication.

3.1.1 Identities

We let n_i be the number of parties present in the model. Each party has a unique identity i and we denote by $\mathcal{I} = \{i\}$ the set of size n_i of all identities.

In certain scenarios, only one of the two participating parties is expected to authenticate itself. To separate the identities of such parties, we specify a subset $\mathcal{S} \subseteq \mathcal{I}$; this is a first addition to previous BR-style models. The parties that are not expected to provide authentication are modeled by the identities in $\mathcal{I} \setminus \mathcal{S}$, these typically would not hold long-term secret information.

This modeling flexibly captures varying forms of authentication: a secure protocol for which $\mathcal{S} = \mathcal{I}$ provides mutual authentication for all sessions, whereas one for which $\mathcal{S} = \emptyset$ provides no authentication whatsoever; for $\mathcal{S}$ a non-empty proper subset of $\mathcal{I}$, it provides one-way authentication of identities in $\mathcal{S}$. We leave the specification of $\mathcal{S}$ within $\mathcal{I}$ as a design choice for protocols.

In this work, we assume that each party is aware of its own identity during an execution. We furthermore work in the *pre-specified peer model* [103] where each party is also aware of its intended partner's identity before the beginning of a protocol execution.

3.1.2 Authenticated key-exchange protocols

During executions, AKE protocols (which we formally define below in Definition 3.1) run locally at each party in the context of a *local session*. These local sessions are identified by a *local session identifier* $\ell \in \mathcal{I} \times \mathcal{I} \times \mathbb{Z}$ where $\ell = (i, j, k)$ refers to the k-th session of identity i with intended peer j. We use $\ell.\mathsf{id}$ and $\ell.\mathsf{pid}$ to refer to i and j respectively. We let n_s be the maximum value of k for any ℓ in a game.

Each such local session maintains a state in order to respond coherently to messages forming part of an execution. The state of the local session with identifier $\ell = (i, j, k)$

is composed of the following:

- $(\mathsf{pk}_i, \mathsf{sk}_i)$, the long-term key pair of identity i, the "owner" of the session. This is initialized to $\ell.\mathsf{id}$'s key pair if $i \in \mathcal{S}$ or set to $\perp$ if $i \in \mathcal{I} \setminus \mathcal{S}$.
- pk_j, the long-term public key of identity j, the intended peer of the session. This is initialized to $\ell.\mathsf{pid}$'s public key if $j \in \mathcal{S}$ or set to $\perp$ if $j \in \mathcal{I} \setminus \mathcal{S}$.
- $\rho \in \{\mathsf{init}, \mathsf{resp}\}$ is the role of the owner in this session of the protocol, i.e. whether it sent the first message or received a first message from another session.
- $\mathsf{crypt} \in \{0,1\}^*$ is some protocol-specific private session state used to maintain secret values from one invocation to the next.
- $\mathsf{tran} \in \{0,1\}^*$ is the concatenation, with delimiters, of messages received and sent by the session.
- $\mathsf{accept} \in \{\mathsf{true}, \mathsf{false}, \perp\}$ indicates whether the party has *accepted* or *rejected* the session as an successful execution of Π. Initially set to $\perp$, to signify *running*, accept may change to either true or false only upon termination. We assume the value of accept is public.
- $\mathsf{sid} \in \{0,1\}^* \cup \{\perp\}$, the session identifier as specified by the protocol. Initially set to $\perp$, it may be changed once to a non-trivial value. If the sid is different from $\perp$, then accept must be set to true, and vice versa, if accept is set to true, then sid must become different from $\perp$. We assume that the value of sid is made public when accept is set to true. (We further discuss these identifiers and the implications of revealing them in Section 3.3.2.)
- $\mathsf{k} \in \{0,1\}^* \cup \{\perp\}$ is the (session-)key locally derived during the execution. Initially set to $\perp$, it may be changed once to a non-trivial value. If k is different from $\perp$, then accept must be set to true, and vice versa, if accept is set to true then, k must become different from $\perp$. We note that this implies that sessions must terminate with the same call to the protocol as that which sets the k and sid, they cannot continue once k is set.
- $\mathsf{kconf} \in \{\mathsf{full}, \mathsf{almost}, \mathsf{no}, \perp\}$ indicates the form of key confirmation that the owner expects to receive. This addition to the model captures the fact that one partner of a run always terminates first and therefore may not expect a full confirmation of the final session-key. The value of kconf is initialized to $\perp$ and set when the session is first activated.

- $\mathsf{kcid} \in \{0,1\}^* \cup \{\perp\}$ is a key-confirmation identifier, indicating sessions which will eventually derive the same key. Initially set to $\perp$, it may be changed once to a non-trivial value and may not be changed again. If k is different from $\perp$, then kcid must be different from $\perp$.

We write $\mathsf{SST}[\ell] = ((\mathsf{pk}_i, \mathsf{sk}_i), \mathsf{pk}_j, \rho, \mathsf{crypt}, \mathsf{tran}, \mathsf{accept}, \mathsf{sid}, \mathsf{k}, \mathsf{kconf}, \mathsf{kcid})$ to denote the session state of ℓ. We use the notation $\ell.\mathsf{sid}$ or $\ell.\mathsf{k}$ to refer to individual elements and use similar notation for the game, local session or model states which we define below.

Our session state renames some elements present in [32] and augments the state from that work with the kconf and kcid elements from [69]. These are used to modularly capture the formal definition of key confirmation of [69], similarly to the addition of $\mathcal{S}$ to capture different authentication directions.

Given this description of session states, we now state the definition of an authenticated key-exchange protocol.

Definition 3.1 (Authenticated key-exchange protocol)**.** We define an *authenticated key-exchange protocol* as a triple of probabilistic $\mathsf{poly}(\lambda)$-time algorithms, together with the description of two identity sets, $\Pi = (\mathcal{I}, \mathcal{S}, \mathtt{Setup}, \mathtt{KGen}, \zeta)$ such that:

1. The *setup* algorithm $\mathtt{Setup}$ takes in the security parameter 1^λ and outputs a tuple of public parameters params required for the protocol. Among other information, params always contains the parameter 1^λ and specifies a message space $\mathcal{M}$ and a key space $\mathcal{K}$.

2. The *key-generation* algorithm $\mathtt{KGen}$ takes as input the public parameters params and an entity identifier i and outputs an entity-specific public/private key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$.

3. The *protocol function* ζ computes the next step in the protocol execution of session ℓ given a new incoming message m. In doing so, it computes the next message that ℓ should send and also updates internal values within ℓ; i.e. $m' \leftarrow \zeta(\mathsf{SST}[\ell], m)$.

The notion of *correctness* requires that whenever messages are relayed faithfully between two participants, then they both accept and compute identical session keys (except with negligible probability over the randomness used in the algorithms).

In this work we will describe protocols by giving the flows of a single run. A description of the function ζ can be easily inferred. Also, we may abuse notation and write Π (or sometimes Σ) both for the protocol function ζ and the entire protocol which includes key generation, i.e. for $(\mathtt{Setup}, \mathtt{KGen}, \zeta)$.

3.2 Execution environment and experiment states

The concepts of identities, sessions and their states, and protocol execution are part of real-world deployment of AKE protocols. However, to model security definitions and capture categories of adversaries, we require a *security experiment*, sometimes referred to as a security *game*. In such an experiment, a challenger simulates to an adversary an *execution environment* which captures the attack surfaces that the adversary has in the real world. In particular, this environment contains several identities, each capable of running multiple sessions with different intended peers. For each identity and session, the adversary is allowed to obtain information regarding long and short-term secrets, via certain queries.

In order to keep track of this simulated environment, and of which secrets have already been revealed, the challenger maintains several states. These were first introduced in [32] and are as follows:

- LSID: the list of all valid local session identifiers ℓ.
- SST: the list of protocol-related *session state* for each identifier $\ell \in \mathsf{LSID}$.
- LST: the list of game-related *local session state* for each identifier $\ell \in \mathsf{LSID}$.
- EST: the *game execution state* containing global protocol-related information.
- MST: the *model state* containing global game-related information, relevant to the notion being defined (e.g. a hidden bit).

Setup. Simulating an execution environment for a given protocol Π and security definition requires setting up these different states at the beginning. This is captured by the following procedures, run by the challenger at the beginning of security experiments.

- $(\mathsf{SST}, \mathsf{EST}) \leftarrow \mathtt{SetupE}(\mathsf{LSID}, \mathtt{KGen}, 1^\lambda)$: for protocol-relevant components.
- $(\mathsf{LST}, \mathsf{MST}) \leftarrow \mathtt{SetupG}(\mathsf{LSID}, \mathsf{SST}, \mathsf{EST}, 1^\lambda)$: for game-relevant components.

The definition of these procedures forms part of the definition of a security notion since LST and MST contain game-related variables.

3.2.1 Common states for authentication experiments

As many of the authentication security notions we introduce in Chapter 4 share the same execution, game and local states, we present these together here. We note that our notions do not make use of the model state MST. These are in the most part taken from [32], with some renaming, together with some additions.

Game execution state. As in [32], the execution state EST contains a list $\mathcal{L}_{\mathsf{keys}} = \{(i, \mathsf{pk}_i, \mathsf{sk}_i, \delta_i)\}_{i \in \mathcal{S}}$ where i is an identity, $\mathsf{pk}_i, \mathsf{sk}_i$ is the public/secret key pair of i and $\delta_i \in \{\mathsf{honest}, \mathsf{corrupt}\}$ denotes whether sk_i has been corrupted, i.e. leaked to the adversary at some point in the security experiment.

In Brzuska et al.'s work [32], the $\texttt{SetupE}$ algorithm generates $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\!\$\, \texttt{KGen}(1^\lambda)$ for all identities and uses these to initialize EST and $\mathsf{SST}[\ell]$ for every $\ell \in \mathsf{LSID}$. Our $\texttt{SetupE}_{\mathsf{auth}}$ does the same but only generates long-term keys for identities $i \in \mathcal{S}$.

Local session state. The local session state contains modeling information about a particular session, to remember whether long or short-term secrets were revealed to the adversary before or after the session was completed. It consists of:

- $\delta_{\mathsf{ownr}} \in \{\mathsf{honest}, \mathsf{corrupt}\}$: denotes the honesty of the owner of the session before the session was completed (i.e. while $\ell.\mathsf{accept} = \perp$).
- $\delta_{\mathsf{peer}} \in \{\mathsf{honest}, \mathsf{corrupt}\}$: denotes the honesty of the intended peer of the session before the session was completed.
- $\delta_{\mathsf{sess}} \in \{\mathsf{fresh}, \mathsf{revealed}\}$: denotes whether the session-key for this session has been revealed to the adversary after the session was completed.

We write $\mathsf{LST}[\ell] = (\delta_{\mathsf{ownr}}, \delta_{\mathsf{peer}}, \delta_{\mathsf{sess}})$ for the local session state of session ℓ and use the notation $\ell.\delta_{\mathsf{sess}}$ to refer to individual elements. As in [32], keeping track of δ_{ownr} separately from δ_i allows sessions that were completed before their owner was corrupted to still be considered as honest executions; this enables the modeling of *forward secrecy*.

Since our authentication notions do not make use of the model state MST, our $\texttt{SetupG}_{\mathsf{auth}}$ procedure simply initializes each $\mathsf{LST}[\ell]$ as $(\mathsf{honest}, \mathsf{honest}, \mathsf{fresh})$.

3.2.2 Adversarial interaction and common queries

Based on the security parameter λ and descriptions for $\mathcal{I}$ and $\mathcal{S}$, the challenger for a given security notion will first populate the different states described above according to the $\texttt{SetupE}_{\mathsf{auth}}$ and $\texttt{SetupG}_{\mathsf{auth}}$ procedures.

The adversary $\mathcal{A}$ is a probabilistic polynomial-time (PPT) algorithm that then interacts with the execution environment and the sessions it contains using queries specified by a set $\mathcal{Q}$. Upon receiving a query $q \in \mathcal{Q}$, the experiment has a behaviour algorithm χ which takes q together with the current state of the execution environment and returns a response to $\mathcal{A}$. The behaviour χ typically either calls ζ to update session states, updates execution environment information, or communicates protocol information directly to $\mathcal{A}$.

As we wish to obtain "active" security this behaviour allows the adversary to control the running of the protocol between the different sessions. In particular, communication between sessions of the model is controlled by $\mathcal{A}$ which can choose to forward legitimate messages or to insert is own, as well as modify, redirect, delay or erase messages. With different sessions belonging to the same identity, the adversary can engage them in several concurrent executions of the protocol, with the same intended peer or not; all of this using queries specified by $\mathcal{Q}$.

Not every query is always valid; this is captured by a Valid predicate which the challenger evaluates each time a query q is received. Based on q and the current execution state, Valid returns either true or false which determines whether χ is executed on q.

Common queries for authentication experiments

In addition to the common states, our security games for authentication notions also share a query set $\mathcal{Q}_{\mathsf{auth}}$. We specify here the `Send`, `Reveal` and `Corrupt` queries following the work of Brzuska et al. but, as we model key-compromise impersonation (KCI) resistance in addition to usual security, we modify the Valid predicate.

The Send query. Whatever the game, $\mathcal{Q}_{\mathsf{auth}}$ always includes the `Send` query. It takes an identifier $\ell \in \mathsf{LSID}$ and a message $m \in \{0,1\}^*$ as inputs and is processed by χ by running $\zeta(\mathsf{SST}[\ell], m)$. This updates SST and returns a response m' which is given to $\mathcal{A}$ together with the updated value of $\ell.\mathsf{accept}$ if applicable.

If the incoming message $m = \vdash$, this prompts ℓ to initiate a new execution of Π. If the response m' ends with $\dashv$, this indicates that ℓ has completed the session; in this case the value of $\ell.\mathsf{sid}$ is also returned to $\mathcal{A}$.

This gives complete control of the communication network to $\mathcal{A}$ and allows it to forward, alter, delay, create or delete messages.

The Reveal query. When $\mathcal{A}$ submits $\mathtt{Reveal}(\ell)$, χ sets $\ell.\delta_{\mathsf{k}} \leftarrow \mathsf{revealed}$ and returns $\ell.\mathsf{k}$ to $\mathcal{A}$. This query models leakage of session-keys that can take place in the real-world.

The Corrupt query. We formalize the $\mathtt{Corrupt}(i)$ query as follows. First, χ sets the value of δ_i in $\mathcal{L}_{\mathsf{keys}}$ to $\mathsf{corrupt}$. Then, for any session $\ell = (i, *, *) \in \mathsf{LSID}$ for which $\ell.\mathsf{accept} = \perp$, χ sets $\ell.\delta_{\mathsf{ownr}} \leftarrow \mathsf{corrupt}$; similarly for any session $\ell = (*, i, *)$ which is still running, χ sets $\ell.\delta_{\mathsf{peer}} = \mathsf{corrupt}$. Finally, sk_i is returned to $\mathcal{A}$.

The Valid predicate. In Brzuska et al.'s model, the Valid predicate restricts the adversary to make $\mathtt{Send}$ queries only to "un-revealed sessions of honest users where the key has not been accepted or rejected". As a significant difference, our $\mathsf{Valid}_{\mathsf{auth}}$ predicate permits $\mathtt{Send}$ queries to sessions which are still running but whose owner has already been corrupted. This is crucial to model KCI resistance as this notion guarantees a security property to sessions whose owner was corrupted *before* they terminated; this implies that sessions have to continue running after corruption. Furthermore, the Valid predicate returns false if a $\mathtt{Reveal}$ query is made to a session ℓ for which $\ell.\mathsf{k} = \perp$.

3.2.3 Winning condition and security experiment

An experiment considers that the adversary $\mathcal{A}$ has "won", i.e. broken a security property of Π, if it succeeds in triggering a "bad" event. This event is defined by a predicate Pred which is a logical statement evaluated over the state of the execution environment. We denote this by $b \leftarrow \mathsf{Pred}(\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST})$, where $b \in \{0, 1\}$, and $b = 1$ signifies that $\mathcal{A}$ has successfully triggered the "bad" event. We then formally define a generic security experiment as follows.

Definition 3.2 (Authentication game and experiment)**.** A game G is defined by the tuple $(\mathtt{SetupE}_{\mathsf{auth}}, \mathtt{SetupG}_{\mathsf{auth}}, \mathcal{Q}_{\mathsf{auth}}, \mathsf{Valid}_{\mathsf{auth}}, \chi, \mathsf{Pred})$ and runs a challenger that maintains a state $(\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST})$. An experiment $\mathbf{Exp}^{G}_{\mathcal{A},\Pi}$ parameterized by a protocol Π (which includes descriptions of $\mathcal{I}$ and $\mathcal{S}$), a (quantum) probabilistic polynomial-time adversary $\mathcal{A}$ and a game G, is executed as follows.

1. The challenger executes $(\mathsf{SST}, \mathsf{EST}) \leftarrow \mathtt{SetupE}_{\mathsf{auth}}(\mathsf{LSID}, \mathtt{KGen}, 1^{\lambda})$ and $(\mathsf{LST}, \mathsf{MST}) \leftarrow \mathtt{SetupG}_{\mathsf{auth}}(\mathsf{LSID}, \mathsf{SST}, \mathsf{EST}, 1^{\lambda})$.
2. The adversary submits queries from $\mathcal{Q}_{\mathsf{auth}}$ to the challenger which processes them according to $\mathsf{Valid}_{\mathsf{auth}}$ and χ.

3. When $\mathcal{A}$ terminates, $b \leftarrow \mathsf{Pred}(\mathsf{LSID}, \mathsf{SST}, \mathsf{LST}, \mathsf{EST}, \mathsf{MST})$ is evaluated by the challenger which finally returns b as the output of the experiment.

We note that our definition of a game G includes more than [32, Definition 1]; namely we include $\mathcal{Q}_{\mathsf{auth}}$ and Pred, as these also uniquely characterize it.

3.3 Session partnering

In some security experiments there exist trivially successful attack strategies. For example, an adversary could make two sessions execute the protocol with each other, use the `Reveal` query to obtain the key from one of them, and then trivially break any notion of key secrecy for the other session. This must be prohibited by the winning condition defined by Pred; however, this requires a notion of *partnering* to capture which sessions can be said to have engaged with each other in a run of the protocol.

3.3.1 Matching conversations

In the first instance, partnering was captured by the notion of *matching conversations*, defined in the original work of Bellare and Rogaway [15]. This considers two sessions to be *partners* if all messages sent by one have been received by the other, and vice-versa.

Definition 3.3 (Matching conversation)**.** Let $\ell, \ell' \in \mathsf{LSID}$ be two sessions with transcripts $\ell.\mathsf{tran} = \{\vdash, r_1, m_2, r_2, \ldots, m_j, r_j\}$ and $\ell'.\mathsf{tran} = \{m'_1, r'_1, m'_2, r'_2, \ldots, m_k, r_k\}$ such that:

- $m'_i = r_i$ for $i \geq 1$,
- $m_i = r'_{i-1}$ for $i \geq 2$,
- for j even: $r_j = \dashv$ and $k = j - 1$,
- for j odd: $r_k = \dashv$ and $k = j$,
- $\ell.\mathsf{id} = \ell'.\mathsf{pid}$ and $\ell.\mathsf{pid} = \ell'.\mathsf{id}$,

then we say that the sessions ℓ and ℓ' have engaged in a *matching conversation* and that the predicate $\mathsf{BRMatching}(\ell, \ell')$ holds true. We also sometimes say that ℓ and ℓ' are *matching (sessions)*.

This definition has some shortcomings. First, the comparison of the messages exactly as they were sent and received does not leave room for a more detailled analysis of the content that truly matters for identifying a given exchange. For example, if a protocol tolerates the re-randomization of messages then the adversary could easily break the partnership of two sessions that otherwise exchanged the same identifying information.

Secondly, the last criteria of the definition requires that the two sessions agree on their expected identities. While this is a condition that must be imposed on protocols for them to provide authentication guarantees, including it here leads to different security notions being conflated, making them harder to analyse separately.

3.3.2 Session identifiers

Another notion to partner two sessions as having engaged in the same execution is that of *session identifiers*, originally proposed by Bellare, Pointcheval and Rogaway [14]. These are computed by the protocol itself, and are not explicitly linked to the messages exchanged between parties; thus the protocol is free to include whatever information is deemed necessary to the identification of the session. (We note that these are different from the *local* session identifier ℓ which is an artifact of the model.)

As they are defined separately from the transcript of each session, we see that these session identifiers address the shortcomings of the notion of matching conversations that we outlined above.

Beyond their abstract use in the modeling of the execution environment, the session identifiers of [14] were intended for parties to identify the session keys for further use.

Brzuska et al. note in [32, Section 3] that a weak form of (public) session matching (i.e. a mechanism by which an eavesdropper can identify which sessions are partnered together) is necessary to ensure certain composition results. They point out, however, that the original practice in [14] of giving the identifier to the adversary upon acceptance of the session straightforwardly yields such a matching mechanism. We maintain this original approach of making these session identifiers public upon acceptance, thus preventing them from containing any secret information.

Definition 3.4 (Partners)**.** We say that two sessions ℓ and ℓ' are *partners* if the predicate $\mathsf{Partner}(\ell, \ell')$ holds true, where

$$\mathsf{Partner}(\ell, \ell') \iff \big[(\ell \neq \ell') \wedge (\ell.\mathsf{sid} = \ell'.\mathsf{sid} \neq \bot)\big].$$

Thus, to be partners, two sessions need to be administratively different in the model and to both have set a non-trivial sid. This does not exclude the possibility that they belong to the same identity, i.e. that $\ell.\mathsf{id} = \ell'.\mathsf{id}$.

For correctness, we require that two sessions executing Π without adversarial interaction derive identical sids upon accepting and are therefore partnered. We also require that two such sessions derive identical keys, k, and kcids.

While this definition of partnering appears similar to that of other "matching" sessions in CK-like models [35, 92, 96], it differs in two important aspects. The first is that it does not involve the sessions' identities, thus separating out authentication notions. The second is that our sids are derived by the protocol itself rather than arbitrarily set by a higher protocol layer. Furthermore, as mentioned above, our notion of sids supersedes that of matching conversations which is also used for partnering sessions in [92] and [96].

3.4 The original Bellare–Rogaway definition

In the original work of Bellare and Rogaway [15], security for AKE protocols is defined in two steps: first the protocol needs to provide *mutual authentication*, and secondly it needs to provide *key secrecy.*

3.4.1 Secure mutual authentication

Entity authentication is captured in the original BR model using the **No-Matching** event. This is triggered if an adversary manages to make a session accept without a matching session, under the notion of matching conversations of Definition 3.3.

To work this event into our framework, we define the BRmAuth predicate to capture the negation of the **No-Matching** event. Thus this new predicate holds true if and only if the event is not triggered. Here the aim of $\mathcal{A}$ is to make a session accept without having perfectly relayed the messages to and from its intended partner, and to do so *without corrupting either party* at any time. This restriction on the corruptions that $\mathcal{A}$ is able to perform introduces the notion of *freshness.*

Session freshness

A session in the execution environment is considered *fresh* for a particular purpose if it satisfies certain "health" criteria. Typically these criteria prevent the adversary from submitting queries that would allow it to win the security game trivially.

In the case of mutual authentication, Blake-Wilson, Johnson and Menezes [23] chose to exclude sessions whose owner or peer was corrupted at any time during the execution in their adaptation of the BR model to the asymmetric setting. This can be captured in our framework with the $\mathsf{BRaFresh}$ predicate.

Definition 3.5 (BR authentication freshness)**.** For any session $\ell \in \mathsf{LSID}$, the *Bellare–Rogaway authentication freshness* predicate $\mathsf{BRaFresh}(\ell)$ evaluates to 1 if and only if

$$\delta_{\ell.\mathsf{id}} = \delta_{\ell.\mathsf{pid}} = \mathsf{honest}.$$

Note that this is different from requiring $\ell.\delta_{\mathsf{ownr}} = \ell.\delta_{\mathsf{peer}} = \mathsf{honest}$ as the latter only speaks to the corruption status of the session *at the time of* the execution of ℓ. This would not be equivalent to the definition of Blake-Wilson et al. which requires that the parties remain honest during the whole execution of the protocol environment.

Original BR definition

Using the freshness condition above then gives us the following definition for the original BR authentication experiment.

Definition 3.6 (BR authentication security game)**.** The *BR mutual authentication* predicate $\mathsf{BRmAuth}$ evaluates to 1 if and only if

$$\forall \ell \in \mathsf{LSID}, (\ell.\mathsf{accept} \wedge \mathsf{BRaFresh}(\ell)) \implies \exists \ell' \in \mathsf{LSID} : \mathsf{BRMatching}(\ell, \ell').$$

The *BR mutual authentication* game G_{BRmAuth} is then defined as in Definition 3.2 with $\mathsf{Pred} = \mathsf{BRmAuth}$.

We denote $\mathcal{A}$'s advantage in the $\mathsf{BRmAuth}$ security game as

$$\mathbf{Adv}^{\mathsf{BRmAuth}}_{\mathcal{A},\Pi} = \Pr\left[\mathbf{Exp}^{\mathsf{BRmAuth}}_{\mathcal{A},\Pi} = 1\right].$$

and below we state the original (albeit reworded) definition of *secure mutual authentication* which includes a requirement for correctness.

Definition 3.7 (Secure mutual authentication)**.** An AKE protocol $\Pi = (\mathcal{I}, \mathcal{S}, \mathtt{Setup}, \mathtt{KGen}, \Pi)$ provides *(post-quantum) secure mutual authentication* if the following hold.

- ($\mathsf{Matching\ conversations} \implies \mathsf{Acceptance}$) If two sessions ℓ and ℓ' have matching conversations in the sense of Definition 3.3, then both sessions accept.
- ($\mathsf{Acceptance} \implies \mathsf{Matching\ conversations}$) For all probabilistic $\mathsf{poly}(\lambda)$-time (quantum) adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that $\mathbf{Adv}^{\mathsf{BRmAuth}}_{\mathcal{A},\Pi} \leq \mathsf{negl}(\lambda)$.

3.4.2 Session-key forward secrecy

After defining secure mutual authentication, Bellare and Rogaway then provide an experiment for key secrecy. This follows the now-standard indistinguishability paradigm where security is achieved when an adversary cannot distinguish a true key from one sampled independently at random.

We capture the original notion with the BR-secrecy game $G_{\mathsf{BRSec},D}$, where D denotes the distribution of the session-keys output by an honest run of the protocol. As this secrecy notion differs fundamentally from the authentication notion we presented above, we now present the additional modeling components required in our security experiment framework.

Modeling components

This game uses the same execution, session and local session states as the default ones describes in Sections 3.1.2 and 3.2.1; thus the $\texttt{SetupE}_{\mathsf{auth}}$ algorithm is also the same here. However, the model state used for game-specific components is different. Here MST contains two bits, $b_{\mathsf{test}} \in \{0,1\}$ and $b_{\mathsf{guess}} \in \{0,1\}$ as well as a session identifier $\ell_{\mathsf{test}} \in \mathsf{LSID}$; these will capture the real-or-random challenge submitted to the adversary. To interact with this, the adversary also has access to two new queries: `Test` and `Guess`.

- The $\texttt{Test}(\ell)$ query sets $\ell_{\mathsf{test}} \leftarrow \ell$ and returns either $\mathsf{k} = \ell.\mathsf{k}$ if $b_{\mathsf{test}} = 1$ or $\mathsf{k} \leftarrow_{\$} D$ otherwise. This query enables $\mathcal{A}$ to choose the session for which it must distinguish a real key from a random one.
- The $\texttt{Guess}(b)$ query sets $b_{\mathsf{guess}} \leftarrow b$ to record $\mathcal{A}$'s guess for the hidden value of b_{test}.

Given the usage made of the new components $b_{\mathsf{test}}, b_{\mathsf{guess}}$ and ℓ_{test}, we let the $\texttt{SetupG}_{\mathsf{sec}}$ algorithm initialize them as $b_{\mathsf{test}} \leftarrow_{\$} \{0,1\}$, $b_{\mathsf{guess}} \leftarrow \perp$ and $\ell_{\mathsf{test}} \leftarrow \perp$.

With the addition of these new queries, we must specify how the $\mathsf{Valid}_{\mathsf{sec}}$ predicate processes them. It requires that only one `Test` query is made, and to a session that has already derived a session-key, and that also only one `Guess` query is made. At this moment, we note that nothing yet prevents the adversary from submitting queries that will allow him to trivially win the game, such as submitting $\texttt{Reveal}(\ell)$ followed by $\texttt{Test}(\ell)$ and comparing the two keys obtained. We therefore must impose *freshness* requirements for this experiment too.

BR secrecy freshness and forward secrecy

Definition 3.8 (BR secrecy freshness)**.** For any session $\ell \in \mathsf{LSID}$, the *Bellare–Rogaway secrecy freshness* predicate $\mathsf{BRsFresh}(\ell)$ evaluates to 1 if and only if

$$\begin{aligned} &\ell.\delta_{\mathsf{k}} = \mathsf{fresh} \\ \wedge\, &\ell.\delta_{\mathsf{ownr}} = \ell.\delta_{\mathsf{peer}} = \mathsf{honest} \\ \wedge\, &\forall \ell' \in \mathsf{LSID}, \mathsf{BRMatching}(\ell, \ell') \implies \ell'.\delta_{\mathsf{k}} = \mathsf{fresh}. \end{aligned}$$

Note that this definition does not require ℓ to have a matching partner. The session is still considered to be fresh even if the adversary has managed to make ℓ accept by generating and sending messages by itself.

In addition, and in contrast to the $\mathsf{BRaFresh}$ condition, a session only needs to be "secrecy fresh" during its execution; after it has accepted, the adversary can corrupt the parties involved. This change of condition on the corruption of the parties captures forward secrecy exactly: the secrecy should be preserved even if the long-term keys are lost after the session has completed. The only restrictions on future queries, ensured by the $\mathsf{Valid}_{\mathsf{sec}}$ predicate, is that $\mathcal{A}$ may not query $\mathtt{Reveal}(\ell_{\mathsf{test}})$ or $\mathtt{Reveal}(\ell')$ with $\mathsf{BRMatching}(\ell_{\mathsf{test}}, \ell') = \mathsf{true}$.

BR forward secrecy.

To capture the secrecy experiment originally described by Bellare and Rogaway, we define the BRSec predicate below. Due to the distinguishing nature of the secrecy challenge, this predicate does not evaluate to 1 if a certain condition holds; instead it evaluates to the value of $\mathsf{MST}.b_{\mathsf{guess}}$.

Definition 3.9 (BR forward secrecy)**.** The *BR secrecy* predicate BRSec evaluates to $\mathsf{MST}.b_{\mathsf{guess}}$ if and only if

$$\mathsf{MST}.\ell_{\mathsf{test}} \neq \perp \wedge \mathsf{BRsFresh}(\mathsf{MST}.\ell_{\mathsf{test}})$$

and evaluates to $\perp$ otherwise. The security game $G_{\mathsf{BRSec},D}$ is then defined as in Definition 3.2 with $\mathsf{Pred} = \mathsf{BRSec}$, together with $\mathtt{SetupE}_{\mathsf{auth}}$, $\mathtt{SetupG}_{\mathsf{sec}}$, $\mathcal{Q}_{\mathsf{sec}} = \mathcal{Q}_{\mathsf{auth}} \cup \{\mathtt{Test}, \mathtt{Guess}\}$ and $\mathsf{Valid}_{\mathsf{sec}}$. We also denote by $G^{b_{\mathsf{test}}}_{\mathsf{BRSec},D}$ the security game initialised with a specific value for b_{test}.

We say that an AKE protocol $\Pi = (\mathcal{I}, \mathcal{S}, \mathtt{Setup}, \mathtt{KGen}, \Pi)$ provides *(post-quantum) forward secrecy (w.r.t. key distribution D)* if, for all probabilistic $\mathsf{poly}(\lambda)$-time (quantum) adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{\mathsf{BRSec}} = \left| \Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{G^0_{\mathsf{BRSec},D}} = 1 \right] - \left[\mathbf{Exp}_{\mathcal{A},\Pi}^{G^1_{\mathsf{BRSec},D}} = 1 \right] \right| \leq \mathsf{negl}(\lambda)\,.$$

3.4.3 Full BR security definition

Finally we combine both the notions of mutual authentication and session key forward secrecy into a single security definition.

This is captured in our model as the adversary is allowed, before it submits a `Guess` query, to submit `Corrupt` queries to the identities that took part in the test session, as captured by the BRsFresh predicate. With that possibility in mind, we still require that its advantage in the BRSec experiment remains negligible. Thus, proving that an AKE protocol satisfies this definition of security also proves that it possesses forward secrecy, in which case we say it is a *forward-secure AKE protocol.* Additionally, our definition also captures the usual properties of AKE protocols such as *session-key reveal secrecy* and *third-party compromise security.*

Definition 3.10 (BR AKE security)**.** An AKE protocol Π with key distribution D is *(post-quantum) BR-(forward-)secure* if, for all probablisitic $\mathsf{poly}(\lambda)$-time (quantum) adversaries $\mathcal{A}$, the following conditions hold.

1. If messages are relayed faithfully (by a benign or an active adversary) between two sessions, then both sessions accept holding identical session keys, and each session's key is distributed according to D.

2. Π is a (post-quantum) secure mutual authentication protocol (Definition 3.7).

3. Π provides (post-quantum) forward secrecy w.r.t. key distribution D (Definition 3.9).

Chapter 4

New definitions of authentication

While the original BR definition of security for AKE protocols captures the most important notions, namely entity authentication and key secrecy, it is not very detailled and is not suited to more detailed analysis. It also conflates certain properties which are then hard to analyze independently from one another. As described in Section 1.1.1, this trend was further exacerbated as authentication-specific requirements were removed from security definitions in later models. In this chapter we present a comprehensive study of the various forms of authentication in AKE protocols and present a formal definition for each of them. We also discuss how different combinations of properties relate to each other through implications and/or equivalence relations.

We first focus on the notion of Match-security which captures correctness against active adversaries; as this was where authentication notions were often covered in previous models, we discuss how our definition relates to these. We then study key authentication where three distinct flavours emerge: implicit, confirmation and explicit; we discuss each in turn and also demonstrate some relations between them and with Match- and BR-security. Next we study entity authentication and describe how it both differs from and is similar to key authentication. Here again the same three flavours emerge albeit with less meaning than their counterparts related to keys. We also prove different relations between notions of key and entity authentication. We next discuss how our notions can adapt to models of AKE security that consider stronger adversaries, such as the model of Canetti and Krawczyk (CK model) [35].

In the last two sections of this chapter, we prove the folklore result that an implicitly authenticated key which is then validated through secure use in fact provides explicit authentication to its users. To do so we first define the class of key-confirming protocol and then prove that the composition of the two attains the desired notion.

As for the previous chapter, the contents of this chapter have been selected from the paper published at CSF [56]; the ideas and strategies were the joint result of this author and his two co-authors. Except where specified, the discussions and proofs are the work of this author in the most part.

4.1 Match security

In this section we build on the work of [32] to separate a first notion, that of *match security*, which can be intuitively described as guarantee of correctness even in the presence of active adversaries, but without any guarantees of authentication. This removal of the authentication guarantees provided by match security is a new component of this work; previous definitions of this notion include requirements that matching sessions agree on each other's identities in some form.

The first step for a more fine-grained analysis of authentication is to no longer use the $\mathsf{BRMatching}$ predicate for session partnering and instead use the more generic $\mathsf{Partner}$ predicate. We also introduce components of newer models, such as key-confirmation identifiers, which were not used in the previous chapter. Given this, a Match-secure AKE protocol should ensure that:

1. Partner sessions derive the same k and kcid (properties 4.1 and 4.2 below);
2. At most two sessions derive the same sid (property 4.3);
3. Sessions with the same kcid accept with the same k (property 4.4).

This guarantees disagreements cannot be created between partnered sessions and that the AKE protocol in question is *robust* against such attacks. Formally, we define the following predicate.

Definition 4.1 (Match predicate)**.** The Match predicate evaluates to 1 iff $\forall \ell, \ell', \ell'' \in \mathsf{LSID}$,

$$\begin{aligned}
&\big(\mathsf{Partner}(\ell,\ell') \wedge \ell.\mathsf{k} \neq \perp \neq \ell'.\mathsf{k}\big) \implies \mathsf{Samekey}(\ell,\ell') && (4.1)\\
\wedge\ &\big(\mathsf{Partner}(\ell,\ell') \wedge \ell.\mathsf{kcid} \neq \perp \neq \ell'.\mathsf{kcid}\big) \implies \mathsf{Samekcid}(\ell,\ell') && (4.2)\\
\wedge\ &\big(\mathsf{Partner}(\ell,\ell') \wedge \mathsf{Partner}(\ell,\ell'')\big) \implies \ell' = \ell'' && (4.3)\\
\wedge\ &\big(\mathsf{Samekcid}(\ell,\ell') \wedge \ell.\mathsf{k} \neq \perp \neq \ell'.\mathsf{k}\big) \implies \mathsf{Samekey}(\ell,\ell'). && (4.4)
\end{aligned}$$

where $\mathsf{Samekey}(\ell, \ell') \iff [\ell' \neq \ell \wedge \ell'.\mathsf{k} = \ell.\mathsf{k} \neq \perp]$ and $\mathsf{Samekcid}(\ell, \ell')$ is defined analogously.

We then define the Match security game G_{Match} in the sense of Definition 3.2 with $\mathsf{Pred} = \mathsf{Match}$. The advantage of an adversary $\mathcal{A}$ against Π in the game G_{Match} is written as

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{G_{\mathsf{Match}}} = \Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{G_{\mathsf{Match}}} = 0\right].$$

Definition 4.2 (Match security)**.** An AKE protocol Π is Match-secure if, for all probabilistic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that $\mathbf{Adv}_{\mathcal{A},\Pi,\mathcal{I},\mathcal{S}}^{G_{\mathsf{Match}}} = \mathsf{negl}(\lambda)$.

4.1.1 Comparison to previous definitions.

Considered in the context of the BR model, our definition of Match security refines that of [32] in two ways. First we incorporate the conditions of the $\mathsf{KCIDbind}$ predicate of [69] as conditions (4.2) and (4.4). As this work builds a unified model of AKE protocols with both authentication and key confirmation, it is reasonable to add this predicate to the definition of Match since it concerns notions of correctness and soundness, as the previous definition already did for keys and partnering.

Secondly, we remove the requirement that partnered sessions should agree on each other's identities (which was also present in the original $\mathsf{BRMatching}$ predicate). This condition would imply that Match-secure AKE protocols already provided some form of authentication, albeit very weak. This mixes an authentication with design and soundness and we therefore remove it here. We present separate definitions for authentication in the next sections.

In the context of the CK-style models, as summarized in [49], the usual first requirement of security is for matching sessions of honest identities to derive equal keys. We note that our notion of Match security would capture and extend this requirement if a `StateReveal` query were added to its game. We note that we do not restrict to honest identities, as we do not consider that the adversary takes control of sessions ℓ managed by the game; thus our Match predicate is only evaluated for honestly-behaving sessions.

4.2 Key authentication and confirmation

Next we present new predicate-based definitions for *key authentication* notions. We define three distinct flavours: *implicit*, *confirmation* and *explicit*. For each, we first discuss the intuitive understanding that motivates our definition and then give a formal statement. We then show that our definitions are consistent with each other by proving that a protocol that combines implicit key authentication and key confirmation also provides explicit key authentication.

4.2.1 Implicit authentication

We take *implicit* to mean: “should there be a session ℓ' that holds the same key as session ℓ, then the owner of session ℓ' *must* be the identity designated as the peer of session ℓ.” Equivalently, this means that any session whose owner is *not* designated by the peer of ℓ should not be able to derive the same session-key. (Recall that the term “session” refers to sessions executed by the model and that this does not forbid the adversary from deriving the key itself; this therefore does not guarantee the secrecy of the key. It also doesn’t even guarantee the existence of such a session ℓ' when ℓ eventually terminates.)

This informal notion of implicit authentication raises the question whether to only consider sessions which interact with an honest peer, or to also allow those with a corrupted peer (unbeknown to the owner). The impact of this distinction was first observed by Diffie et al. [61]. Their design of the station-to-station (STS) protocol aimed to prevent an attack in which one can make an honest identity B believe it is sharing a key with a malicious E, whereas the actual other honest key holder A intends to communicate with B. This later took on the name of an *unknown key-share (UKS)* attack [24] which, ironically, was shown to apply to the STS protocol in the same work [24].

For our formal definition, the question is then either to restrict the adversary’s valid targets to the sessions that were executed with an honest peer, or to allow all sessions as valid targets, even those that accepted with a corrupted peer. The first choice would comply with the idea stated in [24] that “the provision of implicit key authentication is only considered in the case where B engages in the protocol with an honest entity (which E isn’t).” The second choice, would instead lead to a definition where UKS attack scenarios, even with dishonest peers, are accounted for. Such a scenario would, for example, include a corrupt server causing a client to exchange a key with another, unintended, server. It is clear that this second formulation yields a stronger security guarantee and it is the one we choose to capture in our definitions.

We stress, furthermore, that our model also captures *key-compromise impersonation* (KCI) resistance [23]. This property guarantees that an adversary that knows the long-term key of a party is still unable to impersonate another identity to them. Such resistance is captured in our formal definition below since it also allows the *owner* of target sessions (and not only their intended peers) to be corrupted.

Definition 4.3 (Implicit key authentication)**.** The $\mathsf{iKeyAuth}$ predicate evaluates to 1 if and only if

$$\forall \ell \in \mathsf{LSID}, (\ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{accept}) \implies \forall \ell' \in \mathsf{LSID}, \big(\mathsf{Samekey}(\ell', \ell) \implies \ell'.\mathsf{id} = \ell.\mathsf{pid}\big)\,,$$

where $\ell.\mathsf{accept}$ holds true if and only if $\ell.\mathsf{accept} = \mathsf{true}$. We then say that the AKE protocol Π provides *implicit key authentication* if, for all probabilistic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{G_{\mathsf{iKeyAuth}}} = \Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{G_{\mathsf{iKeyAuth}}} = 0\right] = \mathsf{negl}(\lambda)\,,$$

where G_{iKeyAuth} is the same as G of Definition 3.2 with $\mathsf{Pred} = \mathsf{iKeyAuth}$.

Note that the $\mathsf{iKeyAuth}$ predicate only applies to sessions that expect authentication (captured by the condition that $\ell.\mathsf{pid} \in \mathcal{S}$). This models the fact that one can only provide authentication to keys if one possesses authenticating information.

An artefact of this formal definition is that protocols without authenticating information (i.e. with $\mathcal{S} = \emptyset$) strictly speaking provide implicit key authentication as there is no authenticating party which the adversary can attack. Mathematically, this corresponds to a quantification over the empty set.

On the other end of the spectrum, we see that the case where $\mathcal{S} = \mathcal{I}$ rejoins mutual authentication where, upon completion, any session ℓ has authenticated to session ℓ' and vice-versa. Indeed, if $\ell.\mathsf{id} \in \mathcal{S}$ and also $\ell.\mathsf{pid} \in \mathcal{S}$, then both sessions expect to receive authentication and are also expected to provide it. Upon both sessions completing, the predicate therefore induces a symmetry in the authentication guarantees which we term *mutual implicit key authentication.*

As explained above the definition, this first flavour of key authentication considers the strongest attack model for the adversary. We remark that we could strengthen the above requirement and only consider sessions ℓ with an honest peer at the time of session execution, i.e. with $\ell.\delta_{\mathsf{peer}} = \mathsf{honest}$. Indeed, this notion sometimes appears in the literature: it still provides guarantees for parties who engage in sessions of the protocol with an honest peer as intended partner but it also neglects executions in which the intended peer is dishonest in which case one could be vulnerable to certain UKS attacks.

4.2.2 Key confirmation

The second notion of *key confirmation* is, intuitively, "the guarantee that, when a session ℓ terminates, another session ℓ' holds the same key." While this does not provide authentication in the sense of binding an identity to a key, we define it here because its existential guarantee is a link between implicit and explicit authentication.

In contrast to implicit authentication, we note here that key confirmation only makes sense for honest peers. This is because an adversary impersonating an honest party (by knowing its authenticating information) can always compute the key and provide confirmation to the target session on its own. To prevent this trivial attack, we must introduce a freshness condition as we did for the original BR notions of authentication and secrecy in Section 3.4. This will remove any session whose peer was corrupted at the time of the execution from the set of valid target sessions.

Definition 4.4 (Authentication freshness)**.** For any $\ell \in \mathsf{LSID}$, $\mathsf{aFresh}(\ell)$ evaluates to $\mathtt{true}$ if and only if

$$\ell.\delta_{\mathsf{peer}} = \mathsf{honest}.$$

We note that this freshness notion does not prohibit `Reveal` queries (which only affect $\ell.\delta_{\mathsf{sess}}$); this is because key authentication properties are expected to hold upon *derivation* of the key, and knowledge of other sessions' keys should not help the adversary in breaking these.

Furthermore, we show that the `Reveal` query is not useful for the adversary to wrongfully provide confirmation. Two cases are possible: either another session derives the same key and the adversary reveals it, but then another session with the same key does exist, and therefore confirmation holds even though it does with the adversary's intervention; or the adversary reveals the target session itself. However, the second option is somewhat of a tautology, as, in our model, setting the key to a non-trivial value is synonymous with accepting and terminating. Therefore the adversary cannot submit a `Reveal` query before the session has already accepted, at which point the adversary has already won if the session does not share a key with any other. Hence our freshness predicate does not need to eliminate trivial attacks using the `Reveal` query.

In their work defining key-confirmation, Fischlin et al. [69] introduced the distinction between *full* and *almost-full* key confirmation which captures the differences in guarantees that the last sender and last receiver in an AKE session can expect. We present here their definitions and refer to [69] for a discussion. The first says that if an "authentication fresh" session with full key confirmation accepts, then there must be at least one other session holding the same key.

Definition 4.5 (Full key confirmation)**.** The $\mathsf{fKeyConf}$ predicate evaluates to 1 if and only if

$$\forall \ell \in \mathsf{LSID}, \begin{pmatrix} & \mathsf{aFresh}(\ell) \\ \wedge & \ell.\mathsf{kconf} = \mathsf{full} \\ \wedge & \ell.\mathsf{pid} \in \mathcal{S} \\ \wedge & \ell.\mathsf{accept} \end{pmatrix} \implies \exists \ell' \in \mathsf{LSID} :: \mathsf{Samekey}(\ell', \ell).$$

We then say that the AKE protocol Π provides *full key confirmation* if, for all probabilistic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{G_{\mathsf{fKeyConf}}} = \Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{G_{\mathsf{fKeyConf}}} = 0\right] = \mathsf{negl}(\lambda),$$

where G_{fKeyConf} is defined similarly to G_{iKeyAuth}.

We see that the session's expected level of key confirmation is captured by the condition that $\ell.\mathsf{kconf} = \mathsf{full}$ — for a given protocol Π, a session can decide which key confirmation to expect if it is activated as an initiator or a responder. Also, the session ℓ in question is excluded from the existence condition by the $\mathsf{Samekey}$ predicate and a session therefore cannot confirm its own key. We note that $\mathsf{fKeyConf}$ is only tested against sessions that expect authentication, with $\ell.\mathsf{pid} \in \mathcal{S}$, as is discussed in [69, Section III.D]. While this condition is not strictly required in the predicate for our result in Section 4.2.4, we adopt it here to align ourselves on the stand-alone definition of key confirmation.

As pointed out by Fischlin et al. [69], almost-full key confirmation is delicate to define. We adopt their notion saying that if an aFresh session accepts, then there must be a another session holding the same key-confirmation identifier and, moreover, if that other session has already derived a key, then it is the same one as the original session.

Definition 4.6 (Almost-full key confirmation)**.** The $\mathsf{afKeyConf}$ predicate is defined as

$$\forall \ell \in \mathsf{LSID}, \left(\begin{array}{ll} & \mathsf{aFresh}(\ell) \wedge \ell.\mathsf{kconf} = \mathsf{almost} \\ \wedge & \ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{accept} \end{array} \right) \implies$$
$$\exists \ell' \in \mathsf{LSID} :: \left(\begin{array}{ll} & \mathsf{Samekcid}(\ell', \ell) \\ \wedge & \left[\ell'.\mathsf{k} \neq \bot \implies \mathsf{Samekey}(\ell', \ell)\right] \end{array} \right).$$

We then say that the AKE protocol Π provides *almost-full key confirmation* if, for all probabilistic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{G_{\mathsf{afKeyConf}}} = \Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{G_{\mathsf{afKeyConf}}} = 0\right] = \mathsf{negl}(\lambda)$$

where $G_{\mathsf{afKeyConf}}$ is defined similarly to G_{iKeyAuth}.

4.2.3 Explicit authentication

This third notion is much stronger than the first and intuitively rejoins the original BR definition of mutual authentication. Indeed, we take *explicit* to mean that authentication is obtained *at termination*, and therefore it does not rely on the potential use of the key at a later time. In other words, a session ℓ, upon accepting, knows that there is another session ℓ' which holds the same key and whose identity is bound to it. Intuitively, it is a combination of implicit authentication and key confirmation, and indeed it was informally defined as such in Menezes, van Oorschot and Vanstone's *Handbook of Applied Cryptography* [104]; this intuition is reflected in the predicates below.

Similarly to key confirmation, the existence of a session which has already derived the same key cannot always be guaranteed due to the asymmetry of the final message flow in AKE protocols. We therefore define the two analogous notions of *full* and *almost-full explicit key authentication.*

As before, we study the requirements of a freshness predicate. As in the case of implicit authentication, we do not stipulate that the peer should be honest for the target session when it comes to the condition that any partner holding the same key is correctly identified ($\forall \ell' \in \mathsf{LSID}, \mathsf{Samekey}(\ell', \ell) \implies \ell'.\mathsf{id} = \ell.\mathsf{pid}$). This again provides safety against all possible UKS attacks. Only for the "liveness" condition (i.e. that there exists a party with the same key) do we require that the intended peer is honest ($\mathsf{aFresh}(\ell) \implies \exists \ell' \in \mathsf{LSID} :: \mathsf{Samekey}(\ell', \ell)$); otherwise the session may have communicated with an impersonating adversary which could trivially compute the key. Similarly to key confirmation, the `Reveal` query would not enable the adversary to conduct trivial attacks; this implies that the aFresh predicate defined in the context of key confirmation is also the correct one here.

In summary, full explicit key authentication demands that for any fresh accepting session, any other session deriving the same key has the correct identity and there exists at least one other session holding the same key, if the peer is honest.

Definition 4.7 (Full explicit key authentication)**.** The $\mathsf{fexKeyAuth}$ predicate evaluates to 1 if and only if

$$\forall \ell \in \mathsf{LSID}, \begin{array}{l} \ell.\mathsf{pid} \in \mathcal{S} \wedge \\ \ell.\mathsf{kconf} = \mathsf{full} \wedge \\ \ell.\mathsf{accept} \end{array} \Rightarrow \begin{array}{l} (\forall \ell' \in \mathsf{LSID},\ \mathsf{Samekey}(\ell', \ell) \Rightarrow \ell'.\mathsf{id} = \ell.\mathsf{pid}) \wedge \\ (\mathsf{aFresh}(\ell) \Rightarrow \exists \ell' \in \mathsf{LSID} :: \mathsf{Samekey}(\ell', \ell)). \end{array}$$

We then say that the AKE protocol Π provides *full explicit key authentication* if, for all probabilistic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{G_{\mathsf{fexKeyAuth}}} = \Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{G_{\mathsf{fexKeyAuth}}} = 0\right] = \mathsf{negl}(\lambda)$$

where $G_{\mathsf{fexKeyAuth}}$ is defined similarly to G_{iKeyAuth}.

We see that a session's expectation of both authentication and confirmation appears as $\ell.\mathsf{pid} \in \mathcal{S}$ and $\ell.\mathsf{kconf} = \mathsf{full}$ in the predicate. Similarly to key confirmation, we define the almost-full variant of explicit key authentication.

Definition 4.8 (Almost-full explicit key authentication)**.** The predicate $\mathsf{afexKeyAuth}$ evaluates to 1 if and only if

$$\forall \ell \in \mathsf{LSID}, (\ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{kconf} = \mathsf{almost} \wedge \ell.\mathsf{accept}) \implies \left(\begin{array}{ll} & \forall \ell' \in \mathsf{LSID}, \mathsf{Samekey}(\ell', \ell) \implies \ell'.\mathsf{id} = \ell.\mathsf{pid} \\ \wedge & \mathsf{aFresh}(\ell) \Rightarrow \exists \ell' \in \mathsf{LSID} :: \left(\begin{array}{ll} & \mathsf{Samekcid}(\ell', \ell) \\ \wedge & (\ell'.\mathsf{k} \neq \perp \implies \mathsf{Samekey}(\ell', \ell)) \end{array} \right) \end{array} \right).$$

We then say that the AKE protocol Π provides *almost-full explicit key authentication* if, for all probabilistic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$, there exists a negligible function

$\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{G_{\mathsf{afexKeyAuth}}} = \Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{G_{\mathsf{afexKeyAuth}}} = 0\right] = \mathsf{negl}(\lambda)$$

where $G_{\mathsf{afexKeyAuth}}$ is defined similarly to G_{iKeyAuth}.

4.2.4 Equivalence results

We now formally prove the coherence of the authentication definitions presented above. Namely, we show that a protocol which satisfies both implicit key authentication and key confirmation also satisfies explicit key authentication, and we show that the converse holds.

Theorem 4.9. *Let* Π *be an AKE protocol; it holds for* Π *that*

$$\mathsf{iKeyAuth} \land \mathsf{fKeyConf} \iff \mathsf{fexKeyAuth}, \tag{4.5}$$

$$\mathsf{iKeyAuth} \land \mathsf{afKeyConf} \iff \mathsf{afexKeyAuth}. \tag{4.6}$$

Proof. We first focus on equation (4.5) and show that $\mathsf{iKeyAuth} \land \mathsf{fKeyConf} \implies \mathsf{fexKeyAuth}$; we proceed by proving the contrapositive. Let $\mathcal{A}$ be a successful adversary against the $\mathsf{fexKeyAuth}$ predicate; i.e. $\mathcal{A}$ reaches an execution state where $\neg\mathsf{fexKeyAuth}$ holds true. This is equivalent to

$$\begin{aligned}\exists \ell^* \in \mathsf{LSID} :: \ell^*.\mathsf{pid} \in \mathcal{S} \land \ell^*.\mathsf{kconf} = \mathsf{full} \land \ell^*.\mathsf{accept} \\ \land \left(\begin{array}{ll} & \exists \ell' :: \mathsf{Samekey}(\ell', \ell^*) \land \ell'.\mathsf{id} \neq \ell^*.\mathsf{pid} \\ \lor & \mathsf{aFresh}(\ell^*) \land \forall \ell'', \neg\mathsf{Samekey}(\ell'', \ell^*) \end{array}\right)\end{aligned} \tag{4.7}$$

Thus if $\neg\mathsf{fexKeyAuth}$ holds true, either the first expression of the OR clause holds, which implies $\neg\mathsf{iKeyAuth}$, or the second one holds and implies $\neg\mathsf{fKeyConf}$. We therefore obtain that

$$\neg\mathsf{fexKeyAuth} \implies \neg\mathsf{iKeyAuth} \lor \neg\mathsf{fKeyConf} \tag{4.8}$$

which completes the first part of the proof.

We now show that $\mathsf{fexKeyAuth} \implies \mathsf{iKeyAuth} \land \mathsf{fKeyConf}$. We first show that $\mathsf{fexKeyAuth} \implies \mathsf{iKeyAuth}$. Let $\mathcal{A}$ be a successful adversary against the $\mathsf{iKeyAuth}$ predicate; i.e. $\mathcal{A}$ reaches an execution state where $\neg\mathsf{iKeyAuth}$ holds true which is equivalent to

$$\exists \ell^* :: \ell^*.\mathsf{pid} \in \mathcal{S} \wedge (\ell^*.\mathsf{kconf} = \mathsf{full}) \wedge \ell^*.\mathsf{accept} \\ \wedge \exists \ell' :: \mathsf{Samekey}(\ell^*, \ell') \wedge (\ell'.\mathsf{id} \neq \ell^*.\mathsf{pid}). \quad (4.9)$$

Note that we include $\ell^*.\mathsf{kconf} = \mathsf{full}$ in $\neg\mathsf{iKeyAuth}$ as we only aim to prove that $\mathsf{fexKeyAuth}$ implies $\mathsf{iKeyAuth}$ for sessions that expect full explicit key authentication. We now assume, for contradiction, that $\mathsf{fexKeyAuth}$ holds; this implies that

$$\forall \ell \in \mathsf{LSID}, \begin{pmatrix} & \mathsf{Samekey}(\ell, \ell^*) \implies \ell.\mathsf{id} = \ell.\mathsf{pid}) \\ \wedge & \mathsf{aFresh}(\ell^*) \implies \exists \ell'' :: \mathsf{Samekey}(\ell^*, \ell'') \end{pmatrix} \quad (4.10)$$

for ℓ^* as in (4.9). We see that the existence of ℓ' that holds from (4.9) contradicts the first condition of (4.10) which shows that $\mathsf{fexKeyAuth} \implies \mathsf{iKeyAuth}$ as expected from the formulation of the predicates.

We now show that $\mathsf{fexKeyAuth} \implies \mathsf{fKeyConf}$. Let $\mathcal{A}$ be a successful adversary against the $\mathsf{fKeyConf}$ predicate; i.e. $\mathcal{A}$ reaches an execution state where $\neg\mathsf{fKeyConf}$ holds true. This is equivalent to

$$\exists \ell^* :: \begin{pmatrix} & \mathsf{aFresh}(\ell^*) \wedge \ell^*.\mathsf{pid} \in \mathcal{S} \\ \wedge & (\ell^*.\mathsf{kconf} = \mathsf{full}) \wedge \ell^*.\mathsf{accept} \\ \wedge & \forall \ell, \neg\mathsf{Samekey}(\ell^*, \ell) \end{pmatrix}.$$

This ℓ^* is now exactly one that satisfies $\neg\mathsf{fexKeyAuth}$ and hence we immediately have that $\mathsf{fexKeyAuth} \implies \mathsf{fKeyConf}$. As we have that $\mathsf{fexKeyAuth}$ implies both $\mathsf{iKeyAuth}$ and $\mathsf{fKeyConf}$, combined with (4.8) this concludes the proof that

$$\mathsf{iKeyAuth} \wedge \mathsf{fKeyConf} \iff \mathsf{fexKeyAuth}.$$

The proof of the same equivalence for almost-full confirmation notions, equation (4.6), follows from a similar argument. □

4.2.5 Key-match soundness

We next define the $\mathsf{KMSoundness}$ property which captures the essence of BR secrecy as a predicate without `Test` and `Guess` queries. It says that for any *authentication* fresh and accepting session ℓ, there does not exist another session ℓ' which holds the same key but is not partnered with ℓ. This captures exactly the proof strategy often used in the literature to argue that authentication is captured by secrecy notions; indeed, the strategy is to create two non-partnered sessions which have the same key (i.e. a break in authentication) and then revealing one and testing the other to win the secrecy experiment. The issue with that approach is that the requirements for secrecy freshness are often stricter than those for authentication freshness, thus certain authentication attacks may be eliminated from the model despite not being necessarily trivial.

Definition 4.10 (Key-match soundness)**.** The $\mathsf{KMSoundness}$ predicate evaluates to 1 if and only if

$$\forall \ell \in \mathsf{LSID}, \begin{pmatrix} & \mathsf{aFresh}(\ell) \\ \wedge & \ell.\mathsf{pid} \in \mathcal{S} \\ \wedge & \ell.\mathsf{accept} \end{pmatrix} \implies \forall \ell' \in \mathsf{LSID} :: (\mathsf{Samekey}(\ell', \ell) \implies \mathsf{Partner}(\ell', \ell)).$$

We then say that the AKE protocol Π provides *key-match soundness* if, for all probabilistic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{G_{\mathsf{KMSoundness}}} = \Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{G_{\mathsf{KMSoundness}}} = 0\right] = \mathsf{negl}(\lambda)$$

where $G_{\mathsf{KMSoundness}}$ is defined similarly to G_{iKeyAuth}.

The next theorem bridges the previous authentication notions with the original Bellare–Rogaway definition of secrecy by showing that BR-secrecy and Match-security together imply Key-Match soundness.

The definition above and the theorem below and its proof are in the most part the work of M. Fischlin and are reproduced here as published in [56].

Theorem 4.11. *Let Π be an AKE protocol with Match security and BR-secrecy w.r.t. D. Then it also provides key-match soundness. More precisely, for any probabilistic $\mathsf{poly}(\lambda)$-time algorithm $\mathcal{A}$ attacking $\mathsf{KMSoundness}$ in at most n sessions, it holds that for some probabilistic $\mathsf{poly}(\lambda)$-time algorithms $\mathcal{B}_1$, $\mathcal{B}_2$ and the output length $|\mathsf{k}|$ of keys,*

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{G_{\mathsf{KMSoundness}}} \leq n^2 \cdot \mathbf{Adv}_{\mathcal{B}_2,\Pi}^{G_{\mathsf{BRSec},D}} + \mathbf{Adv}_{\mathcal{B}_1,\Pi}^{G_{\mathsf{Match}}} + 2^{-|\mathsf{k}|}.$$

Proof. The first observation is that Match security implies that any session partnered to either ℓ or to ℓ' must hold the same key as the corresponding session. If this would not hold with overwhelming probability, we could build an algorithm $\mathcal{B}_1$ to refute

Match security in a straightforward way. This enables us to assume that there are two sessions ℓ_0 and ℓ_0' with the above property and that both accept without a partnered session. That is, ℓ_0 and ℓ_0'

- hold identical keys, i.e. $\mathsf{Samekey}(\ell_0, \ell_0')$,
- are not partnered, i.e. $\neg\mathsf{Partner}(\ell_0, \ell_0')$, and
- for neither of the two sessions, in the moment when the session accepts, there is another session which is yet partnered with the session.

Note that ℓ_0 must have accepted, by assumption, such that $\ell_0.\mathsf{k} \neq \perp$, and therefore $\mathsf{Samekey}(\ell_0, \ell_0')$ implies that session ℓ_0' must have a valid key too. In particular it must have accepted the session (and can both be tested and revealed in an attack on secrecy).

Assume now that there was a successful adversary $\mathcal{A}$ against key-match soundness (with two sessions ℓ_0, ℓ_0' as above). We show how to break BRSec through an adversary $\mathcal{B}_2$ with non-negligible probability in this case.

Our adversary $\mathcal{B}_2$ will try to predict the sessions ℓ_0, ℓ_0' by picking two session numbers i, j at random from $\{1, 2, \ldots, n\}$, where we count sessions according to their initialisation in $\mathcal{A}$'s simulated attack. Next, $\mathcal{B}_2$ runs $\mathcal{A}$'s attack, relaying all inputs and oracle queries and answers between $\mathcal{B}_2$'s game and $\mathcal{A}$. Note that $\mathcal{B}_2$ has the same oracle interfaces as $\mathcal{A}$, but in addition may call the `Test` and the `Guess` oracle.

Adversary $\mathcal{B}_2$ diverges from $\mathcal{A}$ with respect to two points: if the i-th session in the attack accepts, then $\mathcal{B}_2$ immediately asks to `Reveal` the session key k_i. If the j-the session accepts, then $\mathcal{B}_2$ immediately calls `Test` to get a key value k_j. Adversary $\mathcal{B}_2$ makes the $\mathtt{Guess}(b_{\mathsf{guess}})$ query and stops, where the bit b_{guess} is set to 1 if $\mathsf{k}_i = \mathsf{k}_j$, and to 0 otherwise.

Note that up to the point when $\mathcal{B}_2$ makes the `Test` query, the simulation to $\mathcal{A}$ is perfect. Assume that $\mathcal{B}_2$ predicts ℓ_0, ℓ_0' correctly for the `Test` query resp. the `Reveal` query, which happens with probability at least $1/n^2$. Then the `Test` session ℓ_0 does not have a partner yet, is authentication fresh and the authenticating partner is still honest, such that the session is still secrecy fresh. In this case, if $b_{\mathsf{test}} = 1$ the `Test` oracle returns the actual session key, such that the keys match, and we have $\mathcal{B}_2$ output 1, too. In summary, the probability of this happening is at least $\frac{1}{n^2}$ times the probability that $\mathcal{A}$ succeeds, minus a negligible term for refuting Match security. This is non-negligible.

Next consider the case $b_{\mathsf{test}} = 0$ such that the test session ℓ_0 returns a random key. Then the probability that this independent random key matches the other key is $2^{-|\mathsf{k}|}$ and thus negligible, such that $\mathcal{B}_2$ only returns 1 with this negligible probability. □

4.3 Entity authentication

This second form of authentication does not involve the session-key in its security guarantees; instead it provides authentication guarantees based only on the session that just took place. Our *entity authentication* definitions are based on the Partner predicate, instead of Samekey, and intuitively require that if two sessions terminate with the same sid, then they should agree on each other's identities. This corresponds to the intuitive notion of identification protocols where sessions obtain guarantees upon accepting and deriving an identifier. We first present our definitions and then demonstrate that entity and key authentication are in fact equivalent for Match-secure and BR-secret AKE protocols (since these two properties imply a binding between session identifiers and session-keys).

4.3.1 Entity authentication: implicit, confirmation and explicit

To adapt our definitions of Section 4.2 to entity authentication, we replace Samekey by Partner in the security definitions. Below we present the predicates for implicit entity authentication, entity confirmation and full explicit entity authentication. The full case requires the definition of the session state variable $\mathsf{econf} \in \{\mathsf{full}, \mathsf{almost}, \mathsf{no}, \perp\}$, indicating, analogously to kconf, which form of entity authentication a session expects to obtain. The almost-full case is more involved because it also requires entity confirmation identifiers, similarly to the kcid; those definitions can be derived from the almost-full case for key authentication.

Definition 4.12 (Implicit entity authentication)**.** The iEntAuth predicate evaluates to 1 if and only if

$$\forall \ell \in \mathsf{LSID}, (\ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{accept}) \implies \forall \ell' \in \mathsf{LSID}, \big(\mathsf{Partner}(\ell', \ell) \implies \ell'.\mathsf{id} = \ell.\mathsf{pid}\big)\,.$$

We then say that the AKE protocol Π provides *implicit entity authentication* if, for all probabilistic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{G_{\mathsf{iEntAuth}}} = \Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{G_{\mathsf{iEntAuth}}} = 0\right] = \mathsf{negl}(\lambda)\,,$$

where G_{iEntAuth} is defined similarly to G_{iKeyAuth}.

In short, this notion says that the owner of an implicitly entity-authenticated session is guaranteed to have been partnered with (i.e. had the same protocol execution as) sessions that belong to its intended peer. We note that this is a very weak notion as the session terminates neither with an explicit guarantee nor with a secret element, such as a key, that it may use later to obtain a stronger guarantee. In fact, this was the notion that was typically included in previous definitions of Match security (e.g. in the $\mathsf{P}_{\mathsf{sid}}$ predicate of [32]).

Definition 4.13 (Full entity confirmation)**.** The fEntConf predicate evaluates to 1 if and only if

$$\forall \ell \in \mathsf{LSID}, \begin{pmatrix} & \mathsf{aFresh}(\ell) \\ \wedge & \ell.\mathsf{econf} = \mathsf{full} \\ \wedge & \ell.\mathsf{pid} \in \mathcal{S} \\ \wedge & \ell.\mathsf{accept} \end{pmatrix} \implies \exists \ell' \in \mathsf{LSID} :: \mathsf{Partner}(\ell', \ell).$$

We then say that the AKE protocol Π provides *full entity confirmation* if, for all probabilistic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{G_{\mathsf{fEntConf}}} = \Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{G_{\mathsf{fEntConf}}} = 0\right] = \mathsf{negl}(\lambda),$$

where G_{fEntConf} is defined similarly to G_{iKeyAuth}.

As for key authentication, the full explicit variant of entity authentication is captured by a combination of the previous two predicates.

Definition 4.14 (Full explicit entity authentication)**.** The fexEntAuth predicate evaluates to 1 if and only if

$$\forall \ell \in \mathsf{LSID}, \begin{array}{l} \ell.\mathsf{pid} \in \mathcal{S} \wedge \\ \ell.\mathsf{econf} = \mathsf{full} \wedge \\ \ell.\mathsf{accept} \end{array} \Rightarrow \begin{array}{l} (\forall \ell' \in \mathsf{LSID},\ \mathsf{Partner}(\ell', \ell) \Rightarrow \ell'.\mathsf{id} = \ell.\mathsf{pid}) \wedge \\ (\mathsf{aFresh}(\ell) \Rightarrow \exists \ell' \in \mathsf{LSID} :: \mathsf{Partner}(\ell', \ell)). \end{array}$$

We then say that the AKE protocol Π provides *full explicit entity authentication* if, for all probabilistic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{G_{\mathsf{fexEntAuth}}} = \Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{G_{\mathsf{fexEntAuth}}} = 0\right] = \mathsf{negl}(\lambda)$$

where $G_{\mathsf{fexEntAuth}}$ is defined similarly to G_{iKeyAuth}.

Separating the two different properties that constitute explicit entity authentication may be helpful to understand and guide protocol design. For example, with this separation in mind, a signature over the transcript sent at the end of an execution can be seen as providing entity confirmation, and therefore boosting implicit entity authentication to explicit authentication. A similar argument could show that password authentication over a secure channel can serve a similar purpose.

4.3.2 Key and entity authentication relationships

We now present the necessary conditions for key and entity authentication notions to be equivalent to one another. We use the "secrecy and match-security predicate" KMSoundness from Definition 4.10 to state the relationships in terms of predicates. Recall that this predicate holds (with overwhelming probability) for a Match-secure and BR-secret protocol. Under the assumption that partnered sessions derive equal keys and equal keys can only be derived in partnered sessions, it holds intuitively that authentication guarantees obtained via keys are equivalent to those obtained directly via session identifiers. We formalize this below in Propositions 4.15 and 4.16.

Proposition 4.15. *Let* Π *be an AKE protocol; it holds that*

$$\mathsf{iKeyAuth} \wedge \mathit{Match} \Longrightarrow \mathsf{iEntAuth}, \tag{4.11}$$

$$\mathsf{fexKeyAuth} \wedge \mathit{Match} \wedge \mathsf{KMSoundness} \Longrightarrow \mathsf{fexEntAuth}, \tag{4.12}$$

Proof. The first implication (4.11) can be viewed as follows. Both predicates are identical, except that entity authentication uses the Partner predicate instead of Samekey. Hence, a mismatch—in the sense that iKeyAuth holds but iEntAuth does not—can only occur if

$$\exists \ell \in \mathsf{LSID} :: \left(\begin{array}{ll} & \ell.\mathsf{pid} \in \mathcal{S} \\ \wedge & \ell.\mathsf{accept} \\ \wedge & \exists \ell' \in \mathsf{LSID} :: (\mathsf{Partner}(\ell', \ell) \wedge \neg\mathsf{Samekey}(\ell', \ell)) \end{array} \right).$$

If the sessions ℓ, ℓ' would have the same key then they would also satisfy the identity requirement $\ell'.\mathsf{id} = \ell.\mathsf{pid}$, because of the iKeyAuth property.

Note that the partnering predicate stipulates that the session identifiers of ℓ and ℓ' are equal (and different from $\perp$). According to our specification of key exchange protocols this, in turn, implies that both sessions must have accepted and, if so, that they have set the keys to values different from $\perp$. But now we would get an immediate contradiction to Property (4.1) of Match security with the statement that

$$\exists \ell, \ell' \in \mathsf{LSID} :: \mathsf{Partner}(\ell, \ell') \wedge (\ell.\mathsf{k} \neq \perp \neq \ell'.\mathsf{k}) \wedge \neg \mathsf{Samekey}(\ell, \ell').$$

The second implication (4.12) follows similarly; there are two possibilities for a mismatch ($\mathsf{fexKeyAuth}$ holds, but $\mathsf{fexEntAuth}$ does not). Either the first property in the implication in $\mathsf{fexEntAuth}$, which also appears in the implicit definition, is false, in which case we get the same contradiction as before, or the second property in the implication ($\mathsf{aFresh}(\ell) \implies \exists \ell' :: \mathsf{Partner}(\ell', \ell)$) is false, although it holds in $\mathsf{fexKeyAuth}$ for the $\mathsf{Samekey}$ case. This means that

$$\exists \ell \in \mathsf{LSID} :: \begin{pmatrix} & \mathsf{aFresh}(\ell) \wedge \ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{accept} \\ \wedge & \exists \ell' \in \mathsf{LSID} :: \mathsf{Samekey}(\ell', \ell) \\ \wedge & \forall \ell'' \in \mathsf{LSID} :: \neg \mathsf{Partner}(\ell'', \ell) \end{pmatrix}.$$

Note that this means that there will be a session ℓ' which has the same key as ℓ, and since ℓ has accepted it must be a valid key $\ell.\mathsf{k} \neq \perp$, but such that no other session is partnered with ℓ. This, however, contradicts the $\mathsf{KMSoundness}$ predicate. □

Proposition 4.16. *Let* Π *be an AKE protocol; it holds that*

$$\mathsf{iKeyAuth} \Longleftarrow \mathsf{iEntAuth} \wedge \mathit{Match} \wedge \mathsf{KMSoundness}, \tag{4.13}$$
$$\mathsf{fexKeyAuth} \Longleftarrow \mathsf{fexEntAuth} \wedge \mathit{Match} \wedge \mathsf{KMSoundness} \tag{4.14}$$

Proof. We start with the first implication (4.13). Similar to the previous proposition one can show that any mismatch in the predicates implies

$$\exists \ell \in \mathsf{LSID} :: \begin{pmatrix} & \ell.\mathsf{pid} \in \mathcal{S} \\ \wedge & \ell.\mathsf{accept} \\ \wedge & \exists \ell' \in \mathsf{LSID} :: (\mathsf{Samekey}(\ell', \ell) \wedge \neg \mathsf{Partner}(\ell', \ell)) \end{pmatrix}.$$

This would also contradict the $\mathsf{KMSoundness}$ predicate.

The second implication (4.14) leads either to a contradiction as in the implicit case, or we can analogously to the other direction conclude that

$$\exists \ell \in \mathsf{LSID} :: \left(\begin{array}{ll} & \mathsf{aFresh}(\ell) \wedge \ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{accept} \\ \wedge & \exists \ell' \in \mathsf{LSID} :: \mathsf{Partner}(\ell', \ell) \\ \wedge & \forall \ell'' \in \mathsf{LSID} :: \neg\mathsf{Samekey}(\ell'', \ell) \end{array} \right).$$

This, of course, would contradict Property (4.1) of the Match predicate, as we would have partnered sessions ℓ, ℓ' which do not hold the same (valid) key $\ell.\mathsf{k} \neq \perp$. Here we use the fact that partnering implies non-trivial session identifiers, and if session ℓ' has set the identifier, it has accepted and set a key $\ell'.\mathsf{k} \neq \perp$, too. □

Finally, we show that implicit entity authentication together with (full) explicit entity confirmation is equivalent to full explicit entity authentication.

Proposition 4.17. *Let* Π *be an AKE protocol; it holds that*

$$\mathsf{iEntAuth} \wedge \mathsf{fEntConf} \iff \mathsf{fexEntAuth}$$

Proof. The proof is the same as that of Theroem 4.9 with the Samekey predicate replaced by Partner. □

4.4 Relation with CK-style security

In all CK-style models (CK, $\mathrm{CK}_{\mathrm{HMQV}}$ and eCK) the definition of "matching sessions" includes the requirement that parties agree on each other's identities. As there is no other mention of matching expected identities, this seems to be the only capture of authentication in such models. The argument here being the same as the one captured by our KMSoundness predicate: if one can break authentication, then they are breaking matching which can in turn be used to break secrecy.

Capturing authentication in CK-style models

In [49], Cremers states that since "the test session-key must be indistinguishable from keys computed by non-matching sessions", then sessions with the same key must be matching sessions. This is analogous to our Theorem 4.11 concerning KMSoundness except that the CK-style definition of matching includes expected identities, and therefore this implies that CK-style security also guarantees (at least) implicit authentication.

Crucially, this argument however only applies to sessions for which CK-style security holds, that is sessions that remain $\mathsf{sFresh}_{\mathrm{CK}}$, for a suitable definition of this that captures the restriction of the CK-style models as presented in [49]. In contrast, our standalone Definition 4.3 of implicit authentication captures security against a wider range of attacks due to the absence of freshness requirements on the target session.

Extending authentication freshness

To establish our authentication definitions of Section 4.2 in a CK-style model, we consider whether the freshness conditions change with the addition of a `StateReveal` query. This reveals to the adversary either the entire state or only the ephemeral key, as defined by the protocol, for the CK and eCK models respectively [49].

Following from Section 4.2.1, we allow the adversary to make `StateReveal` queries against the $\mathsf{iKeyAuth}_{\mathrm{CK}}$ predicate to capture the widest possible range of attacks. This implies that $\mathcal{A}$ can both `StateReveal` a session and `Corrupt` its owner which is not allowed by CK-style models. Following similarly from Sections 4.2.2 and 4.2.3, we restrict the adversary from trivially obtaining the key when attacking key confirmation and explicit key authentication. Therefore we state that it cannot both corrupt the intended partner and also `StateReveal` a partner session against CK variants of these predicates.

Separating authentication from secrecy

Let $\mathcal{E}$ denote the event, in a CK-style secrecy experiment, where $\mathcal{A}$ succeeds in causing two game-controlled sessions to share a key without matching in the CK-style sense, i.e. without agreeing on each other's identities. Let G_{CKSec} denote the usual CK-style secrecy game and G^{-}_{CKSec} denote the game which instead uses our definition of partnering of Section 3.3 without expected identities (i.e. an extension of the secrecy game of Section 3.4.2). Using a rather informal terminology, we then have

$$\begin{aligned}\Pr\left[\mathcal{A} \text{ wins } G_{\mathsf{CKSec}}\right] = {} & \Pr\left[\mathcal{A} \text{ wins } G_{\mathsf{CKSec}} | \mathcal{E}\right] \cdot \Pr\left[\mathcal{E}\right] \\ & + \Pr\left[\mathcal{A} \text{ wins } G_{\mathsf{CKSec}} | \neg\mathcal{E}\right] \cdot \Pr\left[\neg\mathcal{E}\right].\end{aligned}$$

First, $\mathcal{E}$ corresponds to a break of $\mathsf{iKeyAuth}_{\mathrm{CK}}$ and so we have

$$\Pr\left[\mathcal{A} \text{ wins } G_{\mathsf{CKSec}} | \mathcal{E}\right] \cdot \Pr\left[\mathcal{E}\right] \leq \Pr\left[\neg\mathsf{iKeyAuth}_{\mathrm{CK}}\right].$$

Second, if $\mathcal{A}$ wins G_{CKSec} without triggering $\mathcal{E}$, then his attack can be reproduced in G^{-}_{CKSec} and so we have

$$\Pr\left[\mathcal{A} \text{ wins } G_{\mathsf{CKSec}} | \neg\mathcal{E}\right] \cdot \Pr\left[\neg\mathcal{E}\right] \leq \Pr\left[\mathcal{A} \text{ wins } G^{-}_{\mathsf{CKSec}}\right].$$

In conclusion,

$$\Pr\left[\mathcal{A} \text{ wins } G_{\mathsf{CKSec}}\right] \leq \Pr\left[\neg\mathsf{iKeyAuth}_{\mathrm{CK}}\right] + \Pr\left[\mathcal{A} \text{ wins } G^{-}_{\mathsf{CKSec}}\right],$$

which shows that we can represent CK-style key secrecy through our separate notions of authentication and key secrecy when adapted to these models.

4.5 Key-confirming protocols

We now define symmetric *key-confirming protocols* and give a simple example. These provide guarantees on the existence of a session with the same key which may be secondary to the main purpose of these protocols. For example, we expect that protocols for authenticated message transmission would belong to this class as they are expected to return error messages if an incorrect key is used.

4.5.1 Definition

Based on [32, Section 4], we describe the syntax and the security game for such protocols. We denote these as $\pi = (\mathcal{I}, \mathtt{KGen}, \zeta)$ and write D for the output distribution of the randomized algorithm $\mathtt{KGen}$. To model such protocols, we use the same mechanism of local session identifiers. Here, EST is not defined as there are no long-term keys.

Session state. For key-confirming protocols, it consists of:

- $\mathsf{crypt} \in \{0,1\}^*$: protocol-specific private session state.
- $\mathsf{k} \in \{0,1\}^* \cup \{\bot\}$: the symmetric key used.
- $\mathsf{kcind} \in \{\mathsf{true}, \mathsf{false}, \bot\}$: indicates if key confirmation is achieved. Initially set to $\bot$, it must be changed to true or false before termination. Its value is always public.

The difference with [32] is the addition of the *key confirmation indicator* kcind. We stress that setting kcind is done independently of termination. For example, a secure channel protocol could achieve key confirmation only after the first messages but continue running for much longer as the channel is used for communication. We focus on guarantees obtained from the setting of kcind and do not make assumptions or requirements on termination.

Local session state. As in [32], it consists of:

- $\delta_{\mathsf{k}} \in \{\mathsf{fresh}, \mathsf{revealed}\}$ denoting whether the key is known to the adversary.
- $\mathsf{lst} \in \{0,1\}^*$ any other local session state required to model the protocol's other security requirements.

Setup. The $\mathtt{SetupE}$ algorithm only initialises $\mathsf{crypt}, \mathsf{k}$ and kcind to $\perp$ for each $\ell \in \mathsf{LSID}$. The $\mathtt{SetupG}$ algorithm also only initialises $\delta_{\mathsf{k}} \leftarrow \mathsf{fresh}$ for every session as our security game for key confirmation does not require any model-wide state.

Queries. As in [32], our model allows $\mathcal{A}$ to initialise sessions with three different queries. The first, $\mathtt{InitS}(\ell)$, initialises a session with an honestly generated key, $\ell.\mathsf{k} \leftarrow \mathtt{KGen}(1^\lambda)$, which remains hidden from $\mathcal{A}$. The second, $\mathtt{InitP}(\ell_1, \ell_2)$, initialises a session with the same key as another. The game sets $\ell_2.\mathsf{k} \leftarrow \ell_1.\mathsf{k}$ and $\ell_2.\delta_{\mathsf{k}} \leftarrow \ell_1.\delta_{\mathsf{k}}$. The third, $\mathtt{InitK}(\ell, \kappa)$, allows $\mathcal{A}$ to set his own key. It sets $\ell.\mathsf{k} \leftarrow \kappa$ and immediately sets $\ell.\delta_{\mathsf{k}} \leftarrow \mathsf{revealed}$. As before, $\mathtt{Send}(\ell, m)$ and $\mathtt{Reveal}(\ell)$ allow $\mathcal{A}$ to control the network and view honestly generated keys.

The Valid predicate verifies that $\mathtt{Send}$ and $\mathtt{Reveal}$ queries are made to initialised sessions and that initialisation queries are made to sessions without keys. For the $\mathtt{InitP}$ query, it also verifies that ℓ_1 is initialised.

Key confirmation guarantee. Here there no longer is a distinction between full and almost-full key confirmation since keys are set upon initialisation. This notion says that for any session which has set the key confirmation identifier to true, there is another session which uses the same key.

Definition 4.18 (Key confirmation guarantee)**.** The $\mathsf{symKeyConf}$ predicate evaluates to 1 if and only if

$$\forall \ell \in \mathsf{LSID}, (\ell.\delta_{\mathsf{k}} = \mathsf{fresh} \wedge \ell.\mathsf{kcind} = \mathsf{true}) \implies \exists \ell' \in \mathsf{LSID} :: \mathsf{Samekey}(\ell', \ell),$$

where $\mathsf{Samekey}$ is defined as before, including the condition that $\ell' \neq \ell$[1]. The game $G_{\mathsf{symKeyConf}}$ is then defined with state, algorithms $\mathtt{SetupE}$ and $\mathtt{SetupG}$, and behaviour as above, with query set $Q = \{\mathtt{Send}, \mathtt{InitS}, \mathtt{InitP}, \mathtt{InitK}, \mathtt{Reveal}\}$ and winning predicate $P = \mathsf{symKeyConf}$.

The protocol π provides *(secure) key confirmation*, or is a *key-confirming protocol*, if, for all probabilistic $\mathsf{poly}(\lambda)$-time adversaries $\mathcal{A}$, there exists a negigible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{A},\pi,\mathcal{I}}^{G_{\mathsf{symKeyConf}}} = \Pr\left[\mathbf{Exp}_{\mathcal{A},\pi}^{G_{\mathsf{symKeyConf}}} = 0\right] = \mathsf{negl}(\lambda)\,.$$

We note that a symmetric protocol π which always sets $\mathsf{kcind} = \mathsf{false}$ trivially achieves secure key confirmation. This is similar to an AKE protocol formally achieving implicit key authentication by setting $\mathcal{S} = \emptyset$.

4.5.2 Protocol example

We present an example of key-confirming protocols. Let $\mathtt{MAC} = (\mathtt{KGen}, \mathtt{Tag}, \mathtt{Vfy})$ be an unforgeable MAC (Definition 2.7). From such a MAC, we construct the protocol π_{kconf} as follows. If a session is activated with first message $m = \mathsf{init}$, it sends $t = \mathtt{Tag}(\mathsf{k}, 1)$. When it receives a second message m^*, it verifies that it is a tag for the message "2" by checking if $\mathtt{Vfy}(\mathsf{k}, 2, m^*) = 1$. If this holds, then it sets $\mathsf{kcind} \leftarrow \mathsf{true}$, otherwise it sets $\mathsf{kcind} \leftarrow \mathsf{false}$. If a session is instead activated as a receiver, then it plays the counterpart and checks that it correctly receives a tag for the message "1" and, if so, sets $\mathsf{kcind} \leftarrow \mathsf{true}$ and replies with a tag for "2". We present the formal description of π_{kconf} in Figure 4.1.

To show that the protocol π_{kconf} is key-confirming, we use the concept of single-session reducible games presented in [32]. Without specifying the formal details, we see that the independence of sessions in the game $G_{\mathsf{symKeyConf}}^{\pi_{\mathsf{kconf}}}$, apart from their potential partner, implies that this game is session restricted. Theorem 2 of [32, Appendix B] then gives us that $G_{\mathsf{symKeyConf}}^{\pi_{\mathsf{kconf}}}$ is single session reducible and therefore that the key-confirmation property of π_{kconf} depends only on the security of a single session.

We then reduce the security of one session to the unforgeability of $\mathtt{MAC}$. The reduction sets up a session and waits for the adversary to submit a message. If it submits $m^* = \mathsf{init}$, then the reduction uses the oracle $\mathtt{Tag}(\mathsf{k}, \cdot)$ to respond with a correct tag. If it submits any other message, then the reduction submits $(1, m^*)$ or $(2, m^*)$ as its forgery, depending on its role. If the reduction has queried the $\mathtt{Tag}$

[1]We note that $\ell.\delta_{\mathsf{k}} = \mathsf{fresh}$ is analogous to a freshness condition.

```
$\pi_{\mathsf{kconf}}$ (with secret key $\mathsf{k}$)
1 :  Initialise $\mathsf{kcind} \leftarrow \bot$ and role $\rho \leftarrow \bot$
2 :  while $\mathsf{kcind} = \bot$ do
3 :    Receive $m^*$
4 :    if $m^* = \mathsf{init}$ and $\rho = \bot$ then
5 :      Set $\rho \leftarrow \mathsf{init}$
6 :      Send $t = \mathtt{Tag}(\mathsf{k}, 1)$
7 :    elseif $\rho = \mathsf{init}$ then
8 :      if $\mathtt{Vfy}(\mathsf{k}, 2, m^*) = 1$ then
9 :        Set $\mathsf{kcind} \leftarrow \mathsf{true}$
10 :     else set $\mathsf{kcind} \leftarrow \mathsf{false}$
11 :   else
12 :     if $\mathtt{Vfy}(\mathsf{k}, 1, m^*) = 1$ then
13 :       Set $\mathsf{kcind} = \mathsf{true}$
14 :       Send $t = \mathtt{Tag}(\mathsf{k}, 2)$
15 :     else set $\mathsf{kcind} \leftarrow \mathsf{false}$
```

Figure 4.1: A simple key confirmation protocol that expects a tag on the message "1" or "2" depending on the role (initiator or responder) played by the session.

oracle on "1" in response to an init message, then the adversary must create a tag for the message "2" to make π_{kconf} accept and therefore there is no risk of the reduction outputting a message which it has already queried. This shows that the reduction creates a forgery exactly when the adversary is capable of winning is $G^{\pi_{\mathsf{kconf}}}_{\mathsf{symKeyConf}}$ and thus π_{kconf} provides secure key confirmation if $\mathtt{MAC}$ is EUF-CMA-secure.

While this protocol is a simple example that has no objective beyond providing secure key confirmation, it nonetheless provides justification for expecting useful constructions, such as authenticated encryption schemes or secure channel protocols, to provide the same guarantees.

4.6 Key-confirming protocols and explicit authentication

We now define the composition of AKE protocols with key-confirming protocols. As a significant difference to [32, Section 5], we consider the composition *as an AKE protocol*, not as a symmetric protocol, and we prove that composing an AKE protocol with implicit key authentication and key secrecy together with a secure key-confirming protocol yields an *AKE protocol* with *explicit* key authentication.

4.6.1 Syntax of composed protocols

The composition first runs the AKE protocol and then, once this accepts, runs the symmetric protocol, initialised with the key from the first step, until key confirmation is obtained. Recall that a key-confirming protocol $\pi = (\mathtt{KGen}_\pi, \zeta_\pi)$ sets kcind before terminating and that it can then continue its execution; this is not the case when forming an AKE protocol since it must terminate upon acceptance of the key, as in Section 3.1.2. We therefore define $\bar{\pi}$ to be π with the algorithm $\zeta_{\bar{\pi}}$ the same as ζ_π but halted after kcind is set. Thus the key derived by ζ_{ke} is only accepted as the final key for the composition once $\zeta_{\bar{\pi}}$ has set kcind to true. Given an AKE protocol $\mathsf{ke} = (\mathtt{KGen}_{\mathsf{ke}}, \zeta_{\mathsf{ke}})$, we write $\mathsf{ke}; \bar{\pi} = (\mathtt{KGen}_{\mathsf{ke};\bar{\pi}}, \zeta_{\mathsf{ke};\bar{\pi}})$ for the composition.

As we consider $\mathsf{ke}; \bar{\pi}$ as an AKE protocol, it uses the same long-term key generation as ke and therefore $\mathtt{KGen}_{\mathsf{ke};\bar{\pi}} = \mathtt{KGen}_{\mathsf{ke}}$. The algorithm $\zeta_{\mathsf{ke};\bar{\pi}}$ first runs ζ_{ke}. If this rejects, then $\zeta_{\mathsf{ke};\bar{\pi}}$ rejects the session; otherwise it runs $\zeta_{\bar{\pi}}$ with the derived key and accepts or rejects depending on kcind. Formally, given an incoming message m, the algorithm $\zeta_{\mathsf{ke};\bar{\pi}}$ first examines the value of $\mathsf{accept}_{\mathsf{ke}}$ from the ke protocol. If $\mathsf{accept}_{\mathsf{ke}} = \perp$, it passes m to ζ_{ke}; if $\mathsf{accept}_{\mathsf{ke}} = \mathsf{true}$, it passes m to $\zeta_{\bar{\pi}}$. We note that this value of $\mathsf{accept}_{\mathsf{ke}}$ from ke is different from the value of $\mathsf{accept}_{\mathsf{ke};\bar{\pi}}$ for $\mathsf{ke}; \bar{\pi}$ which is only set once both ke and $\bar{\pi}$ have terminated.

4.6.2 Syntax of composed games

The game $G^{\mathsf{ke};\bar{\pi}}_{\mathsf{Pred}}$ enables $\mathcal{A}$ to interact with simultaneous sessions of the composed protocol. Here, the adversary's goal is to attack the authentication property Pred of $\mathsf{ke}; \bar{\pi}$ seen as an AKE protocol. We build this game from the elements of the games for the protocols ke and π and use indices to distinguish them.

Game state. The execution state $\mathsf{EST}_{\mathsf{ke};\bar{\pi}}$ is only $\mathsf{EST}_{\mathsf{ke}}$ as key-confirming protocols do not have one. The session state $\mathsf{SST}_{\mathsf{ke};\bar{\pi}}$ is made up of the same elements as for key exchange protocols but constructed from the composing session states as follows:

- The long-term keying information is as in $\mathsf{SST}_{\mathsf{ke}}$.
- The protocol private session state is the concatenation of both states: $\mathsf{crypt}_{\mathsf{ke};\bar{\pi}} = \mathsf{crypt}_{\mathsf{ke}} \| \mathsf{crypt}_{\bar{\pi}}$.
- The $\mathsf{accept}_{\mathsf{ke};\bar{\pi}}$ indicator is set to true once $\mathsf{kcind}_{\bar{\pi}}$ is set to true. Note that this only happens if $\mathsf{accept}_{\mathsf{ke}}$ is also already set to true as $\bar{\pi}$ does not begin until ke has accepted.

- The session identifier $\mathsf{sid}_{\mathsf{ke};\bar{\pi}}$ is set to $\mathsf{sid}_{\mathsf{ke}}$ *only* when $\mathsf{accept}_{\mathsf{ke};\bar{\pi}}$ is set to true.
- The $\mathsf{k}_{\mathsf{ke};\bar{\pi}}$ is set to k_{ke} *only* when $\mathsf{accept}_{\mathsf{ke};\bar{\pi}}$ is set to true. Before then, k_{ke} is kept internally and passed on to $\bar{\pi}$, so that $\mathsf{k}_{\bar{\pi}} \leftarrow \mathsf{k}_{\mathsf{ke}}$, when $\mathsf{accept}_{\mathsf{ke}} \leftarrow \mathsf{true}$.
- As $\bar{\pi}$ always provides full key confirmation, we have that $\mathsf{kconf}_{\mathsf{ke};\bar{\pi}} = \mathsf{full}$ for all sessions.

The local session state $\mathsf{LST}_{\mathsf{ke};\bar{\pi}}$ is the same as for AKE protocols and the model state remains undefined as it is not required for authentication.

Setup, queries and Valid predicate. These are the same as in Section 3.1.2 with the addition that the Valid predicate uses $\mathsf{Valid}_{\bar{\pi}}$ for `Send` and `Reveal` queries to sessions executing $\bar{\pi}$.

Winning predicates. Any predicate from Sections 3.4 (Bellare–Rogaway security), 4.1 (match security), 4.2 (key authentication) and 4.3 (entity authentication).

4.6.3 Composition result

We show that a implicitly authenticated key exchange protocol composed with a key-confirming protocol produces an explicitly authenticated key exchange protocol. Our choice of public session identifiers means we do not require a session matching algorithm as in [32, Section 3].

Theorem 4.19. *Let* ke *be a* Match*-secure key exchange protocol which provides implicit key authentication and BR-secrecy w.r.t. key distribution* D*. Let* π *be a symmetric-key protocol with key generation distribution* D *which provides secure key confirmation. Then* $\mathsf{ke};\bar{\pi}$ *is a key exchange protocol which provides explicit key authentication.*

Proof. We make use of Theorem 4.9 to separate the work into two steps. First we prove in Lemma 4.20 that $\mathsf{ke};\bar{\pi}$ provides implicit key authentication under the assumption that ke does. Then we prove in Lemma 4.21 that $\mathsf{ke};\bar{\pi}$ provides full key confirmation under the assumption that ke is BR-secret and that $\bar{\pi}$ provides secure key confirmation. As the two properties hold separately, Theorem 4.9 immediately gives us that $\mathsf{ke};\bar{\pi}$ provides explicit key authentication. □

Lemma 4.20. *Let* ke *be a key exchange protocol and let* π *be a symmetric-key protocol. For any probabilistic* $\mathsf{poly}(\lambda)$*-time adversary* $\mathcal{A}$*, it holds that, for some probabilistic*

$\mathsf{poly}(\lambda)$*-time algorithm* $\mathcal{B}$*,*

$$\mathbf{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi}}^{G_{\mathsf{iKeyAuth}}^{\mathsf{ke};\bar{\pi}}} = \mathbf{Adv}_{\mathcal{B},\mathsf{ke}}^{G_{\mathsf{iKeyAuth}}^{\mathsf{ke}}}.$$

Proof. Let $\mathcal{A}$ be an adversary against $\mathsf{ke};\bar{\pi}$ in the $G_{\mathsf{iKeyAuth}}^{\mathsf{ke};\bar{\pi}}$ game, which we denote as $G^{\mathsf{ke};\bar{\pi}}$ when the context is clear. We build an adversary $\mathcal{B}$ against ke in the $G_{\mathsf{iKeyAuth}}^{\mathsf{ke}}$ game, which we similarly denote as G^{ke}.

The reduction $\mathcal{B}$ sets up $G^{\mathsf{ke};\bar{\pi}}$ for $\mathcal{A}$ as described above using elements from G^{ke}. It then responds to $\mathcal{A}$'s queries in the following way. (We use the notation $\ell_{\mathsf{ke};\bar{\pi}}$ to denote session identifiers used by $\mathcal{A}$ in $G^{\mathsf{ke};\bar{\pi}}$ and the notation ℓ_{ke} to denote the corresponding identifiers used by $\mathcal{B}$ in G^{ke}.)

- When $\mathcal{A}$ submits $\mathtt{Send}(\ell_{\mathsf{ke};\bar{\pi}}, m)$, $\mathcal{B}$ checks the value of $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept}$. If it is either true or false, $\mathcal{B}$ responds $\perp$ to $\mathcal{A}$ as the sessions has either already accepted or rejected. If it is still $\perp$, $\mathcal{B}$ examines the value of $\ell_{\mathsf{ke}}.\mathsf{accept}$.
  - If $\ell_{\mathsf{ke}}.\mathsf{accept} = \perp$, $\mathcal{B}$ submits $\mathtt{Send}(\ell_{\mathsf{ke}}, m)$ to G^{ke} and responds to $\mathcal{A}$ with m' returned by G^{ke}. If $\ell_{\mathsf{ke}}.\mathsf{sid}$ is set at that step, $\mathcal{B}$ remembers $\ell_{\mathsf{ke}}.\mathsf{sid}$ for when it needs to set $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{sid}$. If $\ell_{\mathsf{ke}}.\mathsf{accept} \leftarrow \mathsf{false}$, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} \leftarrow \mathsf{false}$ and makes this known to $\mathcal{A}$. If $\ell_{\mathsf{ke}}.\mathsf{accept} \leftarrow \mathsf{true}$, $\mathcal{B}$ submits $\mathtt{Reveal}(\ell_{\mathsf{ke}})$ to G^{ke} to obtain $\ell_{\mathsf{ke}}.\mathsf{k}$.
  - If $\ell_{\mathsf{ke}}.\mathsf{accept} = \mathsf{true}$ and $\ell_{\bar{\pi}}.\mathsf{kcind} = \perp$, $\mathcal{B}$ has obtained $\ell_{\mathsf{ke}}.\mathsf{k}$ so it can respond to $\mathcal{A}$ according to $\bar{\pi}$ by computing the response internally. If $\ell_{\bar{\pi}}.\mathsf{kcind} \leftarrow \mathsf{false}$, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} \leftarrow \mathsf{false}$. If $\ell_{\bar{\pi}}.\mathsf{kcind} \leftarrow \mathsf{true}$, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} \leftarrow \mathsf{true}$ and $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{k} \leftarrow \ell_{\mathsf{k}}.\mathsf{k}$.
  - The case of $\ell_{\mathsf{ke}}.\mathsf{accept} = \mathsf{false}$ is never reached as it would already hold that $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} = \mathsf{false}$.
- When $\mathcal{A}$ submits $\mathtt{Reveal}(\ell_{\mathsf{ke};\bar{\pi}})$, $\mathcal{B}$ checks the value of $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept}$. If it is either $\perp$ or false, $\mathcal{B}$ responds $\perp$ to $\mathcal{A}$ as the $\mathtt{Reveal}$ query is invalid. If it is true, then $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{k}$ was set when $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} \leftarrow \mathsf{true}$ so $\mathcal{B}$ responds with $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{k}$ to $\mathcal{A}$ and sets $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{sess}} \leftarrow \mathsf{revealed}$.
- When $\mathcal{A}$ submits $\mathtt{Corrupt}(i)$, $\mathcal{B}$ submits $\mathtt{Corrupt}(i)$ to G^{ke} and receives sk_i which it returns to $\mathcal{A}$. At that moment, G^{ke} will mark the values of $\ell_{\mathsf{ke}}.\delta_{\mathsf{ownr}}$ and $\ell_{\mathsf{ke}}.\delta_{\mathsf{peer}}$ as $\mathsf{corrupt}$ for relevant sessions as decribed in Section 3.2.1. However, $\mathcal{B}$ will not update the corresponding sessions in $G^{\mathsf{ke};\bar{\pi}}$ in the same way as this would leak information to $\mathcal{A}$ about the internal stage of the sessions. Instead, $\mathcal{B}$

marks the values as corrupt for the sessions $\ell_{\mathsf{ke};\bar{\pi}}$ which have not completed the entire composed protocol, even if they have already completed the key exchange protocol and would not be marked as corrupt in G^{ke}.

We now argue that if $\mathcal{A}$ is able to reach an execution state in $G^{\mathsf{ke};\bar{\pi}}$ which contradicts the $\mathsf{iKeyAuth}$ predicate, then $\mathcal{B}$, by behaving as described above, reaches a state in G^{ke} which also contradicts the $\mathsf{iKeyAuth}$ predicate. This means that the composed protocol preserves the implicit key authentication of the key-exchange protocol ke. If $\mathcal{A}$ reaches such a state, then we have that

$$\begin{aligned}\exists \ell_{\mathsf{ke};\bar{\pi}} \in \mathsf{LSID}_{\mathsf{ke};\bar{\pi}} :: (\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{pid} \in \mathcal{S} \wedge \ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept})\\ \wedge \big(\exists \ell'_{\mathsf{ke};\bar{\pi}} \in \mathsf{LSID}_{\mathsf{ke};\bar{\pi}} :: \mathsf{Samekey}(\ell'_{\mathsf{ke};\bar{\pi}}, \ell_{\mathsf{ke};\bar{\pi}})\\ \wedge\, \ell'_{\mathsf{ke};\bar{\pi}}.\mathsf{id} \neq \ell_{\mathsf{ke};\bar{\pi}}.\mathsf{pid}\big).\end{aligned}$$

We show that this also holds for the corresponding sessions ℓ_{ke} and ℓ'_{ke} in G^{ke}. We first have that $\ell_{\mathsf{ke}}.\mathsf{pid} \in \mathcal{S}$ as all the sessions and the set $\mathcal{S}$ match one-to-one between the two games. We then have that $\ell_{\mathsf{ke}}.\mathsf{accept} = \mathsf{true}$ as $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept}$ is set to true only if the ke session accepts, and $\mathcal{B}$ relays $\mathcal{A}$'s $\mathtt{Send}$ queries identically which causes ℓ_{ke} to accept in G^{ke}.

As per the definition of the composed protocol $\mathsf{ke};\bar{\pi}$, the final key is fixed as soon as the ke part completes, therefore it holds that if two sessions accept with the same key in $G^{\mathsf{ke};\bar{\pi}}$, then they have derived that same key in the first part. As $\mathcal{B}$ relays $\mathcal{A}$'s queries identically, we have that $\mathsf{Samekey}(\ell'_{\mathsf{ke}}, \ell_{\mathsf{ke}})$ holds in G^{ke} for the sessions corresponding to $\ell'_{\mathsf{ke};\bar{\pi}}$ and $\ell_{\mathsf{ke};\bar{\pi}}$. Furthermore, $\ell'_{\mathsf{ke}}.\mathsf{id} \neq \ell_{\mathsf{ke}}.\mathsf{pid}$ also holds as these values are the same as the ones for the sessions in $G^{\mathsf{ke};\bar{\pi}}$. This shows that the following holds for the corresponding sessions:

$$\begin{aligned}\exists \ell_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}} :: (\ell_{\mathsf{ke}}.\mathsf{pid} \in \mathcal{S} \wedge \ell_{\mathsf{ke}}.\mathsf{accept})\\ \wedge \big(\exists \ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}} :: \mathsf{Samekey}(\ell'_{\mathsf{ke}}, \ell_{\mathsf{ke}}) \wedge \ell'_{\mathsf{ke}}.\mathsf{id} \neq \ell_{\mathsf{ke}}.\mathsf{pid}\big)\,,\end{aligned}$$

which implies that $\mathcal{B}$ is successful for the game $G^{\mathsf{ke}}_{\mathsf{iKeyAuth}}$ exactly when $\mathcal{A}$ is succesful for the game $G^{\mathsf{ke};\bar{\pi}}_{\mathsf{iKeyAuth}}$. □

Lemma 4.21. *Let* ke *be a* Match*-secure key exchange protocol with output key distribution* D*. Let* π *be a symmetric-key protocol with key generation distribution* D*. Let*

$n = n_i^2 \cdot n_s$. *For any probabilistic* $\mathsf{poly}(\lambda)$*-time adversary* $\mathcal{A}$*, it holds that*

$$\mathbf{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi}}} \leq n \cdot \mathbf{Adv}_{\mathcal{B}_1,\mathsf{ke}}^{G_{BRSec,D}} + \mathbf{Adv}_{\mathcal{B}_2,\pi}^{G_{\mathsf{symKeyConf}}},$$

for some probabilistic $\mathsf{poly}(\lambda)$*-time algorithms* $\mathcal{B}_1$ *and* $\mathcal{B}_2$*.*

Proof. We use a strategy similar to the proof of Theorem 1 in [32], namely we first replace all the keys derived by the ke part of the composed protocol by randomly sampled keys from the correct distribution, using BR-secrecy to show that the final game is indistinguishable from the first. Then we show, similarly to Lemma 4.20, that if an adversary manages to break the key confirmation property of the composed protocol, then a reduction can break the key confirmation property of the symmetric protocol π.

To replace all the keys used, we proceed with a hybrid argument. Let the game $G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},\Sigma,D}$ be the game G_{fKeyConf} played against protocol $\mathsf{ke};\bar{\pi}$, where the first Σ sessions to accept a new key, i.e. where a partner session has not already accepted a key, have their keys from ke replaced by a random value from D for the π part, where $D = D_{\mathtt{KGen}}$ is the output distribution of the key generation algorithm for π. We remove the mention of $\mathsf{fKeyConf}$ when the context is clear. The original game G_{fKeyConf} for $\mathcal{A}$ is therefore $G^{\mathsf{ke};\bar{\pi},0,D}$ where only honestly computed keys are used for π.

The game $G^{\mathsf{ke};\bar{\pi},\Sigma,D}$ runs just as $G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi}}$ does with the following modifications. It maintains a counter σ to keep track of the number of new keys that are accepted (not counting those which the adversary might already know by corrupting one of the parties); this is set to 0 initially. The behaviour of $G^{\mathsf{ke};\bar{\pi},\Sigma,D}$ is then the same as $G^{\mathsf{ke};\bar{\pi}}$ with the following differences to the $\mathtt{Send}(\ell_{\mathsf{ke};\bar{\pi}}, m)$ query:

- If $\sigma \geq \Sigma$, behave as in $G^{\mathsf{ke};\bar{\pi}}$, otherwise:
- If ℓ_{ke} has accepted already, simulate the π part honestly with $\ell_{\bar{\pi}}.\mathsf{k}$;
- Compute the response and the state update according to the ke algorithm;
- If $\ell_{\mathsf{ke}}.\mathsf{accept} \leftarrow \mathsf{true}$:
  - If there exists an $\ell'_{\mathsf{ke};\bar{\pi}} \in \mathsf{LSID}$ such that $\mathsf{Partner}(\ell_{\mathsf{ke};\bar{\pi}}, \ell'_{\mathsf{ke};\bar{\pi}}) = \mathsf{true}$ and $\ell'_{\mathsf{ke}}.\mathsf{accept} = \mathsf{true}$, then set $\ell_{\pi}.\mathsf{k} \leftarrow \ell'_{\pi}.\mathsf{k}$;
  - If there does not exist such an $\ell'_{\mathsf{ke};\bar{\pi}}$ that is partnered and whose ke part has already accepted, but either $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{ownr}} = \mathsf{corrupt}$ or $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{peer}} = \mathsf{corrupt}$ then set $\ell_{\pi}.\mathsf{k} \leftarrow \ell_{\mathsf{ke}}.\mathsf{k}$;

- If both identities are still honest, and no partner session exists or has already accepted a key for the ke part, then set $\ell_\pi.\mathsf{k} \leftarrow_\$ D$ and update $\sigma \leftarrow \sigma + 1$.

With this new behaviour, we have that the first Σ new keys that are unknown to the adversary at the time of their acceptance are replaced with keys sampled from D for the π part of the protocol.

Lemma 4.22 now allows us to change the game $G^{\mathsf{ke};\bar{\pi},0,D}$ into the game $G^{\mathsf{ke};\bar{\pi},n,D}$ for $n = n_i^2 \cdot n_s$ where the indinstinguishability of the two games is guaranteed by the BR-secrecy of the ke protocol. This yields

$$\left|\mathbf{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},0,D}} - \mathbf{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},n,D}}\right| \leq n \cdot \mathbf{Adv}_{\mathcal{B}_1,\mathsf{ke}}^{G_{\mathsf{BRSec}}^{D}},$$

for a first reduction $\mathcal{B}_1$. In Lemma 4.23, we then show that key confirmation of the composed protocol follows from key confirmation of the symmetric-key protocol:

$$\mathbf{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},n,D}} = \mathbf{Adv}_{\mathcal{B}_2,\pi}^{G_{\mathsf{symKeyConf}}}$$

for a second reduction $\mathcal{B}_2$. This allows us to conclude that

$$\mathbf{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi}}} \leq n \cdot \mathbf{Adv}_{\mathcal{B}_1,\mathsf{ke}}^{G_{\mathsf{BRSec},D}} + \mathbf{Adv}_{\mathcal{B}_2,\pi}^{G_{\mathsf{symKeyConf}}}.$$

□

Lemma 4.22. *Let* ke *be a* Match*-secure key exchange protocol with output key distribution D. Let π be a symmetric-key protocol with key generation distribution D. For $\Sigma = 1, \ldots, n_i^2 \cdot n_s$ and for any probabilistic* $\mathsf{poly}(\lambda)$*-time adversary $\mathcal{A}$, we have*

$$\mathbf{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},\Sigma-1,D}} \leq \mathbf{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},\Sigma,D}} + \mathbf{Adv}_{\mathcal{B},\mathsf{ke}}^{G_{\mathit{BRSec},D}},$$

for some probabilistic $\mathsf{poly}(\lambda)$*-time algorithm* $\mathcal{B} = \mathcal{B}(\Sigma)$.

Proof. Given an adversary $\mathcal{A}$ against the game $G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},\Sigma-1,D}$, we construct an algorithm $\mathcal{B}$ against the game $G_{\mathsf{BRSec},D}$. The reduction $\mathcal{B}$ sets up the game for $\mathcal{A}$ as described at the beginning of this section and keeps track of the internal variable of each of the stages of the protocol. It also initialises $\sigma \leftarrow 0$.

As $\mathcal{A}$ runs, $\mathcal{B}$ responds to a $\mathtt{Send}(\ell_{\mathsf{ke};\bar{\pi}}, m)$ query as follows. (We recall that $\ell_{\mathsf{ke};\bar{\pi}}$ refers here to the variables of $G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},\Sigma,D}$ simulated by $\mathcal{B}$ to $\mathcal{A}$, ℓ_{ke} refers here to the variables of $G_{\mathsf{BRSec},D}$ played by $\mathcal{B}$ and that $\ell_{\bar{\pi}}$ refers to the variables for the execution of $\bar{\pi}$ simulated by $\mathcal{B}$.)

- If $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} \in \{\mathsf{true}, \mathsf{false}\}$, $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$; otherwise:
- If $\ell_{\mathsf{ke}}.\mathsf{accept} = \perp$, $\mathcal{B}$ submits $\mathtt{Send}(\ell_{\mathsf{ke}}, m)$ to $G_{\mathsf{BRSec},D}$ and receives an updated state for ℓ_{ke} and a response m'. If $\ell_{\mathsf{ke}}.\mathsf{sid}$ is set, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{sid} \leftarrow \ell_{\mathsf{ke}}.\mathsf{sid}$ for $\mathcal{A}$. If $\ell_{\mathsf{ke}}.\mathsf{accept} \leftarrow \mathsf{false}$, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} \leftarrow \mathsf{false}$ and informs $\mathcal{A}$. If $\ell_{\mathsf{ke}}.\mathsf{accept} \leftarrow \mathsf{true}$, the following takes place:
  - If $\sigma = \Sigma$, $\nexists \ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}} :: \mathsf{Partner}(\ell_{\mathsf{ke}}, \ell'_{\mathsf{ke}})$ and $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{ownr}} \neq \mathsf{corrupt}$ and $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{peer}} \neq \mathsf{corrupt}$, then
    - ∗ Submit $\mathtt{Test}(\ell)$ to $G_{\mathsf{BRSec},D}$ and receive k_{ke}.
    - ∗ Set $\ell_{\bar{\pi}}.\mathsf{k} \leftarrow \mathsf{k}_{\mathsf{ke}}$.
    - ∗ Update $\sigma \leftarrow \sigma + 1$.
  - Else, if $\sigma \leq \Sigma$ then
    - ∗ If there does not exist $\ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}}$ such that $\mathsf{Partner}(\ell_{\mathsf{ke}}, \ell'_{\mathsf{ke}})$ and $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{ownr}} \neq \mathsf{corrupt} \neq \ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{peer}}$, then sample a random key $\mathsf{k}_{\pi} \leftarrow_{\$} D$ and set $\ell_{\bar{\pi}}.\mathsf{k} \leftarrow \mathsf{k}_{\pi}$. Update $\sigma \leftarrow \sigma + 1$.
    - ∗ Else, if $\nexists \ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}} :: \mathsf{Partner}(\ell_{\mathsf{ke}}, \ell'_{\mathsf{ke}})$ and either $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{ownr}} = \mathsf{corrupt}$ or $\ell_{\mathsf{ke};\bar{\pi}}.\delta_{\mathsf{peer}} = \mathsf{corrupt}$, then submit the query $\mathtt{Reveal}(\ell_{\mathsf{ke}})$ to $G_{\mathsf{BRSec},D}$ and receive k_{ke}. Then set $\ell_{\bar{\pi}}.\mathsf{k} \leftarrow \mathsf{k}_{\mathsf{ke}}$.
    - ∗ Else, there exists an $\ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}} :: \mathsf{Partner}(\ell_{\mathsf{ke}}, \ell'_{\mathsf{ke}})$ for which $\ell'_{\bar{\pi}}.\mathsf{k}$ has already been set. Then set $\ell_{\bar{\pi}}.\mathsf{k} \leftarrow \ell'_{\bar{\pi}}.\mathsf{k}$.
  - Else $\sigma > \Sigma$ so perform the following:
    - ∗ If $\exists \ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}} :: \mathsf{Partner}(\ell_{\mathsf{ke}}, \ell'_{\mathsf{ke}})$ then set $\ell_{\bar{\pi}}.\mathsf{k} \leftarrow \ell'_{\bar{\pi}}.\mathsf{k}$.
    - ∗ Else submit the query $\mathtt{Reveal}(\ell_{\mathsf{ke}})$ to $G_{\mathsf{BRSec},D}$, receive k_{ke} and set $\ell_{\bar{\pi}}.\mathsf{k} \leftarrow \mathsf{k}_{\mathsf{ke}}$. Update $\sigma \leftarrow \sigma + 1$.

If $\mathcal{A}$ submits a $\mathtt{Reveal}(\ell_{\mathsf{ke};\bar{\pi}})$ query, $\ell_{\mathsf{ke};\bar{\pi}}$ must have accepted for it to be valid. Therefore $\mathcal{B}$ has already manually set the internal key $\ell_{\bar{\pi}}.\mathsf{k}$ and it can return it to $\mathcal{A}$ consistently.

If $\mathcal{A}$ submits a $\mathtt{Corrupt}(i)$ query, $\mathcal{B}$ marks all relevant sessions $\ell_{\mathsf{ke};\bar{\pi}} \in \mathsf{LSID}_{\mathsf{ke};\bar{\pi}}$ as $\mathsf{corrupt}$ if they are still running and then submits $\mathtt{Corrupt}(i)$ to $G_{\mathsf{BRSec},D}$ to receive sk_i and return it to $\mathcal{A}$.

In the processing of a $\mathtt{Send}$ query, when $\sigma > \Sigma$ and there is an existing partner session, we initialise the key directly from the partner session's. As we assume that ke

is Match-secure, these two partner sessions will derive the same key with overwhelming probability.

We note that if the $\texttt{Test}$ query returns the real key, then $\mathcal{B}$ will perfectly simulate $G^{\mathsf{ke};\bar{\pi},\Sigma-1,D}_{\mathsf{fKeyConf}}$ to $\mathcal{A}$, but if it returns a random key from D, then $\mathcal{B}$ will perfectly simulate $G^{\mathsf{ke};\bar{\pi},\Sigma,D}_{\mathsf{fKeyConf}}$. When $\mathcal{A}$ terminates and was successful, $\mathcal{B}$ submits $\texttt{Guess}(1)$ to $G_{\mathsf{BRSec},D}$; it submits $\texttt{Guess}(0)$ otherwise. The advantage of $\mathcal{B}$ in $G_{\mathsf{BRSec},D}$ therefore corresponds to the difference in the success probability of $\mathcal{A}$ as we have

$$\Pr\left[\mathbf{Exp}^{G^0_{\mathsf{BRSec},D}}_{\mathcal{B},\mathsf{ke}} = 1\right] = \mathbf{Adv}^{G^{\mathsf{ke};\bar{\pi},\Sigma,D}_{\mathsf{fKeyConf}}}_{\mathcal{A},\mathsf{ke};\bar{\pi}}$$

and

$$\Pr\left[\mathbf{Exp}^{G^1_{\mathsf{BRSec},D}}_{\mathcal{B},\mathsf{ke}} = 1\right] = \mathbf{Adv}^{G^{\mathsf{ke};\bar{\pi},\Sigma-1,D}_{\mathsf{fKeyConf}}}_{\mathcal{A},\mathsf{ke};\bar{\pi}}$$

which gives

$$\mathbf{Adv}^{G_{\mathsf{BRSec},D}}_{\mathcal{B},\mathsf{ke}} = \left|\mathbf{Adv}^{G^{\mathsf{ke};\bar{\pi},\Sigma-1,D}_{\mathsf{fKeyConf}}}_{\mathcal{A},\mathsf{ke};\bar{\pi}} - \mathbf{Adv}^{G^{\mathsf{ke};\bar{\pi},\Sigma,D}_{\mathsf{fKeyConf}}}_{\mathcal{A},\mathsf{ke};\bar{\pi}}\right|$$

and yields the desired result. □

Lemma 4.23. *Let* ke *be a* Match*-secure key exchange protocol with output key distribution* D *and* π *be a symmetric-key protocol with key generation distribution* D*. Let* $n = n_i^2 \cdot n_s$*. For any probabilistic* $\mathsf{poly}(\lambda)$*-time adversary* $\mathcal{A}$*, it holds that*

$$\mathbf{Adv}^{G^{\mathsf{ke};\bar{\pi},n,D}_{\mathsf{fKeyConf}}}_{\mathcal{A},\mathsf{ke};\bar{\pi}} = \mathbf{Adv}^{G_{\mathsf{symKeyConf}}}_{\mathcal{B},\pi},$$

for some probabilistic $\mathsf{poly}(\lambda)$*-time algorithm* $\mathcal{B}$*.*

Proof. Similarly to the proof of Lemma 4.20, we build a reduction $\mathcal{B}$ against π in $G_{\mathsf{symKeyConf}}$ which uses an adversary against $\mathsf{ke};\bar{\pi}$ in $G^{\mathsf{ke};\bar{\pi},n,D}_{\mathsf{fKeyConf}}$, which we refer to as $G^{\mathsf{ke};\bar{\pi},n}$ in this proof for simplicity.

The algorithm $\mathcal{B}$ sets up $G^{\mathsf{ke};\bar{\pi},n}$ for $\mathcal{A}$ by simulating all the elements relevant to the ke stage of the composed protocol. It then responds to $\mathcal{A}$'s queries as follows (we once again use $\ell_{\mathsf{ke};\bar{\pi}}$ to refer to identifiers used by $\mathcal{A}$, ℓ_{ke} for corresponding identifiers simulated internally by $\mathcal{B}$ and ℓ_{π} for those used by $\mathcal{B}$ in $G_{\mathsf{symKeyConf}}$).

- When $\mathcal{A}$ submits $\texttt{Send}(\ell_{\mathsf{ke};\bar{\pi}}, m)$: if $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} \in \{\mathsf{true}, \mathsf{false}\}$, $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$. Otherwise:
  - If $\ell_{\mathsf{ke}}.\mathsf{accept} = \perp$, $\mathcal{B}$ simulates the execution of ke. If $\ell_{\mathsf{ke}}.\mathsf{accept} \leftarrow \mathsf{false}$, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\bar{\pi}}.\mathsf{accept} \leftarrow \mathsf{false}$. If $\ell_{\mathsf{ke}}.\mathsf{accept} \leftarrow \mathsf{true}$, $\mathcal{B}$ leaves $\ell_{\mathsf{ke}}.\mathsf{k} = \perp$ and then:

    * If $\ell_{\mathsf{ke}}.\delta_{\mathsf{peer}} = \mathsf{honest} \land \nexists \ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}} :: (\mathsf{Partner}(\ell_{\mathsf{ke}}, \ell'_{\mathsf{ke}}) \land \ell'_{\mathsf{ke}}.\mathsf{accept} = \mathsf{true})$, then $\mathcal{B}$ submits $\mathtt{InitS}(\ell_\pi)$ to $G_{\mathsf{symKeyConf}}$.
    * If $\ell_{\mathsf{ke}}.\delta_{\mathsf{peer}} = \mathsf{corrupt} \land \nexists \ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}} :: (\mathsf{Partner}(\ell_{\mathsf{ke}}, \ell'_{\mathsf{ke}}) \land \ell'_{\mathsf{ke}}.\mathsf{accept} = \mathsf{true})$, then $\mathcal{B}$ submits $\mathtt{InitS}(\ell_\pi)$ and then $\mathtt{Reveal}(\ell_\pi)$ to $G_{\mathsf{symKeyConf}}$ to generate and obtain k_π which it saves by setting $\ell_{\mathsf{ke}}.\mathsf{k} \leftarrow \mathsf{k}_\pi$.
    * If $\exists \ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}} :: (\mathsf{Partner}(\ell_{\mathsf{ke}}, \ell'_{\mathsf{ke}}) \land \ell'_{\mathsf{ke}}.\mathsf{accept} = \mathsf{true})$, then $\mathcal{B}$ submits $\mathtt{InitP}(\ell'_\pi, \ell_\pi)$ to $G_{\mathsf{symKeyConf}}$ and sets $\ell_{\mathsf{ke}}.\mathsf{k} \leftarrow \ell'_{\mathsf{ke}}.\mathsf{k}$.

  – If $\ell_{\mathsf{ke}}.\mathsf{accept} = \mathsf{true}$, $\mathcal{B}$ submits $\mathtt{Send}(\ell_\pi, m)$ to $G_{\mathsf{symKeyConf}}$ and returns the reply to $\mathcal{A}$. If $\ell_\pi.\mathsf{kcind} \leftarrow \mathsf{false}$, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\overline{\pi}}.\mathsf{accept} \leftarrow \mathsf{false}$. If $\ell_\pi.\mathsf{kcind} \leftarrow \mathsf{true}$, $\mathcal{B}$ sets $\ell_{\mathsf{ke};\overline{\pi}}.\mathsf{accept} \leftarrow \mathsf{true}$ and sets $\ell_{\mathsf{ke};\overline{\pi}}.\mathsf{k} \leftarrow \ell_{\mathsf{ke}}.\mathsf{k}$.

- When $\mathcal{A}$ submits $\mathtt{Reveal}(\ell_{\mathsf{ke};\overline{\pi}})$: if $\ell_{\mathsf{ke};\overline{\pi}}.\mathsf{accept} \in \{\perp, \mathsf{false}\}$, $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$. Otherwise:

  – If $\ell_{\mathsf{ke};\overline{\pi}}.\mathsf{k} \neq \perp$, $\mathcal{B}$ returns $\ell_{\mathsf{ke};\overline{\pi}}.\mathsf{k}$ to $\mathcal{A}$.

  – If $\ell_{\mathsf{ke};\overline{\pi}}.\mathsf{k} = \perp$, $\mathcal{B}$ submits $\mathtt{Reveal}(\ell_\pi)$ to $G_{\mathsf{symKeyConf}}$ to obtain k_π, sets $\ell_{\mathsf{ke};\overline{\pi}}.\mathsf{k} \leftarrow \mathsf{k}_\pi$ and sets $\ell'_{\mathsf{ke};\overline{\pi}}.\mathsf{k} \leftarrow \mathsf{k}_\pi$ for any $\ell'_{\mathsf{ke}} \in \mathsf{LSID}_{\mathsf{ke}}$ such that $\mathsf{Partner}(\ell_{\mathsf{ke}}, \ell'_{\mathsf{ke}}) \land \ell'_{\mathsf{ke}}.\mathsf{accept} = \mathsf{true}$.

- When $\mathcal{A}$ submits $\mathtt{Corrupt}(i)$: $\mathcal{B}$ marks all relevant sessions $\ell_{\mathsf{ke};\overline{\pi}} \in \mathsf{LSID}_{\mathsf{ke};\overline{\pi}}$ as $\mathsf{corrupt}$ (either δ_{ownr} or δ_{peer}) and returns sk_i to $\mathcal{A}$.

By processing each query as above, the algorithm $\mathcal{B}$ ensures that the first session that accepts the ke stage within a potential partnership pair is mapped to a new session in $G_{\mathsf{symKeyConf}}$ by an $\mathtt{InitS}$ query. If the peer of that session was already corrupt, then $\mathcal{B}$ submits a $\mathtt{Reveal}$ query so that this session is flagged as $\mathsf{revealed}$ in the game for π. If a session is the second to accept within a partnership pair at the ke stage, then $\mathcal{B}$ uses an $\mathtt{InitP}$ query to initialise it with the same key as its partner and to give it the same value for δ_{k} within $G_{\mathsf{symKeyConf}}$. As we assume that ke is Match-secure, these two partner sessions will derive the same key with overwhelming probability. This, together with $\mathcal{B}$'s handling of $\mathcal{A}$'s $\mathtt{Reveal}$ and $\mathtt{Corrupt}$ queries ensures that every session for which $\mathcal{A}$ could trivially obtain the session key is immediately marked as revealed in $G_{\mathsf{symKeyConf}}$.

Furthermore, since the key derived by $\mathcal{B}$'s internal simulation of the ke stage is never used by the π stage, but instead replaced with a randomly generated key using an $\mathtt{InitS}$ query, $\mathcal{B}$ provides a perfect simulation of $G^{\mathsf{ke};\overline{\pi},n}$ to $\mathcal{A}$. Therefore, as $\mathcal{B}$ relays $\mathcal{A}$'s $\mathtt{Send}$ queries exactly, we see that if $\mathcal{A}$ wins against the $\mathsf{fKeyConf}$ predicate in $G^{\mathsf{ke};\bar{\pi},n}$ then $\mathcal{B}$ will also reach a state that wins against $G_{\mathsf{symKeyConf}}$. We therefore have

$$\mathbf{Adv}_{\mathcal{A},\mathsf{ke};\bar{\pi}}^{G_{\mathsf{fKeyConf}}^{\mathsf{ke};\bar{\pi},n,D}} = \mathbf{Adv}_{\mathcal{B},\pi}^{G_{\mathsf{symKeyConf}}}.$$

□

Chapter 5

From definitions to protocols

In this chapter we first take the model for authentication security of AKE protocols developed in Chapters 3 and 4 and apply it to the study of existing protocols. We consider three emblematic protocols of the AKE literature (Diffie–Hellman, HMQV and TLS 1.3) and analyze which security guarantees are achieved. These analyses allow us to derive general principles to achieve certain notions of authentication. We then design a generic transformation from passive key-exchange protocols to explicitly authenticated ones which achieves BR security.

The contents of Section 5.1 have appeared in [56]; the ideas of analysis were the joint work of this author and his co-authors and the proof of Proposition 5.1 is this author's own work. The contents of Section 5.2 are the joint work of N. P. Smart[1], B. Warinschi[2] and this author; it was published in the proceedings of *Information Security - 20th International Conference, ISC 2017* and presented by this author at the conference in Ho Chi Minh City, Vietnam [59]. The ideas and the design of the transformation were the joint work of this author and his co-authors and, except where mentioned, the content reproduced here is this author's own work.

5.1 Studying authentication in practice

In this section we present established protocols and study which of our authentication notions they achieve.

Our results confirm that a "rule of thumb" for protocol design to achieve implicit key authentication is to include the parties' identities in the key derivation step,

[1]University of Bristol, U.K., later KU Leuven, Belgium.
[2]University of Bristol, U.K., later Dfinity, Switzerland.

Client A		**Server** B
id_B is known	$\mathbb{G} = \langle g \rangle$ of order q	id_A is known
$x \leftarrow_\$ \mathbb{Z}_q, X = g^x$	$\xrightarrow{\quad X \quad}$	$y \leftarrow_\$ \mathbb{Z}_q, Y = g^y$
$\mathsf{k} = \mathsf{KDF}(Y^x, (X, Y, [\mathsf{id}_A, \mathsf{id}_B]))$	$\xleftarrow{\quad Y \quad}$	$\mathsf{k} = \mathsf{KDF}(X^y, (X, Y, [\mathsf{id}_A, \mathsf{id}_B]))$
$\mathsf{sid} = \mathsf{kcid} = (\mathsf{id}_A, X, \mathsf{id}_B, Y)$		$\mathsf{sid} = \mathsf{kcid} = (\mathsf{id}_A, X, \mathsf{id}_B, Y)$

Figure 5.1: The plain Diffie–Hellman protocol; $[\mathsf{id}_A, \mathsf{id}_B]$ means the values are optional.

$\mathsf{k} = \mathsf{KDF}(K, (\mathsf{id}_A, \mathsf{id}_B, \dots))$. If the key derivation function is collision-resistant then different identities immediately imply different session keys. Indeed this strategy has already been applied in very early protocols proposals, such as [14], and has even been sometimes used to fix insecure protocols, e.g., [42]. We note that this method is also used in the TLS 1.3 protocol.

From the analysis of key confirmation in TLS 1.3 of [69], we see that a good strategy to obtain full or almost-full key confirmation is to send a MAC computed over a known value (such as the transcript) with a key derived from the same material as the final session key. This is supported by our proof that the simple protocol of Figure 4.1 in Section 4.5 is sufficient to provide key confirmation.

Furthermore, the key-entity authentication equivalence also shows that, similarly to including the identities in the key to ensure implicit key authentication, including the identities in the session identifiers ensures implicit entity authentication. It also suggests that involving the sids in a MAC is a good method for ensuring entity confirmation.

5.1.1 Plain Diffie–Hellman

We begin with the plain Diffie–Hellman (DH) protocol, presented in Figure 5.1, in which the parties exchange g^x and g^y to derive a key from g^{xy} and the communication transcript. One may also use the identities in the key derivation as indicated by the square brackets. The exchanged values are elements of a cyclic group $\mathbb{G}$ of cryptographic prime order $|\mathbb{G}| = q$ with generator g. We assume that this group is known to all parties. Since this is an unauthenticated protocol, it sets $\mathcal{S} = \emptyset$.

Match security. The plain DH protocol provides Match security. Indeed both the session and key-confirmation identifiers fully determine the key (Properties 4.1

and 4.2). Furthermore, since the key-confirmation and session identifiers are identical, equal key-confirmation identifiers imply identical keys (Property 4.4). Finally, an honest party will contribute a Diffie–Hellman share at random such that the probability of matching any other share is at most $n_s^2 \cdot \frac{1}{q}$, for a total number of n_s sessions, and thus negligible (Property 4.3).

Implicit key authentication. While it is counterintuitive, the plain DH protocol formally provides implicit key authentication, since $\mathcal{S} = \emptyset$. We note that setting $\mathcal{S} \neq \emptyset$ immediately allows an adversary to break implicit key authentication: it can relay messages between sessions that are not expecting each other as intended partners.

However, by including the identities in the key derivation function (as shown in Figure 5.1 with the square bracket notation) this protocol can provide implicit key authentication even in the setting where $\mathcal{S} \neq \emptyset$. Indeed, the adversary has no control over a session's owner identity $\ell.\mathsf{id}$ which implies that if $\ell.\mathsf{pid} \neq \ell'.\mathsf{id}$ then $\ell.\mathsf{k} = \ell'.\mathsf{k}$ only if there is a collision in the KDF. In the random oracle model, or assuming a collision-resistant KDF, this happens only with negligible propability. By selecting an appropriate post-quantum KDF, the same holds against quantum adversaries.

We note that this argument relies on the *pre-specified* peer model; in the setting where parties learn the identity of their intended peer from the transcript of the session, an adversary can potentially trick an honest session in accepting a different pid than the one from which the message truly comes from.

Key confirmation and explicit key authentication. This protocol does not provide key confirmation since no information about the key is exchanged after it is derived. We can show this formally by taking an adversary which initiates a session with an honest party (either client or server) without initiating a matching partner session. It then creates the initial or response message g^x or g^y to complete the exchange with the honest party. There is then no other session in the model which holds the same key nor the same key-confirmation identifier.

As the protocol does not provide key confirmation, Theorem 4.9 implies that it cannot provide explicit key authentication either. Intuitively this is clear since no authentication mechanism is part of the protocol and the parties do not hold long-term authenticating information.

Client A $(a, A = g^a, \mathsf{cert}_A)$		**Server** B $(b, B = g^b, \mathsf{cert}_B)$
cert_B is known	$\mathbb{G} = \langle g \rangle$ of order q	cert_A is known
$x \leftarrow_{\$} \mathbb{Z}_q, X = g^x$		$y \leftarrow_{\$} \mathbb{Z}_q, Y = g^y$
	$\xrightarrow{\quad X \quad}$	
	$\xleftarrow{\quad Y \quad}$	
$d = H(X, \mathsf{cert}_B)$		$d = H(X, \mathsf{cert}_B)$
$e = H(Y, \mathsf{cert}_A)$		$e = H(Y, \mathsf{cert}_A)$
$K = (YB^e)^{x+da}$		$K = (XA^d)^{y+eb}$
$\mathsf{k} = \mathsf{KDF}(K)$		$\mathsf{k} = \mathsf{KDF}(K)$
$\mathsf{sid} = \mathsf{kcid} = (\mathsf{cert}_A, X, \mathsf{cert}_B, Y)$		$\mathsf{sid} = \mathsf{kcid} = (\mathsf{cert}_A, X, \mathsf{cert}_B, Y)$

Figure 5.2: The HMQV protocol. Note that the certificates contain the corresponding public key and identity.

5.1.2 HMQV

We next look at one of the most prominent candidates for implicitly authenticated key-exchange, the HMQV protocol [92]. The idea here is to use the same communication as in the plain DH key-exchange but to mix Schnorr-type signatures under the parties' public keys in the key derivation. These signatures are not sent but only used locally, thus "implicitly" authenticating the key.

The protocol works over a group $\mathbb{G} = \langle g \rangle$ and uses a hash function H to compute the Schnorr signature. It is mutually authenticating, i.e. $\mathcal{S} = \mathcal{I}$, for which both parties use a long-term key. We assume that each party holds a certificate cert_i for its public key, and also a certificate cert_j for the public key of its intended peer. We also assume that the public key and the owner's identity can be recovered from the certificate.

As in the plain DH case, the session and key-confirmation identifiers fully determine the key, they are identical and each DH share is sampled at random; therefore, Match security of the HMQV protocol follows from the same argument.

Implicit key authentication. We prove that the HMQV protocol achieves our strong notion of implicit key authentication and is secure against all possible UKS and KCI attacks. Recall that we need to show that

$$\forall \ell \in \mathsf{LSID}, (\ell.\mathsf{pid} \in \mathcal{S} \wedge \ell.\mathsf{accept}) \implies \\ \forall \ell' \in \mathsf{LSID}, \big(\mathsf{Samekey}(\ell', \ell) \implies \ell'.\mathsf{id} = \ell.\mathsf{pid}\big) .$$

Due to the differences in the corruption queries that the adversary is allowed to make, the proof of secrecy for HMQV in [92] is not immediately sufficient to imply our strong notion of authentication. Indeed, this proof holds only when the test-session in the secrecy experiment is fresh *in the sense of secrecy freshness* (see Definition 3.8). An attack on implicit key authentication which would require the corruption of the owner before the session took place would not, therefore, be considered as valid against key secrecy and would not be ruled out by the proof of [92].

Proposition 5.1. *In the random oracle model, HMQV provides unconditional mutual implicit key authentication, with*

$$\mathbf{Adv}^{G_{\mathsf{iKeyAuth}}}_{\mathcal{A},\mathrm{HMQV}} \leq \frac{n_i^2 \cdot n_s \cdot h}{2q} + \mathsf{negl}(\lambda) ,$$

where h is the number of queries made to H.

Proof. To break the $\mathsf{iKeyAuth}$ predicate, it must be that a session $\ell_A = (A, B, *)$ shares a key with a session $\ell_C = (C, D, *)$ where $C \neq B$. This can happen either if $K_A = K_C$, or if $K_A \neq K_C$ but $\mathsf{KDF}(K_A) = \mathsf{KDF}(K_C)$. The later implies a collision in the KDF and we assume that this happens only with negligible probability. The only degree of freedom that $\mathcal{A}$ then has is to modify the Y value sent to ℓ_A as a response to its first message. Since the value of K_C, x, d and a are already fixed, $\mathcal{A}$ must choose a value of Y such that YB^e, where $e = H(Y, \mathsf{cert}_A)$ is exactly the right value such that $K_A = K_C$. Modeling H as a random oracle ensures that each value of Y yields a new random value of e and therefore that there is a probability of $1/q$ that a given value of Y will yield the correct value of YB^e, where q is the size of the group $\mathbb{G}$. Given that there are at $n_i^2 \cdot n_s/2$ pairs of sessions, it holds that the adversary has at most a $n_i^2 \cdot n_s \cdot h/2q$ probability of finding a suitable Y for which the equality holds.

Considering quantum adversaries, these would have access to the oracle H in superposition. Modeling it as a quantum random oracle is not an issue for the proof strategy above, as H is not programmed. However, if a good value of Y exists, then it will be found using Grover's algorithm [76] with high probability in time $O(\sqrt{q})$. By selecting an appropriate value of q, and an appropriate post-quantum collision-resistant KDF, then the result also holds against quantum adversaries. (Note that this does not imply that HMQV provides post-quantum key-secrecy.) $\square$

Similarly to the plain DH protocol, setting $\mathsf{k} = \mathsf{KDF}(K, (\mathsf{cert}_A, \mathsf{cert}_B))$ immediately provides implicit key authentication if the KDF is (quantum) collision resistant; again this argument holds because of the pre-specified peer model: the adversary cannot influence a session's intended peer. One could also include the values X and Y in the key derivation. In that setting, continuing with the notation of the proof, any change in Y to ensure $K_A = K_C$ would still result in different inputs to the KDF therefore leaving an attack on the collision-resistance of the KDF as the only avenue.

Key confirmation and explicit key authentication. HMQV does not provide key confirmation in the same way that the plain DH does not. It immediately follows that it does not provide explicit key authentication either.

5.1.3 TLS 1.3

We give a simplified version of the DH mode of the TLS 1.3 protocol suite in Figure 5.3 which omits certain steps unrelated to the authentication of the parties and the session-keys, such as the derivation of the handshake key and the encryption of the handshake protocol. We only look at the case of server-only authentication, where $B \in \mathcal{S}$ and $A \in \mathcal{I} \setminus \mathcal{S}$, without session resumption or pre-shared secrets. We also assume that the protocol is in a "pre-specified peer mode" where the client knows the certificate of the server in advance.

Match security and implicit key authentication. TLS 1.3 is Match-secure; the argument is identical to the plain DH case and appears in [64]. Implicit key authentication follows as for the HMQV variant with identifiers, if we assume that the KDF function is collision-resistant; as the server authenticates and cert_B appears in the key derivation, equal keys imply a correct authentication. Similarly to the HMQV protocol, key secrecy of TLS is not enough to imply implicit key authentication.

Key confirmation and explicit key authentication. Key confirmation for TLS 1.3 (`draft-10`) was shown in [69] and our version corresponds to this variant. The idea of the proof is that the parties use the key in the handshake protocol within the MAC. This holds for full and almost-full key confirmation, such that our version also provides explicit key authentication.

Client A		**Server** B $(\mathsf{sk}_B, \mathsf{pk}_B, \mathsf{cert}_B)$
cert_B is known	$\mathbb{G} = \langle g \rangle$ of order q	
$r \leftarrow_{\$} \{0,1\}^n$ $x \leftarrow_{\$} \mathbb{Z}_q, X = g^x$		$s \leftarrow_{\$} \{0,1\}^n$ $y \leftarrow_{\$} \mathbb{Z}_q, Y = g^y$
	$\xrightarrow{r, X}$	
	$\xleftarrow{s, Y}$	
		$\sigma_B \leftarrow \mathsf{Sig}(\mathsf{sk}_B, (r, \ldots, Y))$ $k_B = \mathsf{KDF}(X^y,$ $\quad \texttt{"server"}, (r, \ldots, Y))$ $\tau_B \leftarrow \texttt{MAC}(k_B, (r, \ldots, \sigma_B))$
verify cert_B	$\xleftarrow{\mathsf{cert}_B, \sigma_B, \tau_B}$	
$\texttt{Vfy}(\mathsf{pk}_B, \sigma_B, (r, \ldots, Y))$ $k_B = \mathsf{KDF}(Y^x,$ $\quad \texttt{"server"}, (r, \ldots, Y))$ $\texttt{Vfy}(k_B, \tau_B, (r, \ldots, \sigma_B))$		
$k_A = \mathsf{KDF}(Y^x,$ $\quad \texttt{"client"}, (r, \ldots, \tau_B))$		
$\tau_A \leftarrow \texttt{MAC}(k_A, (r, \ldots, \tau_B))$	$\xrightarrow{\tau_A}$	
		$k_A = \mathsf{KDF}(X^y,$ $\quad \texttt{"client"}, (r, \ldots, \tau_B))$ $\texttt{Vfy}(k_A, \tau_A, (r, \ldots, \tau_B))$
$\mathsf{k} = \mathsf{KDF}(Y^x,$ $\quad \texttt{"app"}, (r, \ldots, \tau_A))$ $\mathsf{sid} = \mathsf{kcid} = (r, X, s, Y, \mathsf{cert}_B)$		$\mathsf{k} = \mathsf{KDF}(X^y,$ $\quad \texttt{"app"}, (r, \ldots, \tau_A))$ $\mathsf{sid} = \mathsf{kcid} = (r, X, s, Y, \mathsf{cert}_B)$

Figure 5.3: (Simplified) TLS 1.3 in (EC)DHE key-exchange mode, without handshake encryption and with server-only authentication. Notation $x, \ldots, y$ means all transmitted communication data, ranging from x to y.

5.2 Transformation for post-quantum AKE

In this next section we present a new transformation which turns a two-round un-authenticated key-exchange (UKE) protocol into a three-round AKE protocol. We first present the definition of two-round protocols that we will start from together with some new security notions which we will assume for the security of the transformation. We then present the AKE protocol itself, discuss the motivation of its construction and prove its security in the BR model.

5.2.1 Un-authenticated key-exchange protocols

First, we formalise (two-round) UKE protocols and what it means for such protocols to be passively secure. Informally we consider a protocol passively secure if the transcript of one session does not leak any information about the resulting session-key (in an indistinguishability notion similar to BR secrecy). As they do not aim to provide authentication, such protocols make no usage of long term keys. We note that for these protocols, for a given session ℓ, its session state is $\mathsf{SST}[\ell] = (\rho, \mathsf{crypt}, \mathsf{tran}, \mathsf{accept}, \mathsf{sid}, \mathsf{k})$ where each element is defined as in Section 3.1.2.

Definition 5.2 (Unauthenticated key exchange protocol). An *un-authenticated key exchange protocol* is a pair of probabilisitc $\mathsf{poly}(\lambda)$-time algorithms $\Pi = (\mathtt{Setup}, \zeta)$ such that:

1. The *setup* algorithm $\mathtt{Setup}$ takes in the security parameter 1^λ and outputs a tuple of public parameters, params, required by the protocol. Among other information params always contains the parameter 1^λ and specifies a message space $\mathcal{M}$ and a key space $\mathcal{K}$.

2. The *protocol function* ζ computes the next step in the protocol execution of session ℓ given a new incoming message m. In doing so, it computes the next message that ℓ should send and also updates internal values within ℓ; i.e. $m' \leftarrow \zeta(\mathsf{SST}[\ell], m)$. Sometimes we write $m' \leftarrow \zeta(\mathsf{SST}[\ell], m; \$)$ to make the randomness used by ζ explicit.

A UKE protocol is said to be *correct* if when the messages are relayed faithfully, i.e. unmodified and in the correct order, between two participants, then they both accept and compute identical session keys, except with negligible probability over the randomness used in the algorithms.

$$\begin{array}{lcl} \text{Client } A & & \text{Server } B \\ m_1 \leftarrow \zeta(\mathsf{SST}[\ell_A], \vdash; \$_1) & \xrightarrow{m_1} & \\ & \xleftarrow{m_2} & m_2 \| \dashv \leftarrow \zeta(\mathsf{SST}[\ell_B], m_1; \$_2) \\ \dashv \leftarrow \zeta(\mathsf{SST}[\ell_A], m_2; \$_3) & & \\ \text{Output } \ell_A.\mathsf{sid}, \ell_A.\mathsf{k} & & \text{Output } \ell_B.\mathsf{sid}, \ell_B.\mathsf{k} \end{array}$$

Figure 5.4: A two-round un-authenticated key exchange protocol.

As for AKE protocols, in practice we define a specific key-exchange protocol by defining how each new input message is responded to, given the current session state. We present a two-round example of such a UKE protocol in Figure 5.4.

For example, the plain DH is an instantiation where A's ephemeral key is e_A and m_1 is g^{e_A}, and B's ephemeral key is e_B (which can be deleted as soon as it is used to derive $m_2 = g^{e_B}$ and $\ell_B.\mathsf{k} = m_1^{e_B}$), finally the computation of $\ell_A.\mathsf{k}$ is done $m_2^{e_A}$.

We next recall basic notions of security for passive key-exchange protocols and introduce two new formal definitions.

Passive security against eavesdroppers

For UKE protocols the best security guarantee we can obtain is that of passive security. Such a protocol is said to be passively secure if a single session of the protocol does not leak any information regarding the computed session key to an arbitrary $\mathsf{poly}(\lambda)$-time adversary $\mathcal{A}$ that only eavesdrops on the conversation. For a UKE protocol Π and an adversary $\mathcal{A}$, this is formalised in the $\mathsf{EAV\text{-}KE}$ experiment described in Figure 5.5 for a given bit $b \in \{0, 1\}$. We denote $\mathcal{A}$'s advantage in the $\mathsf{EAV\text{-}KE}$ game as

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{\mathsf{EAV\text{-}KE}} = \left| \Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{\mathsf{EAV\text{-}KE},1} = 1\right] - \Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{\mathsf{EAV\text{-}KE},0} = 1\right] \right|.$$

Definition 5.3 (Passive UKE security)**.** A UKE protocol Π is *passively (post-quantum) secure (in the presence of an eavesdropper)* if for all probabilistic $\mathsf{poly}(\lambda)$-time (quantum) adversaries $\mathcal{A}$, the following conditions hold:

1. If messages are relayed faithfully by a benign adversary between two sessions, then both accept holding identical session keys, and each session's key is distributed uniformly at random over $\mathcal{K}$.

1. The challenger sets 1^λ, runs $\mathtt{Setup}$ to obtain params and makes two blank sessions execute protocol Π with one another. This results in a transcript tran of the messages, and a key k output by each of the sessions.
2. If $b = 1$, set $\hat{\mathsf{k}} \leftarrow \mathsf{k}$, and if $b = 0$ then sample $\hat{\mathsf{k}} \leftarrow\!\!\$\, \mathcal{K}$ uniformly at random.
3. $\mathcal{A}$ is given tran and $\hat{\mathsf{k}}$, and outputs a guess bit b'.
4. The output of the experiment is 1 if $b' = b$, and 0 otherwise.

Figure 5.5: The $\mathsf{EAV\text{-}KE}, b$ security experiment $\mathbf{Exp}_{\mathcal{A},\Pi}^{\mathsf{EAV\text{-}KE},b}$.

2. There exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{\mathsf{EAV\text{-}KE}} \leq \mathsf{negl}(\lambda)\,.$$

We note that this syntax captures that of the plain Diffie–Hellman of Figure 5.1, as well as that of the lattice-based Frodo and NewHope key-exchange protocols [26, 4]. In addition it is straightforward to show that the plain DH protocol meets our passive UKE security defnition, assuming that the decisional Diffie–Hellman problem is hard, and to check that the proofs of security of Frodo and NewHope also imply quantum security under our passive UKE definition.

Minor active security properties

We now introduce two new simple active security notions relevant to UKE protocols. Most well-designed passive UKE schemes are implicitly understood to satisfy these two notions, but we choose to make them explicit (with the definition of two new security experiments) as the security of the transformation we present below will rely on these.

The first of these formalises the notion of the first protocol message being sufficiently "unpredictable"; i.e. the adversary is not able to guess what the first message m_1 of the transcript tran is going to be. We define the $\mathsf{M1\text{-}Guess}$ experiment in Figure 5.6 and denote an arbitrary adversary $\mathcal{A}$'s advantage in that game as

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{\mathsf{M1\text{-}Guess}} = \Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{\mathsf{M1\text{-}Guess}} = 1\right].$$

Definition 5.4 (First-message randomness)**.** A UKE protocol Π provides *(quantum) first-message randomness* if for all probabilistic $\mathsf{poly}(\lambda)$-time (quantum) adversaries $\mathcal{A}$

1. The challenger sets 1^λ, runs `Setup` to obtain params and computes $m_1 \leftarrow \zeta(\mathsf{SST}[\ell], \vdash; \$)$ for a blank session ℓ and fresh randomness \$.
2. $\mathcal{A}$ is given 1^λ and params and outputs a guess message m_1'.
3. The output of the experiment is 1 if $m_1' = m_1$, and 0 otherwise.

Figure 5.6: The M1-Guess security experiment $\mathbf{Exp}_{\mathcal{A},\Pi}^{\mathsf{M1\text{-}Guess}}$.

there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{\mathsf{M1\text{-}Guess}} \leq \mathsf{negl}(\lambda)\,.$$

The second security notion models the property that an adversary should not be able to obtain information about the final key k even if it may choose the first protocol message. This definition applies only to two-round UKE protocols. To this intent, we define the experiment $\mathsf{Key\text{-}Force}\text{-}b$ in Figure 5.7 and denote an arbitrary adversary $\mathcal{A}$'s advantage as

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{\mathsf{Key\text{-}Force}} = \left|\Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{\mathsf{Key\text{-}Force\text{-}0}} = 1\right] - \Pr\left[\mathbf{Exp}_{\mathcal{A},\Pi}^{\mathsf{Key\text{-}Force\text{-}1}} = 1\right]\right|$$

Definition 5.5 (Key-force security)**.** A UKE protocol Π provides *(quantum) key-force security* if for all probabilistic $\mathsf{poly}(\lambda)$-time quantum adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{A},\Pi}^{\mathsf{Key\text{-}Force}} \leq \mathsf{negl}(\lambda)\,.$$

5.2.2 A new transformation

We now present a transformation which boostraps a two-round UKE protocol into an AKE protocol with full mutual authentication. The transformation furthermore inherits (post-quantum) forward-secrecy from the authentication-independent $\mathsf{EAV\text{-}KE}$ security of the two-round protocol. Our construction, presented in Figure 5.8, requires a two-round UKE protocol $\Pi = (\mathtt{Setup}, \zeta)$, a public key encryption scheme secure under chosen ciphertext attacks, a strongly unforgeable message authentication code, and two key derivation functions H_1 and H_2 which we model as random oracles.

The transformation works by encrypting the message flows, m_1 and m_2, of the two-round UKE protocol under the long term keys of the two parties. Interestingly,

1. The challenger sets 1^λ and runs $\texttt{Setup}$ to obain params.
2. $\mathcal{A}$ is given 1^λ and params and ouputs a first message m_1.
3. If $m_1 \notin \mathcal{M}$ the experiment outputs 0. Otherwise, the challenger initialises a blank session ℓ and computes $m_2 \leftarrow \zeta(\mathsf{SST}[\ell], m_1; \$)$. It sets $\mathsf{k}_0 \leftarrow \ell.\mathsf{k}$ (since the protocol is two-round, this key is already set) and samples $\mathsf{k}_1 \leftarrow_{\$} \mathcal{K}$, from the UKE key-space.
4. $\mathcal{A}$ is given k_b and returns a guess $\tilde{b}$.
5. The experiment outputs 1 if and only if $\tilde{b} = b$, and 0 otherwise.

Figure 5.7: The $\mathsf{Key\text{-}Force\text{-}}b$ security experiment $\mathbf{Exp}_{\mathcal{A},\Pi}^{\mathsf{Key\text{-}Force\text{-}}b}$.

the main role played here by encryption is to authenticate the parties and ensure integrity of the exchanged messages, rather than its more traditional role of providing confidentiality. Indeed, one can think of the first two messages of the protocol as a challenge-response exchange where A attempts to authenticate B by sending an encryption of m_1 and expecting to receive the same m_1 in the next flow. Similarly, the second and third flow can be interpreted as a challenge-response where B sends m_2 to A and expects to receive a message that depends on m_2. In addition, the MAC sent as the last message also ties the identities of the parties involved with this particular execution of the protocol run. The final application key is derived from the same session-key derived by Π, but in a way that decouples it from the MAC key and also incorporates the identities of the participants.

The last message flow and key derivation methodology also thwart an analogue of the (in)famous attack against the Needham-Schroeder protocol which aims to break authentication. A malicious B could reencrypt the first message for a third party C who would reply with its own encrypted m_2 for A; B could then simply forward this message to A. Parties A and C would thus derive the same key for the underlying passively secure protocol. However, C will no longer accept the MAC sent by A as it will be on the wrong message ($A\|B$ as opposed to $A\|C$), thus thwarting the attack. In addition, since it depends on the participants' identities, the final derived session-key will also be different for A and C.

Proving formally that parties authenticate each other successfully in fact requires additional properties on the underlying protocol Π. As explained above, one should think of the first two messages as a challenge-response protocol to authenticate B.

Party A (ℓ_A)		Party B (ℓ_B)
$m_1 \leftarrow \zeta(\mathsf{SST}[\ell_A], \vdash; \$_1)$		
$\mathfrak{m}_1 \leftarrow \mathsf{Enc}_{\mathsf{pk}_B}(A \| m_1)$	$\xrightarrow{\mathfrak{m}_1}$	$A \| m_1 \leftarrow \mathsf{Dec}_{\mathsf{sk}_B}(\mathfrak{m}_1)$
		$m_2 \leftarrow \zeta(\mathsf{SST}[\ell_B], m_1; \$_2)$
$m_1' \| m_2 = \mathsf{Dec}_{\mathsf{sk}_A}(\mathfrak{m}_2)$	$\xleftarrow{\mathfrak{m}_2}$	$\mathfrak{m}_2 \leftarrow \mathsf{Enc}_{\mathsf{pk}_A}(m_1 \| m_2)$
if $m_1' \neq m_1$ **then**		
reject		
$\vdash \leftarrow \zeta(\mathsf{SST}[\ell_A], m_2; \$_3)$		
$\mathsf{k}_{A,1} \leftarrow H_1(\ell_A.\mathsf{k})$		
$\mathfrak{m}_3 \leftarrow \mathtt{MAC}_{\mathsf{k}_{A,1}}(A \| B)$	$\xrightarrow{\mathfrak{m}_3}$	$\mathsf{k}_{B,1} \leftarrow H_1(\ell_B.\mathsf{k})$
		if $\mathtt{Vfy}_{\mathsf{k}_{B,1}}(A \| B, \mathfrak{m}_3) = 0$ **then**
		reject
$\mathsf{k} \leftarrow H_2(\ell_A.\mathsf{k} \| A \| B)$		$\mathsf{k} \leftarrow H_2(\ell_B.\mathsf{k} \| A \| B)$

Figure 5.8: A new AKE protocol transformation.

Notice that for security of authentication, this requires that message m_1 of Π has sufficient entropy; otherwise, an adversary who guesses m_1 can reply with an appropriately message which encrypts m_1 and some m_2 and cause A to accept.

Similarly, thinking of the second and third messages as a challenge-response protocol that authenticates A: the last message should only be computable by some party which received m_2 and derived the MAC key from it. This intuition is valid only if m_2 actually helps determine the MAC key, which is not necessarily the case. Consider a two message protocol where, if the first message of A for B is some fixed message m^*, then B sets the local key to, say, 0^n. Such a protocol may still be secure against a passive adversary as an honest execution by A would never send m^*. Yet, the protocol obtained by applying our transformation is not actively secure since the adversary can send the encryption of m^* to B. More generally, a close look shows that the problem is that the adversary can send an appropriately crafted message m_1 which coerces the key into one which can be easily guessed (even if B behaves honestly). This justifies the introduction of the $\mathsf{M1\text{-}Guess}$ and $\mathsf{Key\text{-}Force}$ security notions in Section 5.2.1.

We now present in more detail our new construction of an AKE protocol. Let $E = (\mathtt{Setup}_E, \mathsf{KGen}_E, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme. Let $M = (\mathsf{KGen}_M, \mathtt{MAC},$

$\texttt{Vfy}$) be a message authentication code such that its key space is $\mathcal{K}_M = \{0,1\}^{l(\lambda)}$ for some polynomial function l, and its KGen_M algorithm simply selects a key from $\mathcal{K}_M$ uniformly at random. Let $\Pi = (\texttt{Setup}_\Pi, \zeta)$ be a two-round UKE protocol and finally, let $H_1 : \{0,1\}^* \to \{0,1\}^{l(\lambda)}$ and $H_2 : \{0,1\}^* \to \{0,1\}^{h(\lambda)}$, where h is a polynomial function, be two key derivation functions. Using these elements, we construct the mutually-authenticating AKE protocol $\Sigma = (\mathcal{I}, \mathcal{S} = \mathcal{I}, \texttt{Setup}_\Sigma, \mathsf{KGen}_\Sigma, \eta)$ where:

1. $\texttt{Setup}_\Sigma$ takes as input the security parameter 1^λ and outputs public parameters params_Σ which contain the parameters of the encryption scheme E output by $\texttt{Setup}_E(1^\lambda)$ and the parameters of the UKE protocol Π output by $\texttt{Setup}_\Pi(1^\lambda)$.

2. KGen_Σ takes as input params_Σ and an indentifier i. It then outputs a key pair for i by setting $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KGen}_E(\mathsf{params}_E)$, i.e. a normal public-key encryption scheme key pair.

3. The function η is as specified by the protocol run described in Figure 5.8.

5.2.3 Proof of security

We now prove the security of the transformation described above in the Bellare–Rogaway model for public-key schemes with forward secrecy; i.e. by first showing that it is mutually authenticating and then showing that it provides BR-secrecy.

The intuition behind the proof is that authentication of B to A is obtained by B prefixing the plaintext m_1 to his response m_2 in the second message flow $\mathfrak{m}_2$. In this way A can verify that the message m_1' that she receives is identical to the one she sent out, i.e. m_1, and therefore B must have decrypted it; since only B has B's decryption key. Authentication of A to B is obtained by A sending a valid MAC tag on the identities under a key derived from the underlying unauthenticated key-exchange scheme. Since only A can decrypt B's message m_2, only A could compute the underlying session-key and therefore the associated MAC key. Notice that the these forms of authentication also imply liveness of the parties.

Bellare–Rogaway security

The above intuition is formalized by the following theorem in the BR model.

Theorem 5.6. *If Π is M1-Guess-secure and Key-Force-secure, E is 2-IND-CCA-secure and M is sEUF-CMA-secure, then Σ is a secure mutual authentication protocol.*

We note that the proof below is in the classical random oracle model.

Proof. We prove correctness first and secure $\mathsf{BRmAuth}$-security second.

$\mathsf{Matching\ Conversation} \implies \mathsf{Acceptance.}$
Suppose two sessions $\ell = (A, B, s)$ and $\ell' = (A', B', s')$ have matching conversations. This first implies $A' = B$ and $B' = A$. Furthermore, the three protocol messages $\mathfrak{m}_1, \mathfrak{m}_2$ and $\mathfrak{m}_3$ are relayed faithfully.

Since ℓ' receives the same $\mathfrak{m}_1$ that ℓ sent out, by the correctness of the encryption scheme E, ℓ' decrypts m_1 correctly and prefixes m_2 with the correct message. Hence, since ℓ receives the same $\mathfrak{m}_2$ that ℓ' sent out, the correctness of E implies again that indeed $m_1' = m_1$ and therefore that ℓ does not reject this session when it receives $\mathfrak{m}_2$. As ℓ does not receive another response from ℓ' before it terminates, we see that ℓ accepts this session.

As ℓ has received the same $\mathfrak{m}_2$ that ℓ' has sent out, it receives the same m_2 and therefore the correctness of Π implies that $\ell.\mathsf{k} = \ell'.\mathsf{k}$. By equality of the inputs to the KDF H_1, we have that $\mathsf{k}_{A,1} = \mathsf{k}_{B,1}$. Also, since ℓ' receives the same $\mathfrak{m}_3$ that ℓ has sent out, the correctness of M and the equality of the keys imply that $\mathtt{Vfy}_{\mathsf{k}_{B,1}}(A\|B, \mathtt{MAC}_{\mathsf{k}_{A,1}}(A\|B)) = 1$ and hence ℓ' does not reject the session when it receives $\mathfrak{m}_3$. As ℓ' does not receive another response from ℓ before it terminates, we see that ℓ' accepts this session as well. This completes the proof of correctness.

$\mathsf{Acceptance} \implies \mathsf{Matching\ Conversation.}$
We now prove that an arbitrary $\mathsf{poly}(\lambda)$-time adversary $\mathcal{A}$ has a negligible chance of winning the $\mathsf{BRmAuth}$ security experiment. Assume that $\mathcal{A}$ reaches an environment such that $\neg\mathsf{BRmAuth}$ holds; i.e.

$$\exists \ell^* \in \mathsf{LSID} :: \ell^*.\mathsf{accept} \wedge \mathsf{BRaFresh}(\ell^*) \wedge \forall \ell' \in \mathsf{LSID}, \neg\mathsf{BRMatching}(\ell^*, \ell')$$

We first observe that the following holds:

$$\mathbf{Adv}_{\mathcal{A},\Sigma}^{\mathsf{BRmAuth}} \leq \Pr[\mathsf{BRmAuth} = 1 \wedge \ell^*.\rho = \mathsf{init}] + \Pr[\mathsf{BRmAuth} = 1 \wedge \ell^*.\rho = \mathsf{resp}]$$

and we therefore deal with the initiator and responder cases separately.

Bounding success against initiator. We proceed via a series of hybrid games.

Game 0. The initial $\mathsf{BRmAuth}$ security experiment with the difference that the adversary loses by default if it returns a responder session.

Game 1. The challenger first selects a random session $\tilde{\ell} \in \mathsf{LSID}$ and then runs Game 0 with the adversary. The adversary loses by default if it outputs a session different from $\tilde{\ell}$.

Game 2. The challenger runs Game 1 with the adversary but replaces the message m_1, honestly generated by $\tilde{\ell}$, by a random UKE message $\tilde{m}_1$ in the rest of the protocol (including in the computation of m_2). However, $\tilde{\ell}$ still only accepts if it receives the honest m_1 in the second message flow.

We denote by G_i the event that Game i outputs 1, for which we sometimes say that an arbitrary adversary $\mathcal{A}$ "wins" Game i. We see that we have $\Pr[G_0] = \Pr\left[\mathsf{BRmAuth} = 1 \wedge \ell^*.\rho = \mathsf{init}\right]$ and we argue the following to obtain an upper bound.

Game 0 to Game 1. First, we see that $\mathcal{A}$ wins Game 1 exactly when it wins Game 0 and the challenger has guessed the output session correctly out of $n_P^2 \cdot n_S$ possibilities, i.e. guessed $\tilde{\ell} = \ell^*$. We therefore have

$$\Pr[G_1] = \frac{1}{n_P^2 \cdot n_S} \cdot \Pr[G_0]. \tag{5.1}$$

Game 1 to Game 2. Next, we build a reduction $\mathcal{B}_1$ that uses an adversary that is able to distinguish between Game 1 and Game 2 to attack the 2-IND-CCA security of the encryption scheme E.

The reduction uses the $\mathcal{O}_{\mathsf{LR}}$ oracle given by the 2-IND-CCA challenger to encrypt either m_1 or $\tilde{m}_1$ in the protocol flows. Whenever the output of the $\mathcal{O}_{\mathsf{LR}}$ oracle is submited to a different session, $\mathcal{B}_1$ uses both m_1 and $\tilde{m}_1$ to compute two responses and encrypts either $m_1||m_2$ or $\tilde{m}_1||\tilde{m}_2$ using the $\mathcal{O}_{\mathsf{LR}}$ oracle. If the secret bit $b = 0$, then it is the honest m_1 that is used throughout the protocol thus simulating Game 1 perfectly. If the secret bit is 1, $\tilde{m}_1$ is used and Game 2 is simulated. If $\mathcal{A}$ still manages to make $\tilde{\ell}$ accept without a matching conversation, by submitting a message $\mathfrak{m}_2^*$ containing the honest m_1, then the reduction returns $b' = 1$ to the 2-IND-CCA challenger. Therefore we see that we have

$$\begin{aligned}\mathbf{Adv}^{\text{2-IND-CCA}}_{\mathcal{B}_1,E} &= \left|\Pr\left[\mathbf{Exp}^{\text{2-IND-CCA-0}}_{\mathcal{B}_1,E} = 1\right] - \Pr\left[\mathbf{Exp}^{\text{2-IND-CCA-1}}_{\mathcal{B}_1,E} = 1\right]\right| \\ &= \left|\Pr[G_1] - \Pr[G_2]\right|.\end{aligned}$$

Bounding Game 2. Finally we see that an adversary can win Game 2 only by guessing the honest m_1 that $\tilde{\ell}$ will accept since no information regarding m_1 is contained within the message flows. Given an adversary $\mathcal{A}$ capable of winning Game 2, we can easily build a reduction $\mathcal{B}_2$ that plays the M1-Guess experiment against the UKE protocol Π. This reduction simulates Game 2 to $\mathcal{A}$ and simply returns whichever m_1 is contained within $\mathfrak{m}_2$. As the reduction is given no information regarding the message it has to guess, the adversary is equally likely to guess the correct message. We thus have

$$\mathbf{Adv}^{\mathsf{M1\text{-}Guess}}_{\mathcal{B}_2,\Pi} = \Pr[G_2].$$

Advantage terms. As E is 2-$\mathsf{IND\text{-}CCA}$-secure and Π is $\mathsf{M1\text{-}Guess}$-secure, there exist negligible functions such that

$$\begin{aligned} |\Pr[G_1] - \Pr[G_2]| &= \mathbf{Adv}^{\mathsf{2\text{-}IND\text{-}CCA}}_{\mathcal{B}_1,E} \leq \mathsf{negl}(\lambda) \\ \text{and} \quad |\Pr[G_2]| &= \mathbf{Adv}^{\mathsf{M1\text{-}GUESS}}_{\mathcal{B}_2,\Pi} \leq \mathsf{negl}(\lambda)\,. \end{aligned}$$

This, combined with (5.1), yields

$$\Pr[G_0] = \Pr\left[\mathsf{BRmAuth} = 1 \wedge \ell^*.\rho = \mathsf{init}\right] \leq \mathsf{negl}(\lambda)\,. \tag{5.2}$$

Bounding success against responder. We also proceed via a series of games.

Game 0. The initial $\mathsf{BRmAuth}$ security experiment with the difference that the adversary loses by default if it returns an initiator session.

Game 1. The challenger first selects a random session $\tilde{\ell} = (\tilde{A}, \tilde{B}, \tilde{s}) \in \mathsf{LSID}$ and then runs Game 0 with the adversary. The adversary loses by default if it outputs a session different from $\tilde{\ell}$.

Game 2. In all of the message flows sent by initiator sessions of $\tilde{B}$ intended for $\tilde{A}$ and responder sessions of $\tilde{A}$ intended for $\tilde{B}$ (including $\tilde{\ell}$), the challenger replaces the honest UKE protocol messages m_1 and m_2 by independently randomly sampled messages $\tilde{m}_1$ and $\tilde{m}_2$. Internally, the oracles still compute and verify with the honest messages and keys. The victory conditions for $\mathcal{A}$ are the same as for Game 1.

Game 3. The challenger runs Game 2 with the adversary but replaces the UKE key $\tilde{\ell}.\mathsf{k}$ derived from the honestly computed messages by a uniformly sampled key $\tilde{\mathsf{k}} \leftarrow_\$ \mathcal{K}_\Pi$ (independent from the random message $\tilde{m}_2$). The session $\tilde{\ell}$ only accepts a MAC tag validly computed using $\tilde{\mathsf{k}}$ as the UKE key.

Game 4. The challenger runs Game 3 with the adversary but replaces the MAC key derived from the uniformly sampled $\tilde{\mathsf{k}}$ by a uniformly sampled MAC key $\tilde{\mathsf{k}}_M \leftarrow_\$ \mathcal{K}_M$. The session $\tilde{\ell}$ only accepts a MAC tag validly computed under $\tilde{\mathsf{k}}_M$.

Game 0 to Game 1. First, we see that $\mathcal{A}$ wins Game 1 exactly when it wins Game 0 and the challenger has guessed the output session correctly out of $n_P^2 \cdot n_S$ possibilities.

We therefore have

$$\Pr[G_1] = \frac{1}{n_P^2 \cdot n_S} \cdot \Pr[G_0]. \tag{5.3}$$

Game 1 to Game 2. In order to bound the difference between Game 1 and Game 2, we build the reduction $\mathcal{B}_3$ to attack the 2-IND-CCA security of the encryption scheme E. As for the previous cases, our aim is to build a reduction that simulates Game 1 or Game 2 perfectly when the $\mathcal{O}_{\mathsf{LR}}$ oracle's bit is 0 or 1. To this effect, $\mathcal{B}_3$ proceeds as follows.

Whenever an initiator session of $\tilde{B}$ with $\tilde{A}$ is activated, the reduction queries the $\mathcal{O}_{\mathsf{LR}}$ oracle to send either an honestly computed m_1 or a randomly sampled $\tilde{m}_1$. It also records both messages and the resulting ciphertext in a list L_1.

Whenever a responder session of $\tilde{A}$ with $\tilde{B}$ receives a ciphertext that appears on L_1, it is able to compute an honest response m_2 (thus deriving an honest session-key), sample a random one and send either an honest message $m_1 \| m_2$ or the random $\tilde{m}_1 \| \tilde{m}_2$ using the $\mathcal{O}_{\mathsf{LR}}$ oracle. If the ciphertext does not appear on L_1, it can be decrypted using Dec to obtain the plaintext. If that plaintext does not appear on L_1, then, as before, $\mathcal{B}_3$ derives a key and samples a uniform $\tilde{m}_2$ before replying with either the honest or random message (unless the content of the plaintext deviates from the protocol). Whenever $\mathcal{B}_3$ replies, it records both plaintexts, the session-key and the resulting ciphertext on a list L_2. If, however, the decrypted plaintext appears on L_1, then $\mathcal{B}_3$ learns the bit of the challenger oracle.

Finally, if an initiator session of $\tilde{B}$ with $\tilde{A}$ receives a ciphertext (as a second protocol message) that appears on L_2, then it is able to check if the accompanying plaintexts match the ones computed or sampled earlier. If they do, $\mathcal{B}_3$ uses the accompanying session key to proceed. If they don't, then this ciphertext was not intended for this session and the session rejects. If the ciphertext received does not appear on L_2, it can be decrypted using Dec to obtain the plaintext. If that plaintext appears on L_2, $\mathcal{B}_3$ learns the bit of the challenger oracle. If it does not, $\mathcal{B}_3$ checks if the first message matches either of the plaintexts computed or sampled earlier. If it does, $\mathcal{B}_3$ learns the bit of the challenger oracle. If it doesn't, then the ciphertext is an invalid protocol message and the session rejects.

If the adversary wins the experiment under the same conditions as for Game 1 and 2, $\mathcal{B}_3$ returns 1 to the 2-IND-CCA-b challenger. Let $\mathcal{E}_1$ denote the event that a session receives a ciphertext that leads to $\mathcal{B}_3$ learning the challenger bit. We then have

$$\begin{aligned}\mathbf{Adv}^{\textsf{2-IND-CCA}}_{\mathcal{B}_3,E} = \Big| 0 \cdot \Pr[\mathcal{E}_1] + \Pr\left[\mathbf{Exp}^{\textsf{2-IND-CCA-0}}_{\mathcal{B}_3,E} = 1 \mid \neg\mathcal{E}_1\right] \cdot \Pr[\neg\mathcal{E}_1] - \\ 1 \cdot \Pr[\mathcal{E}_1] - \Pr\left[\mathbf{Exp}^{\textsf{2-IND-CCA-1}}_{\mathcal{B}_3,E} = 1 \mid \neg\mathcal{E}_1\right] \cdot \Pr[\neg\mathcal{E}_1]\Big|\end{aligned}$$

which implies

$$\mathbf{Adv}^{\textsf{2-IND-CCA}}_{\mathcal{B}_3,E} + \Pr[\mathcal{E}_1] \geq \left|\Pr[G_1] - \Pr[G_2]\right| \cdot (1 - \Pr[\mathcal{E}_1]) . \tag{5.4}$$

It is easy to see from $\mathcal{B}_3$ how one can build a reduction $\mathcal{B}_3'$ which wins the 2-IND-CCA experiment against E whenever $\mathcal{E}_1$ happens. This allows us to write

$$\mathbf{Adv}^{\textsf{2-IND-CCA}}_{\mathcal{B}_3',E} = \Pr[\mathcal{E}_1]. \tag{5.5}$$

Game 2 to Game 3. The reduction $\mathcal{B}_4$ acts as an attacker in a Key-Force experiment against the UKE protocol Π. To simulate Game 2, it chooses a session $\tilde{\ell} = (\tilde{A}, \tilde{B}, \tilde{s})$ at random and runs the execution environment, sampling protocol messages at random where required, maintaining consistency. When $\tilde{\ell}$ receives a ciphertext, $\mathcal{B}_4$ decrypts it and checks whether the plaintext was sampled by an initiator session $\ell' = (\tilde{B}, \tilde{A}, \tilde{t})$. If it was, then $\mathcal{B}_4$ recovers the honest m_1 that was computed and returns it to the Key-Force challenger. If it was not, then $\mathcal{B}_4$ returns the plaintext it decrypted. When the Key-Force challenger returns a key $\tilde{\mathsf{k}}$, $\mathcal{B}_4$ sets the UKE key as $\mathsf{k} = \tilde{\mathsf{k}}$ and proceeds with its simulation of Game 2. If $\mathcal{A}$ wins the simulation of Game 2 according to the same conditions, $\mathcal{B}_4$ returns $\tilde{b} = 1$ to the challenger.

It is easy to see that if the Key-Force challenger returns the real key, then $\mathcal{B}_4$ simulates Game 2 perfectly, and that if it returns a random UKE key, then it is Game 3 that is simulated perfectly. We therefore have

$$\begin{aligned}\mathbf{Adv}^{\textsf{Key-Force}}_{\mathcal{B}_4,\Pi} &= \left|\Pr\left[\mathbf{Exp}^{\textsf{Key-Force-0}}_{\mathcal{B}_4,\Pi} = 1\right] - \Pr\left[\mathbf{Exp}^{\textsf{Key-Force-1}}_{\mathcal{B}_4,\Pi} = 1\right]\right| \\ &= \left|\Pr[G_2] - \Pr[G_3]\right| .\end{aligned}$$

Game 3 to Game 4. We see that, as the UKE key $\tilde{\mathsf{k}}$ is selected uniformly at random, the derived MAC key $\mathsf{k}_M = H_1(\tilde{\mathsf{k}})$ follows the same distribution. Therefore this is a simple re-wording and we have

$$\Pr[G_4] = \Pr[G_3]. \tag{5.6}$$

Bounding Game 4. Finally, we see that Game 4 is very similar to the $\mathsf{sEUF\text{-}CMA}$ game against the MAC M. Indeed, the adversary has no information regarding the MAC key $\tilde{\mathsf{k}}_M$ used by the sessions except perhaps a valid tag $\mathfrak{m}_3$ if it relayed the two first messages correctly. We therefore make use of a final reduction $\mathcal{B}_5$ to bound $\Pr[G_4]$.

This reduction works as follows. If the adversary presents the chosen session $\tilde{\ell} = (\tilde{A}, \tilde{B}, \tilde{s})$ with a first UKE message $\tilde{m}_1$, $\mathcal{B}_5$ replies by inserting a random message $\tilde{m}_2$ into $\tilde{\mathfrak{m}}_2$ without deriving a session key. If an instance of $\tilde{B}$ receives the correct $\tilde{m}_1 \| \tilde{m}_2$ that matches $\tilde{\ell}$, then it queries the MAC oracle for a tag $\tilde{\mathfrak{m}}_3$ on $\tilde{B} \| \tilde{A}$. When $\tilde{\ell}$ receives a tag $\mathfrak{m}_3 \neq \tilde{\mathfrak{m}}_3$, it returns $\mathfrak{m}_3$ to the $\mathsf{sEUF\text{-}CMA}$ challenger.

We can see from the description above that if $\mathcal{A}$ manages to make a responder accept in Game 4 without a matching conversation, then $\mathcal{B}_5$ is able to win the $\mathsf{sEUF\text{-}CMA}$ experiment. We therefore have

$$\mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathcal{B}_5,M} = \Pr[G_4].$$

Advantage terms. As E is $\mathsf{2\text{-}IND\text{-}CCA}$-secure, there exist negligible functions such that

$$\mathbf{Adv}^{\mathsf{2\text{-}IND\text{-}CCA}}_{\mathcal{B}_3,E} \leq \mathsf{negl}(\lambda) \quad \text{and} \quad \mathbf{Adv}^{\mathsf{2\text{-}IND\text{-}CCA}}_{\mathcal{B}_3',E} \leq \mathsf{negl}(\lambda)\,.$$

Combining this with equations (5.4) and (5.5) yields

$$|\Pr[G_1] - \Pr[G_2]| \leq \mathsf{negl}(\lambda)\,.$$

As Π is $\mathsf{Key\text{-}Force}$-secure, there exists a negligible function such that

$$|\Pr[G_2] - \Pr[G_3]| = \mathbf{Adv}^{\mathsf{Key\text{-}Force}}_{\mathcal{B}_4,\Pi} \leq \mathsf{negl}(\lambda)\,.$$

Finally, as M is $\mathsf{sEUF\text{-}CMA}$-secure, there exists a negligible function such that

$$|\Pr[G_4]| = \mathbf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathcal{B}_5,M} \leq \mathsf{negl}(\lambda)\,.$$

Combining the above, together with (5.3) and (5.6), yields

$$\Pr[G_0] = \Pr\left[\mathsf{BRmAuth} = 1 \land \ell^*.\rho = \mathsf{resp}\right] \leq \mathsf{negl}(\lambda)\,. \tag{5.7}$$

Final advantage statement. Combining (5.2) and (5.7) yields

$$\mathbf{Adv}^{\mathsf{BRmAuth}}_{\mathcal{A},\Sigma} \leq \mathsf{negl}(\lambda)$$

which completes the proof that Σ is a secure mutual authentication protocol. $\square$

Finally, we show that our construction yields a protocol that guarantees key secrecy.

Theorem 5.7. *If* Π *is* *EAV-KE-secure, M1-Guess-secure and Key-Force-secure,* E *is* 2-*IND-CCA-secure and* M *is sEUF-CMA-secure, then* Σ *is is BR-forward-secure.*

Proof. We show that the three conditions required for BR-forward-security hold.

1. Correctness. The proof of Theorem 5.6 gives us that if two sessions have matching conversations, then both of them accept. We now show that they derive identical sessions-keys which are uniformaly distributed.

As the two sessions have had matching conversations, the two messages m_1, m_2, of the UKE protocol have been correctly received by both entities. The correctness of Π therefore implies that the keys derived are identical. Furthermore, the matching conversations also imply that the two sessions agree on each other's identities. This all together implies that the inputs to the KDF H_2 are identical for each sessions, therefore they derive the same session-key.

Finally, as Π is passively secure, we see that the session keys k_A and k_B are uniformly distributed over $\mathcal{K}_\Pi$. Therefore, we have that the output of H_2, modelled as a random oracle, is also uniformly distributed over $\mathcal{K}_\Sigma$.

2. Σ *is a secure mutual authentication protocol.* This condition is proved by Theorem 5.6 which holds as we make the same assumptions on the security of Π, E and M.

3. Σ *provides forward secrecy w.r.t the key distribution of* Π. We build a reduction $\mathcal{B}_6$ that uses an arbitrary $\mathsf{poly}(\lambda)$-time adversary $\mathcal{A}$ against the BRSec security of Σ in order to win the EAV-KE experiment against Π.

$\mathcal{B}_6$ receives a transcript from the EAV-KE challenger and guesses which session $\mathcal{A}$ will return. It assumes that this will be a initiator session. In order for it to be BR-fresh, and therefore a valid `Test` query target, the adversary cannot have corrupted either participating party and must have made it accept. As shown by Theorem 5.6, this implies that this session must have had a matching conversation with another. Therefore, when its chosen session is first activated, $\mathcal{B}_6$ inserts the first message of the UKE transcript in place of the honestly computed one.

When a session recieves this first ciphertext (and there is one as there must be a matching conversation), $\mathcal{B}_6$ replaces the second message with the one from the UKE transcript. It also replaces the session-key with that given by the EAV-KE challenger.

When the second AKE protocol message is relayed to the initiator session, $\mathcal{B}_6$ once again replaces the session key with the challenger's.

When the adversary requests to be tested on the chosen session, $\mathcal{B}_6$ does not flip a coin, instead, it simply returns the challenger's session key. When $\mathcal{A}$ terminates and returns a guess bit, $\mathcal{B}_6$ returns that same bit to the $\mathsf{EAV\text{-}KE}$ challenger.

If $\mathcal{A}$ submits any query that invalidates the freshness of $\mathcal{B}_6$'s chosen session, or requests to be tested on a different session, the reduction aborts the simulation and guesses the challenger's bit at random.

As with previous reductions, we see that the probability of $\mathcal{B}_6$ picking the correct test session is $1/n_P^2 \cdot n_S$, in which case, $\mathcal{B}_6$ has the same probability of success in the $\mathsf{EAV\text{-}KE}$ game as $\mathcal{A}$ has in the BRSec game with the same bit. If $\mathcal{B}_6$ guesses the session wrongly, then it must output a random guess which will be equal to 1 with probability 1/2. Therefore we have, for $b \in \{0,1\}$:

$$\Pr\left[\mathbf{Exp}_{\mathcal{B}_6,\Pi}^{\mathsf{EAV\text{-}KE},b} = 1\right] = \frac{1}{n_P^2 \cdot n_S} \cdot \Pr\left[\mathbf{Exp}_{\mathcal{A},\Sigma}^{G^b_{\mathsf{BRSec}}} = 1\right] + \left(1 - \frac{1}{n_P^2 \cdot n_S}\right) \cdot \frac{1}{2}.$$

This then implies that

$$\mathbf{Adv}_{\mathcal{A},\Sigma}^{\mathsf{BRSec}} = n_P^2 \cdot n_S \cdot \mathbf{Adv}_{\mathcal{B}_6,\Pi}^{\mathsf{EAV\text{-}KE}}.$$

We conclude the proof by stating that as Π is $\mathsf{EAV\text{-}KE}$-secure, there exists a negligible function $\mathsf{negl}(\lambda)$ which, combined with the above, gives the desired relation,

$$\mathbf{Adv}_{\mathcal{A},\Sigma}^{\mathsf{BRSec}} \leq \mathsf{negl}(\lambda)\,.$$

$\square$

Part II

Supersingular isogeny-based cryptography

Chapter 6

Encryption from supersingular isogenies

In this chapter we present isogenies of supersingular elliptic curves and describe how they have been used to construct cryptographic primitives and schemes in recent years. We then describe a new construction for public-key encryption which achieves IND-CCA security based on a novel trapdoor one-way function.

Except where specified, the contributions in this chapter are selected from the work of P. Kutas[1], C. Petit[2], J. Silva[3] and this author; it was uploaded as Report 2019/1291 in the *Cryptology ePrint Archive* of the IACR [57]. Except where specified, the contributions and proofs are the work of this author in the most part.

6.1 Background material

We first summarize the relevant mathematical notions of supersingular isogenies and the computational problems associated with them. We then describe the supersingular isogeny Diffie–Hellman (SIDH) protocol of [66] as an introduction to isogeny-based protocols.

6.1.1 Isogenies of supersingular elliptic curves

Let q be a power of a prime p such that the finite field $\mathbb{F}_q$ has characteristic different from 2 or 3. An *elliptic curve* E over $\mathbb{F}_q$ (denoted $E/\mathbb{F}_q$) is a non-singular projective

[1]University of Birmingham, U.K.
[2]University of Birmingham, U.K.
[3]Universitat Pompeu Fabra, Barcelona, Spain.

algebraic curve over $\mathbb{F}_q$ with genus 1 and endowed with a distinguished point at infinity: ∞. Every such curve $E/\mathbb{F}_q$ can be described in its short Weierstrass form:

$$E : y^2 = x^3 + Ax + B, \qquad A, B \in \mathbb{F}_q. \tag{6.1}$$

We therefore identify such curves with its two coefficients: $E \sim (A, B)$.

The set $E(\overline{\mathbb{F}_q})$ denotes the set of pairs $(x, y) \in \overline{\mathbb{F}_q}^2$ which satisfy (6.1). Under the usual addition law on elliptic curves, this set forms an Abelian group for which the distinguished point ∞ is the identity. For any integer $N \in \mathbb{N}$, the notation $[N]P$ denotes the N-fold addition of the point $P \in E(\overline{\mathbb{F}_q})$ to itself.

Torsion groups and the Weil pairing. For an integer $N \in \mathbb{N}$ coprime to p, we denote by $E[N]$ the set $\{P \in E(\overline{\mathbb{F}_q}) : [N]P = \infty\}$ of $\mathbb{F}_q$-rational points of E of order dividing N, called the *N-torsion* of E.

We let $\zeta_N = \{u \in \mathbb{F}_{p^2}^* : u^N = 1\}$ denote the set of N-th roots of unity. For any curve $E/\mathbb{F}_{p^2}$ the Weil pairing is a map $e_N : E[N] \times E[N] \to \zeta_N$ that is both bilinear and non-degenerate:

$$\begin{gathered} e_N(P_1 + P_2, Q) = e_N(P_1, Q) \cdot e_N(P_2, Q) \\ e_N(P, Q_1 + Q_2) = e_N(P, Q_1) \cdot e_N(P, Q_2) \\ \forall P \in E[N] \setminus \{\infty\}, \exists Q \in E[N] : e_N(P, Q) \neq 1. \end{gathered}$$

Elliptic curve isomorphism invariants and quadratic twists. Given a curve $E/\mathbb{F}_q \sim (A, B)$, its *$j$-invariant* is the quantity

$$j(E) = 1728 \cdot \frac{4A^3}{4A^3 + 27B^2}.$$

This is invariant under any isomorphism of E defined over $\overline{\mathbb{F}_q}$.

Given $d \neq 0$ not a square in $\mathbb{F}_q$ the *quadratic twist* of a curve $E/\mathbb{F}_q$ is the curve E^d defined by

$$E^d : dy^2 = x^3 + Ax + B.$$

Two such curves E and E^d will not be isomorphic over $\mathbb{F}_q$ but rather over $\mathbb{F}_q(\sqrt{d})$; they will therefore still have the same j-invariant.

Isogenies of elliptic curves. Let $E_1/\mathbb{F}_q$ and $E_2/\mathbb{F}_q$ be two elliptic cxurves, an *isogeny* $\varphi : E_1 \to E_2$ over $\mathbb{F}_q$ is a surjective morphism which sends the point at infinity of E_1 to that of E_2. In this work we only consider isogenies φ that are separable which implies that $\#\ker\varphi = \deg\varphi$.

An isogeny can also be defined as a group homomorphism from $E_1(\overline{\mathbb{F}_q})$ to $E_2(\overline{\mathbb{F}_q})$ with a finite kernel. Furthermore, if $E/\mathbb{F}_q$ is an elliptic curve and $G \subseteq E(\overline{\mathbb{F}_q})$ is a finite subgroup, then there exists a unique isogeny $\varphi : E \to E'$ (up to automorphisms of the target curve) whose kernel is exactly G. We therefore identify the target curve E' with the notation E/G.

Two curves are said to be $\mathbb{F}_q$-isogenous if there exists an $\mathbb{F}_q$-isogeny between them; by Tate's theorem this holds if and only if the two curves have the same number of $\mathbb{F}_q$-rational points, i.e. $\#E_1(\mathbb{F}_q) = \#E_2(\mathbb{F}_q)$. A set of elliptic curves over $\mathbb{F}_q$ that are all isogenous to one another is called an *isogeny class*. If there exists an isogeny $\varphi : E_1 \to E_2$, then there exists a unique isogeny $\hat{\varphi} : E_2 \to E_1$, called the *dual isogeny* of φ, such that $\varphi \circ \hat{\varphi} = [n]$ where $n = \deg\varphi$ and $[n]$ denotes the multiplication-by-n map on $E_2(\mathbb{F}_q)$. This shows that being isogenous is a symmetric relation.

We note that, for any isogeny φ, the Weil pairing defined above satisfies the relation

$$e_N(\varphi(P), \varphi(Q)) = e_N(P, Q)^{\deg\varphi}.$$

Endomorphisms and supersingular curves. Let E be an elliptic curve over $\mathbb{F}_q$; an isogeny from E to itself is called an *endomorphism* of E. Together with the zero isogeny and addition and composition operations, the set of endomorphisms of E forms a ring, denoted $\mathrm{End}(E)$. A theorem of Deuring states that $\mathrm{End}(E)$ is isomorphic to either an order in an imaginary quadratic field or a maximal order in a quaternion algebra. In the first case, the curve E is said to be *ordinary*; in the second E is said to be *supersingular*.

Given p and q, every supersingular curve over $\mathbb{F}_q$ is isomorphic to another curve defined over $\mathbb{F}_{p^2}$. Furthermore, a curve $E/\mathbb{F}_{p^2}$ is supersingular if and only if $\#E(\mathbb{F}_{p^2}) \equiv 1 \mod p$. Tate's theorem mentioned above therefore implies that supersingularity is preserved under isogenies and that curves that belong to the same isogeny class are either all ordinary or all supersingular.

From now on we restrict the previous discussion to $\mathbb{F}_q = \mathbb{F}_{p^2}$. To set notation, we let $\mathcal{J}_p$ denote the set of j-invariants of all supersingular elliptic curves defined over $\mathbb{F}_{p^2}$; we then identifiy the set of isomorphism classes of such curves with the set $\mathcal{J}_p$. We then have $\#\mathcal{J}_p = \lfloor \frac{p}{12} \rfloor + k$, with $k \in \{0, 1, 2\}$ depending on p.

Canonical choices. Following [71, Section 2.4] we fix the following method to select canonical choices of curves. Given $j \in \mathcal{J}_p$, we define the curve E_j as $E_j \sim (0, 1)$ when $j = 0$, $E_j \sim (1, 0)$ when $j = 1728$ and $E_j \sim (\frac{3j}{1728-j}, \frac{2j}{1728-j})$ otherwise.

For a given curve E and torsion order N dividing $p-1$, there exists a deterministic and efficient algorithm $\mathsf{Basis}(E, N)$ which produces a basis $\{P, Q\}$ of the N-torsion of E [7, Section 3.2]. This algorithm first uses a deterministic PRG to sample a point $P' \in E(\mathbb{F}_{p^2})$ and multiplies it by $\frac{p-1}{N}$ to obtain a point P whose order divides N. (This is efficient if N is a powersmooth number.) In fact, with high probability, the order of P is exactly N; if not, then the algorithm samples a new point P' and checks again. To generate the other basis point, the same method is applied to find a second point Q of order N. Then the Weil pairing is used to check if P and Q are independent of each other. If $e_N(P, Q) \neq 1$ then $\{P, Q\}$ is returned as the output of $\texttt{Basis}(E, N)$; if $e_N(P, Q) = 1$, then a new Q is sampled. As long as the seed for the PRG is fixed in advance, then this procedure will always produce the same points for a given curve and torsion order.

Isogeny graphs. Let ℓ be a prime number different from p. We define the graph $G_\ell = G_\ell(\mathbb{F}_{p^2}) = (\mathcal{V}, \mathcal{E})$ where the vertex set $\mathcal{V} = \mathcal{J}_p$ and the edge set $\mathcal{E}$ is defined as follows. Given any isogeny $\varphi : E_{j_1} \to E_{j_2}$ of degree ℓ, for two vertices $j_1, j_2 \in V$, we identify it with its dual isogeny and any other isogeny that is equivalent with either, up to post-composition with automorphisms of the target curve; we then let this isogeny class define an undirected edge $(j_1, j_2) \in \mathcal{E}$.

The edges of G_ℓ can also be defined using the modular polynomial $\Phi_\ell(x, y) \in \mathbb{Z}[x, y]$ [117]. This polynomial is symmetric, i.e. $\Phi_\ell(x, y) = \Phi_\ell(y, x)$, and is of degree $\ell + 1$ in both x and y. Given two j-invariants $j_1, j_2 \in \mathcal{V}$, it holds that $\Phi_\ell(j_1, j_2) = 0$ if and only if there exists an ℓ-isogeny between E_{j_1} and E_{j_2} and therefore this exactly characterises the edge set $\mathcal{E}$. In particular, this implies that for any such prime ℓ, there exist exactly $\ell + 1$ distinct isogenies originating from any given supersingular curve, up to isomorphism. Given the degree and symmetry of Φ_ℓ, this also implies that the graph G_ℓ is $(\ell + 1)$-regular.

6.1.2 Computational problems

Given an elliptic curve, the foundational problem of isogeny-based cryptography is to compute its endomorphism ring.

Problem 6.1 (Supersingular endomorphism ring computation)**.** Let p be a prime number and let E be a supersingular curve over $\mathbb{F}_{p^2}$, chosen uniformly at random. Determine the endomorphism ring $\mathrm{End}(E)$.

A second problem of importance is that of computing an isogeny between two given elliptic curves. The existence of such an isogeny can easily be decided by testing the equality of the number of points on both given curves. If this isogeny has low degree, then it can easily be guessed by enumeration. The computationally interesting version is therefore to compute such an isogeny with the guarantee that it is of relatively high degree.

Problem 6.2 (Generic supersingular isogeny computation)**.** Let D be a smooth number and let E_1 and E_2 be isogenous elliptic curves over $\mathbb{F}_{p^2}$ for an isogeny of degree D. Compute an isogeny of degree D between E_1 and E_2.

We note that, heuristically, restricting Problem 6.2 to isognies of smooth degree does not change its hardness, which is in fact equivalent to that of Problem 6.1 [111, 65]. Furthermore, fixing E_1 arbitrarily and letting only E_2 vary at random also does not change the complexity.

As described in more detail in the next section, relying only on isogenies does not provide sufficient commutative structure to realize some protocols like key-exchange. To work around this, De Feo, Jao and Plût proposed to also include images of torsion points under secret isogenies [66]. As more information is now revealed about the isogeny, this specializes the computational problem.

Problem 6.3 (Computational supersingular isogeny (CSSI) problem [66])**.** Let p be a prime, let D and N be coprime integers and let $E_0/\mathbb{F}_{p^2}$ be a fixed supersingular elliptic curve. Let $\varphi : E_0 \to E_1$ be an isogeny whose kernel is $\langle K \rangle$, where K is sampled at random from points of order D. Let P, Q be a basis of the torsion group $E_0[N]$. Given $E_0, P, Q, E_1, \varphi(P)$ and $\varphi(Q)$, compute φ.

As can be seen in its statement, Problem 6.3 assumes that E_0 is a fixed curve. This ensures that the assumption is close to the reality of protocol deployment where a fixed starting curve can be an advantage for efficiency.

Petit showed in 2017 that including such images wasn't always secure and that polynomial-time computations of isognenies were possible if the imbalance between D and N was sufficient [110]. Recently, further work has improved on this technique and lowered the imbalance requirements [94]. Since Petit's algorithm requires knowledge of some of the endomorphism ring of the starting curve E_0, fixing this curve allows for better performance. While removing the images of torsion points is not possible without sacrificing the protocol benefits, we can at least first remove the dependence of the above problem on a particular fixed curve.

Problem 6.4 (Random-start computational supersingular isogeny (RCSSI) problem)**.** Let p be a prime and D and N be coprime integers, let $E_1/\mathbb{F}_{p^2}$ be a uniformly random supersingular elliptic curve and let $\varphi : E_1 \to E_2$ be a random isogeny of degree D sampled from a distribution X with min-entropy $H_\infty(X) = O(\lambda)$. Let P, Q be a basis of the torsion group $E_1[N]$. Given $E_1, P, Q, E_2, \varphi(P)$ and $\varphi(Q)$, compute φ.

We note this RCSSI problem differs from the CSSI problem in two aspects. The first is that the starting curve E_1 is uniformly random instead of being a special fixed curve. Selecting E_1 at random means that computing $\mathrm{End}(E_1)$ is exactly an instance of Problem 6.1; as this problem is assumed to be hard on average, this prevents the attacks in [110, 94] from applying to Problem 6.4, even for unbalanced D and N.

The second difference is the specification of the entropy of the distribution from which the challenge isogeny is sampled. Note that in the statement of Problem 6.4 we have allowed arbitrary distributions with sufficient min-entropy for convenience, but in fact we will only require the problem to be hard for specific distributions. In Section 6.1.4, we discuss how this modification to the original CSSI problem is in fact not specific to the protocol presented in this chapter, as a similar modification seems to be needed to formally prove the security of the NIST submission SIKE [6] derived from SIDH.

Finally we also include here a decisional problem introduced by De Feo et al. [66].

Problem 6.5 (Decisional supersingular isogeny (DSSI) problem)**.** Let p be a prime, let D be an integer and let $E_0/\mathbb{F}_{p^2}$ be a fixed supersingular curve. Given another curve $E/\mathbb{F}_{p^2}$, decide whether E is D-isogenous to E_0.

Since both E and E_0 are both supersingular, they will both have the same number of points and will therefore be isogenous. The problem here is more specific and asks whether E and E_0 are connected by an isogeny of degree exactly D.

6.1.3 The SIDH key-exchange and semi-commutativity

In the supersingular case, recall that the endomorphism ring $\mathrm{End}(E)$ is isomorphic to a maximal order in a quaternion algebra and is therefore non-commutative. Despite this, a notion of *semi-commutativity* is possible and we describe here the SIDH two-party key-exchange protocol to illustrate this.

We generalise slightly the presentation of [66] and discuss the case where p is a prime of the form $\ell_1^{e_1}\ell_2^{e_2}\cdots\ell_n^{e_n}\cdot f \pm 1$ for n small primes $\ell_1,\dots,\ell_n$ and a small cofactor f. By construction of p, there is a curve $E/\mathbb{F}_{p^2}$ in each isomorphism class of $\mathcal{J}_p$ such that the torsion group $E[\ell_i^{e_i}]$ contains $\ell_i^{e_i-1}(\ell_i+1)$ cyclic subgroups of order $\ell_i^{e_i}$ (which each define a distinct isogeny up to isomorphism).

To generate a secret key, each party generates a random point K_i of order $\ell_i^{e_i}$ on a fixed public curve E and computes a public ephemeral key by computing the unique isogeny with kernel $\langle K_i\rangle$ and publishing the domain curve $E/\langle K_i\rangle$. The computation of this isogeny is efficient due to its very smooth degree. The issue here is that the structure of $\mathrm{End}(E)$ no longer allows for the composition of arbitrary isogenies to commute and an analogue of the $(g^a)^b = (g^b)^a$ equality is not immediate. However, with isogenies of co-prime degrees some commutative structure can still be achieved.

To do so, in addition to the curve E, the parties agree on bases $\{P_i, Q_i\}$ for each of the torsion groups $E[\ell_i^{e_i}]$. The semi-commutative structure then comes from the fact that applying an isogeny of degree $\ell_i^{e_i}$ preserves the torsion groups $E[\ell_j^{e_j}]$ for $j \neq i$ since every point in $E[\ell_j^{e_j}]$ has order co-prime to $\ell_i^{e_i}$. Therefore, alongside publishing the domain curve $E/\langle K_i\rangle$ for their secret isogeny ϕ_i, parties also publish $\{\{\phi_i(P_j),\phi_i(Q_j)\}_{j\neq i}\}$, the images under ϕ_i of the bases for the other torsion groups. By expressing their secret kernel $K_j = [\alpha_j]P_j + [\beta_j]Q_j$ in the bases of the initial torsion groups and applying these coefficients to the images $\{\phi_i(P_j),\phi_i(Q_j)\}$, the other party can then compute an isogeny $\varphi_j : E/\langle K_i\rangle \to E/\langle K_i, K_j\rangle$ which can be seen as "parallel" to the isogeny $\phi_j : E \to E/\langle K_j\rangle$.

Whilst the two resulting curves $E/\langle K_i, K_j\rangle$ and $E/\langle K_j, K_i\rangle$ may not be identical, they will be isomorphic, as the kernel $\langle K_i, K_j\rangle$ defines a unique isogeny up to isomorphism, and the parties can then take the j-invariants of their respective curves as an identical shared value.

This protocol is provably secure in the Auth model of Canetti and Krawczyk [35] under the assumption that the SSDDH problem is hard. The SSDDH problem is a variant of the DSSI problem (Problem 6.5) which takes into account the additional points revealed by the protocol described above.

6.1.4 Isogeny sampling in SIKE and the CSSI problem

In their paper introducing SIDH [66], De Feo et al. specify that the kernel generator of the secret isogeny is selected as $[m]P + [n]Q$, with $m, n \leftarrow_{\$} \mathbb{Z}_{\ell^e}$ not both divisible by ℓ (taking $D = \ell^e$ in this case). This ensures that every one of the $(\ell+1)\ell^{e-1}$ degree-D

isogenies can be selected as the challenge. The CSSI problem defined in that paper therefore naturally assumes that the isogenies are sampled uniformly at random.

For increased efficiency, however, it is proposed in [47, Section 4] to sample the generator points as $P + [\ell \cdot m]Q$ for $m \in [\ell^{e-1}]$. This has the consequence of only sampling from 1/3, resp. 1/4, of the possible isogenies, for $\ell = 2$ and $\ell = 3$ respectively. A similar method is included in the SIKE specification [6, Section 1.3.5] which, furthermore, samples m only in the set $[2^{\lfloor \log 3^{e_3} \rfloor}]$, therefore not reaching the full range of possible values. It is not expected that such imperfect sampling makes the CSSI problem easier, especially since such sampling methods still yield distributions of isogenies with min-entropy of the order of $O(1^\lambda)$. Nonetheless, we have included this difference into Problem 6.4 to make this sampling discrepancy more explicit.

6.2 Injective trapdoor one-way function family

We first present a generalisation of the CGL hash function [37] and then introduce a new family of trapdoor OWFs. We show that, for certain parameters, we can efficiently sample a statistically uniform function from the family and that any such function is injective and one-way. Finally, we show that sampling a function at random yields a trapdoor, i.e. a secret isogeny, which we can use to efficiently invert the function.

6.2.1 Extending the Charles-Goren-Lauter hash function

We first present the CGL hash function family $\mathcal{H}^{p,\ell,e} = \{h_j^{\ell,e} : [\ell]^e \to \mathcal{J}_p\}_{j \in \mathcal{J}_p}$ as introduced in [37]. We omit ℓ, e from notation when the context is clear. To select a hash function from the family, one selects a j-invariant $j \in \mathcal{J}_p$ which fixes the canonical curve $E = E_j/\mathbb{F}_{p^2}$. The graph G_ℓ is $(\ell + 1)$-regular so there are $\ell + 1$ isogenies of degree ℓ connecting E to other vertices. A canonical one of these is ignored and the other ℓ are numbered arbitrarily. Then, given a message $m = b_1 b_2 \dots b_e$, with $b_i \in [\ell]$, hashing starts by choosing a degree-ℓ isogeny from E according to symbol b_1 to arrive at a first curve E_1. Not allowing backtracking, there are then only ℓ isogenies out of E_1 and one is chosen according to b_2 to arrive at a second curve E_2. Continuing in the same way, m determines a unique walk of length e along the edges of G_ℓ which can be identified with a degree-ℓ^e isogeny φ_m.

We modify this hash function family in three ways. First, we consider a generalisation where we do not ignore one of the $\ell + 1$ isogenies from the starting curve E. That is, we take inputs $m = b_1 b_2 \dots b_e$ where $b_1 \in [\ell + 1]$ and $b_i \in [\ell]$ for $i > 1$; this introduces a one-to-one correspondence between inputs and isogenies φ_m of degree ℓ^e originating from E.

Secondly, we consider a generalisation where the walk takes place over multiple graphs G_{ℓ_i}. Given an integer $D_M = \prod_{i=1}^n \ell_i^{e_i}$ where the ℓ_i are its prime factors, we introduce the notation $\mu(D_M) \coloneqq \prod_{i=1}^n (\ell_i + 1) \cdot \ell_i^{e_i - 1}$. We take the message m to be an element of $[\mu(D_M)]$ represented as a tuple $(m_1, \dots, m_n)$ and each m_i is then hashed along the graph G_{ℓ_i}. To ensure continuity, the j-invariants are chained along the hash functions, that is, we write $j_i = h_{j_{i-1}}(m_i)$, where j_{i-1} is the hash of m_{i-1}. Each such hash can be identified with the computation of an isogeny $\varphi_i : E_{j_{i-1}} \to E_{j_i}$ of degree $\ell_i^{e_i}$. Thus, only j_0 parametrises the overall hash function, which we denote by j. As before, this generalization returns the final j-invariant $j_n = h_{j_{n-1}}(m_n)$ as the hash of m.

Thirdly, we also modify the CGL hash function to return the images of two given points of the starting curve E under the D_M-isogeny with target curve E_{j_n}. Under the notation from the previous modification, we have $\varphi_m = \varphi_n \circ \cdots \circ \varphi_1$. For the rest of this work, as we will only make use of this family of generalised functions, we therefore refer by $\mathcal{H}^{p,D_m}$ to the hash function family

$$\mathcal{H}^{p,D_M} = \left\{ h_j^{D_M} : m, P, Q \mapsto j_n, \varphi_m(P), \varphi_m(Q) \right\}.$$

6.2.2 Defining a new one-way function family

Given a prime number p and two integers D_M and N, we define a family of functions

$$\mathcal{F}^{p,D_M,N} : \mathcal{J}_p \times [\mu(D_M)] \to \mathcal{J}_p \times (\overline{\mathbb{F}_{p^2}})^2 \times (\overline{\mathbb{F}_{p^2}})^2$$

which makes use of the extended CGL hash function family $\mathcal{H}^{p,D_M}$ described in the previous section. Each function f_j in the family, for $j \in \mathcal{J}_p$, first computes the canonical curve E_j together with a canonical basis (P_j, Q_j) of the torsion group $E_j[N]$. It then maps an input $m \in [\mu(D_M)]$ as:

$$f_j : m \mapsto (j_c, P_c, Q_c) = h_j^{D_M}(m, P_j, Q_j).$$

Statistically random sampling from the family. We let the canonical curve E_{1728} be part of the global parameters of the family $\mathcal{F}^{p,D_M,N}$ as a fixed starting curve for sampling purposes. Let D_S be an integer coprime to D_M. To then select a random $f_j \in \mathcal{F}$, a random D_S-isogeny with kernel K_S is sampled which fixes the target curve

$E_S = E_{1728}/K_S$. Taking its j-invariant j_S, this selects the function f_{j_S} from the family.

Theorem 6.6 ([71, Theorem 1]). *For an integer $D_S = \prod_i \ell_i^{e_i}$, the distribution of the j-invariant j_S sampled as the last j-invariant of a random walk through the graphs G_{ℓ_i} of length D_S is within statistical distance $\prod_i \left(\frac{2\sqrt{\ell_i}}{\ell_i+1}\right)^{e_i}$ of uniform.*

Similarly to [71, Lemma 1], letting $\epsilon > 0$ be arbirarily small and setting $D_S = \prod_i \ell_i^{e_i}$ with all ℓ_i being all primes less than $2(1+\epsilon)\log p$ and taking $e_i = \max\{e \in \mathbb{N} : \ell_i^e < 2(1+\epsilon)\log p\}$ leads to a statistical distance of less than $1/p^{1+\epsilon}$.

Injectivity. We show that the functions in the family are injective for the right choice of parameters.

Lemma 6.7 ([57, Lemma 1]). *For $N^2 > 4D_M$, any function $f_j \in \mathcal{F}^{p,D_M,N}$ is injective.*

Proof. Suppose that a function f_j is not injective, i.e. that there are two distinct isogenies φ and φ' of degree D_M from E_j to E_c, corresponding to two distinct messages, with the same action on $E_j[N]$, implied by the colliding images of P_j and Q_j. Then, following from [101, Section 4], their difference $\varphi - \varphi'$ is also an isogeny between the same curves whose kernel contains the entire N-torsion. This, together with [118, Lemma V.1.2], implies that $4D_M \geq \deg(\varphi - \varphi') \geq N^2$. Taking $N^2 > 4D_M$ ensures that in fact $\varphi = \varphi'$ and therefore that f_j is injective. $\square$

One-wayness. We finally prove that a randomly sampled function from the family $\mathcal{F}$ is one-way under the assumption of hardness of the RCSSI problem.

Lemma 6.8 ([57, Lemma 2]). *Let D_S be such that the distribution of the resulting j_S is statistically close to uniform. A function $f_j \in \mathcal{F}^{p,D_M,N}$ sampled at random as described above is quantum one-way under the hardness of Problem 6.4 with isogeny degree $d = D_M$ and torsion degree N.*

Proof. Suppose that there is a PPT quantum adversary $\mathcal{A}$ that can break the one-wayness of f_j; that is, given j and $(j_c, P_c, Q_c) = f_j(m^*)$ for $m^* \leftarrow_{\$} [\mu(D_M)]$, $\mathcal{A}$ can recover m^* with non-negligible probability. We build a reduction $\mathcal{B}$ which receives a challenge $(E_1, P_1, Q_1, E_2, P_2, Q_2)$ for Problem 6.4, with X being the uniform distribution over isogenies of degree D_M, and returns an isogeny $\varphi : E_1 \to E_2$ such that $\varphi(P_1) = P_2$ and $\varphi(Q_1) = Q_2$.

We first observe that since E_1 is uniformly distributed, then the distribution of its j-invariant statistically close to that expected by $\mathcal{A}$ for j, so $\mathcal{A}$ is not able to distinguish such distributions. We also observe that the distribution of isogenies resulting from hashing a uniform $m^* \leftarrow_{\$} [\mu(D_M)]$ is exactly the distribution X of D_M-isogenies. The reduction therefore passes $j(E_1)$ and $(j(E_2), P_2', Q_2')$ to $\mathcal{A}$, where P_2' and Q_2' are the images of P_2 and Q_2 under the isomorphism from E_2 to $E_{j(E_2)}$, and $\mathcal{A}$ will return a corresponding input m with high probability. By reproducing the hashing of m, the reduction $\mathcal{B}$ can then recompute an isogeny $\tilde{\varphi} : E_{j(E_1)} \to E_{j(E_2)}$. Note here that if m is a correct pre-image of (j_c, P_c, Q_c) under the function f_j, then we are certain that it is the only one as, by Lemma 6.7, f_j is injective. With its knowledge of E_1, E_2, P_1 and Q_1, $\mathcal{B}$ can then compute the appropriate isomorphism from E_1 to $E_{j(E_1)}$ use that from $E_{j(E_2)}$ to E_2 and compose them with $\tilde{\varphi}$ to obtain φ. □

6.2.3 Inverting functions with an isogeny trapdoor

We now show how to use the algorithm of [110] to invert a given function $f_j \in \mathcal{F}^{p,D_M,N}$. We are given $\mathsf{ct} = (j_c, P_c, Q_c)$ as the output of $f_j(m)$ for some unknown m, and also the random isogeny $\varphi_S : E_{1728} \to E_j$ of degree D_S used to select E_j at random; this random isogeny is the trapdoor of this inversion mechanism. This gives us the composed isogeny $\varphi = \varphi_m \circ \varphi_S : E_{1728} \to E_m$ of degree $D = D_M \cdot D_S$, where φ_m is the walk determined by m, used in the computation of $f_j(m)$.

We now assume that we know a particular endomorphism $\theta \in \mathrm{End}(E_{1728})$ together with an integer $d \in \mathbb{Z}$ such that $\mathrm{Tr}(\theta) = 0$ and $\deg(\varphi \circ \theta \circ \hat{\varphi} + [d]) = N$. The other authors of [57] describe in Section 3.4 how such a θ can be computed. (This technique was recently further improved to have lower requirements on the imbalance between D and N [94].) Furthermore we assume that D is odd, that $\gcd(D, N) = 1$ and that $-4\deg(\theta)$ is not a square modulo every prime divisor of D. It was shown, by the other authors of [57], how to find such a suitable θ as part of the global parameters of the function family $\mathcal{F}^{p,D_M,N}$.

To invert a given output, we first let $\psi = \varphi \circ \theta \circ \hat{\varphi} + [d] \in \mathrm{End}(E_m)$. As described in [110], we can compute ψ by noting that it has degree N and that we know its action on the torsion group $E_m[N]$ (because of the basis P_c, Q_c). This means that we can compute its kernel, as it is contained within $E_m[N]$.

Since we can compute ψ, we can compute $G = \ker(\psi - [d]) \cap E_m[D]$ efficiently. Lemma 6.9 below, proved by the other authors of [57], in fact shows that $G = \ker(\hat{\varphi})$ directly and therefore we can recover first $\ker(\varphi)$ and then $\ker(\varphi_m)$ by separating out φ_S. This finally allows us to recover $m \in [\mu(D_M)]$ by retracing φ_m through the isogeny graph. These steps are summarized in Algorithm 1.

Algorithm 1 Inverse computation

```
Require: $\mathsf{ct}, \varphi_S, \theta \in \mathrm{End}(E_{1728}), d \in \mathbb{Z}$.
Ensure: $m \in [\mu(D_M)]$ such that $f_j(m) = \mathsf{ct}$.
1: Parse ct as $(j_c, P_c, Q_c) \in \mathcal{J}_p \times (\overline{\mathbb{F}_{p^2}})^2 \times (\overline{\mathbb{F}_{p^2}})^2$.
2: Compute the canonical curve $E_m = E_{j_c}$.
3: Let $\varphi = \varphi_m \circ \varphi_S : E_{1728} \to E_m$.
4: Let $\psi = \varphi \circ \theta \circ \hat{\varphi} + [d] \in \mathrm{End}(E_m)$.    ▷ Choices of $\theta$ and $d$ ensure $\deg \psi = N$.
5: Compute $K_1 = \ker \psi \subset E_m[N]$ using $d, \theta, \phi_S$ and $P_c, Q_c \in E_m[N]$.
6: Compute $K_2 = \ker(\varphi \circ \theta \circ \hat{\varphi}) \cap E_m[D] = \ker(\psi - [d]) \cap E_m[D] = \ker(\hat{\varphi})$.
7: Compute $\ker(\varphi_m)$ using $\ker(\hat{\varphi})$.
8: return $m \in [\mu(D_M)]$ that corresponds to $\ker(\phi_m)$.
```

Lemma 6.9 ([57, Lemma 3])**.** *Let θ be such that $-\deg(\theta)$ is a quadratic nonresidue modulo every prime dividing $D = D_M \cdot D_S$. Then G is cyclic and $G = \ker(\hat{\varphi})$.*

Avoiding a timing dependency.

The condition that $-\deg(\theta)$ is a quadratic nonresidue modulo every prime dividing D may seem strange at first since in [110] the case when G is not cyclic is also considered. In the original attack algorithm of in [110], there is a guessing step where a θ-invariant subgroup of $E_{1728}[k]$ has to be found, where k is the largest divisor of D such that $E_m[k] \subset G$. Since k depends on m, and since this guessing step takes time exponential in the number of prime factors of k, this leads to a dependency of the inversion alorithm on the message. Not only does this open the avenue to timing attacks further on, it also raises the question of whether the inversion runs efficiently for all possible inputs. However, with the particular condition on $-\deg(\theta)$ mentioned above, it holds that $k = 1$ for all messages m. This extra condition on θ only increases the parameters slightly and represents a conservative choice for the sake of security.

Detection of invalid inputs.

When provided with a valid ciphertext ct, Algorithm 1 will always return the corresponding plaintext. To detect invalid inputs we proceed as follows. If any of the steps fails we return $\perp$ to indicate that the ciphertext is invalid. If the algorithm returns an output $\tilde{m}$ then we recompute the image $\tilde{\mathsf{ct}}$ from it; if that matches the original ct, then we return $\tilde{m}$ as a valid message; otherwise we return $\perp$.

Algorithm 2 The KGen algorithm for $\mathsf{S\acute{E}TA}_{\mathsf{OW\text{-}CPA}}$.

```
Require: params = $(\lambda, p, j_{1728}, D_S, D_M, N, \theta)$.
Ensure: pk = $j_S$, sk = $K_S$.
1: Sample a random cyclic subgroup $K_S \subset E_{1728}(\overline{\mathbb{F}_{p^2}})$ of size $D_S$.
2: Compute the isogeny $\varphi_S : E_{1728} \to E_S := E_{1728}/\langle K_S \rangle$.
3: Compute the $j$-invariant $j_S = j(E_S)$ and its canonical curve $E_{j_S}$.
4: Set pk := $j_S$ and sk := $K_S$.
5: Return (pk, sk).
```

Algorithm 3 The Enc algorithm for $\mathsf{S\acute{E}TA}_{\mathsf{OW\text{-}CPA}}$.

```
Require: Parameters params, pk = $j_S$ and plaintext $m$.
Ensure: Ciphertext ct.
1: Cast $m$ as an integer in $[\mu(D_M)]$
2: Compute $(j_c, P_c, Q_c) \leftarrow f_{j_S}(m)$, where $f_{j_S} \in \mathcal{F}^{p,D_M,N}$.
3: Embed $(j_c, P_c, Q_c)$ as a binary string ct $\in \{0,1\}^{n_C}$.
4: Return ct.
```

6.3 IND-CCA public-key encryption from isogenies

We now build a PKE scheme using the family of trapdoor OWFs of Section 6.2 and show that it is OW-CPA secure; then we modify it using a generic transformation to achieve IND-CCA security.

6.3.1 OW-CPA encryption scheme

We present the $\mathsf{S\acute{E}TA}_{\mathsf{OW\text{-}CPA}}$ PKE scheme as the tuple (Setup, KGen, Enc, Dec) of PPT algorithms described below, as originally published in [57].

Parameters. Let E_{1728} be a fixed supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ with j-invariant 1728. Let D_S, D_M and N be integers chosen according to the requirements of Section 6.2. Let $\theta \in \mathrm{End}(E_0)$ be a particular endomorphism computed as in [57, Section 3.4]. We let params $= (\lambda, p, j_{1728}, D_S, D_M, N, \theta)$ be the output of the Setup algorithm.

Key generation. The KGen(params) algorithm is decribed in Algorithm 2.

Encryption. Let $n_M = \lfloor \log_2 \mu(D_M) \rfloor$ and n_C be sufficiently many bits to represent one j-invariant in $\mathcal{J}_p$ and two points in $(\overline{\mathbb{F}_{p^2}})^2)$. For a message $m \in \{0,1\}^{n_M}$, the Enc(params, pk, m) algorithm is decribed in Algorithm 3.

Decryption. The Dec(params, pk, sk, ct) algorithm is decribed in Algorithm 4.

Algorithm 4 The $\mathtt{Dec}$ algorithm for $\mathsf{S\acute{E}TA}_{\mathsf{OW\text{-}CPA}}$.

```
Require: Parameters $\mathsf{params}$, keys $\mathsf{pk}, \mathsf{sk}$ and ciphertext $\mathsf{ct}$.
Ensure: Plaintext $m$ or error message $\perp$.
1: Parse $\mathsf{ct}$ as $(j_c, P_c, Q_c) \in \mathbb{F}_{p^2} \times (\overline{\mathbb{F}_{p^2}})^2 \times (\overline{\mathbb{F}_{p^2}})^2$; if that fails, return $\perp$.
2: Execute Algorithm 1 to recover $\tilde{m} \in [\mu(D_M)]$; if this fails, set $\tilde{m} = \perp$.
3: If $\tilde{m} \neq \perp$; verify that $f_{j_S}(\tilde{m}) \overset{?}{=} \mathsf{ct}$. If not, set $\tilde{m} = \perp$.
4: If $\perp$ was recovered, return $\perp$.
5: Otherwise, from $\tilde{m} \in [\mu(D_M)]$, recover $m \in \{0,1\}^{n_M}$ and return it.
```

With the algorithms defined above, we now prove the $\mathsf{OW\text{-}CPA}$ security of the scheme.

Theorem 6.10. *Let D_S be such that the distribution of j_S is statistically close to uniform. If Problem 6.4 with $p, d = D_M, N$ and X such that $H_\infty(X) = O(\lambda)$ is hard for quantum PPT adversaries, then the PKE scheme above is quantum $\mathsf{OW\text{-}CPA}$-secure.*

Proof. In the notation of Definition 2.11, we have $\mathcal{M} = \{0,1\}^{n_m}$. We see that a randomly sampled $m \leftarrow_{\$} \mathcal{M}$ directly embedded as an integer $m \in [\mu(D_M)]$ yields a distribution Y with min-entropy $H_\infty(Y) = \lambda$ on isogenies of degree D_M starting from E_S. Similarly to the proof of Lemma 6.2, the challenge of inverting a given ciphertext ct reduces to recovering the secret isogeny of Problem 6.4 with $X = Y$. $\square$

Ciphertext size. A ciphertext is composed of a j-invariant $j_c \in \mathbb{F}_{p^2}$, which can be represented with $2\log p$ bits, and two torsion points $P_c, Q_c \in E_{j_c}[N]$, each of which can be represented with $2\log N$ bits by identifying each N-torsion point with a pair of elements in $\mathbb{Z}_N$. Therefore, the bit size of a ciphertext is

$$2\log p + 4\log N.$$

Further compression is possible, representing both torsion points with $3\log N$ bits, using the techniques in [46, Section 6.1].

6.3.2 IND-CCA encryption scheme from OAEP transformation

We now show how to construct $\mathsf{S\acute{E}TA}_{\mathsf{IND\text{-}CCA}}$, an $\mathsf{IND\text{-}CCA}$ secure PKE scheme based on our OWF of Section 6.2. We do so with the post-quantum OAEP transformation of [122, Section 5] which we state below. We then prove that our function f is quantum partial-domain one-way which is the required property to make use of the transformation. We then recall the transformation's security theorem and comment on the efficiency of the resulting IND-CCA scheme.

Post-quantum OAEP transformation

Let

$$f : \{0,1\}^{\lambda+k_1} \times \{0,1\}^{k_0} \to \{0,1\}^{n_C}$$

be an invertible injective function. The function f is the public key of the scheme, its inverse f^{-1} is the secret key. The scheme makes use of three hash functions

$$\begin{aligned} G &: \{0,1\}^{k_0} \to \{0,1\}^{k-k_0}, \\ H &: \{0,1\}^{k-k_0} \to \{0,1\}^{k_0}, \\ H' &: \{0,1\}^{k} \to \{0,1\}^{k}, \end{aligned}$$

modelled as random oracles, where $k = \lambda + k_0 + k_1$. Given those, the `Setup` and `KGen` algorithms remain as before, but the `Enc` and `Dec` algorithms are defined as follows:

- `Enc`: given a message $m \in \{0,1\}^\lambda$, sample $r \leftarrow_{\$} \{0,1\}^{k_0}$ and set

  $$\begin{aligned} s &= m||0^{k_1} \oplus G(r), & t &= r \oplus H(s), \\ c &= f(s,t), & d &= H'(s||t), \end{aligned}$$

  and output the ciphertext (c,d).

- `Dec`: given a ciphertext (c,d), use the secret key to compute $(s,t) = f^{-1}(c)$. If $d \neq H'(s||t)$ output $\perp$. Otherwise, compute $r = t \oplus H(s)$ and $\tilde{m} = s \oplus G(r)$. If the last k_1 bits of $\tilde{m}$ are 0, output the first n bits of $\overline{m}$, otherwise output $\perp$.

Quantum partial-domain one-wayness

Lemma 6.11. *The function f defined in Section 6.2.2 is a quantum partial-domain one-way function, under the hardness of Problem 6.4.*

Proof. We note that in our case, partial domain inversion is the same as domain inversion where only the first part of the path is required. More precisely, we factor D_M as $D'_M \cdot D''_M$ such that $\gcd(D'_M, D''_M) = 1$, $2^{\lambda+k_1} \leq \mu(D'_M)$ and $2^{k_0} \leq \mu(D''_M)$ (where $\lambda + k_0 + k_1$ is the bit-length of input strings) and then embed each of s and t in the respective factors when computing $f(s,t)$. If D'_M is appropriately set, then recovering s from $c = f(s,t)$ is hard under the same assumption as Theorem 6.10 with D_M replaced by D'_M. $\square$

Theorem 6.12 ([122, Theorem 2])**.** *If f is a quantum partial-domain one-way function, then the OAEP-transformed scheme is* IND-CCA *secure in the quantum random oracle model (QROM).*

Ciphertext size. The IND-CCA version adds an output of a hash function H', which has the same size as the input of the one-way function f. Thus the total bit size of the ciphertext is

$$2\log p + 4\log N + k,$$

where $k = \lambda + k_0 + k_1$.

Chapter 7

Invertible semi-commutative masking

In this chapter, we construct (invertible) semi-commutative masking structures as a generalisation of exponentiation-only mechanisms in protocol design and formulate abstract computational problems that derive from such structures. We then present two concrete instantiations, one based on group actions and one based on isogenies of supersingular elliptic curves as defined in Chapter 6.

Except where specified, the material in this chapter is selected from the work of E. Orsini[1], C. Petit[2], N. P. Smart[3] and this author; it was accepted for publication in the proceedings of the *19th International Conference on Cryptology and Network Security, CANS 2020* and will be presented at the virtual edition of the conference [58]. The ideas and analyses presented here were the joint work of this author and his co-authors and, except where specified, the final content reproduced here is this author's own work.

7.1 Semi-commutative masking structures

We define the abstraction of *semi-commutative invertible masking structures.* It allows us to define protocols succinctly without dealing with implementation details.

To help ideas we illustrate the presentation of these structures with the case of discrete logarithms in a finite field $\mathbb{F}_p$ where p and $q = (p-1)/2$ are primes and $g \in \mathbb{F}_p$

[1]University of Bristol, U.K., later KU Leuven, Belgium.
[2]University of Birmingham, U.K.
[3]University of Bristol, U.K., later KU Leuven, Belgium.

is an element of order q.

7.1.1 Definition of masking structures

Elements and representatives. A masking structure $\mathcal{M}$ is defined over a set X. Each element $x \in X$ may have multiple *representations*, and we define R_x to be the set of representations of an element $x \in X$. We require that it be efficient to recover x from any representation in R_x. We denote the set of all such sets by $R_X = \{R_x\}_{x \in X}$.

The sets of representatives are assumed to be disjoint, i.e. $\forall x, x' \in X$ s.t. $x \neq x'$: $R_x \cap R_{x'} = \emptyset$, and we define $R = \cup_{x \in X} R_x$ to be the set of all representatives.

For example, if we take $X = \langle g \rangle \subset \mathbb{F}_p^*$, then the usual choice for R is to let $R_x = \{x\}$ for every $x \in X$. One could also take a redundant representation with two elements, letting $R_x = \{x, x+p\}$.

Masking sets and invertible masks. Given a structure $\mathcal{M}$ with its set X and its set of representatives R, a *mask* is a function $\mu : R \to R$, and a masking set M is a set of such functions.

In the discrete logarithm analogue M can be a set of exponentiation maps on the representatives of the elements of $X = \langle g \rangle$ indexed by elements in $\mathbb{Z}_q^*$. That is $M = \{\mu_a, a \in \mathbb{Z}_q^*\}$ with $\mu_a : h \mapsto h^a$.

A masking function $\mu \in M$ is said to be *invertible* if

$$\forall x \in X, \quad \forall r \in R_x, \quad \exists\, \mu^{-1} \in M \quad : \quad \mu^{-1}(\mu(r)) \in R_x.$$

We only require that the inverse mask outputs a representative in the same set, not necessarily the same representative. For a given masking set M, if all masks $\mu \in M$ are invertible, then we say that the masking set M is *invertible.*

In the discrete logarithm example if μ_a corresponds to the map $h \mapsto h^a$, then μ_a^{-1} corresponds to the map $\mu_{1/a} : h \mapsto h^{1/a}$.

Semi-commutative structures. An *invertible masking structure* $\mathcal{M}$ for a set X is then a collection of sets of representative R_X, along with a collection of n invertible masking sets $[M_i]_{i=1}^n$, and we write $\mathcal{M} = \{X, R_X, [M_i]_{i=1}^n\}$. Such an invertible masking structure is said to be *semi-commutative* if

$$\forall i \neq j, \;\; \forall \mu \in M_i, \;\; \forall \mu' \in M_j, \;\; \forall r \in R, \;\; \mu(\mu'(r)) \in R_x \iff \mu'(\mu(r)) \in R_x.$$

1. The challenger sets 1^λ and $\mathcal{M} = \{X, R_X, [M_i]\}$.
2. $\mathcal{A}$ is given 1^λ and $\mathcal{M}$ and outputs (r, μ_0, μ_1, i) such that $r \in R$, $i \in [n]$, $\mu_0, \mu_1 \in M_j$, $j \neq i$.
3. The challenger computes $r_0 \leftarrow \mu_0(r)$ and $r_1 \leftarrow \mu_1(r)$ and also samples a bit $\mathsf{b} \leftarrow_\$ \{0,1\}$ at random.
4. The challenger samples a random mask $\mu \leftarrow_\$ M_i$ and computes $\tilde{r} \leftarrow \mu(r_\mathsf{b})$.
5. $\mathcal{A}$ is given $\tilde{r}$ and outputs a guess $\tilde{\mathsf{b}}$.
6. The experiment ouputs 1 if and only if $\tilde{\mathsf{b}} = \mathsf{b}$, and 0 otherwise.

Figure 7.1: The $\mathsf{IND\text{-}Mask}$ security experiment $\mathbf{Exp}^{\mathsf{IND\text{-}Mask}}_{\mathcal{A},\mathcal{M}}$.

In our discrete logarithm setting, M being the set of exponentiation maps considered earlier, we see that $\mathcal{M} = \{X, R_X, [M, M]\}$ is a semi-commutative invertible masking structure since commutativity of elements of $\mathbb{Z}_q^*$ is immediate.

7.1.2 Problems and properties.

We now present a distinguishing experiment and several computational problems for such masking structures. We will prove the security of our protocols to hold under the assumption of the difficulty of these problems, and the precise security level will then be established when we instantiate our generic structure from concrete primitives.

The first security notion of indistinguishable masking, $\mathsf{IND\text{-}Mask}$, whose security experiment is described in Figure 7.1, guarantees that a random mask hides distinguishing information about the representatives that it acts on.

Definition 7.1 ($\mathsf{IND\text{-}Mask}$ security)**.** We say that a semi-commutative invertible masking structure $\mathcal{M} = \{X, R_X, [M_i]\}$ is *(quantum) $\mathsf{IND\text{-}Mask}$-secure* if for all probabilistic $\mathsf{poly}(\lambda)$-time (quantum) adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\left| \Pr\left[\mathbf{Exp}^{\mathsf{IND\text{-}Mask}}_{\mathcal{A},\mathcal{M}} = 1\right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda).$$

In the discrete logarithm setting, when $R_x = \{x\}$, the map $h \mapsto h^a$ for random $a \in \mathbb{Z}_q^*$ induces a permutation of the group elements in $\langle g \rangle$. Therefore for a secret a and two group elements h_0, h_1, the distribution of h_b^a is perfectly uniform, independently of b. This shows that such an $\mathcal{M}$ is perfectly $\mathsf{IND\text{-}Mask}$-secure, even against quantum adversaries.

Note 7.2. In some settings (but not in the discrete logarithm one), it may be possible to distinguish the action of two masks that belong to separate masking sets. It is also possible that this difference is preserved under the action of a mask from a third masking set. Therefore, if an adversary was able to submit arbitrary r_0 and r_1 to the IND-Mask experiment, it could ensure that the difference between them is preserved by the action of μ and hence win the experiment with certainty. By forcing $\mathcal{A}$ to submit a single $r \in R$ and two maps μ_0, μ_1 belonging to the same masking set M_j, the experiment prevents that strategy.

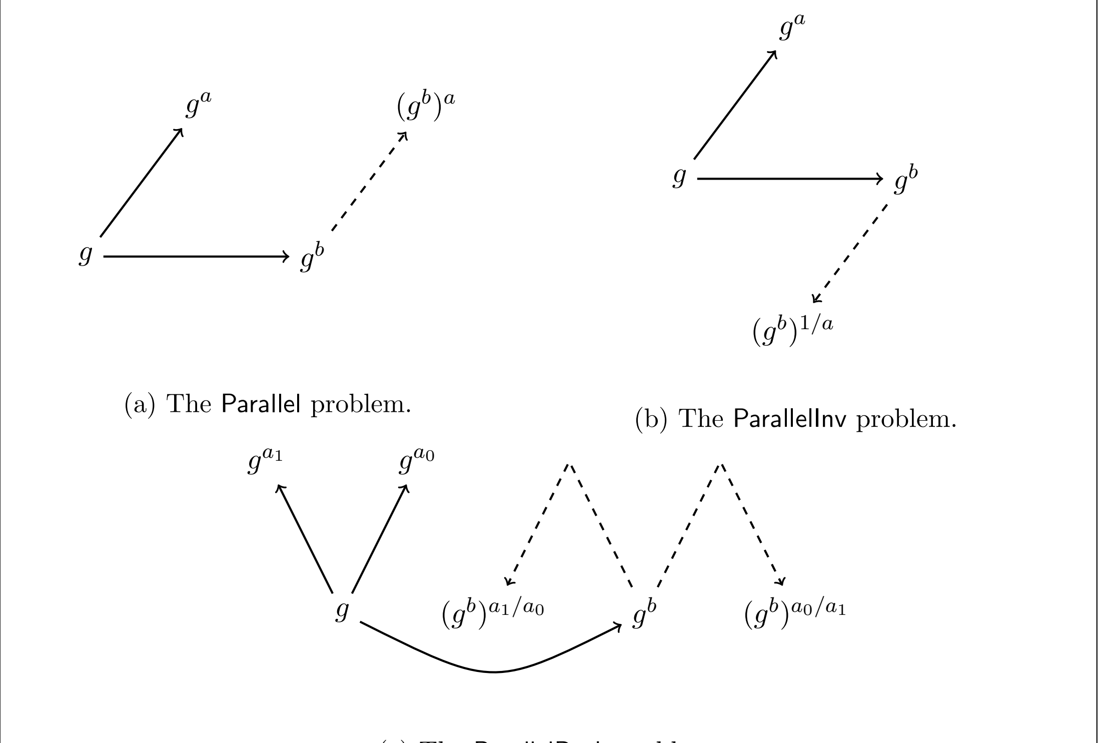
Definition 7.3 (Computational problems)**.** Given a semi-commutative invertible masking structure $\mathcal{M} = \{X, R_X, [M_i]\}$, we define the following computational problems:

1. Demask: Given (i, r, r_x) such that $r_x = \mu_x(r)$ for a uniformly random $\mu_x \leftarrow\$ M_i$, return μ_x.
2. Parallel: Given (i, j, r, r_x, r_y) such that $i \neq j$ and that $r_x = \mu_x(r)$ and $r_y = \mu_y(r)$ for uniformly random $\mu_x \leftarrow\$ M_i, \mu_y \leftarrow\$ M_j$, return $z \in X$ such that $\mu_x(r_y) \in R_z$.
3. ParallelInv: Given (i, j, r, r_x, r_y) such that $i \neq j$ and that $r_x = \mu_x(r)$ and $r_y = \mu_y(r)$ for uniformly random $\mu_x \leftarrow\$ M_i, \mu_y \leftarrow\$ M_j$, return $z \in X$ such that $\mu_x^{-1}(r_y) \in R_z$.
4. ParallelEither: Given (i, j, r, r_x, r_y) such that $i \neq j$ and that $r_x = \mu_x(r)$ and $r_y = \mu_y(r)$ for uniformly random $\mu_x \leftarrow\$ M_i, \mu_y \leftarrow\$ M_j$, return $z \in X$ such that either $\mu_x(r_y) \in R_z$ or $\mu_x^{-1}(r_y) \in R_z$.
5. ParallelBoth: Given $(i, j, r, r_{x_0}, r_{x_1}, r_y)$ such that $i \neq j$ and that $r_{x_\mathsf{b}} = \mu_\mathsf{b}(r), \mathsf{b} \in \{0,1\}$ and $r_y = \mu_y(r)$ for uniformly random $\mu_\mathsf{b} \leftarrow\$ M_i, \mu_y \leftarrow\$ M_j$, return $z \in X$ such that either $\mu_{1-\mathsf{b}}^{-1}(\mu_\mathsf{b}(r_y)) \in R_z$ or $\mu_\mathsf{b}^{-1}(\mu_{1-\mathsf{b}}(r_y)) \in R_z$.

To make explicit the given structure $\mathcal{M}$ to which the (say) Demask problem refers, we write $\mathsf{Demask}^{\mathcal{M}}$. The name "Parallel" is inspired by a similar problem defined by Couveignes [48] which we describe in Section 7.2.1.

We motivate these problems in the discrete logarithm setting, where we take our masking structure as before to have $X = \langle g \rangle$, $R_x = \{x\}$ and to have each M_i to be identical to the set of exponentiation maps indexed by $\mathbb{Z}_q^*$.

- We see that the Demask problem is, given (g, h) such that $h = g^a$ for a random a, to return a. This is exactly the discrete logarithm problem (DLP).



(a) The Parallel problem.

(b) The ParallelInv problem.

(c) The ParallelBoth problem.

Figure 7.2: Representations of computational problems.

- Similarly, the Parallel problem is, given (g, g^a, g^b) for random a, b, to return $g^{a \cdot b}$ which is exactly the computational Diffie–Hellman (CDH) problem. The name “Parallel” of [48] is thought to be derived from the representation shown in Figure 7.2a of the problem where the challenge is to compute the parallel operation.

- In the discrete logarithm setting, the ParallelInv problem is to compute $g^{b/a}$ given (g, g^a, g^b).

  We show here that in this setting it is equivalent to the Parallel problem. Given a challenge (g, g^a, g^b) for Parallel, we let (g^a, g, g^b) be a challenge for ParallelInv We rewrite this as $(h, h^{a'}, h^{b'})$ with $h = g^a$, $a' = 1/a$ and $b' = b/a$. As a and b are uniformly random, so are a' and b' and hence our ParallelInv solver returns

$$h^{b'/a'} = h^{a \cdot b/a} = (g^a)^b = g^{a \cdot b}$$

  which is exactly the solution to the Parallel challenge that was given. A similar reduction shows that the ParallelInv problem can be solved using a solver for the Parallel problem.

We note that this reduction does not immediately hold in the abstract case, due to the unspecified relation between r and $\mu^{-1}(\mu(r))$, but it can nonetheless be shown to hold for individual instantiations.

- The ParallelEither problem is an instance where both the solutions to the Parallel and to the ParallelInv problems, for the same challenge, are accepted. While it is immediate that the ParallelEither problem is at most as hard as any of the other two, a formal reduction to show the reverse implication does not appear to be as trivial. We conjecture that in most settings, and in the discrete logarithm setting in particular, allowing for two possible answers which are both hard to compute on their own does not significantly decrease the hardness of the ParallelEither problem.

- The solution of the ParallelBoth problem can be seen as a combination of both Parallel and ParallelInv solutions together with the choice of the ParallelEither problem as is shown in Figure 7.2c.

  Indeed, one can first use a Parallel oracle to compute $\mu_{\mathsf{b}}(r_y)$ for either $\mathsf{b} \in \{0,1\}$ and then use a ParallelInv oracle to compute $\mu^{-1}_{1-\mathsf{b}}(\mu_{\mathsf{b}}(r_y))$ which shows that ParallelBoth is at most as hard as those two problems. Similarly to the ParallelEither problem, we conjecture that in most settings the ParallelBoth will not be significantly easier as it requires solutions which are both hard to compute.

7.2 Instantiation from one-way group actions

We now present a generalisation of the discrete logarithm setting instantiation of our new semi-commutative masking structure. Specifically, we show that *hard homogeneous spaces*, as given in [36], and which are based on Couveignes's original definition [48], are an example of such structures. This is also the case for the action, via isogenies, of the class group of the ring of $\mathbb{F}_p$-rational endomorphisms of supersingular isogenies over $\mathbb{F}_p$ on the isomorphism classes of such curves.

7.2.1 Hard homogeneous spaces

We first give a summarized definition of hard homogeneous spaces and then formally instantiate a semi-commutative masking structure from such spaces. Throughout this section, we let G be a finite commutative group with identity element e and we denote the *group action* of G on a set X with the operator $*$ defined as: $* : G \times X \to X$, with $g * x \mapsto y$.

Definition 7.4 (Hard (efficient) homogeneous space)**.** A *homogeneous space* X for G is a finite set X on which G acts freely and transitively. This implies that for any $g \in G$ different from e, the permutation of X induced by the action of g has no fixed points; i.e. for given $x, y \in X$, there exists a unique $g \in G$ such that $y = g * x$. The space X is *efficient* if the following tasks are computationally easy (i.e. polynomial-time):

- evaluation of the group operation, inversion and equality testing of elements of G;
- sampling a random element from G with (close to) uniform distribution;
- deciding membership and equality of a representation of elements of X;
- evaluation of the action of a group element $g \in G$ on a set element $x \in X$.

The space X is *(quantum) hard* if the following tasks are computationally hard (i.e. not (quantum) polynomial-time):

- Vectorization: given $x, y \in X$, return $g \in G$ such that $y = g * x$; this is the analogue of the DLP for the group action.
- Parallelization: given $x, y, z \in X$ such that $y = g * x$, return $g * z$; this is the analogue of the CDH problem for the group action.

We then instantiate a masking structure and show that it realises our definition of a semi-commutative invertible masking structure.

Definition 7.5 (Masking structure from homogeneous space)**.** Given a homogeneous space X for G we define a masking structure $\mathcal{M}_{X,G} = \{X, R_X, [G, G]\}$ for X as follows:

- We let $R_x = \{x\}$ for each $x \in X$ and therefore have $R = X$.
- The masking tuple $[G, G]$ consists of two identical copies of the group G that acts on X.

Lemma 7.6. *Let X be an efficient homogeneous space for a commutative group G, then the masking structure $\mathcal{M}_{X,G} = \{X, R_X, [G, G]\}$ of Definition 7.5 is a semi-commutative masking structure.*

Proof. First we see that all the elements of $\mathcal{M}_{X,G}$ are well-defined and that the masking action of $\mu \in G : R \to R$ also is, where $\mu : r \mapsto \mu * r$. Next, we have that by definition of a group action, the masking of any $r \in R$ by any $\mu \in M_i$ for all i is indeed invertible. Also, since every M_i is a copy of the group G, the commutativity of G induces the semi-commutativity of $\mathcal{M}_{X,G}$. Finally, the properties of an efficient homogeneous space imply the efficiency of the operations required for a semi-commutative masking structure. □

We see here that a group action is stronger than our semi-commutative structure since any mask is in fact able to commute with any other. However the advantage of our weaker structure will become apparent in Section 7.3 with the instantiation from supersingular isogenies over $\mathbb{F}_{p^2}$.

Note 7.7. Before we discuss the instantiation of the computational problems, we briefly note that the two requirements for the hardness of a homogeneous space correspond exactly to the Demask and Parallel problems for a semi- commutative masking structure. Also, we have that the Parallel and ParallelInv problems are equivalent as it suffices to swap the first two elements of a challenge (x, y, z) for one problem to obtain a challenge (y, x, z) for the other which yields the same solution. Finally we have that ParallelEither is at most as hard as Parallel or ParallelInv. Hence we have

$$\mathsf{ParallelEither}^{\mathcal{M}_{X,G}} <_P \mathsf{Parallel}^{\mathcal{M}_{X,G}} \cong_P \mathsf{ParallelInv}^{\mathcal{M}_{X,G}}.$$

We also note that $\mathcal{M}_{X,G}$ is perfectly IND-Mask-secure since the action by a uniformly random element in G induces a perfect randomization of any element in X.

7.2.2 Discrete logarithm setting

The traditional Diffie-Hellman setting presented in Section 7.1 is a straightforward realisation of the hard homogeneous space presented above. Indeed, for any finite abelian group $\langle g\rangle$ of prime order in which the CDH problem is hard, we can let X be the set $\langle g\rangle$ and G be the set of exponentiation maps.

7.2.3 Class group of $\mathbb{F}_p$-rational endomorphisms

A second realisation of hard homogeneous spaces we present is a summary of work by Castryck et al. [36]. This work builds upon the Couveignes–Rostovtsev–Stolbunov scheme of [48, 114] where the public key space is the set of $\mathbb{F}_p$-isomorphism classes of *ordinary* elliptic curves over $\mathbb{F}_p$ whose endomorphism ring is a given order $\mathcal{O}$ in an imaginary quadratic field and whose trace of the Frobenius map has prescribed sign. The key ideas of the scheme of Couveignes et al. is that the ideal class group $\mathrm{cl}(\mathcal{O})$ acts freely and transitively on that set, and that this class group is commutative which allows for a natural key exchange protocol.

However, and despite recent improvements [67, 91], the scheme of Couveignes et al. is inefficient for the following reason. In order to decompose the action of an element of $\mathrm{cl}(\mathcal{O})$ into several smaller actions that are quicker to compute, De Feo, Kieffer and Smith [67] had the idea to chose $p \equiv -1 \mod \ell$ for several small odd primes ℓ. They then searched for an ordinary elliptic curve $E/\mathbb{F}_p$ such that $\#E(\mathbb{F}_p) \equiv 0$ modulo as many ℓ's as possible. This would ensure that the ideal $\ell\mathcal{O}$ decomposes as the product of two prime ideals $\mathfrak{l}$ and $\bar{\mathfrak{l}}$ for which the action of the ideal classes $[\mathfrak{l}]$ and $[\bar{\mathfrak{l}}]$ can be computed efficiently. If this works for sufficiently many ℓ's, then a generic element of $\mathrm{cl}(\mathcal{O})$ can be written as a product of small integral powers of such $[\mathfrak{l}]$ and the class group action can be computed efficiently. However, finding a curve $E/\mathbb{F}_p$ such that $\#E(\mathbb{F}_p) \equiv 0$ is hard and they only manage to obtain practical solutions for 7 different values of ℓ.

In order to increase the efficiency of this methodology, Castryck et al. adapt it to make use of supersingular elliptic curves, instead of ordinary ones, but still defined only over $\mathbb{F}_p$ [36]. Instead of the full ring of endomorphisms of such curves, which is not commutative, they consider the subring of $\mathbb{F}_p$-rational endomorphisms which is again an order $\mathcal{O}$ in an imaginary quadratic field. As before, the ideal class group $\mathrm{cl}(\mathcal{O})$ acts via isogenies on the set of $\mathbb{F}_p$-isomorphism classes of elliptic curves with $\mathbb{F}_p$-rational endomorphism ring equal to $\mathcal{O}$, we denote this set by $\mathcal{E}_p(\mathcal{O})$. Furthermore, contrary to the ordinary case, this action only has a single orbit.

The reason why this yields an increase in efficiency is that, in the supersingular case, $\#E(\mathbb{F}_p) = p + 1$ and hence $\#E(\mathbb{F}_p) \equiv 0$ modulo *all* primes $\ell \mid p + 1$ used in building p. This allows for many more values of ℓ to be used which in turn reduces the integral powers of each $[\mathfrak{l}]$ that appear in the decomposition of generic elements in $\mathrm{cl}(\mathcal{O})$. Concretely, Castryck et al. use 74 small odd primes in their implementation for which they heuristically expect that each element in $\mathrm{cl}(\mathcal{O})$ can be written as

$[\mathfrak{l}_1]^{e_1}[\mathfrak{l}_2]^{e_2}\cdots[\mathfrak{l}_{74}]^{e_{74}}$ with each $e_i \in \{-5,\ldots,5\}$. In contrast, for a class group of equivalent 256-bit size, using 7 small primes for the same approach would require exponents in the range of 2^{36} which leads to much slower computations.

We note that, typically, the exact structure of the class group $\mathrm{cl}(\mathcal{O})$ is unknown which requires the use of heuristics for actions such as the uniform sampling of elements. Furthermore, as a set of generators of $\mathrm{cl}(\mathcal{O})$ is not known, it is not possible to establish a canonical representation of elements which implies that evaluating the group action may not always be efficient. For the CSIDH-512 parameter set [36], which uses the 74 small primes mentioned above, Beullens, Kleinjung and Vercauteren performed the expensive computation of the exact structure of the corresponding class group [20]. This implies that the operations mentioned above can be computed efficiently for these specific parameters.

Lemma 7.8. *For a fixed prime field $\mathbb{F}_p$ and appropriate order $\mathcal{O}$ of an imaginary quadratic field, let $X = \mathcal{E}_p(\mathcal{O})$, and let $G = \mathrm{cl}(\mathcal{O})$. If the structure of $\mathrm{cl}(\mathcal{O})$ is known, then X is an efficient homogeneous space for G.*

Proof. As stated in the discussion above, we have that G acts freely and transitively on X, that it inherits the commutative structure of $\mathcal{O}$ and that, therefore, this is a well-defined homogeneous space. Also, due to the decomposition into classes of small prime ideals with small integral exponents the evaluation of the group operation, inversion, equality and sampling, as well as the action of a group element on a set element x are all efficient. Furthermore, as X can be represented as the set of Montgomery coefficients of the $\mathbb{F}_p$-isomorphism classes, equality of elements of X is efficient as well. □

As in the previous setting, the Demask and Parallel problems for the masking structure $\mathcal{M}_{X,G}$ induced by the homogeneous space of Lemma 7.8 immediately translate to analogues of the DLP and CDH in the class group action setting; and so does our prior discussion on the equivalence of ParallelInv and Parallel and on the hardness of ParallelEither. The classical and post-quantum security of the DLP analogue in this setting was already succinctly discussed in [36, Section 7] and was addressed in greater detail in [25] which provides a finer estimation of the required security parameters.

7.3 Instantiation from supersingular isogenies over $\mathbb{F}_{p^2}$

In Section 7.2 above, the commutative property of G gives stronger algebraic properties to the induced masking structure than the weaker semi-commutativity which the definition requires. In addition, the first realisation presented in Section 7.2.2 also possesses a group structure on the set X which is compatible with the action of G. This structure on X plays a key role in the design of several protocols as it enables increased flexibility, such as the OT protocol of Chou and Orlandi [43], but it also leads to new attack vectors such as Pohlig–Hellman-style attacks. The second realisation of Section 7.2.3 does not possess such a structure on X compatible with the action G which eliminates this attack vector. However the commutative property of G itself still enables the Demask problem for $\mathcal{M}_{X,G}$ to be presented as an instance of the abelian shift problem for which a sub-exponential quantum algorithm with time complexity $L_p[1/2]$ is known to exist [40]. When the restriction to $\mathbb{F}_p$-rational endomorphism rings is lifted, and the full space of isogenies over $\mathbb{F}_{p^2}$ is considered, this commutative property disappears and the Demask problem can no longer be attacked by the algorithm above. This is what gave rise to De Feo, Jao and Plût's setting of supersingular elliptic curves over $\mathbb{F}_{p^2}$ that we presented in Chapter 6 and was used to build the SIDH protocol.

In this section we draw from elements described in Section 6.1 to construct a semi-commutative masking structure from this SIDH-style setting and discuss the hardness of the induced problems.

7.3.1 Masking structure

To start defining a semi-commutative masking structure, we fix $p = \ell_1^{e_1} \ell_2^{e_2} \cdots \ell_n^{e_n} \cdot f \pm 1$ as in Section 6.1.3. In this setting, there are (at most) five supersingular isogeny classes and we let $X \subset \mathcal{J}_p$ denote the j-invariants of one of the two classes with curves $E/\mathbb{F}_{p^2}$ with trace $t = p^2 + 1 - \#E(\mathbb{F}_{p^2}) \in \{-2p, 2p\}$; these two classes are the largest of the five [1].

Representatives

For each j-invariant $x \in X$ there is a canonical choice of curve E_x, as presented in Section 6.1.1 following from [71, Section 2.4]. We define the set of representatives to be the set of tuples

$$R_x = \{(E_x, \{\{P_i, Q_i\}_{i \in [n]}\})\}$$

where E_x is the canonical curve for x or an appropriate twist and $\{P_i, Q_i\}$ is any basis of the torsion group $E_x[\ell_i^{e_i}]$ as in Section 6.1.3. For each torsion order $\ell_i^{e_i}$, we fix a generator $q_i \in \zeta_{\ell_i^{e_i}}$ such that for any curve E, the value of the Weil pairing evaluated on the basis points output by `Basis`(E, i) is equal to q_i. This will be used to derive new torsion points when required but these are still free to be modified within representative tuples under the action of isogenies. Hence for each x, there will be a unique choice of E_x but many choices of bases of torsion groups that originate from transformations of the deterministic one.

In protocols, when a common element $x \in X$ is required to be shared between parties in advance, then the parties should agree on a j-invariant and then derive the canonical curve and all the basis points in the deterministic way described above.

Masking sets

We first observe that for any $K_i = [\alpha_i]P_i + [\beta_i]Q_i$ on a curve E, the point $[\gamma]K_i$, for $\gamma \in (\mathbb{Z}/\ell_i^{e_i}\mathbb{Z})^*$, generates the same subgroup of $E[\ell_i^{e_i}]$ as K_i. By defining the equivalence relation $\sim_R$ by

$$(\alpha, \beta) \sim_R (\alpha', \beta') \qquad \Longleftrightarrow \qquad \exists \gamma \in (\mathbb{Z}/\ell_i^{e_i}\mathbb{Z})^* \text{ s.t. } (\alpha', \beta') = (\gamma\alpha, \gamma\beta),$$

we can then identify any such kernel K_i with the equivalence class of (α_i, β_i) which we denote $[\alpha_i : \beta_i]$. We recall that the projective line $\mathbb{P}^1(\mathbb{Z}/\ell_i^{e_i}\mathbb{Z})$ is the set of equivalence classes $[\alpha_i : \beta_i]$ such that $\gcd(\alpha_i, \beta_i) = 1$.

Since K_i has exact order $\ell_i^{e_i}$, at least one of α_i and β_i must not be divisible by ℓ_i and hence the ideal of the ring $\mathbb{Z}/\ell_i^{e_i}\mathbb{Z}$ generated by α_i, β_i is always the unit ideal, i.e. the whole of $\mathbb{Z}/\ell_i^{e_i}\mathbb{Z}$. This implies that all the possible choices for K_i can be exactly identified with the points on the projective line $\mathbb{P}^1(\mathbb{Z}/\ell_i^{e_i}\mathbb{Z})$. We therefore define n masking sets $[M_i]_{i\in[n]}$ where each M_i is the projective line $\mathbb{P}_i := \mathbb{P}^1(\mathbb{Z}/\ell_i^{e_i}\mathbb{Z})$.

Masking action

Given a mask $\mu \in M_i$, which is a point on $\mathbb{P}_i$, computing the result of $\mu(r) \in R_y$ on a representative $r = (E_x, \{\{P_i, Q_i\}_{i\in[n]}\}) \in R_x$ then consists in computing one of its representatives $K_i = [\alpha_i]P_i + [\beta_i]Q_i$ in $E_x[\ell_i^{e_i}]$ and the isogeny $\varphi_i : E_x \to E_x/\langle K_i\rangle$. Note that the curve $E_x/\langle K_i\rangle$ with j-invariant $y \in X$ may not be the same curve as the canonical choice E_y. However they will be isomorphic over $\mathbb{F}_{p^2}$, due to the appropriate choice of twist in the definition of our set R_y, and the isomorphism $\chi : E_x/\langle K_i\rangle \to E_y$ will be easy to compute.

To be able to compose isogenies in a semi-commutative way, computing $\mu(r)$ also requires computing the images of $\{\{P_j, Q_j\}\}$ for $j \neq i$ first under φ_i and then under the isomorphism χ to obtain bases of the torsion groups of E_y. It also requires generating a new basis for $E_y[\ell_i^{e_i}]$ using the $\texttt{Basis}(E_y, i)$ algorithm.

The output of the computation of the mask $\mu(r)$ is therefore the curve $E_y \overset{\chi}{\simeq} E_x/\langle K_i \rangle$ together with the basis points $\{\{\chi \circ \varphi_i(P_j), \chi \circ \varphi_i(Q_j)\}\}$ for $j \neq i$ and the output of $\texttt{Basis}(E_y, i)$.

Inverting the mask

Since our masking sets M_i do not derive from a group structure, we do not have an immediate instantiation of an inverse operation. However, recall that for every isogeny $\varphi : E \to E'$ of degree ℓ, there is a unique dual isogeny $\hat{\varphi} : E' \to E$ also of degree ℓ such that the composition is the multiplication-by-ℓ map: $\hat{\varphi} \circ \varphi = [\ell] : E \to E$. Whilst not a perfect inverse operation, in this setting the multiplication-by-$\ell_i^{e_i}$ map not only preserves the j-invariant of the curve, but also preserves the structure of the $\ell_j^{e_j}$-torsion groups for all $j \neq i$ which we require for semi-commutativity to hold.

Hence, given a kernel generator $K_i \in E[\ell_i^{e_i}]$ for some curve E, one can compute a generator of the image $\varphi_i(E[\ell_i^{e_i}]) \subset E'[\ell_i^{e_i}]$ of the $\ell_i^{e_i}$-torsion group under the isogeny $\tilde{\varphi}_i$ defined by K_i and an appropriate isomorphism, to obtain $\hat{K}_i \in E/\langle K_i \rangle$ which is a generator of the kernel of the unique dual isogeny $\hat{\varphi}_i$.

Given a mask $\mu \in M_i = \mathbb{P}_i$ and r, $r' = \mu(r)$ with $r' = (E', \{\{P_j, Q_j\}_{j \in [n]}\})$, computing the inverse μ^{-1} amounts to computing a point $\hat{K}_i$ as above and expressing it as $\hat{K}_i = (\hat{\alpha}_i, \hat{\beta}_i)$ in the deterministically generated basis for $E'[\ell_i^{e_i}]$ using the $\texttt{Basis}$ algorithm. This then allows us to define μ^{-1} uniquely as $[\hat{\alpha}_i : \hat{\beta}_i] \in \mathbb{P}_i$, given μ and r. We note that the dependency of μ^{-1} on μ and also r is consistent with the definition of the inverse of a mask as stated in Section 7.1.1.

This necessary dependency of the inverse for this setting is the principal factor for our definition of masks as points on $\mathbb{P}_i$ as opposed to only isogenies. This also allows us to capture isogenies that are different but have the same kernel and therefore induce identical masking actions.

Masking structure

We are then able to formally define a masking structure in this setting of isogenies of supersingular elliptic curves over $\mathbb{F}_{p^2}$.

Definition 7.9 (Masking structure from supersingular isogenies)**.** Let p be a prime defining the finite field $\mathbb{F}_{p^2}$ as above, we define the masking structure $\mathcal{M}_p = \{X, R_X, [M_i]_{i \in [n]}\}$ where the individual components are defined as above.

Lemma 7.10. *The masking structure $\mathcal{M}_p$ of Definition 7.9 is semi-commutative.*

Proof. First we see that the elements of $\mathcal{M}_p$ together with the action of any $\mu \in M_i$ on any r are well-defined. Then, since the composition of any isogeny with its dual results in an endomorphism of the starting curve, our method of inverting a given mask yields the same j-invariant regardless of the starting r or masking index i. Also, the semi-commutative property of our structure follows from the semi-commutative property of isogenies of co-prime degrees. Finally, the required efficiency of the computations for $\mathcal{M}_p$ follows from the comments above regarding the computation of isogenies of smooth degrees and expression of points in arbitrary torsion bases. Equality in X and M_i and membership in X are immediate to check. □

7.3.2 Computational problems

We first present a further specialization of the original CSSI problem (Problem 6.3) to the case where the prime p has the particular form $p = \ell_1^{e_1}\ell_2^{e_2} \cdot f \pm 1$ which was chosen in [66] to ensure that isogenies of degree dividing $p \mp 1$ could be decomposed into many small isogenies (typically $\ell_1 = 2$ and $\ell_2 = 3$). We state here a definition merged from that of Galbraith and Vercauteren [72] and Urbanik and Jao [123].

Problem 7.11 (2-i-CSSI problem [72, Definition 2][123, Problem 4.1])**.** Let (E, P_1, Q_1, P_2, Q_2) be such that $E/\mathbb{F}_{p^2}$ is a supersingular curve and P_j, Q_j is a basis for $E[\ell_j^{e_j}]$ for $j \in \{1, 2\}$. Let E' be such that there is an isogeny $\varphi : E \to E'$ of degree $\ell_i^{e_i}$. Let P'_j, Q'_j be the images under φ of P_j, Q_j for $j \neq i$. The *2-i-CSSI problem*, for $i \in \{1, 2\}$, is, given $(E, P_1, Q_1, P_2, Q_2, E', P'_j, Q'_j)$, to determine an isogeny $\tilde{\varphi} : E \to E'$ of degree $\ell_i^{e_i}$ such that $P'_j = \tilde{\varphi}(P_j)$ and $Q'_j = \tilde{\varphi}(Q_j)$.

This definition leads to the following natural generalisation which we show corresponds exactly to the computational problem that we need.

Definition 7.12 (n-i-CSSI problem)**.** Let $(E, \{P_j, Q_j\}_{j=1}^n)$ be a tuple such that $E/\mathbb{F}_{p^2}$ is a supersingular curve and P_j, Q_j is a basis for $E[\ell_j^{e_j}]$ for $j \in [n]$. Let E' be such that there is an isogeny $\varphi : E \to E'$ of degree $\ell_i^{e_i}$. Let $\{P'_j, Q'_j\}$ be the images under φ of $\{P_j, Q_j\}$ for $j \neq i$. The *n-i-CSSI problem*, for $i \in [n]$, is, given $(E, \{P_j, Q_j\}_{j=1}^n, E', \{P'_j, Q'_j\}_{j\neq i})$, to determine an isogeny $\tilde{\varphi} : E \to E'$ of degree $\ell_i^{e_i}$ such that $P'_j = \tilde{\varphi}(P_j)$ and $Q'_j = \tilde{\varphi}(Q_j)$ for all $j \neq i$.

Lemma 7.13. *Let $p = \ell_1^{e_1}\ell_2^{e_2} \cdots \ell_n^{e_n} \cdot f \pm 1$ be a prime and let $\mathcal{M}_p$ be a masking structure as defined in Definition 7.9. Then the* Demask *problem for $\mathcal{M}_p$ is an instance of the n-i-isogeny problem.*

Proof. The specification of i in (i, r, r_x) together with the random mask μ_x satisfies the promise of existence of an isogeny φ of degree $\ell_i^{e_i}$. Also, by definition of R_x for each $x \in X$ for $\mathcal{M}_p$, the representative r_x contains exactly the information of the curve E' together with the images of the appropriate torsion points. We note that r_x does not contain additional information as the basis points of $E'[\ell_i^{e_i}]$ are derived deterministically from E'. □

Computational SIDH problems

Together with the original CSSI problem (Problem 6.3), the isogeny problems defined above can be viewed as the analogues of the discrete logarithm problem of computing an unknown exponent in the case of generic primes p or these of a particular form. This naturally leads to an analogue of the CDH problem which is defined as follows in the case of $n = 2$.

Problem 7.14 (2-computational SIDH problem [123, Problem 4.3])**.** Let E, E_A, E_B be supersingular curves such that there exist isogenies $\varphi_A : E \to E_A$ and $\varphi_B : E \to E_B$ with kernels K_A and K_B and degrees $\ell_1^{e_1}$ and $\ell_2^{e_2}$ respectively. Let P_1, Q_1 and P_2, Q_2 be bases of $E[\ell_1^{e_1}]$ and $E[\ell_2^{e_2}]$ respectively, and let $P_1' = \varphi_B(P_1)$, $Q_1' = \varphi_B(Q_1)$ and $P_2' = \varphi_A(P_2)$, $Q_2' = \varphi_A(Q_2)$ be the images of the bases under the isogeny of coprime degree. The 2-*computational SIDH problem* is, given $(E, P_1, Q_1, P_2, Q_2, E_A, P_2', Q_2', E_B, P_1', Q_1')$, to identify the isomorphism class of the curve $E/\langle K_A, K_B\rangle$.

Note 7.15. We abbreviate the previous problem as 2-CSIDH and we stress that it has no relation to the CSIDH scheme of [36].

This problem can also be generalised in a natural way to the following which then yields the appropriate instantiation for our structure.

Problem 7.16 (n-i, j-computational SIDH problem)**.** Let E, E_A, E_B be supersingular curves such that there exist isogenies $\varphi_A : E \to E_A$ and $\varphi_B : E \to E_B$ with kernels K_A and K_B and degrees $\ell_i^{e_i}$ and $\ell_j^{e_j}$ respectively with $i \neq j$. Let $\{P_k, Q_k\}$ be bases of $E[\ell_k^{e_k}]$, for $k \in [n]$, and let $P_k^A = \varphi_A(P_k)$, $Q_k^A = \varphi_A(Q_k)$, for $k \neq i$, and $P_k^B = \varphi_B(P_k)$, $Q_k^B = \varphi_B(Q_k)$, for $k \neq j$ be the images of the bases under the isogeny of coprime degree. The *n-i, j-computational SIDH problem*, for $i, j \in [n]$, is, given $(E, \{P_k, Q_k\}_{k\in[n]}, E_A, \{P_k^A, Q_k^A\}_{k\neq i}, E_B, \{P_k^B, Q_k^B\}_{k\neq j})$, to identify the isomorphism class of the curve $E/\langle K_A, K_B\rangle$.

Lemma 7.17. *Let $p = \ell_1^{e_1}\ell_2^{e_2}\cdots\ell_n^{e_n}\cdot f \pm 1$ be a prime and let $\mathcal{M}_p$ be a masking structure* *as defined in Definition 7.9. Then the* Parallel *problem for $\mathcal{M}_p$ is an instance of the n-i, j-CSIDH problem.*

Proof. As for Lemma 7.13, the specification (i, j, r, r_x, r_y) of the Parallel problem for $\mathcal{M}_p$ satisfies the promise of existence of the two isogenies of coprime degrees and contains all the required information on the images of the torsion bases. Also, the goals of the problems agree since the solution to the Parallel problem for $\mathcal{M}_p$ requires $z \in X$ which is exactly the j-invariant which identifies the isomorphism class uniquely. Again, r_x and r_y do not contain additional information since the bases for the ith and jth torsion groups are computed deterministically. □

Regarding the ParallelInv problem for $\mathcal{M}_p$, we do not have an immediate reduction to the Parallel problem as we had for the previous instantiation. To follow the same proof strategy as for masking structures from homogeneous spaces (sketched in Note 7.7), one would have to swap r and r_x to submit a challenge to the oracle for the ParallelInv problem. The map from r_x to r would then be the inverse of the one from r to r_x but the map from r_x to r_y would no longer satisfy the promise of the ParallelInv problem. We discuss this interesting subtlety in the definitions of the CDH problem in Section 7.3.3. We nonetheless conjecture that, as they are very similar, the hardness of the ParallelInv problem is close to that of the Parallel problem. We similarly conjecture that the hardness of the ParallelEither and ParallelBoth problems is comparable to that of the Parallel and ParallelInv problems as no additional information is revealed and only similarly hard-to-compute solutions are required.

Decisional SIDH problem

Galbraith and Vercauteren also formalise a decisional variant of the SIDH problem in the case of $n = 2$ [72].

Problem 7.18 (2-i-decisional SIDH problem [72, Definition 3])**.** Let (E, P_1, Q_1, P_2, Q_2) be such that $E/\mathbb{F}_{p^2}$ is a supersingular curve and P_j, Q_j is a basis for $E[\ell_j^{e_j}]$ for $j \in \{1, 2\}$. Let E' be an elliptic curve and let $P'_j, Q'_j \in E'[\ell_j^{e_j}]$ for $j \neq i$. Let $0 < d < e_i$. The 2-i-*decisional SIDH problem* is, given $(E, P_1, Q_1, P_2, Q_2, E', P'_j, Q'_j, d)$ for $j \neq i$, to determine if there exists an isogeny $\varphi : E \to E'$ of degree ℓ_i^d such that $\varphi(P_j) = P'_j$ and $\varphi(Q_j) = Q'_j$.

As for the computational problems, we can generalise the above problem to our setting.

```
$\mathsf{CDH}_1(G = \langle g \rangle)$
1 : $a \leftarrow\!\!\$\ \mathbb{Z}_q^*$
2 : $b \leftarrow\!\!\$\ \mathbb{Z}_q^*$
3 : output $(g, g^a, g^b)$
```

(a) Traditional CDH.

```
$\mathsf{CDH}_2(G = \langle g \rangle)$
1 : $h \leftarrow\!\!\$\ G$
2 : $a \leftarrow\!\!\$\ \mathbb{Z}_q^*$
3 : $b \leftarrow\!\!\$\ \mathbb{Z}_q^*$
4 : output $(h, h^a, h^b)$
```

(b) Random base CDH.

```
$\mathsf{CDH}_3(G = \langle g \rangle)$
1 : $h \leftarrow\!\!\$\ G$
2 : $a \leftarrow\!\!\$\ \mathbb{Z}_q^*$
3 : $i \leftarrow\!\!\$\ G$
4 : output $(h, h^a, i)$
```

(c) Single promise CDH.

Figure 7.3: Three versions of the CDH challenge creation.

Definition 7.19 (n-i-decisional SIDH problem)**.** Let $(E, \{P_j, Q_j\}_{j\in[n]})$ be such that $E/\mathbb{F}_{p^2}$ is a supersingular curve and P_j, Q_j is a basis for $E[\ell_j^{e_j}]$ for $j \in [n]$. Let E' be an elliptic curve and let $P'_j, Q'_j \in E'[\ell_j^{e_j}]$ for $j \neq i$. Let $0 < d < e_i$. The *n-i-decisional SIDH problem* is, given $(E, \{P_j, Q_j\}_{j\in[n]}, E', \{P'_j, Q'_j\}_{j\neq i}, d)$, to determine if there exists an isogeny $\varphi : E \to E'$ of degree ℓ_i^d such that $\varphi(P_j) = P'_j$ and $\varphi(Q_j) = Q'_j$ for $j \neq i$.

Whilst we do not have an equivalence between the $\mathsf{IND\text{-}Mask}$ experiment and the n-i-DSIDH as presented above, we see that an oracle for the latter with $d = e_i$ is sufficient to obtain a noticeable advantage against the former. Also, it would seem that our $\mathsf{IND\text{-}Mask}$ experiment corresponds to a worst case of the n-i-DSIDH as it uses a maximal degree of $d = e_i$. We conjecture that the $\mathsf{IND\text{-}Mask}$ problem for $\mathcal{M}_p$ is not significantly easier than the n-i-DISDH for the same parameters.

As hinted at in Note 7.2, the Weil pairing is in fact a useful tool against the $\mathsf{IND\text{-}Mask}$ experiment. Indeed, if the adversary had free control over the values r_0 and r_1 of the experiment, it could give two representatives whose basis points of the same torsion group evaluated to different values under the Weil pairing. This difference would be preserved under the secret masking action of the experiment and this would enable it to win trivially. Restricting the adversary's input to be a single representative r and two masks of the same set that determine r_0 and r_1 and preserve the values of Weil pairing on the points of r thus prevents this strategy.

7.3.3 Different formulations of the CDH problem

In Figure 7.3, we present three subtly different versions of the CDH problem, simplified to their challenge creation and written using group exponentiation notation.

The first formulation, in Figure 7.3a, reflects the original definition of the CDH problem where the first element of the tuple (g, g^a, g^b) is always the pre-defined generator g. This formulation differs from our definition of the Parallel problem as the r element of our challenge tuple (i, r, r_x, r_y) does not have to be any pre-defined value. Instead our formulation is aligned on the second version, presented in Figure 7.3b. This then allows for the equivalence between the Parallel and ParallelInv problems to be proven formally in the setting of homogeneous spaces. Indeed we can construct a tuple (i, j, r', r'_x, r'_y), with $r' = r_x, r'_x = r$ and $r'_y = r_y$, where the promise of a map in M_i between r' and r'_x is satisfied because of the inverse, and the promise of a map in M_j taking r' to r'_y holds because, in this setting, there necessarily exists a map between any two elements.

However, the second implication does not hold in the setting of supersingular isogenies. Indeed, swapping r and r_x results in a isogeny of degree $\ell_i^{e_i} \cdot \ell_j^{e_j}$ between the curves in r'_x and r'_y as opposed to an isogeny of degree $\ell_j^{e_j}$ as promised by the problem.

Formulating the CDH challenge differently and removing the promise between r and r_y, as presented in Figure 7.3c, would enable a formal reduction to be built between the Parallel and ParallelInv problems in this less structured setting. We note that the CDH_2 and CDH_3 formulations are in fact equivalent in our first setting of homogeneous spaces.

Changing our definitions to allow for this reduction to be proven would however cause the computational problems to be further removed from their usage in practice. Indeed, the messages exchanged in protocols constructed in this setting typically satisfy the promises of our problems as they currently stand and we therefore chose not to modify our definitions.

Chapter 8

OT protocols from semi-commutative masking

In this chapter we construct two OT protocols from a semi-commutative structure $\mathcal{M}$ as constructed in Chapter 7. We first recall preliminary material on oblivious transfer and the universal composability (UC) framework of security before presenting each protocol and its proof of UC-security against passive adversaries with static corruptions in the ROM.

The material in this chapter is also selected from the work of Orsini, Petit, Smart, and this author [58]. The design of the protocols was joint work of all four authors; E. Orsini contributed the majority of Section 8.1; this author contributed the majority of the proofs in Sections 8.2 and 8.3. The contents of Section 8.2.3 were the joint work of this author and E. Orsini.

8.1 Preliminaries on oblivious transfer

Oblivous transfer, originally proposed by Rabin in 1981 [112], is a two-party primitive which enables a sender to transfer one of two messages to a receiver. Without knowing them in advance, the receiver selects one of the two messages and remains oblivious to the other one; similarly, the sender remain oblivious to which of the two messages was transfered. We formally describe the ideal OT functionality in Figure 8.1.

Universal composability security model

We present an overview of the universally composability (UC) framework of security established by Canetti [34]. Protocols that aim to achieve security in this model are

$\mathcal{F}^{\mathsf{OT}}$ functionality

PARAMETER: length n of the messages (formalized as bit-strings).

- Upon receiving $(P_S, \mathsf{sid}, m_0, m_1)$ from the sender, check if a $(\mathsf{sid}, \mathsf{c})$ was previously stored. If yes, send m_{c} to P_R; if not, store (sid, m_0, m_1) and continue to run.
- Upon receiving $(P_R, \mathsf{sid}, \mathsf{c})$ from the receiver, check if a (sid, m_0, m_1) was previously stored. If yes, send m_{c} to P_R; if not, store $(\mathsf{sid}, \mathsf{c})$ and continue to run.

Figure 8.1: Oblivious transfer functionality.

defined in three steps. First, the protocol and its execution in the presence of an adversary are formalized; this represents the *real-life model* which we also call the *real world*. Next, an ideal process for executing the task is defined; its role is to act as a trusted party by separately receiving the input of each party, honestly computing the result of the protocol internally and returning the output assigned to each party. In this *ideal world*, the parties do not communicate with one another but instead solely rely on the *ideal functionality* to provide them with their output. Finally, we say that the protocol in question *UC-realizes* the ideal functionality if running the protocol is equivalent to emulating the ideal functionality. We provide a brief discussion with additional formal details for the case of *semi-honest* adversaries with *static corruptions*.

In the real world, the parties involved in the execution of a protocol Π perform their own computation and communicate with one another when required to do so. Also present in the execution model is an adversary $\mathcal{A}$ which not only observes the messages exchanged but is also responsible for their delivery. This implies that it can choose to deliver them in the wrong order or to not deliver them at all. However, we assume that communication is authenticated and that $\mathcal{A}$ can therefore only deliver messages that were previously sent, without modifying them, and that it cannot deliver the same message more than once.

The final entity present in this execution model is the environment $\mathcal{E}$ which represents all of the events happening on the network at the time of the protocol execution. This environment is responsible for deciding the inputs and receiving the outputs of all the parties executing the protocol; this communication takes place outside of the view of $\mathcal{A}$ but we note that $\mathcal{A}$ still learns the inputs and outputs of corrupt parties as it is able to read their internal state. Furthermore, $\mathcal{E}$ interacts with $\mathcal{A}$ *throughout* the execution of the protocol Π.

$\mathcal{F}^{\mathtt{RO}}$ functionality

The functionality is parametrized by a domain $\mathcal{D}$ and range $\mathcal{R}$. It keeps a list L of pairs of values, which is initially empty and proceeds as follows:

- Upon receiving a value $(\mathsf{sid}, m), m \in \mathcal{D}$, if there is a pair $(m, \hat{h})$, $\hat{h} \in \mathcal{R}$, in the list L, set $h = \hat{h}$. Otherwise choose $h \leftarrow_{\$} \mathcal{R}$ and store the pair (m, h) in L.
- Reply to the activating machine with (sid, h).

Figure 8.2: Random oracle functionality.

In the ideal world, the parties instead interact with an ideal functionality $\mathcal{F}$ in a simple way: they pass their private inputs to $\mathcal{F}$ and wait for it to return their assigned output. There is also an adversary $\mathcal{S}$ which is responsible for the delivery of messages. As we assume that the functionality is a trusted third party, this adversary cannot observe the content of the messages. Finally, the same environment $\mathcal{E}$ is present in the ideal world. $\mathcal{E}$ also prescribes the inputs and observes the outputs of all parties and may interact with $\mathcal{S}$ throughout the execution of the ideal process.

In the *static corruptions* strategy, the adversary ($\mathcal{A}$ or $\mathcal{S}$) may choose, at the beginning of the execution only, to corrupt one or more parties in the protocol. After the execution begins, it is forbidden from corrupting new parties.

We also formalize *semi-honest* adversarial behaviour, also called *honest-but-curious*, by saying that the adversary may not send messages on behalf of corrupt parties. Instead, it is given read access to all of their internal state which includes their private input and output as well as their internal computations. In the real world, this forces $\mathcal{A}$ to follow the protocol honestly and in the ideal world, it restricts $\mathcal{S}$ to simply forwarding messages between parties and the functionality.

In addition to these two model of computation, the UC-framework also considers the $\mathcal{G}$-hybrid model where the parties in both real and ideal world have access to a copy of the ideal functionality $\mathcal{G}$. In the real world, this is an independent trusted party that executes the functionality honestly. In the ideal world, $\mathcal{S}$ executes an internal copy of the functionality $\mathcal{G}$ and only interacts with $\mathcal{F}$. Particularly, the ROM is formalized here using an instance of the $\mathcal{F}^{\mathtt{RO}}$ functionality as shown in Figure 8.2 and by proving the security of protocols in the $\mathcal{F}^{\mathtt{RO}}$-hybrid model.

To then prove that a protocol Π securely UC-realizes an ideal functionality $\mathcal{F}$, one must show that, for every adversary $\mathcal{A}$ interacting with Π in the real world, there exists an adversary $\mathcal{S}$ (often called the *simulator*) interacting with $\mathcal{F}$ in the ideal world such that *no environment* $\mathcal{E}$ should be able to distinguish whether it is interacting with $\mathcal{A}$ or with $\mathcal{S}$.

In other words, for every $\mathcal{A}$, one needs to design an $\mathcal{S}$ which is capable of *simulating* the view of $\mathcal{A}$ (which includes the transcript of the protocol and the internal state of the corrupt parties) such that no $\mathcal{E}$ can distinguish the simulation from a real execution. In this work, we restrict all of the entities $\mathcal{A}, \mathcal{S}, \mathcal{E}$ to probabilistic $\mathsf{poly}(\lambda)$-time algorithms.

We then say that the protocol Π securely realises the functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for every environment $\mathcal{E}$,

$$\mathsf{HYBRID}^{\mathcal{G}}_{\Pi,\mathcal{A},\mathcal{E}} \overset{\mathrm{c}}{\approx} \mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{E}},$$

where $\overset{\mathrm{c}}{\approx}$ denotes computational indistinguishability, $\mathsf{HYBRID}^{\mathcal{G}}_{\Pi,\mathcal{A},\mathcal{E}}$ denotes the output of $\mathcal{E}$ in an execution of the real protocol with the adversary $\mathcal{A}$ controlling the corrupted parties, and $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ denotes the output of $\mathcal{E}$ in the ideal execution, where the simulator $\mathcal{S}$ plays the role of the honest parties in Π against an internal $\mathcal{A}$ and interacts as the corrupt parties with the functionality $\mathcal{F}$.

Active security in the UC model

While the extensive proofs of Sections 8.2.2 and 8.3.2 are for semi-honest adversaries, we mention here another kind of adversarial behaviour considered in the UC model: *malicious* behaviour, also called *active* behaviour. In this case, the adversary is no longer restricted to the instructions of the protocol and can instead deviate arbitrarily and compute its own messages on behalf of the corrupt parties. Security against this kind of adversary, called *malicious, or active, security*, is much harder to achieve. Indeed in this case, to prove the indistinguishability of the real (or hybrid) world from the ideal world, the simulator must be able to extract inputs from $\mathcal{A}$'s arbitrary messages in order to cause the functionality to output coherent messages to the honest parties.

8.2 OT from three-pass message transfer

This first OT protocol is inspired by the two-party Shamir three-pass protocol for secure message transmission shown in Figure 8.3 (ignoring the elements in square brackets), also known as the Massey-Omura encryption scheme [102]. Here, Alice's input is a message g together with a secret mask a and Bob's input is another secret

$$\begin{tikzcd}
\text{Alice } (g_0, [g_1], a) & \text{Bob } ([\mathsf{c}], b) \\
(g_0)^a, [(g_1)^a] \arrow[r, "{g_0^a, [g_1^a]}"] & {} \\
{} & (g^a_{[\mathsf{c}]})^b \arrow[l, "g^{ab}_{[\mathsf{c}]}"'] \\
(g^{ab}_{[\mathsf{c}]})^{1/a} \arrow[r, "g^b_{[\mathsf{c}]}"] & (g^b_{[\mathsf{c}]})^{1/b} = g_{[\mathsf{c}]}
\end{tikzcd}$$

Figure 8.3: The Shamir three-pass protocol and its OT variant.

mask b. To transmit g, Alice first sends g^a to Bob who replies by masking it as g^{ab}. Now Alice removes her mask and replies with $g^{ab/a} = g^b$ to Bob who then inverts b and recovers g.

This protocol can be modified to yield an OT protocol as shown in Figure 8.3 (including the elements in square brackets); this was proposed by Wu, Zhang and Wang [127]. This protocol can be seen as being based on a key transport protocol which is expanded to achieve the requirements of oblivious transfer. In contrast, our second construction will later take a key exchange protocol and turn it into an OT protocol.

Alice, acting as sender, now has two inputs g_0 and g_1 and masks both with a to send g_0^a, g_1^a to Bob, the receiver. In addition to his mask b, Bob now also has a choice bit $\mathsf{c} \in \{0, 1\}$ and he replies to Alice with $(g_\mathsf{c}^a)^b$. They then continue as before until Bob recovers g_c.

While the security of the Shamir three-pass protocol holds against external adversaries due to the masking of the message, the OT protocol also needs to provide security guarantees against internal adversaries, namely Alice or Bob. The intuition for security in this case is that the mask a cannot be deduced from either g_0^a or g_1^a and therefore the first message hides both of Alice's inputs from Bob. Also when Bob applies his own mask to one of the two messages, this hides his input bit c from Alice who doesn't know b.

To instantiate this protocol for *general* semi-commutative masking schemes, Alice needs to be able to invert the mask $1/a$ on g_c^{ab} without knowing g_c. While this is easy in the discrete logarithm case, it is not possible in general. This is due to the subtle fact that in the definition of the inverse of a mask $\mu \in M$: $\forall x \in X$, $\forall r \in R_x$, $\exists \mu^{-1} \in M :: \mu^{-1}(\mu(r)) \in R_x$, the μ^{-1} is specified *after* μ and r and may therefore depend on $\mu(r)$. While such an μ^{-1} is required to exist *for all* $r \in R_x$ for a given μ and x, if may be different for each value of r or x. Therefore in a general semi-commutative masking scheme, the $1/a$ mask may be different depending on whether it needs to be

$$\begin{array}{ccc} \text{Alice } (g_0, g_1; a) & & \text{Bob } (g_0, g_1; \mathsf{c}; b) \\ & \xleftarrow{\quad g_\mathsf{c}^b \quad} & (g_\mathsf{c})^b \\ (g_0)^a, (g_1)^a, (g_\mathsf{c}^b)^a & \xrightarrow{\quad g_\mathsf{c}^{ab} \quad} & (g_\mathsf{c}^{ab})^{1/b} = g_\mathsf{c}^a \end{array}$$

Figure 8.4: Sketch of the final Shamir three-pass OT protocol.

applied to g_0^a or g_1^a. As the aim of the protocol is to hide which of these two values was chosen by Bob, Alice lacks some information to compute her un-masking properly.

We therefore modify the OT protocol to remove this operation. In our new (discrete logarithm based) variant, the elements g_0 and g_1 are common to both parties. Rather using a to send g_0^a, g_1^a to Bob (the receiver), Alice (the sender) does not go first. Instead, Bob first communicates his masked choice g_c^b, and then Alice applies her mask a and replies with g_c^{ab}. At that moment, she also computes g_0^a, g_1^a internally. She then uses these internal values to derive two symmetric keys k_0 and k_1. Those are used to encrypt Alice's actual OT inputs m_0 and m_1 as two ciphertexts ct_0 and ct_1 which she sends alongside g_c^{ab}. This allows Bob to recover g_c^a and hence decrypt ct_c to recover m_c.

As she no longer communicates one of g_0 or g_1 to Bob this is no longer exactly a *message transport* protocol. Instead, it can be seen as a *randomness transport* protocol where Alice communicates her random mask a applied to Bob's choice g_c. As g_0 and g_1 are now established once and re-used for every instance of the protocol, this allows the flows to have only two passes rather than three. Figure 8.4 abstracts the symmetric encryption and only shows the flows that lead to Bob receiving the value g_c^a.

8.2.1 Construction

We now formally define the first OT protocol from semi-commutative invertible masking schemes. Let $\mathcal{M} = \{X, R_X, [M_A, M_B, M_C]\}$ be a semi-commutative masking structure with three masking sets; let $E = (\mathtt{KGen}, \mathtt{Enc}, \mathtt{Dec})$ be a symmetric encryption scheme with key space $\mathcal{K}$ and let $\mathcal{F}^{\mathtt{RO}}$ be an instance of the RO ideal functionality with domain $\mathcal{D} = X$ and range $\mathcal{R} = \mathcal{K}$. We assume that random sampling from masking sets $M_i, i \in \{A, B, C\}$, evaluation of masks, evaluation of $\mathtt{Enc}$, $\mathtt{Dec}$, and inversion in M_i are all efficient operations for the masking structure $\mathcal{M}$ and for the symmetric encryption scheme E. The protocol $\Pi^1_{\mathtt{OT}}$ is formally defined in Figure 8.5.

As described above, the idea of the protocol is that both the sender, P_S, and

Protocol $\Pi^1_{\mathtt{OT}}$

Parameters: length n of the sender's input strings; masking structure $\mathcal{M}$; symmetric encryption scheme E.
Common inputs: Arbitrary $x_0 \neq x_1 \in X$ together with $r_0 \in R_{x_0}, r_1 \in R_{x_1}$ are shared and re-used for every instance of the protocol; an instance of the random oracle ideal functionality $\mathcal{F}^{\mathtt{RO}} : \{0,1\}^{1^\lambda} \to \mathcal{K}$.
Sender's Input: m_0, m_1.
Receiver's Input: $\mathsf{c} \in \{0,1\}$.

Receiver 1

1. Sample $\beta \leftarrow\$ M_B$ uniformly at random.
2. Compute $r_{\mathsf{c}}^{\beta} \coloneqq \beta(r_{\mathsf{c}})$ and $\beta^{-1} \in M_B$.
3. Send r_{c}^{β} to P_S.

Sender 1

1. Sample $\alpha \leftarrow\$ M_A$ and compute $r_{\mathsf{b}}^{\alpha} \coloneqq \alpha(r_{\mathsf{b}}) \in R_{x_{\mathsf{b}}^{\alpha}}$, $\mathsf{b} \in \{0,1\}$
2. For $\mathsf{b} \in \{0,1\}$, call $\mathcal{F}^{\mathtt{RO}}$ twice on input x_{b}^{α} obtaining k_{b}, and compute $\mathsf{ct}_{\mathsf{b}} \leftarrow \mathtt{Enc}(\mathsf{k}_{\mathsf{b}}, m_{\mathsf{b}})$
3. Compute $r_{\mathsf{c}}^{\alpha\beta} \coloneqq \alpha(r_{\mathsf{c}}^{\beta})$
4. Send $(r_{\mathsf{c}}^{\alpha\beta}, \mathsf{ct}_0, \mathsf{ct}_1)$ to P_R.

Receiver 2

1. Compute $r_{\mathsf{c}}^{\alpha} \coloneqq \beta^{-1}(r_{\mathsf{c}}^{\alpha\beta})$ and $\mathsf{k}_R \coloneqq \mathcal{F}^{\mathtt{RO}}(x_{\mathsf{c}}^{\alpha})$ where $r_{\mathsf{c}}^{\alpha} \in R_{x_{\mathsf{c}}^{\alpha}}$.
2. Return $m_{\mathsf{c}} \coloneqq \mathtt{Dec}(\mathsf{k}_R, \mathsf{ct}_{\mathsf{c}})$.

Figure 8.5: The protocol $\Pi^1_{\mathtt{OT}}$ for realizing $\mathcal{F}^{\mathtt{OT}}$ from semi-commutative masking.

receiver, P_R, have as common input arbitrary elements $x_0 \neq x_1 \in X$ along with representations $r_0 \in R_{x_0}, r_1 \in R_{x_1}$. In the first pass, P_R takes a random mask $\beta \in M_B$ and sends $r_{\mathsf{c}}^{\beta} = \beta(r_{\mathsf{c}})$ to P_S, where c is its choice bit. In the second pass, P_S samples a random mask $\alpha \in M_A$ and computes $r_0^{\alpha} = \alpha(r_0)$ and $r_1^{\alpha} = \alpha(r_1)$. These elements uniquely determine $x_{\mathsf{b}}^{\alpha} \in X, \mathsf{b} \in \{0,1\}$. Thus the sender can compute two private keys $\mathsf{k}_{\mathsf{b}}, \mathsf{b} \in \{0,1\}$, by invoking twice the random oracle functionality $\mathcal{F}^{\mathtt{RO}}$ on input x_{b}^{α}, and encrypt its input messages m_0, m_1 accordingly. P_S then sends the ciphertexts $\mathsf{ct}_{\mathsf{b}} \leftarrow \mathtt{Enc}(\mathsf{k}_{\mathsf{b}}, m_{\mathsf{b}}), \mathsf{b} \in \{0,1\}$, and $r_{\mathsf{c}}^{\alpha\beta} = \alpha(r_{\mathsf{c}}^{\beta})$ to P_R. The receiver has now all the information needed to recover the message m_{c} corresponding to its choice bit: it can apply the inverse β^{-1} to $r_{\mathsf{c}}^{\alpha\beta}$ using the semi-commutativity of $\mathcal{M}$, so that

$$\beta^{-1}(r_{\mathsf{c}}^{\alpha\beta}) = \beta^{-1}(\alpha(r_{\mathsf{c}}^{\beta})) = \beta^{-1}(\alpha(\beta(r_{\mathsf{c}}))) \in R_{x_{\mathsf{c}}^{\alpha}},$$

and recover $\mathsf{k}_{\mathsf{c}} = \mathcal{F}^{\mathtt{RO}}(x_{\mathsf{c}}^{\alpha})$. This easily implies correctness of the scheme.

8.2.2 UC-security against semi-honest adversaries

The security of protocol $\Pi_{\mathtt{OT}}^{1}$ of figure 8.5 is given by the following theorem.

Theorem 8.1. *The protocol $\Pi_{\mathtt{OT}}^{1}$ securely UC-realizes the functionality $\mathcal{F}^{\mathtt{OT}}$ of Figure 8.1 in the $\mathcal{F}^{\mathtt{RO}}$-hybrid model for semi-honest adversaries and static corruptions, under the assumption that E is IND-CPA-secure, that $\mathcal{M}$ is IND-Mask-secure and that the ParallelEither$^{\mathcal{M}}$ problem is hard.*

Proof. We prove that there exists a PPT simulator $\mathcal{S}$, with access to an ideal functionality $\mathcal{F}^{\mathtt{OT}}$, which simulates the adversary's view. We divide the proof according to the selection of the corrupt parties.

Corrupt receiver and corrupt sender. As both parties are corrupt, the simulator $\mathcal{S}$ may read their inputs from their internal state and use those to create a perfect simulation of the transcript and of the parties' internal states. It presents this simulation to its internal copy of $\mathcal{A}$, together with a perfect simulation of $\mathcal{F}^{\mathtt{RO}}$, with which it is then able to perfectly answer $\mathcal{E}$'s queries by forwarding them to $\mathcal{A}$ and returning the responses. Since it knows all of the inputs, it forwards them to $\mathcal{F}^{\mathtt{OT}}$ at the right moment to ensure that the dummy corrupt parties return the correct output to $\mathcal{E}$.

Corrupt receiver and honest sender. We formally describe the simulator $\mathcal{S}_{R^*}$ in Figure 8.6. We show that for every semi-honest adversary $\mathcal{A}$ who corrupts P_R and any environment $\mathcal{E}$, we have that $\mathsf{HYBRID}_{\Pi_{\mathtt{OT}}^{1},\mathcal{A},\mathcal{E}}^{\mathcal{F}^{\mathtt{RO}}} \overset{\mathrm{c}}{\approx} \mathsf{IDEAL}_{\mathcal{F}^{\mathtt{OT}},\mathcal{S}_{R^*},\mathcal{E}}$, by proceeding via a sequence of hybrid simulators.

We begin with a hybrid $\mathcal{H}_0$ which knows the inputs of the honest sender. As it learns the input c of the corrupt receiver as soon as it is activated by $\mathcal{E}$, it is able to present a perfect simulation of the protocol. The second hybrid $\mathcal{H}_1$ samples $\mathsf{k}_{1-\mathsf{c}} \leftarrow\!\!\$\, \mathcal{K}$ at random. Instead, $\mathcal{F}^{\mathtt{RO}}(x_{1-\mathsf{c}}^{\alpha})$ will be set to a random value if it is queried during the execution.

Claim 8.2. Any environment $\mathcal{E}$ that can distinguish the simulations of $\mathcal{H}_1$ and $\mathcal{H}_0$ can be used to solve the ParallelEither problem for $\mathcal{M}$.

Simulator $\mathcal{S}_{R^*}$

1. Throughout the execution, $\mathcal{S}_{R^*}$ simulates the $\mathcal{F}^{\mathtt{RO}}$ by answering every new query with a random value from $\mathcal{K}$ and maintaining a list of past queries to answer repeated queries consistently. As in the previous case, it presents the simulated transcript and corrupt receiver state as computed below to $\mathcal{A}$ and uses it to answer queries from $\mathcal{E}$.

2. When $\mathcal{E}$ activates the corrupt Receiver, its private input c is visible by $\mathcal{S}_{R^*}$ which can then compute r_{c}^{β} to perfectly simulate Receiver 1.

3. To simulate Sender 1, $\mathcal{S}_{R^*}$ samples $\alpha \leftarrow\!\!\$\, M_A$ and computes $r_{\mathsf{c}}^{\alpha\beta}$ honestly.
Since m_{c} appears on the corrupt Receiver's output tape, the simulator computes k_{c} and ct_{c} as prescribed by the protocol.
However, since $\mathcal{S}_{R^*}$ does not learn the honest input $m_{1-\mathsf{c}}$, it samples $\mathsf{k}_{1-\mathsf{c}} \leftarrow\!\!\$\, \mathcal{K}$ at random and sets $\mathsf{ct}_{1-\mathsf{c}} \leftarrow \mathtt{Enc}(\mathsf{k}_{1-\mathsf{c}}, m)$ for an arbitrary plaintext m.

4. If $\mathcal{E}$ queries either $\mathcal{F}^{\mathtt{RO}}(x_{\mathsf{c}}^{\alpha})$ before activating Sender 1, then $\mathcal{S}_{R^*}$ aborts the simulation by returning $\perp$ to $\mathcal{E}$.

5. Finally, $\mathcal{S}_{R^*}$ finishes the protocol as prescribed.

Figure 8.6: The simulator $\mathcal{S}_{R^*}$ of Theorem 8.1.

Proof. Such an environment is capable of distinguishing if and only if it queries $\mathcal{F}^{\mathtt{RO}}(x_{1-\sigma}^{\alpha})$. Let $\mathcal{A}$ be an adversary for which $\mathcal{E}$ distinguishes between $\mathcal{H}_0$ and $\mathcal{H}_1$ with some advantage ϵ; we use this to build a reduction $\mathcal{B}$ against the $\mathsf{ParallelEither}$ problem for $\mathcal{M}$ which proceeds as follows. Upon receiving a challenge (C, A, r, r_x, r_y), $C \neq A$, $r_x = (r)$ and $r_y = \alpha(r)$, $\mathcal{B}$ simulates an execution with $\mathcal{E}$ as follows:

1. First set $r_0 := r$ and $r_1 := r_x$, and set $r_{\mathsf{c}}^{\alpha} := r_y$.

2. Set the keys and ciphertexts as $\mathcal{H}_1$ does and simulate Receiver 1 honestly.

3. Since $\mathcal{B}$ does not know the $\alpha \in M_A$ such that $r_y = \alpha(r)$, it cannot compute $r_{\mathsf{c}}^{\alpha\beta} = \alpha(r_{\mathsf{c}}^{\beta})$ honestly. Instead, it sets $r_{\mathsf{c}}^{\alpha\beta} = \beta(r_y)$. This can be done since it is simulating the internal value β. This remains consistent with the protocol as we still have that $\beta^{-1}(r_{\mathsf{c}}^{\alpha\beta}) \in R_y$ and $r_y = (r_{\mathsf{c}}^{\alpha}) \in R_y$, as set at the beginning of $\mathcal{B}$.

4. If $\mathsf{c} = 0$, then $r_{1-\mathsf{c}} = \gamma(r_{\mathsf{c}})$ and therefore $\gamma(r_{\mathsf{c}}^{\alpha}) = \gamma(r_y) \in R_{x_{1-\mathsf{c}}^{\alpha}}$. If instead $\mathsf{c} = 1$, then $r_{1-\mathsf{c}}^{\alpha} = \gamma^{-1}(r_{\mathsf{c}}^{\alpha}) = \gamma^{-1}(r_y)$.
Therefore we see that, independently of c, if $\mathcal{E}$ queries $\mathcal{F}^{\mathtt{RO}}(x_{1-\mathsf{c}}^{\alpha})$, then one of the solutions to the $\mathsf{ParallelEither}$ problem is present on the list of past queries.

Simulator $\mathcal{S}_{S^*}$

1. $\mathcal{S}_{S^*}$ simulates $\mathcal{F}^{\mathtt{RO}}$ consistently and presents the state and transcript computed as follows to an internal copy of $\mathcal{A}$ to reply to the queries from $\mathcal{E}$.
2. As it does not know the c of the honest receiver, $\mathcal{S}_{S^*}$ proceeds by setting $\mathsf{c} = 0$ internally which remains out of the view of $\mathcal{A}$. It then samples $\beta \leftarrow_\$ M_B$ and sets $r_{\mathsf{c}}^{\beta} = \beta(r_0)$ consistently.
3. As it knows the inputs m_0, m_1 of the corrupt sender, $\mathcal{S}_{S^*}$ computes Sender 1 consistently with r_{c}^{β} using the correct plaintexts.
4. Finally, $\mathcal{S}_{S^*}$ finishes the protocol as prescribed.

Figure 8.7: The simulator $\mathcal{S}_{S^*}$ of Theorem 8.1.

When $\mathcal{E}$ terminates, $\mathcal{B}$ therefore returns a random entry on the list of random oracle queries. If $\mathcal{E}$ has advantage ϵ in distinguishing between $\mathcal{H}_1$ and $\mathcal{H}_0$, $\mathcal{B}$ then has an advantage ϵ/q_H in solving the $\mathsf{ParallelEither}$ problem, where q_H denotes the number of queries to $\mathcal{F}^{\mathtt{RO}}$ made during the execution. $\square$

The final hybrid $\mathcal{H}_2$ replaces $m_{1-\mathsf{c}}$ by an arbitrary plaintext m in the computation of $\mathsf{ct}_{1-\mathsf{c}}$. This removes the last occurrence of $m_{1-\mathsf{c}}$ in the simulator and we have that $\mathcal{H}_2$ is identical to the original $\mathcal{S}_{R^*}$.

Claim 8.3. Any environment $\mathcal{E}$ that can distinguish between a simulation of $\mathcal{H}_2$ and of $\mathcal{H}_1$ with advantage ϵ can be used to break the $\mathsf{IND\text{-}CPA}$ property of E with advantage at least ϵ.

Proof. We can build an adversary against the $\mathsf{IND\text{-}CPA}$ property of E by querying the challenger for a ciphertext of either m or $m_{1-\mathsf{c}}$. This reduction emulates either $\mathcal{H}_2$ or $\mathcal{H}_1$ perfectly as $\mathsf{k}_{1-\mathsf{c}}$ is not accessible to $\mathcal{E}$ and therefore not required by $\mathcal{H}_2$ or $\mathcal{H}_1$ at any point. $\square$

Under the assumption that E is $\mathsf{IND\text{-}CPA}$-secure and that the $\mathsf{ParallelEither}$ problem is hard for $\mathcal{M}$, we have that the simulation generated by $\mathcal{S}_{R^*}$ is indistinguishable from a real world execution, for any environment $\mathcal{E}$. This concludes the proof that $\mathsf{HYBRID}^{\mathcal{F}^{\mathtt{RO}}}_{\Pi^1_{\mathtt{OT}},\mathcal{A},\mathcal{E}} \overset{c}{\approx} \mathsf{IDEAL}_{\mathcal{F}^{\mathtt{OT}},\mathcal{S}_{R^*},\mathcal{E}}$.

Honest receiver and corrupt sender. We formally describe the simulator $\mathcal{S}_{S^*}$ in Figure 8.7 We show that for every semi-honest adversary $\mathcal{A}$ who corrupts P_S and any environment $\mathcal{E}$, it holds that $\mathsf{HYBRID}^{\mathcal{F}^{\mathtt{RO}}}_{\Pi^1_{\mathtt{OT}},\mathcal{A},\mathcal{E}} \overset{c}{\approx} \mathsf{IDEAL}_{\mathcal{F}^{\mathtt{OT}},\mathcal{S}_{S^*},\mathcal{E}}$.

The simulation of $\mathcal{S}_{S^*}$ is not a perfect simulation of a real world execution only if the honest receiver had actually received input $\mathsf{c} = 1$ from $\mathcal{E}$.

Claim 8.4. In that case, any environment that can distinguish between a simulation of $\mathcal{S}_{S^*}$ and the real world with advantage ϵ can be used to break the $\mathsf{IND\text{-}Mask}$ security of $\mathcal{M}$ with advantage at least ϵ.

Proof. We build a reduction $\mathcal{B}$ against the $\mathsf{IND\text{-}Mask}$ experiment as follows. The reduction first selects an arbitrary r as well as two masks $\gamma_0, \gamma_1 \in M_C$ and sends $(r, \gamma_0, \gamma_1, B)$ to the $\mathsf{IND\text{-}Mask}$ experiment. Upon receiving $\tilde{r}$, $\mathcal{B}$ then begins the distinguishing experiment with $\mathcal{E}$ by setting $r_0 = \gamma_0(r), r_1 = \gamma_1(r)$ and returning $r_{\mathsf{c}}^{\beta} = \tilde{r}$ to the adversary when $\mathcal{E}$ activates Receiver 1. Not knowing β is not a problem for the simulation as the receiver is honest and therefore $\mathcal{B}$ does not need to simulate its state to $\mathcal{A}$. This is a perfect simulation of either the real world or of $\mathcal{S}_{S^*}$ as either r_1 or r_0 is used by the $\mathsf{IND\text{-}Mask}$ experiment in the computation of r_{c}^{β}. Thus if $\mathcal{E}$ distinguishes between the two, then $\mathcal{B}$ can distinguish the hidden bit of the $\mathsf{IND\text{-}Mask}$ experiment. □

Honest receiver and honest sender. In this final case, the simulator $\mathcal{S}$ chooses arbitrary inputs $m_0 = m_1 = m$ and $\mathsf{c} = 1$ and simulates a transcript to $\mathcal{A}$. If an environment $\mathcal{E}$ is capable of distinguishing this simulation from a real execution of the protocol then this implies that it is able to extract information regarding the arbitrary inputs used by $\mathcal{S}$. However the previous two cases show that, even with the additional information of the corrupted party's internal state, any environment is not able to identify a simulation that does not have any information on the honest party's inputs. By combining techniques from both cases above, we can therefore show that the simulation of $\mathcal{S}$ is indistinguishable from a real world execution under the assumption that $\mathcal{S}$ is $\mathsf{IND\text{-}CPA}$-secure, that $\mathcal{M}$ is $\mathsf{IND\text{-}Mask}$-secure and that the $\mathsf{ParallelEither}^{\mathcal{M}}$ problem is hard. □

Note 8.5. Protocol $\Pi_{\mathtt{OT}}^{1}$ only requires the third masking set M_C as a proof artefact and that only two sets would be sufficient to execute the protocol.

8.2.3 Actively secure two-round OT

In this section we make use of a recent transformation by Döttling et al. which compiles two-round OT protocols with minimal security into a fully actively secure two-round

OT protocol [63]. We first present the definitions of security required of the starting protocol and then prove that our protocol satisfies them.

Additional OT security notions

A two-round OT protocol with public setup consists of four algorithms $\Pi_{\mathtt{OT}} = (\mathtt{Setup}, \mathtt{OT}_1, \mathtt{OT}_2, \mathtt{OT}_3)$ such that:

- $\mathtt{Setup}(1^\lambda)$ generates a public input pin.
- $\mathtt{OT}_1(\mathsf{pin}, \mathsf{c})$, where $\mathsf{c} \in \{0,1\}$ is the P_R choice bit, outputs $(\mathsf{state}, \mathsf{ot\_}P_R)$
- $\mathtt{OT}_2(\mathsf{pin}, \mathsf{ot\_}P_R, m_0, m_1)$, where m_0, m_1 are the sender's input messages, outputs $\mathsf{ot\_}P_S$
- $\mathtt{OT}_3(\mathsf{state}, \mathsf{ot\_}P_S)$ outputs m_{c}

First we need to recall some security notions for the receiver P_R and the sender P_S. The first definition states that P_S should not learn anything about P_R's choice bit c.

Definition 8.6 (Receiver's indistinguishability security [63]). For any probabilistic poly-nomial-time adversary $\mathcal{A}$ an OT protocol $\Pi_{\mathtt{OT}} = (\mathtt{Setup}, \mathtt{OT}_1, \mathtt{OT}_2, \mathtt{OT}_3)$ has *receiver indistinguishability* if there exists a negligible function $\mathsf{negl}(\lambda)$ such that:

$$|\Pr[\mathcal{A}(\mathsf{pin}, \mathtt{OT}_1(\mathsf{pin}, 0)) = 1] - \Pr[\mathcal{A}(\mathsf{pin}, \mathtt{OT}_1(\mathsf{pin}, 1)) = 1]| = \mathsf{negl}(\lambda)\,,$$

where pin is the public output of the setup phase.

The next definition concerns the security of the sender; it states that P_R cannot compute both secret values y_0 and y_1 used by $\mathtt{OT}_2$ to protect m_0 and m_1, but not necessarily in the same experiment.

Definition 8.7 (Sender's search security [63]). Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary where $\mathcal{A}_2$ outputs a string y^*. Consider the following experiment $\mathbf{Exp}^{\mathsf{pin},\rho,w}_{\mathsf{sOT}}(\mathcal{A})$, indexed by a pin, random coins $\rho \in \{0,1\}^\lambda$ and a bit $w \in \{0,1\}$.

1. Run $(\mathsf{ot\_}P_R, \mathsf{state}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{pin}; \rho)$.
2. Compute $(\mathsf{ot\_}P_S, \mathsf{y}_0, \mathsf{y}_1) \leftarrow_{\$} \mathtt{OT}_2(\mathsf{pin}, \mathsf{ot\_}P_R)$.
3. Run $\mathsf{y}^* \leftarrow \mathcal{A}_2(\mathsf{state}, \mathsf{ot\_}P_S, w)$ and output 1 iff $\mathsf{y}^* = \mathsf{y}_w$.

We say that $\mathcal{A}$ breaks a scheme's *Sender's search (sOT) security* if there exists a non-negligible function ϵ such that

$$\Pr_{\mathsf{pin},\rho}\left[\Pr[\mathbf{Exp}_{\mathsf{sOT}}^{\mathsf{pin},\rho,0}(\mathcal{A}) = 1] > \epsilon \text{ and } \Pr[\mathbf{Exp}_{\mathsf{sOT}}^{\mathsf{pin},\rho,1}(\mathcal{A}) = 1] > \epsilon\right] > \epsilon,$$

where $\mathsf{pin} \leftarrow_{\$} \mathtt{Setup}$ and $\rho \leftarrow_{\$} \{0,1\}^{\lambda}$.

Two-round OT with active UC-security

We provide an intermediary result which enables the use of the general compiler from [63] to transform Π^1_{OT} into an actively secure two-round OT protocol. First we introduce and discuss a new security assumption derived from the Parallel problem but more suited to active adversaries. Then we show that our protocol satisfies the security notions of Definitions 8.6 and 8.7. Finally, by applying the general transformations from sOT to UC OT described in [63], we obtain a fully UC-secure two-round OT protocol. We note that we are able to remove the random oracle from our protocol to achieve sOT security; therefore the resulting OT protocol requires only the CRS. We define our new computational problem as follows.

Definition 8.8 (ParallelDouble)**.** Given $(i, j, r, r_{x_0}, r_{x_1}, r_y)$ with the promise that $i \neq j$ and that $r_{x_\mathsf{b}} = \mu_{x_\mathsf{b}}(r)$, $\mathsf{b} \in \{0,1\}$ and $r_y = \mu_y(r)$ for random $\mu_{x_\mathsf{b}} \leftarrow_{\$} M_i$ and $\mu_y \leftarrow_{\$} M_j$, and given a *one-time* access to an oracle $\mathcal{O}_y$ which, when given $r \in R$ returns $\mu_y(r)$, compute $z_0, z_1 \in X$ such that *both* $\mu_{x_\mathsf{b}}(r_y) \in R_{z_\mathsf{b}}$.

The instantiation of this problem in the discrete logarithm case is, when given (g, g^a, g^b, g^c) and a one-time access to an exponentiation-by-c oracle, to return both g^{ac} and g^{bc}. For practical efficiency, it is also desirable that g^a and g^b remain constant across multiple instances of the ParallelDouble problem, with only g^c being randomly sampled in each instance. This version of the problem is similar to the one-more static CDH problem where an adversary has to successfully compute one more CDH challenge than it was able to ask from a helper oracle [31].

Security of the Π^1_{OT} protocol. We then prove that protocol Π^1_{OT} achieves Receiver's indistinguishability and Sender's search security.

Proposition 8.9. *The protocol Π^1_{OT} in Figure 8.5 satisfies computational receiver's indistinguishability security and sender's sOT security under the assumption that $\mathcal{M}$ is IND-Mask-secure and that the ParallelDouble$^{\mathcal{M}}$ problem is hard.*

Proof. Receiver's indistinguishability follows from the $\mathsf{IND\text{-}Mask}$-security assumption. By setting the public inputs r_0 and r_1 in $\Pi^1_{\mathtt{OT}}$ as they are computed in the $\mathsf{IND\text{-}Mask}$ experiment, the random mask μ is distributed in the same way as the mask β in $\mathtt{OT}_1$. Therefore if an adversary breaks the receiver's indistinguishability for $\Pi^1_{\mathtt{OT}}$, this can be reduced to a solution to the $\mathsf{IND\text{-}Mask}$ problem.

Sender's search security. To prove sOT security for $\Pi^1_{\mathtt{OT}}$ we assume the existence of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and a non-negligible ϵ such that

$$\Pr_{\mathsf{pin},\rho}[\Pr[\mathbf{Exp}^{\mathsf{pin},\rho,0}_{\mathsf{sOT}}(\mathcal{A}) = 1] > \epsilon \text{ and } \Pr[\mathbf{Exp}^{\mathsf{pin},\rho,1}_{\mathsf{sOT}}(\mathcal{A}) = 1] > \epsilon] > \epsilon,$$

and we build a reduction $\mathcal{B}$ that is given a $\mathsf{ParallelDouble}$ challenge $(i, j, r, r_{x_0}, r_{x_1}, r_y)$ with access to an oracle $\mathcal{O}_y$ (Definition 8.8). Instead of running $\mathtt{Setup}$ to generate r_0 and r_1, $\mathcal{B}$ sets $r_0 \leftarrow r_{x_0}$ and $r_1 \leftarrow r_{x_1}$; also $\mathcal{B}$ samples $\rho \leftarrow\!\!\$ \{0,1\}^\lambda$. As this ensures that pin is distributed identically to the output of $\mathtt{Setup}$, pin and ρ are good for $\mathcal{A}$ with probability at least ϵ.

After $\mathcal{B}$ runs $\mathcal{A}_1$, which outputs $(\mathsf{ot\_}P_R, \mathsf{state})$, it queries the oracle to obtain $\mathsf{ot\_}P_{S,0} \leftarrow \mathcal{O}_y(\mathsf{ot\_}P_R)$. It also computes $\mathsf{ot\_}P_{S,1} \leftarrow \mu(\mathsf{ot\_}P_{S,0})$ for a random $\mu \in M_k$ with $i \neq k \neq j$; it also computes μ^{-1}. Then, for $w \in \{0,1\}$, $\mathcal{B}$ runs $\mathsf{y}^*_w \leftarrow \mathcal{A}_2(\mathsf{state}, \mathsf{ot\_}P_{S,w}, w)$ and updates $\mathsf{y}^*_1 \leftarrow \mu^{-1}(\mathsf{y}^*_1)$. Finally $\mathcal{B}$ returns y^*_0 and the updated y^*_1 as the $\mathsf{ParallelDouble}$ answer.

Since $\Pr[\mathbf{Exp}^{\mathsf{pin},\rho,0}_{\mathsf{sOT}}(\mathcal{A}) = 1] > \epsilon$ and $\Pr[\mathbf{Exp}^{\mathsf{pin},\rho,1}_{\mathsf{sOT}}(\mathcal{A}) = 1] > \epsilon$, with probability ϵ^2, $\mathcal{A}_2$ is successful for both inputs $(\mathsf{state}, \mathsf{ot\_}P_{S,0}, 0)$ and $(\mathsf{state}, \mathsf{ot\_}P_{S,1}, 1)$ as the two messages are made independent by $\mathcal{B}$'s addition of μ. If this happens, then y^*_0 is exactly one of the answers, and the update of y^*_1 by $\mathcal{B}$ removes the extra mask μ and means that y^*_1 is then the other answer to the $\mathsf{ParallelDouble}$ problem. Hence $\mathcal{B}$ is successful with probability at least ϵ^3. □

Theorem 8.10. *Under the assumption that $\mathcal{M}$ is IND-Mask-secure and that the ParallelDouble$^{\mathcal{M}}$ problem is hard, there exists a 2-round UC-secure OT protocol constructed from $\Pi^1_{\mathtt{OT}}$.*

Proof. This follows from the transformations and results of [63, Theorems 8, 9, 11, 12, 14, 19 and 21]. □

Corollary 8.11. *By instantiating the semi-commutative masking scheme, there exists an actively secure 2-round OT protocol based on supersingular isogenies.*

$$\begin{tikzcd}
\text{Alice } (g; a, d_0, d_1) & & \text{Bob } (\mathsf{c}; b) \\
{} \arrow[rr, "{g^a, g^{d_0}, g^{d_1}}"] & & {} \\
g^a, g^{d_0}, g^{d_1} & & \mathsf{k}_R = (g^a)^b \arrow[ll, "(g^{d_\mathsf{c}})^b"'] \\
\mathsf{k}_\mathsf{b} = (g^{d_\mathsf{c} b})^{a/d_\mathsf{b}} & & \\
{} \arrow[rr, dashed, "\{m_\mathsf{b}\}_{\mathsf{k}_\mathsf{b}}"] & & {}
\end{tikzcd}$$

Figure 8.8: Sketch of the OT protocol derived from the key exchange protocol.

We remark here that the isogeny-based OT protocols proposed by Vitse [125], while being semantically secure against malicious adversaries, require three rounds of communication; this implies that they cannot be transformed to achieve two-round OT with fully UC-security using the work of Döttling et al.

8.3 OT from key-exchange

The second OT protocol is inspired by the OT protocol of Chou and Orlandi [43] in that it uses an underlying key exchange mechanism and then transforms it to achieve oblivious transfer. The problems that have emerged in their construction [44, Section 1.1] do not arise when considering passive adversaries so we do not address them here.

Again we motivate our proposed OT protocol by looking at the discrete logarithm variant. Here, Alice's inputs are two messages m_0, m_1 and an ephemeral mask a and Bob's is another mask b together with his choice c. To agree on the key under which the selected message will be encrypted, Alice sends g^a to Bob who derives the decryption key g^{ab}. But Bob cannot simply reply with g^b, since Alice would then not know which of m_0 or m_1 to encrypt. Instead, Alice communicates two random masks g^{d_0} and g^{d_1} to allow Bob to make a selection. By masking $(g^{d_\mathsf{c}})^b$ with the same b as he uses to derive the key, Bob obliviously communicates his choice and his mask to Alice which is then able to derive two keys (by unmasking d_b and then adding her mask a) of which only one will be share with Bob. We sketch the protocol flows in Figure 8.8. The protocol is intuitively secure as Alice cannot deduce b from Bob's message and Bob cannot deduce the key $\mathsf{k}_{1-\mathsf{c}}$ as he is not able to recover $d_{1-\mathsf{c}}^{-1}$ from Alice's first message.

8.3.1 Construction

We now formally define the second OT protocol from semi-commutative invertible masking schemes. Let $\mathcal{M} = \{X, R_X, [M_A, M_B, M_C]\}$ be a semi-commutative masking structure; let $E = (\texttt{KGen}, \texttt{Enc}, \texttt{Dec})$ be a symmetric encryption scheme with key space $\mathcal{K}$ and let $\mathcal{F}^{\texttt{RO}}$ be an instance of the RO ideal functionality with domain $\mathcal{D} = X$ and range $\mathcal{R} = \mathcal{K}$. We formally describe the protocol $\Pi^2_{\texttt{OT}}$ in Figure 8.9. Protocol $\Pi^2_{\texttt{OT}}$ makes use of random sampling from M_i, evaluation of masks, evaluation of H, evaluation of Enc, Dec, as well as membership and equality testing in X and in the ciphertext space and inversion in M_i. All these operations are assumed to be efficient for the masking structure $\mathcal{M}$ and for the symmetric scheme E.

Because $\mathcal{M}$ is semi-commutative, we see that $\alpha(\gamma_{\mathsf{b}}^{-1}(r_{\mathsf{c}}^{\beta})) = \alpha(\gamma_{\mathsf{b}}^{-1}(\beta(\gamma_{\mathsf{c}}(r)))) \in R_{x^{\alpha\beta}}$ if and only if $\mathsf{b} = \mathsf{c}$. This shows that, if both parties execute the protocol honestly, $\mathsf{k}_R = \mathsf{k}_{\mathsf{c}}$ and hence P_R recovers the correct message m_{c}.

8.3.2 UC-security against semi-honest adversaries

The security of protocol $\Pi^2_{\texttt{OT}}$ of figure 8.9 is given by the following theorem.

Theorem 8.12. *The protocol $\Pi^2_{\texttt{OT}}$ securely UC-realizes the functionality $\mathcal{F}^{\texttt{OT}}$ of Figure 8.1 in the $\mathcal{F}^{\texttt{RO}}$-hybrid model for semi-honest adversaries and static corruptions, under the assumption that E is IND-CPA-secure, that $\mathcal{M}$ is IND-Mask-secure and that the ParallelBoth$^{\mathcal{M}}$ problem is hard.*

Proof. We prove that there exists a PPT simulator $\mathcal{S}$, with access to an ideal functionality $\mathcal{F}^{\texttt{OT}}$, which simulates the adversary's view. We divide the proof according to the selection of the corrupt parties.

Corrupt receiver and corrupt sender. As both parties are corrupt, the simulator $\mathcal{S}$ may read their inputs from their internal state and use those to create a perfect simulation of the transcript and of the parties' internal states. It presents this simulation to its internal copy of $\mathcal{A}$, together with an perfect simulation of $\mathcal{F}^{\texttt{RO}}$, with which it is then able to perfectly answer $\mathcal{E}$'s queries by forwarding them to $\mathcal{A}$ and returning the responses. Since it knows all of the inputs, it forwards them to $\mathcal{F}^{\texttt{OT}}$ at the right moment to ensure that the dummy corrupt parties return the correct output to $\mathcal{E}$.

Corrupt receiver and honest sender. We formally describe the simulator $\mathcal{S}_{R^*}$ in Figure 8.10. We show that for every semi-honest adversary $\mathcal{A}$ who corrupts P_R and any

Protocol $\Pi^2_{\mathtt{OT}}$

PARAMETERS: length n of the sender's input strings; masking structure $\mathcal{M}$; symmetric encryption scheme E.
COMMON INPUTS: An arbitrary $x \in X$ together with $r \in R_x$ is shared and re-used for every instance of the protocol; an instance of the random oracle ideal functionality $\mathcal{F}^{\mathtt{RO}} : \{0,1\}^\lambda \to \mathcal{K}$.
SENDER'S INPUT: m_0, m_1.
RECEIVER'S INPUT: $\mathsf{c} \in \{0,1\}$.

Sender 1

1. Sample $\alpha \leftarrow\!\!\$\, M_A, \gamma_{\mathsf{b}} \leftarrow\!\!\$\, M_C$, $\mathsf{b} \in \{0,1\}$, at random, and compute $r^\alpha := \alpha(r)$, $r_{\mathsf{b}} := \gamma_{\mathsf{b}}(r)$ and $\gamma_{\mathsf{b}}^{-1} \in M_C$.
2. Send (r^α, r_0, r_1) to P_R

Receiver 1

1. Sample $\beta \leftarrow\!\!\$\, M_B$ uniformly at random.
2. Compute $r_{\mathsf{c}}^\beta := \beta(r_{\mathsf{c}})$.
3. Compute $\mathsf{k}_{\mathsf{c}} := \mathcal{F}^{\mathtt{RO}}(x^{\alpha\beta})$ where $\beta(r^\alpha) \in R_{x^{\alpha\beta}}$.
4. Send r_{c}^β to P_S.

Sender 2

1. For $\mathsf{b} \in \{0,1\}$, compute $\mathsf{k}_{\mathsf{b}} := \mathcal{F}^{\mathtt{RO}}(x_{\mathsf{b}})$ where $\alpha(\gamma_{\mathsf{b}}^{-1}(r_{\mathsf{c}}^\beta)) \in R_{x_{\mathsf{b}}}$.
2. For $\mathsf{b} \in \{0,1\}$, compute $\mathsf{ct}_{\mathsf{b}} \leftarrow \mathtt{Enc}(\mathsf{k}_{\mathsf{b}}, m_{\mathsf{b}})$.
3. Send $(\mathsf{ct}_0, \mathsf{ct}_1)$ to P_R.

Receiver 2 Return $m_{\mathsf{c}} := \mathtt{Dec}(\mathsf{k}_{\mathsf{c}}, \mathsf{ct}_{\mathsf{c}})$.

Figure 8.9: The protocol $\Pi^2_{\mathtt{OT}}$ for realizing $\mathcal{F}^{\mathtt{OT}}$ from semi-commutative masking.

environment $\mathcal{E}$, it holds that

$$\mathsf{HYBRID}^{\mathcal{F}^{\mathtt{RO}}}_{\Pi^2_{\mathtt{OT}},\mathcal{A},\mathcal{E}} \overset{c}{\approx} \mathsf{IDEAL}_{\mathcal{F}^{\mathtt{OT}},\mathcal{S}_{R^*},\mathcal{E}},$$

by proceeding via a sequence of hybrid simulators, going from the real execution to the ideal execution, defined as follows.

The first hybrid $\mathcal{H}_0$ knows the inputs of the honest sender and is therefore able to compute $\mathsf{ct}_{1-\mathsf{c}}$ honestly using the correct random oracle query to obtain the key. This is then a perfect simulation of a real-world execution.

Simulator $\mathcal{S}_{R^*}$

1. $\mathcal{S}_{R^*}$ simulates $\mathcal{F}^{\texttt{RO}}$ consistently and presents the state and transcript computed as follows to an internal copy of $\mathcal{A}$ to reply to the queries from $\mathcal{E}$.
2. When $\mathcal{E}$ activates the honest sender, $\mathcal{S}_{R^*}$ computes Sender 1 honestly to send (r^α, r_0, r_1) to $\mathcal{A}$.
3. When $\mathcal{E}$ activates the corrupt receiver, its private input c is visible by $\mathcal{S}_{R^*}$ which can then compute r_{c}^{β} to perfectly simulate Receiver 1.
4. To simulate Sender 2, since m_{c} appears on the corrupt receiver's output tape, the simulator computes k_{c} and ct_{c} as prescribed by the protocol.
   However, since $\mathcal{S}_{R^*}$ does not learn the honest input $m_{1-\mathsf{c}}$, it samples $\mathsf{k}_{1-\mathsf{c}} \leftarrow\$ \mathcal{K}$ at random and sets $\mathsf{ct}_{1-\mathsf{c}} \leftarrow \mathtt{Enc}(\mathsf{k}_{1-\mathsf{c}}, m)$ for an arbitrary plaintext m.
5. If $\mathcal{E}$ queries either $\mathcal{F}^{\texttt{RO}}(x_{\mathsf{b}})$ before activating Receiver 1, then $\mathcal{S}_{R^*}$ aborts the simulation by returning $\perp$ to $\mathcal{E}$.
6. Finally, $\mathcal{S}_{R^*}$ finishes the protocol as prescribed.

Figure 8.10: The simulator $\mathcal{S}_{R^*}$ of Theorem 8.12.

The second hybrid $\mathcal{H}_1$ samples $\mathsf{k}_{1-\mathsf{c}} \leftarrow\$ \mathcal{K}$ at random and does not query the random oracle on $x_{1-\mathsf{c}}$ where $\alpha(\gamma_{1-\mathsf{c}}^{-1}(r_{\mathsf{c}}^{\beta})) \in R_{x_{1-\mathsf{c}}}$.

Claim 8.13. Any environment $\mathcal{E}$ that distinguishes an interaction with $\mathcal{H}_1$ from one with $\mathcal{H}_0$ with advantage ϵ can be used to solve the $\mathsf{ParallelBoth}$ problem for $\mathcal{M}$ with advantage at least ϵ/q_H where q_H denotes the number of queries made by $\mathcal{E}$ to the random oracle.

Proof. Such an environment is capable of distinguishing if and only if it submits the query for $\mathsf{k}_{1-\mathsf{c}}$ to the random oracle. We use this to build a reduction $\mathcal{D}$ against the $\mathsf{ParallelBoth}$ problem for $\mathcal{M}$ which proceeds as follows.

Upon receiving a challenge $(C, A, r, r_{x_0}, r_{x_1}, r_y)$, $\mathcal{D}$ first sets $z^\alpha := r_y$ and $z_i := r_{x_i}$ to simulate Sender 1 and then samples $\beta \leftarrow\$ M_B$ to compute Receiver 1 perfectly upon activation of P_R^* which reveals c.

Since it now does not know the $\alpha \in M_A$ such that $r^\alpha = \alpha(r)$, $\mathcal{B}$ computes k_{c} from $\beta(r^\alpha)$ which it can do as it knows β and which yields the correct $x^{\alpha\beta}$ as the masks commute. For the other key, it sets $\mathsf{k}_{1-\mathsf{c}} \leftarrow\$ \mathcal{K}$ as $\mathcal{S}_1$ would. It then returns the ciphertexts encrypting m_0, m_1 under these keys.

When $\mathcal{E}$ terminates, $\mathcal{B}$ selects a random entry on the list of random oracle queries and applies β^{-1}. The un-selected key $\mathsf{k}_{1-\mathsf{c}}$ is the hash of the element of X represented by $\alpha(\gamma_{1-\mathsf{c}}^{-1}(\beta(\gamma_{\mathsf{c}}(r))))$ where $\gamma_i \in M_C$ is such that $r_{x_i} = \gamma_i(r)$. So by applying β^{-1}, $\mathcal{B}$ obtains exactly a representative one of the solutions to the $\mathsf{ParallelBoth}$ problem as long as it selected the correct entry on the hash list. If $\mathcal{E}$ has advantage ϵ in distinguishing between $\mathcal{H}_1$ and $\mathcal{H}_0$, $\mathcal{B}$ then has an advantage ϵ/q_H in solving the $\mathsf{ParallelBoth}$ problem. □

The final hybrid $\mathcal{H}_2$ replaces $m_{1-\mathsf{c}}$ by an arbitrary plaintext m in the computation of $\mathsf{ct}_{1-\mathsf{c}}$. This removes the last occurrence of $m_{1-\mathsf{c}}$ in the simulator and we have that $\mathcal{H}_2$ is identical to $\mathcal{S}_{R^*}$.

Claim 8.14. Any environment $\mathcal{E}$ that can distinguish between a simulation of $\mathcal{H}_2$ and of $\mathcal{H}_1$ with advantage ϵ can be used to break the $\mathsf{IND\text{-}CPA}$ property of E with advantage at least ϵ.

Proof. We can build an adversary against the $\mathsf{IND\text{-}CPA}$ property of E by querying the challenger for a ciphertext of either m or $m_{1-\mathsf{c}}$. This reduction emulates either $\mathcal{H}_2$ or $\mathcal{H}_1$ perfectly as $\mathsf{k}_{1-\mathsf{c}}$ is not accessible to $\mathcal{E}$ and therefore not required by $\mathcal{S}_2$ or $\mathcal{S}_1$ at any point. □

Under the assumption that E is $\mathsf{IND\text{-}CPA}$-secure and that the $\mathsf{ParallelBoth}$ problem is hard for $\mathcal{M}$, we have that the simulation generated by $\mathcal{S}_{R^*}$ is indistinguishable from a real world execution, for any environment $\mathcal{E}$. This concludes the proof that $\mathsf{HYBRID}^{\mathcal{F}^{\mathtt{RO}}}_{\Pi^2_{\mathtt{OT}},\mathcal{A},\mathcal{E}} \overset{\mathrm{c}}{\approx} \mathsf{IDEAL}_{\mathcal{F}^{\mathtt{OT}},\mathcal{S}_{R^*},\mathcal{E}}$.

Honest receiver and corrupt sender. We formally describe the simulator $\mathcal{S}_{S^*}$ in Figure 8.11. We show that for every semi-honest adversary $\mathcal{A}$ who corrupts P_S and any environment $\mathcal{E}$, it holds that

$$\mathsf{HYBRID}^{\mathcal{F}^{\mathtt{RO}}}_{\Pi^2_{\mathtt{OT}},\mathcal{A},\mathcal{E}} \overset{\mathrm{c}}{\approx} \mathsf{IDEAL}_{\mathcal{F}^{\mathtt{OT}},\mathcal{S}_{S^*},\mathcal{E}}.$$

The simulation of $\mathcal{S}_{S^*}$ is not a perfect simulation of a real world execution only if the honest receiver had actually received input $\mathsf{c} = 1$ from $\mathcal{E}$.

Claim 8.15. In that case, any environment that can distinguish between a simulation of $\mathcal{S}_{S^*}$ and the real world with advantage ϵ can be used to break the $\mathsf{IND\text{-}Mask}$ security of $\mathcal{M}$ with advantage at least ϵ.

Proof. We build a reduction $\mathcal{B}$ against the $\mathsf{IND\text{-}Mask}$ experiment as follows. It first simulates Sender 1 as in the protocol and sends $(r, \gamma_0, \gamma_1, B)$ to the $\mathsf{IND\text{-}Mask}$ experiment. Upon receiving $\tilde{r}$, $\mathcal{B}$ then returns $r^{\beta}_{\mathsf{c}} = \tilde{r}$ to the adversary when $\mathcal{E}$ activates

Simulator $\mathcal{S}_{S^*}$

1. $\mathcal{S}_{S^*}$ simulates $\mathcal{F}^{\mathtt{RO}}$ consistently and presents the state and transcript computed as follows to an internal copy of $\mathcal{A}$ to reply to the queries from $\mathcal{E}$.
2. When $\mathcal{E}$ activates the corrupt sender, $\mathcal{S}_{S^*}$ computes Sender 1 honestly to send (r^{α}, r_0, r_1) to $\mathcal{A}$.
3. As it does not know the c of the honest receiver, $\mathcal{S}_{S^*}$ proceeds by setting $\mathsf{c} = 0$ internally which remains out of the view of $\mathcal{A}$. It then samples $\beta \leftarrow\!\!\$\, M_B$ and sets $r_0^{\beta} = \beta(r_0)$ consistently. It also computes k_0 accordingly.
4. As it knows the inputs m_0, m_1 of the corrupt Sender, $\mathcal{S}_{S^*}$ computes Sender 2 consistently with r_{c}^{β} using the correct plaintexts.
5. Finally, $\mathcal{S}_{S^*}$ finishes the protocol as prescribed.

Figure 8.11: The simulator $\mathcal{S}_{S^*}$ of Theorem 8.12.

Receiver 1. Not knowing β is not a problem for the simulation as the receiver is honest and therefore $\mathcal{B}$ does not need to simulate its state to $\mathcal{A}$. This is a perfect simulation of either the real world or of $\mathcal{S}_{S^*}$ as either r_1 or r_0 is used by the $\mathsf{IND\text{-}Mask}$ experiment in the computation of r_{c}^{β}. Thus if $\mathcal{E}$ distinguishes between the two, then $\mathcal{B}$ can distinguish the hidden bit of the $\mathsf{IND\text{-}Mask}$ experiment. □

Honest receiver and honest sender. In this final case, the simulator $\mathcal{S}$ chooses arbitrary inputs $m_0 = m_1 = m$ and $\mathsf{c} = 1$ and simulates a transcript to $\mathcal{A}$ using those. If an environment $\mathcal{E}$ is capable of distinguishing this simulation from a real execution of the protocol then this implies that is is able to extract information regarding the arbitrary inputs used by $\mathcal{S}$. However the previous two cases show that, even with the additional information of the corrupted party's internal state, any environment is not able to identify a simulation that does not have any information the honest party's inputs. By combining techniques from both cases above, we can therefore show that the simulation of $\mathcal{S}$ is indistinguishable from a real world execution under the assumption that $\mathcal{S}$ is $\mathsf{IND\text{-}CPA}$-secure, that $\mathcal{M}$ is $\mathsf{IND\text{-}Mask}$-secure and that the $\mathsf{ParallelBoth}^{\mathcal{M}}$ problem is hard.

This completes the proof that for any $\mathcal{A}$ there exists a $\mathcal{S}$ such that, for any $\mathcal{E}$,

$$\mathsf{HYBRID}^{\mathcal{F}^{\mathtt{RO}}}_{\Pi^2_{\mathtt{OT}},\mathcal{A},\mathcal{E}} \overset{c}{\approx} \mathsf{IDEAL}_{\mathcal{F}^{\mathtt{OT}},\mathcal{S},\mathcal{E}}. \qquad \square$$

Part III

Signature schemes from MPC-in-the-head

Chapter 9

Using AES in Picnic signatures

In this chapter we present the use of the advanced encryption standard (AES) block cipher for signature schemes built in the MPCitH paradigm. We first describe this paradigm and illustrate it with the example of the Picnic signature scheme [38] adapted for arithmetic circuits over fields. We then recall the inner workings of the AES cipher, present mechanisms to optimize its use with the MPCitH paradigm and estimate signature sizes resulting from these mechanisms.

Except where specified, the contributions in this chapter are selected from the work of L. De Meyer[1], E. Orsini[2], N. P. Smart[3] and this author; it was published in the proceedings of *Selected Areas in Cryptography, SAC 2019* and presented at the conference by this author in Waterloo, ON, Canada [55]. The ideas were the joint work of all four authors; L. De Meyer contributed the majority of Section 9.2.1; this author contributed the majority of the rest of Section 9.2 with inputs from E. Orsini and N. P. Smart.

9.1 MPC-in-the-head and the Picnic signature scheme

We present here the paradigm of MPCitH for non-interactive zero-knowledge proofs of knowledge (NIZKPoKs) and the Picnic signature scheme adapted for arithmetic circuits.

[1]KU Leuven, Belgium
[2]University of Bristol, U.K., later KU Leuven, Belgium.
[3]University of Bristol, U.K., later KU Leuven, Belgium.

9.1.1 Efficient NIZKPoK in the MPC-in-the-head paradigm

In 2007, Ishai et al. [80] showed how to use any MPC protocol to construct a zero-knowledge (ZK) proof for an arbitrary NP relation R. The high level idea is the following: zero knowledge can be seen as a special function evaluation, and hence as a two-party computation between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, with common input the statement x, and $\mathcal{P}$'s private input w, which is a witness to the assertion that x belongs to a given NP language $\mathcal{L}$. The function to be evaluated by $\mathcal{P}$ and $\mathcal{V}$ is $f_x(w) \iff (x, w) \in R$, which checks whether w is a valid witness. The verifier $\mathcal{V}$ will accept the proof provided by $\mathcal{P}$ if $f_x(w) = 1$.

In the MPCitH paradigm, the prover $\mathcal{P}$ simulates an n-party MPC protocol Π in "its head". They first sample n random values $w^{(1)}, \dots, w^{(n)}$, subject to the condition that $\sum_{i\in[n]} w^{(i)} = w$. These are a sharing of the witness w and serve as private inputs to the simulated parties. The prover then emulates the evaluation of the protocol Π for the function $f_x(w^{(1)} + \cdots + w^{(n)})$ by choosing uniformly random coins $r^{(i)}$ for each simulated party P_i, $i \in [n]$ and executing the protocol between them. Note that, once the inputs and random coins are fixed, for each round j of communication of the protocol Π and for each party P_i, the messages sent by P_i at round j are deterministically specified as a function of the internal state of P_i, i.e. P_i's private inputs $w^{(i)}$, randomness $r^{(i)}$, and the messages $\{\mathsf{msg}^{(i)}\}$ that P_i received in previous rounds. The set of the state and all messages received by party P_i during the execution of the protocol constitutes the view of P_i, denoted as view_{P_i}.

After this simulated evaluation of Π, the prover $\mathcal{P}$ sends commitments to the view of each party to $\mathcal{V}$. At this point, the verifier "corrupts" a random subset of parties, challenging the prover to open their committed views. They then finally verify that views were computed correctly from the perspective of the corrupt parties; this is achieved by checking that they are all consistent with the protocol Π and with each other (i.e. that messages sent by one party appear identical in the intended recipient's view). To obtain desired soundness guarantees it is often necessary to repeat the above procedure several times in parallel with freshly simulated executions.

Using the MPCitH paradigm, some recent works [5, 39, 74] have constructed efficient NIZKPoKs for Boolean circuits which form the basis of iterations of the Picnic signature scheme [39, 86]. In particular, the work of Katz et al. [86] constructs an honest-verifier zero-knowledge proof by instantiating the MPCitH paradigm using an MPC protocol designed in the preprocessing model. We now describe their protocol adapted for arithmetic circuits.

9.1.2 Arithmetic MPCitH in the preprocessing model

Let $\mathbb{F}$ denote a finite field. The honest-verifier zero-knowledge (HVZK) protocol introduced here provides a proof of knowledge of a witness $\boldsymbol{w} \in \mathbb{F}^\iota$ such that $C(\boldsymbol{w}) = \boldsymbol{y} \in \mathbb{F}^o$, for a given circuit $C : \mathbb{F}^\iota \to \mathbb{F}^o$ and output $\boldsymbol{y}$.

As the underlying protocol Π_C is designed in the preprocessing model (and therefore has a preprocessing phase followed by an online phase), the HVZK proof also happens in two stages. In the first stage, the prover $\mathcal{P}$ commits to a number of preprocessing executions (which consist only of input-independent randomness). Then the verifier $\mathcal{V}$ requests that some, but not all, of these are opened, and checks that they are correct and consistent computations.

In the second stage, the prover uses the unopened preprocessing material to execute parallel and independent executions of the online phase of the MPC protocol Π_C. It then commits to these executions and, for each of them, is challenged by the verifier to open a random selection of $n-1$ parties' views. By receiving these views, $\mathcal{V}$ is able to perform checks to ensure that the prover did not falsify the executions of Π_C.

Since we make use of an arithmetic circuit C over a field $\mathbb{F}$, it is composed of addition and multiplication gates operating on values in $\mathbb{F}$. This is a generalization of [86] as the work of Katz et al. only considers circuits operating on bits with XOR and AND gates. The online evaluation of C is done by a SPDZ-like [54] protocol, simplified as we only require security against semi-honest adversaries (notably, we use broadcast as the only communication channel).

This protocol is executed by n parties. For every value $x \in \mathbb{F}$ in the arithmetic circuit C, the protocol uses an additive secret sharing $\langle x \rangle$ which denotes the sharing $(x^{(1)}, \ldots, x^{(n)})$, such that $\sum_{i=1}^{n} x^{(i)} = x$ and every party P_i holds $x^{(i)}$.

To reconstruct, or "open", a shared value $\langle x \rangle$, each party P_i broadcasts its share $x^{(i)}$ and each party P_j can then reconstruct $x = \sum_{i=1}^{n} x^{(i)}$. With such a secret sharing, the following operations can be performed locally, i.e. without communication between parties:

- *Addition with public constant:* To compute $\langle z \rangle \leftarrow a + \langle x \rangle$ given $\langle x \rangle$ and a public value a, party P_1 sets his share to be $z^{(1)} \leftarrow x^{(1)} + a$ and every other party P_i $(i \neq 1)$ sets $z^{(i)} := x^{(i)}$.

- *Multiplication by public constant:* To compute $\langle z \rangle \leftarrow a \cdot \langle x \rangle$ given $\langle x \rangle$ and a, every party P_i sets his share to be $z^{(i)} \leftarrow a \cdot x^{(i)}$.

- *Addition of two shared values:* To compute $\langle z \rangle \leftarrow \langle x + y \rangle$ given $\langle x \rangle$ and $\langle y \rangle$, every party P_i sets his share to be $z^{(i)} \leftarrow x^{(i)} + y^{(i)}$.

However, computing the multiplication of two shared values, i.e. $\langle z \rangle \leftarrow \langle x \cdot y \rangle$ given $\langle x \rangle$ and $\langle y \rangle$, cannot be done locally; it requires both preprocessing material and communication during the online phase. Namely, given a precomputed triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$, such that $c = a \cdot b$, the parties can compute $\langle z \rangle$ as follows:

1. Locally compute $\langle \alpha \rangle := \langle x - a \rangle$ and $\langle \beta \rangle := \langle y - b \rangle$.
2. Open α and β.
3. Locally compute $\langle z \rangle = \langle c \rangle - \alpha \cdot \langle b \rangle - \beta \cdot \langle a \rangle + \alpha \cdot \beta$.

This technique is due to Beaver [11] and is easily checked to be correct:

$$\begin{aligned} z &= c - \alpha \cdot b - \beta \cdot a + \alpha \cdot \beta \\ &= a \cdot b - (x - a) \cdot b - (y - b) \cdot a + (x - a) \cdot (y - b) \\ &= x \cdot y. \end{aligned}$$

Before we describe the HVZK protocol of [86], we present the execution of both phases of Π_C that the prover will have to simulate; namely the preprocessing and online phases.

1) Preprocessing phase. The only preprocessing required is the generation of random multiplication triples $\{(\langle a_m \rangle, \langle b_m \rangle, \langle c_m \rangle)\}_{m \in [\mathsf{mult}]}$, where mult is the number of multiplication gates in C. As noted in [86] for its preprocessing computation, for each triple, the i-th share of $\langle a \rangle$ and $\langle b \rangle$ is uniform and therefore can be generated by party P_i applying a PRG to a short random seed $\mathsf{sd}^{(i)}$. Each share of $\langle c \rangle$ can also be generated in that way, but then the actual generated value $\tilde{c} = \sum_{i=1}^{n} c^{(i)}$ would not equal $a \cdot b$ with very high probability. Instead, for each triple $(\langle a_m \rangle, \langle b_m \rangle, \langle c_m \rangle)$, party P_n is given a "correction value" $\Delta_m = a \cdot b - \sum_{i=1}^{n-1} c^{(i)}$ and directly sets $c^{(n)} \leftarrow \Delta_m$.

In summary, the outcome of the preprocessing is that every party P_i is given a λ-bit seed $\mathsf{sd}^{(i)} \in \{0,1\}^\lambda$ and P_n is also given mult values $\{\Delta_m\}$ denoted by $\mathsf{aux}^{(n)}$. This information is called the *state* of party P_i and is denoted by $\mathsf{state}^{(i)}$.

2) Online phase. The online computation can itself be divided into three components: *input distribution*, *computation* and *output reconstruction.*

Unlike for real MPC, the input $\boldsymbol{w} = (w_1, \ldots, w_\iota)$ is global and must be distributed by the prover, outside of the view of any $n - 1$ parties. To do so, each party P_i

PARAMETERS: Let $H : \{0,1\}^* \to \{0,1\}^{2\lambda}$ and G be collision-resistant hash functions and `Com` be a commitment scheme. Note G is only used in the non-interactive variant of the protocol.
Other parameters are T and τ, that indicate the total number of preprocessing executions emulated by $\mathcal{P}$ and the number of online executions, respectively.

INPUTS: Both parties hold a description of C over $\mathbb{F}$, the value $\boldsymbol{y} \in \mathbb{F}^o$. The prover $\mathcal{P}$ also holds $\boldsymbol{w} \in \mathbb{F}^\iota$ such that $C(\boldsymbol{w}) = \boldsymbol{y}$.

Round 1: *Protocol execution emulation.*

1. For each $t \in [T]$, $\mathcal{P}$ emulates the preprocessing phase as follows:
   (a) Sample a uniform master seed $\mathsf{sd}_t \in \{0,1\}^\lambda$ and use it to generate $\mathsf{sd}_t^{(1)}, \ldots, \mathsf{sd}_t^{(n)} \in \{0,1\}^\lambda$ and $r_t^{(1)}, \ldots, r_t^{(n)} \in \{0,1\}^\lambda$ used in the commitments.
   (b) For each multiplication gate $m \in [\mathsf{mult}]$:
      i. Use $\mathsf{sd}_j^{(i)}$ to sample $a_{t,m}^{(i)}, b_{t,m}^{(i)}$ and also $c_{t,m}^{(i)}$ for $i = 1, \ldots, n-1$.
      ii. Compute $a_{t,m} = \sum_{i=1}^n a_{t,m}^{(i)}$ and $b_{t,m}$ similarly.
      iii. Compute the offset $\Delta_{t,m} = a_{t,m} \cdot b_{t,m} - \sum_{i=1}^{n-1} c_{t,m}^{(i)}$.
   (c) Set $\mathsf{aux}_t^{(n)} = (\Delta_{t,m})_{m\in[\mathsf{mult}]}$.
   (d) Set $\mathsf{state}_t^{(i)} = \mathsf{sd}_t^{(i)}$ for $i \in [n-1]$ and set $\mathsf{state}_t^{(n)} = (\mathsf{sd}_t^{(n)}, \mathsf{aux}_t^{(n)})$.
   (e) For each $i \in [n]$, compute $\gamma_{\mathsf{s},t}^{(i)} = \mathtt{Com}(\mathsf{state}_t^{(i)}, r_t^{(i)})$.
   (f) Compute $h_{\mathsf{s},t} = H(\gamma_{\mathsf{s},t}^{(1)}, \ldots, \gamma_{\mathsf{s},t}^{(n)})$.
2. For each $t \in [T]$, given $\boldsymbol{w} = (w_1, \ldots, w_\iota)$, $\mathcal{P}$ emulates the online phase as follows:
   (a) Generate the input offsets for each $j \in [\iota]$:
      i. Use $\mathsf{sd}_t^{(i)}$ to sample $w_{t,j}^{(i)}$ for $i \in [n]$.
      ii. Compute $\tilde{w}_{t,j} = \sum_{i=1}^n w_{t,j}^{(i)}$.
      iii. Compute $\Lambda_{t,j} = w_j - \tilde{w}_{t,j}$.
   (b) Compute C by proceeding through the gates in topological order. For each party P_i, record each broadcast message in $\mathsf{msg}_t^{(i)}$.
   (c) Compute $h_{\mathsf{m},t} = H(\{\Lambda_{t,j}\}_{j\in[\iota]}, \mathsf{msg}_t^{(1)}, \ldots, \mathsf{msg}_t^{(n)})$.
3. Compute $h_{\mathsf{s}} = H(h_{\mathsf{s},1}, \ldots, h_{\mathsf{s},T}), h_{\mathsf{m}} = H(h_{\mathsf{m},1}, \ldots, h_{\mathsf{m},T})$ and send $h^* = H(h_{\mathsf{s}}, h_{\mathsf{m}})$ to $\mathcal{V}$.

Figure 9.1: 3-round HVZK proof—part 1.

uses $\mathsf{sd}^{(i)}$ to generate a random share $w_j^{(i)}$, for $j \in [\iota]$. Identically to the generation of multiplication triple, a correction value Λ_j must then be computed so that $w_j = \Lambda_j + \sum_{i=1}^n w_j^{(i)}$. The values $(w_j^{(1)}, \ldots, w_j^{(n)}, \Lambda_j)$ now constitute an n-out-of-$(n+1)$ sharing of w_j and therefore it is safe for $\mathcal{P}$ to communicate the Λ_j values to $\mathcal{V}$ in

Round 2: *Challenge.*
$\mathcal{V}$ challenges $\mathcal{P}$ on the executions indexed by a set $\mathcal{T} \subset [T]$, with $|\mathcal{T}| = \tau$. For each of these, it chooses a party that remains honest, i.e. it chooses a vector $(i_\mathsf{t})_{\mathsf{t}\in\mathcal{T}} \in [n]^\tau$. It then sends $(\mathcal{T}, (i_\mathsf{t})_{\mathsf{t}\in\mathcal{T}})$ to $\mathcal{P}$.
In the NIZKPoK variant, $\mathcal{P}$ locally computes $(\mathcal{T}, (i_\mathsf{t})_{\mathsf{t}\in\mathcal{T}}) = G(h^*)$.

Round 3: *Opening.*
$\mathcal{P}$ sends $(\mathsf{sd}_\mathsf{t}, h_{\mathsf{m},\mathsf{t}})_{\mathsf{t}\in\overline{\mathcal{T}}}$ and $((\mathsf{state}_\mathsf{t}^{(i)}, r_\mathsf{t}^{(i)})_{i\in[n]\setminus\{i_\mathsf{t}\}}, \gamma_{\mathsf{s},\mathsf{t}}^{(i_\mathsf{t})}, \{\Lambda_{\mathsf{t},j}\}_{j\in[\iota]}, \mathsf{msg}_\mathsf{t}^{(i_\mathsf{t})})_{\mathsf{t}\in\mathcal{T}}$, to $\mathcal{V}$.

Verification:

1. For each $\mathsf{t} \in \mathcal{T}$, use $(\mathsf{state}_\mathsf{t}^{(i)}, r_\mathsf{t}^{(i)})_{i\in[n]\setminus\{i_\mathsf{t}\}}$ to reconstruct $\gamma_{\mathsf{s},\mathsf{t}}^{(i)}$ for $i \in [n]\setminus\{i_\mathsf{t}\}$. Then compute $h'_{\mathsf{s},\mathsf{t}} = H(\gamma_{\mathsf{s},\mathsf{t}}^{(1)}, \dots, \gamma_{\mathsf{s},\mathsf{t}}^{(n)})$ using $\gamma_{\mathsf{s},\mathsf{t}}^{(i_\mathsf{t})}$ sent by $\mathcal{P}$.
2. For $\mathsf{t} \in \overline{\mathcal{T}}$, use sd_t to compute $h'_{\mathsf{s},\mathsf{t}}$ as an honest $\mathcal{P}$ would.
3. Then compute $h'_\mathsf{s} = H(h'_{\mathsf{s},1}, \dots, h'_{\mathsf{s},T})$.
4. For each $\mathsf{t} \in \mathcal{T}$, use $\{\mathsf{state}_\mathsf{t}^{(i)}\}_{i\in[n]\setminus\{i_\mathsf{t}\}}$, $(\Lambda_{\mathsf{t},j})_{j\in[\iota]}$ and $\mathsf{msg}_\mathsf{t}^{(i_\mathsf{t})}$ to recompute the online phase. Check that the output reconstruction yields the correct value of $\boldsymbol{y}$, and compute $h'_{\mathsf{m},\mathsf{t}} = H(\{\Lambda_{\mathsf{t},j}\}_{j\in[\iota]}, \mathsf{msg}_\mathsf{t}^{(1)}, \dots, \mathsf{msg}_\mathsf{t}^{(n)})$.
5. Then compute $h'_\mathsf{m} = H(h'_{\mathsf{m},1}, \dots, h'_{\mathsf{m},T})$ using $(h'_{\mathsf{m},\mathsf{t}})_{\mathsf{t}\in\overline{\mathcal{T}}}$ sent by $\mathcal{P}$.
6. Check that $h^* \overset{?}{=} H(h'_\mathsf{s}, h'_\mathsf{m})$.

Figure 9.2: 3-round HVZK proof—part 2.

addition to $n-1$ of the seeds $\mathsf{sd}^{(i)}$. Before the computation begins, we let P_n be the one to set $w_j^{(n)} \leftarrow w_j^{(n)} + \Lambda_j$, for $j \in [\iota]$.

Next, the parties compute the intermediary values in the computation of $C(\boldsymbol{w})$. Whenever multiplication gate m is encountered, each party P_i recalls their share of the next unused triple $a_m^{(i)}, b_m^{(i)}, c_m^{(i)}$ from the preprocessing phase. This yields the triple $(\langle a_m\rangle, \langle b_m\rangle, \langle c_m\rangle)$ which can then be used to compute multiplication gate m. Whenever P_i needs to sample a random value, it does so using $\mathsf{sd}^{(i)}$.

Once the parties have computed the shared output values $\langle y_1\rangle, \dots, \langle y_o\rangle$, they jointly reconstruct $\boldsymbol{y} = C(\boldsymbol{w})$. To do so, they open each value y_j, for $j \in [o]$, by having each party broadcast its share $y_j^{(i)}$

As noted above during the description of the MPCitH paradigm, this online phase is entirely deterministic. In particular, the next broadcast message of party P_i at a given time in Π_C depends only on $\mathsf{state}^{(i)}$ and the messages received by P_i so far. Denoting by A the set of "corrupt" parties, this implies that when the verifier $\mathcal{V}$ checks the views of these $n-1$ parties in the execution of Π_C, they only need to be given the $n-1$ states of parties in A together with the messages broadcast by the unopened party during each multiplication gate; they can then recompute the internal values of $C(\boldsymbol{w})$ since they can infer the broadcast messages sent by the opened parties. This means that instead of sending view_{P_i}, for each $i \in A$, $\mathcal{P}$ only needs to send $\mathsf{view}_A = \{\mathsf{state}^{(i)}\}_{i \in A} \cup \{\mathsf{msg}^{(j)}\}_{j \notin A}$, where $\{\mathsf{msg}^{(j)}\}_{j \notin A}$ are the messages sent by the "honest" party.

The HVZKPoK protocol

In Figures 9.1 and 9.2, we give a slightly modified version of the 3-round protocol presented by Katz et al. [86] to compute their proof of knowledge, so that it can use our Π_C protocol for an arithmetic circuit C over $\mathbb{F}$. We refer the reader to the original paper [86] for a more in-depth explanation of the protocol. We adopt their optimization of performing a more general cut-and-choose by having the prover run T independent preprocessing phases and the verifier choosing τ of them to be used for independent online phases. For each of these online phases, $\mathcal{V}$ challenges $\mathcal{P}$ on $n-1$ parties which it then checks, together with the $T-\tau$ preprocessing phases that were not used.

From the 3-round protocol, it is possible to obtain a non-interactive ZKPoK by applying the Fiat-Shamir [68] transform, as indicated in the description of **Round 2** in Figure 9.2. The system we obtain in this way consists of two stages, Prove and $\mathsf{VerifyProof}$.

- $\mathsf{Prove}(\boldsymbol{y}, \boldsymbol{w})$ takes as input $(\boldsymbol{y}, \boldsymbol{w})$ and consists of **Round 1, 2** and **3** of Fig. 9.1 and 9.2; i.e. $\mathcal{P}$ computes the first round message and then *locally* computes the challenge by hashing this message using a collision-resistant hash function G. The resulting proof, σ, consists of the concatenation of the first-round message and the response to the challenge.

- $\mathsf{VerifyProof}(\boldsymbol{y}, \sigma)$ does the **Verification** step (Fig. 9.2) and returns $b \in \{0, 1\}$.

To obtain a formula for our proof size estimates we analyze each of the elements communicated by the prover in Rounds 1 and 3. We present and incorporate the optimizations discussed in [86] and thus obtain a formula very close to theirs. At the end of Round 1, $\mathcal{P}$ sends $h^* = H(h_{\mathsf{s}}, h_{\mathsf{m}})$ to $\mathcal{V}$, which contributes 2λ bits to the proof size. In Round 3, $\mathcal{P}$ sends two sets of elements; the first corresponds to the $T-\tau$ opened preprocessing executions and the second corresponds to the τ executions of the online phase of the protocol. By generating the master seeds as the leaves of a binary tree expanding from a single root, all-but-τ of the seeds can be sent by only sending $\tau \cdot \log(T/\tau)$ elements, each of λ bits. By computing the hash h_{m} in a similar way, i.e. as the root of a tree where the $h_{\mathsf{m,t}}$ values are the leaves, then it also suffices to send at most $\tau \cdot \log(T/\tau)$ values, each of 2λ bits. This implies that the opening of the $T-\tau$ preprocessing executions adds $\tau \cdot \log(T/\tau) \cdot 3\lambda$ to the proof size. To open the τ online executions, $\mathcal{P}$ sends, for each execution, the commitment $\gamma_{\mathsf{s,t}}^{(i_t)}$ of 2λ bits, the input correction values $\{\Lambda_{\mathsf{t},j}\}_{j\in[\iota]}$ of size $|\boldsymbol{w}|$ bits, the messages of the unopened party of size $|\mathsf{msg}^{(i)}|$ and the states of the openened parties. To reduce the communication here, it is observed in [86] that there is sufficient entropy contained in $\mathsf{state}_t^{(i)}$ to not require a separate randomness $r_t^{(i)}$. Combining this with a tree-like structure thus reduces the communication to only $\lambda \cdot \log n$ for the states. In the worst case when party P_n is opened, the prover also has to send the auxiliary information of size $|\mathsf{aux}^{(n)}|$. This results in the following estimate for the proof sizes.

$$2\cdot\lambda + \tau\cdot\log(T/\tau)\cdot 3\cdot\lambda + \tau\cdot(\lambda\cdot\log n + 2\cdot\lambda + |\mathsf{aux}^{(n)}| + |\boldsymbol{w}| + |\mathsf{msg}^{(i)}|). \qquad (9.1)$$

We have $\mathsf{aux}^{(n)} = \{\Delta_m\}_{m\in[\mathsf{mult}]}$ and typically, the majority of messages in $\mathsf{msg}^{(i)}$ are the openings of α and β during the multiplications. We see that when λ, n, T and τ are fixed, the final proof size is strongly correlated with the number of multiplications in the circuit.

9.1.3 The Picnic signature scheme

It is straightforward to use the NIZKPoK described in the previous section to obtain the Picnic signature scheme [39]. Given a block cipher $F_k(\boldsymbol{x}) : \mathcal{K}\times\mathcal{X}\to\mathcal{Y}$, presented as a binary circuit, the scheme is described by three algorithms:

- $\mathtt{KGen}(1^\lambda)$: Sample $\boldsymbol{x}$ in $\mathcal{X}=\{0,1\}^\lambda$, and $k\in\mathcal{K}=\{0,1\}^\lambda$ and then compute $\boldsymbol{y}=F_k(\boldsymbol{x})$. The public key pk is given by $(\boldsymbol{y},\boldsymbol{x})$ and the secret key is $\mathsf{sk}=k$.
- $\mathtt{Sig}(\mathsf{sk}, m)$: Given a message m to be signed, compute $\sigma \leftarrow \mathsf{Prove}(\mathsf{pk}, k)$. Compute the challenge internally as $H(\tilde{\sigma}, m)$, where $\tilde{\sigma}$ denotes the message sent in **Round 1** of the proof.
- $\mathtt{Vfy}(\mathsf{pk}, m, \sigma)$: Compute $\mathsf{VerifyProof}(\mathsf{pk},\sigma)$ and the challenge as $H(\tilde{\sigma}, m)$. Return 1 if the result of $\mathsf{VerifyProof}$ is 1 and 0 otherwise.

The Picnic signature scheme is thus a NIZKPoK of k realized with the HVZKPoK protocol of [86] for binary circuits. The size of the proof σ then depends on the number of AND gates required for the evaluation of the symmetric primitive F. In the submission to the NIST's Post-Quantum Cryptography project [38], F is instantiated

Scheme	λ	s	r	#ANDs	T	τ	est. size
picnic2-L1-FS	128	10	20	600	343	27	12.7 kB
picnic2-L3-FS	192	10	30	900	570	39	28.1 kB
picnic2-L5-FS	256	10	38	1140	803	50	47.9 kB

Table 9.1: Picnic2 parameters and estimated proof sizes.

with LowMC [3], an "MPC friendly" block-cipher explicitly designed to have low AND depth and lower multiplication complexity. LowMC is a very parametrizable scheme, the number s of 3-bits S-boxes per round and the number of rounds r can both be modified to favor either low round complexity or low multiplication complexity. The block-size and key-size λ does not affect the number of S-boxes, it only imposes the condition that $3s \leq \lambda$; bits of the state that are not affected by the S-box layer are left unchanged. In [38], the Picnic submission team provides parameters (λ, s, r) for the LowMC block-cipher and parameters $(n = 64, \lambda, T, \tau)$ for the NIZKPoK. We reproduce these parameters in Table 9.1 together with the number of AND gates in each circuit and the estimated signature size using the formula given in (9.1). As per the MPC protocol for binary circuits given in [86], we use $|\mathsf{aux}^{(n)}| = |\mathsf{msg}^{(i)}| = (\#\text{ANDs})$ and $|\boldsymbol{w}| = \lambda$ in our estimates. We note that these estimated proof sizes consistently fall in between the maximum and averages sizes reported in the Picnic submission [38] and we will therefore compare our estimations to these.

Key generation security. As discussed in [39, Appendix D], the security of the key generation relies on the assumption that the block cipher $F_k(\boldsymbol{x})$ is a one-way function with respect to k. That is, for a fixed plaintext block $\boldsymbol{x} \in \mathcal{X}$, the function $f_{\boldsymbol{x}} : \mathcal{K} \rightarrow \mathcal{Y}$ defined by $f_{\boldsymbol{x}} : k \mapsto F_k(\boldsymbol{x})$ is a OWF. This is indeed the correct assumption since $\boldsymbol{x}$ and $\boldsymbol{y}$ as part of the public-key, thus fixing the function $f_{\boldsymbol{x}}$ and posing the challenge of recovering a suitable pre-image $\tilde{k} \in \mathcal{K}$ such that $f_{\boldsymbol{x}}(\tilde{k}) = F_{\tilde{k}}(\boldsymbol{x}) = \boldsymbol{y}$. This also shows that first picking $\boldsymbol{x}$ and then picking k during key generation is the natural way of first fixing the function $f_{\boldsymbol{x}}$ and then computing the challenge image $\boldsymbol{y}$.

In the full version of the original Picnic work [39], the authors go on to prove that if the block cipher $F_k(\boldsymbol{x})$ is a PRF family *with respect to* $\boldsymbol{x}$ (that is for a fixed k, the function $F_k : \mathcal{X} \rightarrow \mathcal{Y}$ is a PRF) then it is also a OWF *with respect to* k.

9.2 Revisiting AES for MPC-in-the-head

We now first recall the structure of the AES block-cipher. We next present our chosen method for computing the S-box, the non-linear operation of AES, in MPC and then discuss instantiations of the Picnic signature scheme using this protocol, together with estimates of resulting signature sizes. We finally discuss alternative methods for computing the AES S-box which were not selected.

9.2.1 The AES block cipher and its Rijndael origins

The AES is a 128-bit block-cipher constructed from a substitution-permutation network (SPN). It allows key lengths of 128, 192 or 256 bits and the corresponding number of rounds for the SPN is respectively 10, 12 or 14. The state always consists of 128 bits and can be considered as a 4×4 matrix of elements in $\mathbb{F}_{2^8}$. The cipher can thus be considered either as a Boolean circuit over $\mathbb{F}_2$ or as an arithmetic circuit over $\mathbb{F}_{2^8}$. We consider the latter. The round function is composed of four operations on the state, of which only one is non-linear: SubBytes.

AddRoundKey takes the 128-bit round key produced by the key schedule and performs an exclusive-or (XOR) operation with the current state to obtain the new state. As $\mathbb{F}_{2^8}$ has characteristic 2, the XOR operation can be computed by simply adding two elements together.

SubBytes is the only non-linear block of the round function, which transforms each of the 16 bytes of the state by means of a substitution function, known as the S-box. The AES S-box is a multiplicative inverse computation in the field $\mathbb{F}_{2^8}$, followed by an invertible affine transformation. In some sense, the S-box can be seen as

$$S : s \mapsto \phi^{-1}\left(\mathbf{A} \cdot \phi\left(s^{-1}\right) + \vec{b}\right) \tag{9.2}$$

where $\phi : \mathbb{F}_{2^8} \to (\mathbb{F}_2)^8$ is an isomorphism of vector spaces (mapping bytes in $\mathbb{F}_{2^8}$ to vectors of eight bits in $\mathbb{F}_2$) and $\mathbf{A} \in (\mathbb{F}_2)^{8\times 8}$ and $\vec{b} \in (\mathbb{F}_2)^8$ are the public parameters of the affine transformation.

ShiftRows is a permutation of the 16 state bytes, obtained by rotating row i of the state by i bytes to the left. Hence, the first row (row zero) remains the same, the second row is rotated to the left by one byte, etc.

MixColumns is the final phase and linear component. It consists of a mixing operation that is applied to each column of the state separately. Column i is transformed by a

AES-	128	192	256
# Rounds	10	12	14
# Round S-boxes	160	192	224
# Key schedule S-boxes	40	32	52
Total # S-boxes	200	224	276

Table 9.2: Number of S-boxes in the AES circuits.

Rijndael-	128	192	256
# Rounds	10	12	14
# Round S-boxes	160	288	448
# Key schedule S-boxes	40	48	112
Total # S-boxes	200	336	560

Table 9.3: Number of S-boxes in the Rijndael circuits.

matrix multiplication with a 4×4 matrix defined over $\mathbb{F}_{2^8}$.

Key schedule. The key schedule is mostly linear except for the application of the same AES S-box to up to four bytes of the round key. As the S-box is the only non-linear block, and only non-linear operations contribute to signature sizes in the MPCitH paradigm, Table 9.2 presents how many times it needs to be computed for each AES circuit.

Rijndael. The AES is the standardized version of the Rijndael cipher [51]. The most prominant difference between the two is that Rijndael allows both a variable key size κ and a variable block-length β. The round transformations are the same, with SubBytes performing S-box on each of the $\frac{\beta}{8}$ state bytes and MixColumns transforming each of the $\frac{\beta}{32}$ state columns. The key schedule is identical to that of AES-κ, but keep in mind that the expanded key is larger in the former case and hence the number of S-box calculations also differs. We summarize the number of S-boxes in Rijndael-κ circuits with $\kappa = \beta$ in Table 9.3.

9.2.2 Computing the S-box in MPC

To design an MPC protocol for the circuit $f_{\boldsymbol{x}}(\boldsymbol{k}) := \mathsf{AES}_{\boldsymbol{k}}(\boldsymbol{x})$ over $\mathbb{F}_{2^8}$, we design a communication-efficient computation of the non-linear S-box in a distributed way using the advantages of MPCitH. This computation happens in two stages: first an *inversion* stage described by the map

$$s \mapsto \begin{cases} s^{-1} & \text{if } s \neq 0, \\ 0 & \text{if } s = 0, \end{cases} \quad \text{over } \mathbb{F}_{2^8}; \tag{9.3}$$

and second, an *affine* stage where $s^{-1} \mapsto \phi^{-1}(\mathbf{A} \cdot \phi(s^{-1}) + \vec{b})$ in $(\mathbb{F}_2)^8$ as described in Equation (9.2). As the second stage is in fact local, we present it first and then address the inversion stage below.

The affine transformation (being linear) can be applied by each party independently. Since it operates on individual bits, the parties must first derive a sharing of the bit-decomposition of s. As we only require semi-honest security for our protocol, this can be also be achieved locally. Given the isomorphism $\phi : \mathbb{F}_{2^8} \to (\mathbb{F}_2)^8$, each party can locally obtain $\vec{s}^{(i)} = \phi(s^{(i)})$ which then corresponds to a sharing $\langle \vec{s} \rangle = \phi(\langle s \rangle)$. Each party can then locally compute the affine transformation $\vec{t}^{(i)} := \mathbf{A} \cdot \vec{s}^{(i)} + \vec{b}$; indeed, as $\mathbf{A}$ and $\vec{b}$ are both public values, the transformation can be computed as a series of multiplications by and additions with public constants during the online phase. Finally, they can recompose $\langle t \rangle = \phi^{-1}(\langle \vec{t} \rangle)$. Thus the affine phase of the S-box can be computed entirely locally.

Computing the inversion

To perform the Galois field inversion within the MPCitH paradigm, we use a masked inversion method. Namely, given $\langle s \rangle \in \mathbb{F}_{2^8}$ and a randomly sampled $\langle r \rangle \in \mathbb{F}_{2^8}$ (where each party samples $r^{(i)}$ at random from $\mathsf{sd}^{(i)}$), we run the following:

1. Compute $\langle s \cdot r \rangle$ (by using a preprocessed triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$ and opening $s - a$ and $r - b$).
2. $\mathsf{Open}(s \cdot r)$.
3. Compute $(s^{-1} \cdot r^{-1})$ (done locally by each P_i).
4. Compute $\langle s^{-1} \rangle = (s^{-1} \cdot r^{-1}) \cdot \langle r \rangle$.

Performing the multiplication and the opening of $\langle s \cdot r \rangle$ results in every party broadcasting three elements of $\mathbb{F}_{2^8}$ for each inversion. This implies that we have a base communication cost of 3 bytes per party per inversion in the online phase; i.e. for each P_i,

$$|\mathsf{msg}^{(i)}| = 8 \cdot (3 \cdot \#(\text{S-boxes}) + o),$$

where the additional o bytes come from the opening of the output values as $f_{\boldsymbol{x}}$ outputs tuples in $(\mathbb{F}_{2^8})^o$.

Also considering the inclusion of the offset value $\Delta \in \mathbb{F}_{2^8}$ in the auxiliary information $\mathsf{aux}^{(n)}$, for the triple that is used for the inversion, we also have that

$$|\mathsf{aux}^{(n)}| = 8 \cdot \#(\text{S-boxes}).$$

There are however two complications with this method: when either $r = 0$ or $s = 0$. We first present how to deal with the first one, assuming that the second does not happen. We then discuss how to mitigate the second one.

Need for non-zero randomness

Indeed, assuming that $s \neq 0$, we see that if $r = 0$, then $s \cdot r = 0$ as well and the inversion computation cannot proceed. However, this does not leak any information regarding s, so when the parties observe that $\mathsf{Open}(s \cdot r)$ yields 0, they can restart the computation of the inversion with a fresh $\langle r' \rangle$ and a new multiplication triple.

While this would not be a problem for real MPC executions of this protocol, here we are restricted because of the commitment to $\mathsf{aux}^{(n)}$, and therefore to the number of preprocessed triples, that the prover $\mathcal{P}$ of the HVZKPoK must make before it emulates the online phase of the protocol. The prover does emulate the online phase for every preprocessing phase, and it would therefore be possible for them to observe exactly how many triples are required for an execution. However, this would ultimately be dependent on the input.

In order to commit to the preprocessing, the prover would have to indicate how many triples were generated and using which bits of randomness, given that those bits produced from $\mathsf{sd}^{(i)}$ would be mixed with bits used to produce other randomness, such as the input sharings. We do not discuss here the design of a protocol which accounts for this flexible verification and the evaluation of whether this would leak information regarding the input to the verifier.

Instead, we design our protocol to generate a fixed number m of additional triples

m	1	2	3	4	5	6	7	8	9	10
AES-128	$2^{-2.4}$	$2^{-4.5}$	$2^{-6.9}$	$2^{-9.7}$	$2^{-12.7}$	$2^{-15.9}$	$2^{-19.3}$	$2^{-22.9}$	$2^{-26.6}$	$2^{-30.5}$
AES-192	$2^{-2.2}$	$2^{-4.1}$	$2^{-6.4}$	$2^{-9.0}$	$2^{-11.8}$	$2^{-14.8}$	$2^{-18.1}$	$2^{-21.5}$	$2^{-25.1}$	$2^{-28.8}$
AES-256	$2^{-1.8}$	$2^{-3.4}$	$2^{-5.4}$	$2^{-7.7}$	$2^{-10.2}$	$2^{-13.0}$	$2^{-15.9}$	$2^{-19.0}$	$2^{-22.3}$	$2^{-25.7}$

Table 9.4: Probability of abort of each AES circuit (assuming $\Pr[r = 0] = \frac{1}{256}$).

for every execution. In the rare cases that this additional number is not enough, we say that the prover aborts the proof and restarts. Table 9.4 shows the probability that a proof aborts for each of the AES circuits depending on the number of additional triples provided by P. If we want the probability of needing to abort a proof to be less than $10^{-8} \approx 2^{-20}$, then adding $m = 9$ additional triples is sufficient.

The addition of these m additional triples then implies that, for each execution, the auxiliary information contains $(\#(\text{S-boxes}) + m)$ elements in $\mathbb{F}_{2^8}$ and we therefore have

$$|\mathsf{aux}^{(n)}| = 8 \cdot (\#(\text{S-boxes}) + m).$$

Similarly, since parties are susceptible to repeat the computation of up to m inversions during the protocol and therefore broadcast more than 3 bytes per inversion, we have that for each P_i

$$|\mathsf{msg}^{(i)}| \leq 8 \cdot (3 \cdot (\#(\text{S-boxes}) + m) + o).$$

One advantage of restricting the number of triples produced in this way is that it guarantees a maximum proof size, independently of the randomness used.

Need for non-zero input

In the previous section we assumed that $s \neq 0$ to say that if $s \cdot r = 0$, then it must be that $r = 0$. However, in general, it is possible for the input s to the AES S-box to be 0. If the inversion was computed as above, with this possibility, then opening $s \cdot r = 0$ would in fact reveal information about s since r is not allowed to be zero in a correct inversion.

To avoid this leakage, we restrict our protocol to only prove knowledge of values

Circuit	AES-128	AES-192	AES-256
Number of S-boxes	200	224	276
No-zeroes probability	45.7%	41.6%	34.0%

Table 9.5: Probabilities of no zero-inputs to S-boxes.

of $\boldsymbol{k}$ for which there is no zero as input to any S-box in the computation of $f_{\boldsymbol{x}}(\boldsymbol{k})$. We write $\mathcal{K}_{\boldsymbol{x}} \subset \mathcal{K}$ to denote the subset of such keys. While this reduces the number of applications of this protocol as a general HVZKPoK for AES keys, this is not so significant in the context of Picnic signatures. Indeed, the values $\boldsymbol{x}$ and $\boldsymbol{k}$ for which the proof must be given are fixed during key generation and used for all signatures. It is therefore feasible to restrict the `KGen` algorithm to first select a random $\boldsymbol{x}$ and then sample values of $\boldsymbol{k}$ until one from $\mathcal{K}_{\boldsymbol{x}}$ is found.

We note that a malicious prover could intentionally generate a key for which some zeroes do appear in the computation of the circuit. However, in the context of public-key infrastructures, users typically present a signature of their certificate under their public-key as a proof of possession. A check on such a signature would immediately reveal whether a key was malformed and thus prevent verifiers from accepting further signatures.

Loss of security. This particular design of the `KGen` procedure naturally affects the assumption that $f_{\boldsymbol{x}}(\boldsymbol{k}) := \mathrm{AES}_{\boldsymbol{k}}(\boldsymbol{x})$ is still a OWF family as required for Picnic signatures. While the random selection of $\boldsymbol{x}$ still ensures that a random function $f_{\boldsymbol{x}}$ is selected from the family, the restricted selection of $\boldsymbol{k} \in \mathcal{K}_{\boldsymbol{x}}$ reduces the image space of $f_{\boldsymbol{x}}$ and also the pre-image space for a given $\boldsymbol{y}$.

We can estimate this security loss by assuming that, for a given block $\boldsymbol{x}$ and a randomly sampled key $\boldsymbol{k} \in \mathcal{K}$, the S-boxes are independent from one another and each has a probability of $\frac{1}{256}$ of receiving 0 as input. Table 9.5 presents the proportion of keys that would also belong to $\mathcal{K}_{\boldsymbol{x}}$ for the three different AES circuits, i.e. $(255/256)^{\#\text{ S-boxes}}$. For AES-256, we see that approximately $\frac{1}{3}$ of keys possess this property, which, from an attacker's perspective, reduces by only $\log_2(3) \approx 1.4$ the bit-security of the problem of inverting the OWF $f_{\boldsymbol{x}}$ and recovering $\boldsymbol{k} \in \mathcal{K}_{\boldsymbol{x}}$. The assumption that the S-boxes are independent is justified by the PRF-behaviour or AES. Moreover, our experiments confirmed that, for a fixed $\boldsymbol{x}$, the same proportions of keys as shown in Table 9.5 yielded no zero-inputs.

This also shows that it is not very computationally expensive for `KGen` to re-sample $\boldsymbol{k}$ as it will only have to do so three times on average. To emphasize the low cost of this, we also recall that during `KGen`, the signer samples and verifies that $\boldsymbol{k} \in \mathcal{K}_{\boldsymbol{x}}$ for a given $\boldsymbol{x}$. While they do this, they do not have to execute the AES circuit using the MPC protocol, the S-box inputs can be verified using an un-shared execution.

A more formal analysis. In [39, Appendix D], the authors show that if an adversary $\mathcal{A}$, on input $\boldsymbol{y}$ has a high probability of inverting the OWF $f_{\boldsymbol{x}}(\boldsymbol{k}) = \boldsymbol{y}$, then one can build a distinguisher $\mathcal{D}$ which has a high probability of distinguishing $F_{\boldsymbol{k}}$ from a random function. Recall that in our case, $F_{\boldsymbol{k}}$ is the usual $\mathrm{AES}_{\boldsymbol{k}}$ PRF family and that $f_{\boldsymbol{x}}$ is the AES function family considered on $\boldsymbol{k}$ indexed by $\boldsymbol{x}$.

Let us now assume that the adversary $\mathcal{A}$ has a high probability p of inverting $f_{\boldsymbol{x}}(\boldsymbol{k})$ only if $\boldsymbol{k} \in \mathcal{K}_{\boldsymbol{x}}$. In the reduction proof, $\mathcal{D}$ queries the PRF oracle $\mathcal{O}$ on input $\boldsymbol{x}$ to receive $\boldsymbol{y}$ as its image either under the PRF $F_{\boldsymbol{k}}$ for a random $\boldsymbol{k} \in \mathcal{K}$ or under a random function. It then queries $\mathcal{A}$ on input $\boldsymbol{y}$ in the hope of receiving the correct key $\boldsymbol{k}$. The only difference between our situation and theirs is that the key space $\mathcal{K}_{\boldsymbol{x}}$ for which $\mathcal{A}$ is capable of returning the correct key is smaller than the whole of $\mathcal{K}$. In the case where $\mathcal{O}$ is indeed computing the PRF, there is then a probability of $|\mathcal{K}_{\boldsymbol{x}}|/|\mathcal{K}|$ that the sampled hidden key will belong to the reduced key space and therefore $\mathcal{A}$ will return the correct key with smaller probability $p \cdot |\mathcal{K}_{\boldsymbol{x}}|/|\mathcal{K}|$. All other aspects of the proof are the same as in [39] and therefore, under the assumption that AES, over any key $\boldsymbol{k} \in \mathcal{K}$, is a PRF with respect to $\boldsymbol{x}$, it holds that it also is a OWF with respect to $\boldsymbol{k} \in \mathcal{K}_{\boldsymbol{x}}$ with a tightness gap proportional to $|\mathcal{K}_{\boldsymbol{x}}|/|\mathcal{K}|$ for a given $\boldsymbol{x}$.

We can therefore say that if our AES circuit with restricted keys would yield a weak OWF, then there would be a similar proportion of keys for which it would be easy to distinguish the AES function from a random one, thus contradicting the assumed and observed PRF behaviour of the AES. Furthermore our experiments confirm that this proportion of keys is non-negligible, further increasing confidence in our OWF construction.

9.2.3 Application to Picnic signatures

We now estimate the signature size if our arithmetic circuit $C_{\boldsymbol{x}}(\boldsymbol{k}) = \mathrm{AES}_{\boldsymbol{k}}(\boldsymbol{x})$ were to be used instead of the LowMC binary circuit in Picnic signatures and also present the interesting conclusion that a naive application of AES-192 or AES-256 is not sufficient to achieve stronger levels of security.

Level	Circuit	λ	T	τ	$\lvert\mathsf{aux}^{(n)}\rvert$	$\lvert\mathsf{msg}^{(i)}\rvert$	Est. size
L1	picnic2-AES-128-bin	128	343	27	6400	6400	51.9 kB
	picnic2-AES-128-inv				1664	5120	**31.6 kB**

Table 9.6: Estimates for Picnic signatures with AES-128 circuit.

AES-128

Table 9.6 presents our estimates using the AES-128 circuit instead of the LowMC circuit with parameters ($k = 128, s = 10, r = 20$), specified by the Picnic submission [38] as achieving AES-128-like security. For the picnic2-AES-128-bin circuit, we used the state-of-the-art figure of 32 AND gates per AES S-box reported in the inversion circuit of [27], thus yielding $|\mathsf{aux}^{(n)}| = |\mathsf{msg}^{(i)}| = 200 \cdot 32 = 6400$. We note that this differs from the figure of 5440 AND gates stated in [39, Section 6.1]. While we cite the same work [27], a figure of 5440 would make sense if the AES-128 circuit contained only 170 S-boxes, or 160 S-boxes with 34 AND gates per S-box Our Table 9.2 shows that if the full circuit is considered, including the key schedule, then 200 S-boxes are necessary. Moreover, removing the key schedule is not possible as a fraudulent prover would then be able to create a forgery by selecting a final round key which would make the second-to-last state agree with the public key value $\boldsymbol{y}$. Finally, for our picnic2-AES-128-inv circuit, we used our costs of $|\mathsf{aux}^{(n)}| = 8 \cdot (200 + 8) = 1664$ and $|\mathsf{msg}^{(i)}| = 8 \cdot (3 \cdot (200 + 8) + 16) = 5120$ accounting for $m = 8$ additional inversion operations and $o = 16$ bytes of communication per party for output reconstruction.

We see that implementing AES-128 using the best current binary circuit would increase the signature size by a factor of 4.07 compared to the estimated signature size for picnic2-L1-FS given in Table 9.1. However, our inversion technique yields an increase only by a factor of 2.48, thus a reduction of 39% over the binary circuit. We note that this improvement comes with the caveat that our technique of restricting to non-zero inputs to S-boxes reduces the key-space for $\boldsymbol{k}$ by $|\log_2(0.457)| = 1.13$ bits from the 128 bit-security level.

AES-192 and AES-256

Such a direct comparison as the one presented above is not possible for the AES-192 and AES-256 circuits. While these two algorithms are believed to provide respectively 192 and 256 classical bit-security when used as block-ciphers, this does not hold in this paradigm due to the requirement for a one-way function for the key generation of Picnic-like signatures.

Indeed, as remarked in [39, Appendix D], the security of the key generation requires the block-size and the key-size of the underlying block-cipher to be equal—in part to prevent quantum attacks which benefit from a square-root speedup over the block-size. This is not the case for AES-192 and AES-256 and therefore, interestingly, the shift to longer keys, without an accompanying shift to longer blocks, does not translate into increased security against forgery attacks for the signature scheme. Here, the standardized 128 bit block-length of the AES cipher becomes an obstacle to easily achieving both stronger security and optimal efficiency. For AES-192, for example, a single block of encryption would result in (an expected) 2^{64} spurious keys per single block $\boldsymbol{x}$, $\boldsymbol{y}$ pair. Thus the probability of guessing a valid key, with a single block pair $\boldsymbol{x}$, $\boldsymbol{y}$ will be 2^{-128} and not 2^{-192} as desired, and in addition applying Grover's search for a valid key would only be slightly more complex than in the AES-128 case.

This fixed block-length was actually not a part of the original Rijndael design [51]. Hence, to obtain L3 (resp. L5) constructions, one could alternatively use the Rijndael cipher with 192-bit (resp. 256-bit) blocks and key. Since these are not standardized, we first discuss some constructions using AES.

To design a circuit suitable for 192 (resp. 256) bit-security level Picnic signatures, we propose to combine two copies of an AES-192 (resp. AES-256) circuit, keyed with the same key. To realize such a cipher with longer block- and key-length, we use AES in ECB mode. For our L3 construction using AES-192, we only require $\boldsymbol{x}$ and $\boldsymbol{y}$ to be 192 bits long. We therefore pad $\boldsymbol{x}$ with 64 0s to reach two full blocks, and then truncate the resulting ECB-mode encryption to its first 192 bits (one and a half blocks) to produce the output $\boldsymbol{y}$. For our L5 construction using AES-256, we use an $\boldsymbol{x}$ value of 256 bits, encrypted as two blocks in ECB mode, to produce a two-block value $\boldsymbol{y}$ of 256 bits.

One side-effect of using ECB is that we have the problem of malleable public keys, for example the public key $(\boldsymbol{x}_0 \| \boldsymbol{x}_1, \boldsymbol{y}_0 \| \boldsymbol{y}_1)$ would be equivalent to $(\boldsymbol{x}_1 \| \boldsymbol{x}_0, \boldsymbol{y}_1 \| \boldsymbol{y}_0)$. Whilst this does not break the security of the signature scheme in the standard security game, this could be a problem in practice. However, we can tie signatures to a specific public key by including it in the hash used to generate the challenge in the NIZKPoK.

The security of key generation now depends on the ECB construction which is clearly not a PRF family, but we can instead rely on the assumption that ECB mode is OW-CPA and we provide a similar argument as in [39] to formalise this assumption.

Claim 9.1. If $F_k(x)$ is OW-CPA then $f_x(k)$ is a OWF.

Level	Circuit	λ	T	τ	$\lvert\mathsf{aux}^{(n)}\rvert$	$\lvert\mathsf{msg}^{(i)}\rvert$	Est. size
L3	picnic2-AES-192-bin	192	570	39	13312	13312	149.1 kB
	picnic2-Rijndael-192-bin				10752	10752	124.2 kB
	picnic2-AES-192-inv				3416	10440	**86.9 kB**
	picnic2-Rijndael-192-inv				2768	8496	**74.2 kB**
L5	picnic2-AES-256-bin	256	803	50	16000	16000	233.7 kB
	picnic-Rijndael-256-bin				17920	17920	257.7 kB
	picnic2-AES-256-inv				4096	12544	**133.7 kB**
	picnic2-Rijndael-256-inv				4576	13984	**149.7 kB**

Table 9.7: Estimates for Picnic signatures using AES-192, AES-256 and Rijndael.

Proof. The proof follows similary to the proof in [39, Appendix D]. Indeed, given an adversary $\mathcal{A}$ that can return k on input $y = f_x(k)$, we can build a adversary $\mathcal{B}$ against the OW-CPA challenger for F_k as follows. First, $\mathcal{B}$ submits x to the encryption oracle and receives $y = F_k(x)$ for random $k \in \mathcal{K}$ chosen by the oracle. Then $\mathcal{B}$ runs $\mathcal{A}$ on input y to obtain k with high probability. Finally, $\mathcal{B}$ requests the challenge ciphertext y^* and uses k to compute $x^* = F_k^{-1}(y^*)$ and return it as its answer to the OW-CPA challenge. With high probability, $\mathcal{B}$ returns the correct pre-image. □

Both of the ECB-mode circuits result in executing two AES circuits but this drawback is inherent to the fixed block-length of the AES block-cipher and therefore applies equally to our inversion approach and the naive binary approach. We estimate the proof sizes obtained with these designs in Table 9.7.

For the binary circuits, we use the same figure of 32 AND gates per S-box [27]. In the case of the picnic2-AES-192-bin and picnic2-AES-256-bin circuits, we double the number of round S-boxes presented in Table 9.2 to obtain figures for $|\mathsf{aux}^{(n)}|$ and $|\mathsf{msg}^{(i)}|$. We keep the same number of key-schedule S-boxes as both circuits use the same key $\boldsymbol{k}$ and therefore the key-schedule needs to be computed only once. For the inversion circuits, we also double the number of round S-boxes in our formulæ for $|\mathsf{aux}^{(n)}|$ and $|\mathsf{msg}^{(i)}|$ but we must also change number m of additional triples due to the increased risk of sampling zero randomness during the inversion computation. A quick calculation similar to the one used for Table 9.4 shows that our doubled-up circuits for AES-192 and AES-256 require $m = 11$ and $m = 12$ respectively to have a probability of abort below 2^{-20}.

We see that implementing our doubled-up AES-192 and AES-256 using the best current binary circuit would increase the signature sizes by a factor of 5.30 and 4.87 respectively compared to the estimated signature size for picnic2-L3-FS and picnic2-L5-FS given in Table 9.1. However, our inversion technique yields increases only by a factor of 3.09 and 2.79 respectively, thus reductions of 41.7% and 42.8% over the binary circuits. We note that this improvement comes with the caveat that our technique of non-zero inputs reduces the key-space for $\boldsymbol{k}$ by $|\log_2(0.196)| = 2.35$ bits and $|\log_2(0.141)| = 2.83$ bits from the 192 and 256 bit-security levels respectively.

Rijndael-192 and Rijndael-256

To avoid the reliance on ECB mode and on a slightly different security assumption, we also provide estimates of proof sizes that make use of the original Rijndael-192 and -256 circuits which use an equal block and key size with the same S-box as AES. The number of rounds in Rijndael-192 (resp. Rijndael-256) is the same as in AES-192 (resp. AES-256), i.e. 12 (resp. 14).

We use the figures of Table 9.3 to obtain figures for $|\mathsf{aux}^{(n)}|$ and $|\mathsf{msg}^{(i)}|$, presented in Table 9.7. As above, we calculate that values of $m = 10$ and $m = 12$ are required by the Rijndael-192 and -256 circuits respectively to have a probability of abort below 2^{-20}.

We see that implementing the Rijndael-192 and -256 circuits using the best current binary circuit would increase the signature sizes by a factor of 4.42 and 5.38 respectively compared to the estimated signature size for picnic2-L3-FS and picnic2-L5-FS given in Table 9.1. However, our inversion technique yields increases only by a factor of 2.64 and 3.12 respectively, thus reductions of 40.2% and 41.8% over the binary circuits. We note that this improvement comes with the caveat that our technique of non-zero inputs reduces the key-space for $\boldsymbol{k}$ by $|\log_2(0.268)| = 1.90$ bits and $|\log_2(0.112)| = 3.16$ bits from the 192 and 256 bit-security levels respectively.

It is not surprising that the Rijndael-192 results are overall better those with AES-192, since the ECB construction with extra padding performs more work than it should. On the other hand, the AES-256 constructon is more efficient than the Rijndael-256 design since the latter has a larger key schedule.

9.2.4 Alternative computations of the AES S-box

Representing the AES S-box as the map $s \mapsto s^{-1}$ (with $0 \mapsto 0$) over $\mathbb{F}_{2^8}$ is one of three methods described by Damgård and Keller [53] for this task in standard MPC. In fact, it is the most efficient method of that work to compute the AES arithmetic circuit in real-world MPC. However, their treatment of the $s = 0$ difficulty differs from ours, and they also present two alternative methods to compute the inversion operation. We discuss these methods and their cost here, along with other methods, to study the possibilities for computing the AES circuit in the MPCitH paradigm.

Square-and-multiply

The first method of [53] is not to compute $s \mapsto s^{-1}$ but rather $s \mapsto s^{254}$. As $\mathrm{ord}(\mathbb{F}_{2^8}{}^*) - 1 = 255 - 1 = 254$, this achieves the same result with the additional advantage of not requiring a special case for $s = 0$ which maps to 0 naturally. However, s^{254} requires a square-and-multiply chain to be computed. The shortest such chain given in [53] requires 11 multiplications. Combining the cost of the openings and the auxiliary information required for triple generation, this method would cost $11 \cdot 3 = 33$ bytes per S-box, compared to only 4 bytes per S-box for our method in Section 9.2.2.

Masked exponentiation

This second method of [53] computes the same map $s \mapsto s^{254}$ but using the fact that $(a+b)^{2^i} = a^{2^i} + b^{2^i}$ in fields of characteristic 2. Here, they are able to take advantage of the preprocessing model; they first pre-compute the squares $\langle r \rangle, \langle r^2 \rangle, \langle r^4 \rangle, \ldots, \langle r^{128} \rangle$ for a random value $r \in \mathbb{F}_{2^8}$. Then, to invert $\langle s \rangle$, they additively mask $\langle s \rangle + \langle r \rangle$, open $(s + r)$ and square locally to obtain $(s + r)^2, (s + r)^4, \ldots, (s + r)^{128}$. Finally, they unmask each power-of-two with the corresponding shared power-of-two of r, $(s + r)^{2^i} + \langle r^{2^i} \rangle = \langle s^{2^i} \rangle$ and then perform an online multiplication chain to obtain $\prod_{i=1}^{7} \langle s^{2^i} \rangle = \langle s^{254} \rangle$.

Unlike in [53], the requirement on the MPC protocol for only semi-honest security means that the successive shared powers of $\langle r \rangle$ can be computed locally. Indeed, in characteristic 2, $r^2 = (\sum_{i=1}^{n} r^{(i)})^2 = \sum_{i=1}^{n} (r^{(i)})^2$, so each party can compute their shares locally. Therefore the cost of this method in our paradigm is the opening of $(s + r)$ and the six multiplications required for the computation of $\langle s^{128} \rangle$. This would still amount to $1 + 6 \cdot 3 = 19$ bytes per S-box (broadcast and auxiliary information put together).

Masked inversion

In [53], Damgård and Keller also conclude that the masked inversion method that we presented in Section 9.2.2 is the most efficient. Indeed, at its core it only requires 3 bytes in the online phase and 1 triple in the preprocessing phase. However, the leakage that occurs when $s = 0$ must be prevented. While it is possible in our application to Picnic signatures to generate the public-/private-key pairs so that this leakage does not take place, this is not possible for generic computations of the AES circuit.

The solution given in [53] is to have the parties compute a shared "zero-indicator" value $\langle \delta(s) \rangle$ defined as

$$\delta(s) = \begin{cases} 1 & \text{if } s = 0, \\ 0 & \text{otherwise.} \end{cases}$$

It is then easily verified that the inversion mapping of (9.3) is equivalent to

$$s \mapsto (s + \delta(s))^{-1} + \delta(s)$$

When $\delta(s)$ is computed in shared form, then the parties can perform the inversion computation with $\langle s + \delta(s) \rangle$ instead of $\langle s \rangle$. Once $\langle (s + \delta(s))^{-1} \rangle$ is obtained, the parties can locally compute $\langle (s + \delta(s))^{-1} \rangle + \langle \delta(s) \rangle$. This ensures that if $s = 0$ then $s + \delta(s) = 1$ is instead inverted to $(s + \delta(s))^{-1} = 1$ and that the second addition of $\delta(s)$ returns the value to 0, as $\mathbb{F}_{2^8}$ has characteristic 2.

This technique could be applied to our MPCitH circuit for AES to avoid the loss of generality and security that come with the efficient method presented in the previous section. We present here two methods to compute the zero-indicator $\delta(s)$ in secret-shared form.

Method 1. In [53], the value $\langle \delta(s) \rangle$ is computed by first bit-decomposing $\langle s \rangle$, then NOT-ing each bit and finally computing the joint AND of the eight bits of s. This joint AND requires 7 bit-wise AND gates to output the final value. If these gates are performed so that each party eventually holds $\delta(s)^{(i)} \in \mathbb{F}_2$, then each party can embed its share into $\mathbb{F}_{2^8}$ to obtain its share of $\langle \delta(s) \rangle$.

As before, bit-decomposition is trivially obtained by each party locally bit-decomposing its share into $\vec{s}^{(i)} = \phi(s^{(i)})$. The NOT gates are then also performed locally by each party. For the AND gates, we make use of the same technique of preprocessed multiplication triples, but over $\mathbb{F}_2$ instead of $\mathbb{F}_{2^8}$. For each AND gate, 1 bit then needs to be added to $\mathsf{aux}^{(n)}$ to correct the triple and 2 bits need to be opened in the online phase to compute the multiplication.

This method therefore results in $7 \cdot 3 = 21$ bits per S-box in addition to the 4 bytes required for the inversion of $\langle s + \delta(s) \rangle$. This results in approximately 6.6 bytes per S-box which is only 17.5% less than implementing the best binary circuit for AES in the model of [86].

Method 2. In the world of MPC, the computation of the shared zero-indicator $\langle\delta(s)\rangle$ can be seen as an n-party functionality which takes $\langle s\rangle$ as input and distributes $\langle\delta(s)\rangle$ to all the parties.[4] As we are in the MPCitH setting, the prover can observe the value of s and act as this functionality by providing each party with their share of $\delta(s)$. To further optimize this, we can assume that P_i, for $i = 1 \rightarrow n-1$, samples their share of $\delta(s)$ pseudo-randomly and that the prover only gives P_n a correction value which makes the reconstructed value equal to $\delta(s)$.

This approach is the first here that deviates from a normal MPC scenario and exploits the MPCitH paradigm to reduce the cost. However, it comes with the caveat that a malicious prover is able to select an arbitrary correction value for P_n. This implies that they have the freedom to arbitrarily modify the computation of the circuit. A partial fix for this is to specify that the shares of $\delta(s)$ must be either 0 or 1 within $\mathbb{F}_{2^8}$. This is only natural, as $\delta(s)$ can only take either of these values and the sum of such values is again 0 or 1. In this way, the prover can at most substitute a 1 for a 0 or vice-versa thus greatly limiting their ability to cheat arbitrarily. They can only inject an arbitrary $\mathbb{F}_{2^8}$ value into P_n and not get caught if P_n is the party that is not revealed to the verifier, which corresponds exactly to their usual cheating capability in Picnic. Furthermore, this approach means that the additional auxiliary input required for P_n is only one bit of information, as opposed to one byte.

The remaining drawback is then that a malicious prover can arbitrarily change the value of $\delta(s)$ from 0 to 1 in the computation. Note that this 1-bit fault can only be injected in the least significant bit of the input of the AES inversion, and if injected, the same fault is automatically applied to the inversion output as well, corresponding to the second XOR with $\delta(s)$. Unfortunately, it remains as an open question to compute what the soundness would be of a proof which used this technique for the computation of the AES circuit. This kind of fault injections have not been studied in the side-channel analysis literature, since a fault attacker typically injects faults in order to obtain an exploitable (non-zero) output difference in the ciphertext (i.e. differential fault analysis [22]). A malicious prover in MPCitH on the other hand would be trying to modify an execution yielding a wrong ciphertext into an apparently correct one.

In the normal MPCitH scenario, a malicious prover who does not have knowledge of the secret key $\boldsymbol{k}$, has a probability $2^{-\lambda}$ to produce a valid signature by guessing the key, where $\lambda \in \{128, 192, 256\}$ is the key and block size. In this adapted scenario with very specific 1-bit faults, the size of the prover's search space increases to $2^{\lambda} \cdot 2^{\#\text{ S-boxes}}$, as they can guess both a key and a fault configuration by selecting which of the S-box

[4]The majority of the analysis for this method is the work of L. De Meyer.

computations to infect. Let $\tilde{\boldsymbol{k}}$ be the key guess and $\tilde{F}$ be the circuits of Section 9.2.2 with injected faults; then the prover succeeds if the correct ciphertext is output at the end, i.e. for a public key $(\boldsymbol{x}, \boldsymbol{y})$, $\tilde{F}_{\tilde{\boldsymbol{k}}}(\boldsymbol{x}) = \boldsymbol{y}$. We again assume that all S-box calculations are independent and treat AES as a PRF, hence we assume that each of the 2^λ possible outputs are equiprobable. We thus estimate that the number of correct choices is

$$\#\text{ correct choices} \approx \frac{\text{total }\#\text{ choices}}{\#\text{ outputs}} = \frac{2^{\lambda+\#\text{S-boxes}}}{2^\lambda} = 2^{\#\text{S-boxes}}.$$

This results in the following success probability for a malicious prover:

$$\Pr[\tilde{F}_{\tilde{\boldsymbol{k}}}(\boldsymbol{x}) = \boldsymbol{y}] \approx \frac{\#\text{ correct choices}}{\text{total }\#\text{ choices}} = \frac{2^{\#\text{S-boxes}}}{2^{\lambda+\#\text{S-boxes}}} = 2^{-\lambda}$$

which corresponds exactly to the success probability in the MPCitH scenario without fault injections.

This methods originates from using an n-party functionality within the MPCitH paradigm, which is normally not permitted as a malicious prover is then able to cheat arbitrarily. However our analysis shows that, for this particular application, this additional freedom does not seem to give the Prover further advantages to forge a signature. Therefore, while n-party functionalities may not be permitted in general, this shows that case-by-case analysis may reveal that they do not affect the soundness for specific proofs. This then raises the question of what other proofs may be realized by taking advantage of such functionalities, which are in fact “free”, in terms of communication between parties, as they can be computed outside of the MPC protocol by the prover themselves.

Chapter 10

Faster MPCitH from verification protocols

In the previous chapter, the MPCitH method used consists in computing the entire circuit as it would have been in a real MPC execution. In that case, no advantage is gained from the fact that the prover is in fact able to reconstruct the simulated secret-sharing of the intermediary values. This strategy is only briefly considered in the description of the last method to realize the masked inversion technique. In that work, we did not consider this to be an advantage, only an opportunity for a malicious prover to cheat, and we analyzed the security risk of letting this cheating be possible.

Taking a different approach, Baum and Nof were the first to suggest that this capability of the prover could be used for efficiency gains [10]. In that work, they allowed the prover to "fill-in" certain values in the computation. To prevent cheating, the prover then has to simulate the execution of a verification protocol on these filled-in values, and the verifier then checks that this verification protocol succeeded and was executed correctly. This replaces the prover's commitment to an execution and the verifier's checking of that execution.

In this chapter we describe two further signature schemes derived from this technique. The first is based on the proof of knowledge of a pre-image of the Legendre PRF and the second is an improvement to the proof of knowledge of an AES key presented in the previous chapter.

The contents of Section 10.1 are selected from the work of W. Beullens[1] and this author; it was published in the proceedings of *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020* and was presented virtually at the

[1]KU Leuven, Belgium

conference on 21–23 September 2020 [19] by this author. The majority of the ideas and writing were the work of W. Beullens so we only include a brief description of the principles and of the protocol itself.

The contribution of Section 10.2 is selected from work in progress of C. Baum[2], E. Orsini[3], P. Scholl[4] and this author. The core idea of the polynomial-based verification protocol was the work of C. Baum, P. Scholl and E. Orsini; the formalizing of the verification protocol together with that of the signature scheme and its proof is the work of this author.

10.1 Post-quantum signatures from the Legendre PRF

We first recall the definition of the Legendre PRF and of its generalizations to residue symbols of higher order. After stating the relation that the scheme will provide a proof of, we then describe the verification mechanism used to replace the computation of the actual PRF evaluations. We finally state the formal description of the signature scheme and quote implementation results from [19].

10.1.1 The Legendre PRF and higher order variants

Let p be an odd prime. The Legendre PRF family, indexed by a key $K \in \mathbb{Z}_p$, is the family of functions

$$\begin{aligned}\mathcal{L}_K : \mathbb{F}_p &\to \mathbb{Z}_2, \\ a &\mapsto \left\lfloor \frac{1}{2}\left(1 - \left(\frac{K+a}{p}\right)\right)\right\rfloor,\end{aligned}$$

where $(\frac{a}{p}) \in \{-1, 0, 1\}$ is the quadratic residuosity symbol of a mod p, i.e. whether a is a square (1), not a square (-1) or 0 modulo p. Given the above definition, we have that $\mathcal{L}_0(a \cdot b) = \mathcal{L}_0(a) + \mathcal{L}_0(b)$ for all $a, b \in \mathbb{F}_p^\times$ and that $\mathcal{L}_K(a) = \mathcal{L}_0(K + a)$.

This function family is conjectured to be pseudo-random and was first proposed for use in cryptography by Damgård in 1988 [52]. Since then, its security as a PRF family has been studied in many attack models. In the case of quantum adversaries, if they are only able to make classical queries to a PRF evaluation oracle, then the state-of-the-art is an attack which requires $O(p^{1/2} \log p)$ computation of Legendre symbols and $O(p^{1/2} \log p)$ queries to the PRF oracle [90]. Restricting the adversary to

[2]Aarhus University, Denmark.
[3]KU Leuven Belgium.
[4]Aarhus University, Denmark.

only L queries to the evaluation oracle then increases the required Legendre symbol computations to $O(p \log^2 p/L^2)$ [18].

In his original work, Damgård also considers higher order power residues as a generalization of the Legendre PRF. Instead of using the quadratic residue symbol $(\frac{a}{p}) = a^{\frac{p-1}{2}} \mod p$, these generalizations instead use the k-th power residue symbol $(\frac{a}{p})_k = a^{\frac{p-1}{k}} \mod p$ for any integer k which divides $p-1$. Analogously to the Legendre PRF, the power residue PRF family, indexed by a key $K \in \mathbb{Z}_p$, is defined as

$$\begin{aligned}
\mathcal{L}_K^k : \mathbb{F}_p &\to \mathbb{Z}_k, \\
a &\mapsto \begin{cases} i & \text{if } (a+K)/g^i \equiv h^k \mod p \text{ for some } h \in \mathbb{F}_p^\times, \\ 0 & \text{if } (a+K) \equiv 0 \mod p, \end{cases}
\end{aligned}$$

where g is a fixed generator of $\mathbb{F}_p^\times$. The original Legendre PRF is then a special case with $k = 2$; we therefore use the generic notation for the rest of the section.

For implementations, the power residue PRF has the advantage of providing $\log k$ bits of output, instead of just 1 for the Legendre PRF. In the setting where an adversary is only allowed to query L evaluations of the power residue PRF, the best known attack currently requires computing $O(p \log^2 p/(kL \log^2 k))$ power residue symbols [18].

The power residue PRF relation

In the previous chapter, the language for which we provided a proof was based on the relation $R((\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{k}) = \{((\boldsymbol{x}, \boldsymbol{y} = F_{\boldsymbol{k}}(\boldsymbol{x})), \boldsymbol{k})\}$ for some block cipher F. Here we define the Legendre and power residue PRF relations $R_{\mathcal{L}^k}$ for $k \geq 2$ and their relaxed versions $R_{\beta\mathcal{L}^k}$ for a parameter β.

Let $\mathcal{I} = (i_1, \ldots, i_L)$ denote a list of L elements of $\mathbb{Z}_p$. We denote a length-L k-th power residue PRF as

$$\begin{aligned}
F_{\mathcal{I}}^k : \mathbb{F}_p &\to \mathbb{Z}_k^L \\
K &\mapsto (\mathcal{L}_K^k(i_1), \ldots, \mathcal{L}_K^k(i_L)).
\end{aligned}$$

Definition 10.1 (k-th power residue PRF [19, Definition 1])**.** For an odd prime p, a positive integer $k \mid p-1$ and a list $\mathcal{I}$ of L elements of $\mathbb{Z}_p$, the *k-th power residue PRF relation $R_{\mathcal{L}^k}$* with output length L is defined as

$$R_{\mathcal{L}^k} = \{(F_{\mathcal{I}}^k(K), K) \in \mathbb{Z}_k^L \times \mathbb{F}_p\}.$$

In short, the language defined by this relation consists of all strings of length L defined by K and I. The second relation below is a relaxed version of the first. The induced language consists of strings which are very close to those of the first language, where the Hamming distance d_H is used and β parameterizes the slack.

Definition 10.2 (β-approximate k-th power residue PRF [19, Definition 2])**.** For $\beta \in [0,1]$, an odd prime p, a positive integer $k \mid p-1$ and a list $\mathcal{I}$ of L elements of $\mathbb{Z}_p$, the *β-approximate k-th power residue PRF relation $R_{\beta\mathcal{L}^k}$* with output length L is defined as

$$R_{\beta\mathcal{L}^k} = \{(s,K) \mid \exists a \in \mathbb{Z}_k : d_H(s+(a,\dots,a), F^k_{\mathcal{I}}(K)) \leq \beta L\} \subset \mathbb{Z}_k^L \times \mathbb{F}_p.$$

Theorem 1 of [19] proves that, for sufficiently small β and large L, then the β-approximate language is as hard at the exact one because every witness for the first is also a witness for the second with overwhelming probability over the choice of the list $\mathcal{I}$.

10.1.2 Signature scheme from the power residue PRF relation

The signature scheme we present here is the one of [19, Section 5] based on the identification scheme of [19, Section 4]. Instead of the LowMC block cipher used in Picnic, this scheme uses the PRF $F^k_{\mathcal{I}}$, for some fixed list $\mathcal{I}$. The secret key is then $\mathsf{sk} = K$, corresponding to the public key $\mathsf{pk} = F^k_{\mathcal{I}}(K)$, and a signature consists in a proof of knowledge of sk with a simulated N-party computation of pk.

To avoid expensive computation of so many power residue symbols, three optimizations were proposed. The first is to compute only B symbols out of the L contained in the public key. The B indices of $\mathcal{I}$ that are to be computed are selected at random by the verifier (or by a random oracle, following the Fiat–Shamir paradigm). This turns the scheme into a proof of knowledge for the β-approximate language but, by Theorem 1 of [19], this does not induce a noticeable loss in security.

The second optimization is to multiplicatively mask the B inputs $K + I^{(j)}$ and let the verifier compute their residue symbols. In order to do so, the prover first commits to B random values $r^{(j)}$ together with their residue symbols $s^{(j)} = \mathcal{L}^k_0(r^{(j)})$. After the verifier has selected the B elements $I^{(j)}$ of $\mathcal{I}$ to be checked, the prover makes the simulated MPC protocol compute the B output values $o^{(j)} = (K + I^{(j)})r^{(j)}$. On the one hand, the multiplicative masks ensure that no information about K is leaked, and on the other, the verifier can check himself whether $\mathcal{L}^k_0(o^{(j)}) = \mathsf{pk}_{I^{(j)}} + s^{(j)}$, thus avoiding the expensive computation of the residue symbol in the MPC protocol.

The third optimization is of the kind discussed at the beginning of this chapter. Instead of *computing* the outputs $o^{(j)}$ in the MPC protocol, the prover can instead compute them and send them to the verifier directly. Then the verifier will challenge him to check that these $o^{(j)}$ values are indeed the correct results of the multiplications. The optimization comes from the fact that it is possible to verify a random linear combination of these equations, rather than all B of them individually. Indeed, when given B random coefficients $\lambda^{(j)}$ by the verifier (or the random oracle in the signature scheme), the prover with make the MPC protocol compute the error term

$$E = \sum_{j=1}^{B} \lambda^{(j)} \left((K + I^{(j)}) r^{(j)} - o^{(j)} \right).$$

If all the $o^{(j)}$ products were computed honestly by the prover, E will be 0; otherwise, if one or more are wrong, E will be uniformly random because of the $\lambda^{(j)}$ coefficients, except with probability $1/p$, which is negligible. The value of E can in fact be computed with a single non-linear operation since K can be taken to the outside of the sum:

$$E = K \cdot \sum_{j=1}^{B} \lambda^{(j)} r^{(j)} + \sum_{j=1}^{B} \lambda^{(j)} (I^{(j)} r^{(j)} - o^{(j)}).$$

As non-linear operations are the only ones requiring communication between parties in MPC, this makes for an extremely efficient MPCitH protocol.

We note that despite the $1/p$ probability of cheating on the output values, the prover can still cheat in 1 out of the N parties computing the verification protocol. Since N must be relatively small for computational efficiency, we still run M parallel executions to reduce the cheating probability to a negligible value.

The signature scheme

We now present the formal signature scheme. The verifier challenges are computed using three random oracles H_1, H_2 and H_3 according to the Fiat–Shamir transform [68]. The message to be signed is added to the first challenge computation, together with a 2λ-bit salt value.

The parameters params consist of the values p, k, L, B, N and M together with a pseudo-random list $\mathcal{I}$ as described above. The key-generation algorithm $\mathtt{KGen}$ samples a random secret-key $\mathsf{sk} = K \leftarrow\!\$\, \mathbb{F}_p$ uniformly at random and computes the public-key $\mathsf{pk} = F_{\mathcal{I}}^{k}(K)$. The signing algorithm $\mathtt{Sig}$ is described in Figures 10.1 and 10.2 and the

```
$\texttt{Sig}(\mathsf{params}, \mathsf{sk}, m)$—part 1
Phase 1: Commitment to sharings of $K$, randomness and triples
1: Pick a random salt: $\mathsf{st} \leftarrow \{0,1\}^\lambda$.
2: for $e$ from 1 to $M$ do
3:     Sample a root seed: $\mathsf{sd}_e \leftarrow_\$ \{0,1\}^\lambda$.
4:     Build binary tree from $\mathsf{sd}_e$ with leaves $\mathsf{sd}_{e,1}, \dots, \mathsf{sd}_{e,N}$.
5:     for $i$ from 1 to $N$ do
6:         Sample shares: $K_{e,i}, r_{e,i}^{(1)}, \dots, r_{e,i}^{(B)}, a_{e,i}, b_{e,i}, c_{e,i} \leftarrow \texttt{Expand}(\mathsf{sd}_{e,i})$.
7:         Commit to seed: $\texttt{Com}_{e,i} \leftarrow H_{\mathsf{sd}}(\mathsf{st}, e, i, \mathsf{sd}_{e,i})$.
8:     Compute witness offset: $\Delta K_e \leftarrow K - \sum_{i=1}^N K_{e,i}$.
9:     Adjust first share: $K_{e,1} \leftarrow K_{e,1} + \Delta K_e$.
10:    Compute triple: $a_e \leftarrow \sum_{i=1}^N a_{e,i}$, $b_e \leftarrow \sum_{i=1}^N b_{e,i}$ and $c_e \leftarrow a_e \cdot b_e$.
11:    Compute triple offset: $\Delta c_e \leftarrow c_e - \sum_{i=1}^N c_{e,i}$.
12:    Adjust first share: $c_{e,1} \leftarrow c_{e,1} + \Delta c_e$.
13:    for $j$ from 1 to $B$ do
14:        Compute residuosity symbol: $s_e^{(j)} \leftarrow \mathcal{L}_0^k(r_e^{(j)})$ where $r_e^{(j)} \leftarrow \sum_{i=1}^N r_{e,i}^{(j)}$.
15: Set $\sigma_1 \leftarrow ((\texttt{Com}_{e,i})_{i\in[N]}, (s_e^{(j)})_{j\in[B]}, \Delta K_e, \Delta c_e)_{e\in[M]}$.
Phase 2: Challenge on public key symbols
1: Compute challenge hash: $h_1 \leftarrow H_1(m, \mathsf{st}, \sigma_1)$.
2: Expand hash: $(I_e^{(j)})_{e\in[M],j\in[B]} \leftarrow \texttt{Expand}(h_1)$, where $I_e^{(j)} \in \mathcal{I}$.
Phase 3: Computation of output values
1: for $e$ from 1 to $M$ and for $j$ from 1 to $B$ do
2:     Compute output value: $o_e^{(j)} \leftarrow (K + I_e^{(j)}) \cdot r_e^{(j)}$.
3: Set $\sigma_2 \leftarrow (o_e^{(1)}, \dots, o_e^{(B)})_{e\in[M]}$.
```

Figure 10.1: k-th power residue PRF signature scheme—part 1.

verification algorithm `Vfy` is described in Figure 10.3. We state below the security theorem from [19].

Theorem 10.3 ([19, Theorem 2]). *In the classical random oracle model, the signature scheme defined above is EUF-CMA secure under the assumption that computing β-approximate witnesses for a given public-key is hard.*

To compare with the AES-based proposal of Chapter 9, we note that the Legendre PRF based instantiation (i.e. with $k = 2$) of this scheme achieves signature sizes ranging from 16.0kB to 12.2kB for the number of parties N ranging from 16 to 256 respectively. This is much smaller than the average size of 31.6kB estimated for the AES-based signatures. We note that, as the number of parties is increased to achieve lower signature sizes, so does the signing time increase as well, from 2.8ms to 15.7ms respectively.

As no alternative method of computing the Legendre PRF in MPC is considered

```
$\mathtt{Sig}(\mathsf{params}, \mathsf{sk}, m)$—part 2
**Phase 4: Challenge for sacrificing-based verification**
1: Compute challenge hash: $h_2 \leftarrow H_2(h_1, \sigma_2)$.
2: Expand hash $(\epsilon_e, \lambda_e^{(1)}, \ldots, \lambda_e^{(B)})_{e \in [M]} \leftarrow \mathtt{Expand}(h_2)$, where $\epsilon_e, \lambda_e^{(j)} \in \mathbb{Z}_p$.
**Phase 5: Commitment to views of sacrificing protocol**
1: for $e$ from 1 to $M$ do
2:     for $i$ from 1 to $N$ do
3:         Compute shares: $\alpha_{e,i} \leftarrow a_{e,i} + \epsilon_e K_{e,i}$ and $\beta_{e,i} \leftarrow b_{e,i} + \sum_{j=1}^{B} \lambda_e^{(j)} r_{e,i}^{(j)}$.
4:     Compute values: $\alpha_e \leftarrow \sum_{i=1}^{N} \alpha_{e,i}$ and $\beta_e \leftarrow \sum_{i=1}^{N} \beta_{e,i}$.
5:     for $i$ from 1 to $N$ do
6:         Compute product shares: $z_{e,i} \leftarrow \sum_{j=1}^{B} -\lambda_e^{(j)} r_{e,i}^{(j)} I_e^{(j)}$.
7:         if $i \overset{?}{=} 1$ then $z_{e,i} \leftarrow z_{e,i} + \sum_{j=1}^{B} \lambda_e^{(j)} o_e^{(j)}$.
8:         Compute check value shares: $\gamma_{e,i} \leftarrow \alpha_e b_{e,i} + \beta_e a_{e,i} - c_{e,i} + \epsilon_e z_{e,i}$.
9: Set $\sigma_3 \leftarrow (\alpha_e, \beta_e, (\alpha_{e,i}, \beta_{e,i}, \gamma_{e,i})_{i \in [N]})_{e \in [M]}$.
**Phase 6: Challenge on sacrificing protocol**
1: Compute challenge hash $h_3 \leftarrow H_3(h_2, \sigma_3)$.
2: Expand hash $(\bar{i}_e)_{e \in [M]} \leftarrow \mathtt{Expand}(h_3)$, where $\bar{i}_e \in [N]$.
**Phase 7: Opening the views of sacrificing protocol**
1: for $e$ from 1 to $M$ do
2:     $\mathsf{seeds_e} \leftarrow \{\log_2(N)$ nodes in tree needed to compute $\mathsf{sd}_{e,i}$ for $i \in [N] \setminus \bar{i}\}$.
3: Output: $\sigma = (\mathsf{st}, h_1, h_3, (\Delta K_e, \Delta c_e, o_e^{(1)}, \ldots, o_e^{(B)}, \alpha_e, \beta_e, \mathsf{seeds}_e, \mathtt{Com}_{e,\bar{i}_e})_{e \in [M]})$.
```

Figure 10.2: k-th power residue PRF signature scheme—part 2.

in [19], we do not estimate the signature sizes that would result from a naive MPC protocol. We note that the final optimization of computing only the error term E in the MPC protocol, instead of the B output values, already reduces the number of multiplications from B to only 1. As the value of B ranges from 9 to 16, depending on the number of parties, we could expect the non-optimized computation to result in signatures that many times bigger (approximately).

For the higher order power residue PRF, this scheme achieves even lower signature sizes ranging from 8.6kB to 6.3kB for the same range of N. Due to the increased soundness gained from increasing k, less repetitions M are required, and less symbols of the public key need to be checked; both of these advantages contribute to the reduced size and also to reduced computation times, ranging from 1.2ms to 7.9ms.

10.2 Improving AES in MPCitH with S-box verification

We now present a method similar to Section 10.1's to improve the performance of the proof of knowledge of AES key presented in Chapter 9.

```
Vfy(params, pk, $m$, $\sigma$):
1: Parse $\sigma = (\mathsf{st}, h_1, h_3, (\Delta K_e, \Delta c_e, o_e^{(1)}, \dots, o_e^{(B)}, \alpha_e, \beta_e, \mathsf{seeds}_e, \mathtt{Com}_{e,\bar{i}_e})_{e\in[M]})$.
2: Compute $h_2 \leftarrow H_2(h_1, (o_e^{(j)})_{e\in[M],j\in[B]})$.
3: Expand challenge hash 1: $(I_e^{(1)}, \dots, I_e^{(B)})_{e\in[M]} \leftarrow \mathtt{Expand}(h_1)$, where $I_e^{(j)} \in \mathcal{I}$.
4: Expand challenge hash 2: $(\epsilon_e, \lambda_e^{(1)}, \dots, \lambda_e^{(B)})_{e\in[M]} \leftarrow \mathtt{Expand}(h_2)$.
5: Expand challenge hash 3: $(\bar{i}_e)_{e\in[M]} \leftarrow \mathtt{Expand}(h_3)$.
6: for $e$ from 1 to $M$ do
7:     Use $\mathsf{seeds}_e$ to compute $\mathsf{sd}_{e,i}$ for $i \in [N] \setminus \bar{i}_e$.
8:     for $i$ from 1 to $\bar{i}_e - 1$ and from $\bar{i}_e + 1$ to $N$ do
9:         Sample shares: $K_{e,i}, r_{e,i}^{(1)}, \dots, r_{e,i}^{(B)}, a_{e,i}, b_{e,i}, c_{e,i} \leftarrow \mathtt{Expand}(\mathsf{sd}_{e,i})$.
10:        if $i \stackrel{?}{=} 1$ then
11:            Adjust shares: $K_{e,i} \leftarrow K_{e,i} + \Delta K_e$ and $c_{e,i} \leftarrow c_{e,i} + \Delta c_e$.
12:        Recompute commitments: $\mathtt{Com}_{e,i}^* \leftarrow H(\mathsf{st}, e, i, \mathsf{sd}_{e,i})$
13:        Recompute shares: $\alpha_{e,i}^* \leftarrow a_{e,i} + \epsilon_e K_{e,i}$ and $\beta_{e,i}^* \leftarrow b_{e,i} + \sum_{j=1}^{B} \lambda_e^{(j)} r_{e,i}^{(j)}$.
14:        Recompute product shares: $z_{e,i} \leftarrow \sum_{j=1}^{B} -\lambda_e^{(j)} r_{e,i}^{(j)} I_e^{(j)}$.
15:        if $i \stackrel{?}{=} 1$ then
16:            $z_{e,i} \leftarrow z_{e,i} + \sum_{j=1}^{B} \lambda_e^{(j)} o_e^{(j)}$.
17:        Recompute check value shares: $\gamma_{e,i}^* \leftarrow \alpha_e b_{e,i} + \beta_e a_{e,i} - c_{e,i} + \epsilon_e z_{e,i}$.
18:    Compute missing shares: $\alpha_{e,\bar{i}_e}^* \leftarrow \alpha_e - \sum_{i\neq\bar{i}} \alpha_{e,i}^*$ and $\beta_{e,\bar{i}_e}^* \leftarrow \beta_e - \sum_{i\neq\bar{i}} \beta_{e,i}^*$.
19:    Compute missing check value share: $\gamma_{e,\bar{i}_e}^* = \alpha_e\beta_e - \sum_{i\neq\bar{i}} \gamma_{e,i}^*$.
20:    for $j$ from 1 to $B$ do
21:        Recompute residuosity symbols: $s_e^{(j)*} \leftarrow \mathcal{L}_0^k(o_e^{(j)}) - \mathsf{pk}_{I_e^{(j)}}$.
22: Check 1: $h_1 \stackrel{?}{=} H_1(m, \mathsf{st}, ((\mathtt{Com}_{e,i}^*)_{i\in[N]}, (s_e^{(j)*})_{j\in[B]}, \Delta K_e, \Delta c_e)_{e\in[M]})$
23: Check 2: $h_3 \stackrel{?}{=} H_3(h_2, (\alpha_e, \beta_e, (\alpha_{e,i}^*, \beta_{e,i}^*, \gamma_{e,i}^*)_{i\in[N]})_{e\in[M]})$
24: Output accept if both checks pass.
```

Figure 10.3: k-th power residue PRF signature verification.

Recall that the only non-linear operations that the MPC protocol must compute are the application of the AES S-boxes:

$$S : s \mapsto \phi^{-1}\left(\mathbf{A} \cdot \phi\left(s^{-1}\right) + \vec{b}\right),$$

for which, in fact, only the operation $s \mapsto s^{-1}$ requires communication. To follow the approach of the previous section, we let the prover "open" the MPCitH sharing of s to compute $t = s^{-1}$ directly and insert it into the computation. (We assume that $s \neq 0$.) The prover then simulates a verification protocol which checks that every inserted t is indeed the inverse of the corresponding s.

We first detail the verification protocol, proposed by C. Baum and P. Scholl, before formally describing the resulting signature scheme and giving its security proof.

10.2.1 Polynomial-based verification of inverse pairs

Let m denote the number of S-boxes in the circuit for $f_{\boldsymbol{x}}(\boldsymbol{k}) = \mathsf{AES}_{\boldsymbol{k}}(\boldsymbol{x})$ and let $\langle s_1 \rangle, \dots, \langle s_m \rangle$ denote the secret-shared inputs to each of these S-boxes. As it simulates the execution of the circuit, we let the prover open each $\langle s_i \rangle$ to compute $t_i \leftarrow s_i^{-1}$ and then insert $\langle t_i \rangle$ back into the execution by having each party j sample $t_i^{(j)}$ from seed and then inserting a correction value Δt_i to correct $t_i^{(1)}$. The verification protocol then has, as inputs, the sharings $\langle s_i \rangle$ and $\langle t_i \rangle$, for $i \in [m]$, and needs to verify that $s_i \cdot t_i = 1$ for every i.

The method proposed by Baum and Scholl consists in building two polynomials: $S(x)$ from the s_i values, and $T(x)$ from the t_i values, and then verifying that their product $P = S \cdot T$ satisfies the required relation.

The first idea was to set $S(i) \coloneqq s_i$, and thus have S and T be polynomials of degree $m-1$, and then verify that $P(i) = 1$ for $i \in [m]$. This verification is done by enforcing $P(i) = 1$ when interpolating P and then checking that $P(R) = S(R) \cdot T(R)$ for a randomly sampled point R different from all used to interpolate P. The soundness of this check follows from the following lemma.

Lemma 10.4 (Schwartz–Zippel Lemma [129, 115])**.** *Let $Q \in \mathbb{F}[x]$ be a non-zero polynomial of degree $d > 0$. For any finite subset S of $\mathbb{F}$,*

$$\Pr_{r \leftarrow\$ S}[Q(r) = 0] \leq \frac{d}{|S|}.$$

This check was then improved by constructing S and T using random coefficients together with the pairs to be checked. The idea is to instead check m_2 random linear combinations of m_1 elements each, where $m_1 \cdot m_2 = m$. To further increase the soundness of the check, we also lift the values, which belong to $\mathbb{F}_{2^8}$, to the bigger field $\mathbb{F}_{2^{8\rho}}$, for $\rho \geq 2$. We assume that there exists a non-interactive injective homomorphism to lift from $\mathbb{F}_{2^8}$ to $\mathbb{F}_{2^{8\rho}}$. The verfication protocol, in non-secret-shared form, therefore proceeds as follows.

1. Lift $s_1, \dots, s_m$ and $t_1, \dots, t_m$ to $\mathbb{F}_{2^{8\rho}}$.

2. Sample random challenges $r_i \leftarrow \mathbb{F}_{2^{8\rho}}$ for $i \in \{1, \dots, m_1\}$.

3. Compute $s'_{ij} = r_i \cdot s_{i+m_1 j}$ and write $t'_{ij} = t_{i+m_1 j}$ for $i \in \{1, \dots, m_1\}, j \in$

$\{0, \dots, m_2 - 1\}$. Observe that

$$\begin{pmatrix} s'_{1j} \\ \vdots \\ s'_{m_1 j} \end{pmatrix} \begin{pmatrix} t'_{1j} & \cdots & t'_{m_1 j} \end{pmatrix} = \sum_i r_i \quad \text{for } j \in \{0, \dots, m_2 - 1\}. \tag{10.1}$$

These are the m_2 linear combinations to be verified.

4. For $i \in \{1, \dots, m_1\}$ sample $\bar{s}_i, \bar{t}_i \leftarrow\!\!\$\, \mathbb{F}_{2^{8\rho}}$ and define the polynomials S_i, T_i such that $S_i(m_2) = \bar{s}_i, T_i(m_2) = \bar{t}_i$ and $S_i(j) = s'_{ij}, T_i(j) = t'_{ij}$ for $j \in \{0, \dots m_2 - 1\}$. By definition each such polynomial has degree m_2.

5. Let P be the degree $2m_2$ polynomial $P = \sum_i S_i \cdot T_i$. Observe that $P(j) = \sum_i r_i$ for $j \in \{0, \dots, m_2 - 1\}$ as per (10.1).

6. Sample a random challenge $R \leftarrow\!\!\$\, \mathbb{F}_{2^{8\rho}} \setminus \{0, \dots, m_2 - 1\}$.

7. Compute $P(R)$ as well as $S_i(R)$ and $T_i(R)$ for $i \in \{1, \dots, m_1\}$ and verify whether $P(R) = \sum_i S_i(R) \cdot T_i(R)$.

The secret-shared verification protocol

Since polynomial interpolation is linear, parties can interpolate shares of polynomials S_i and T_i from shares of s'_{ij} and t'_{ij}, since the challenges r_i are public. From this, each party can compute their share of $S_i(R)$ and $T_i(R)$ for each i and then reveal it in order to perform the check.

So the only non-linear part is the computation of the polynomial P. Here we let the prover inject the shares for $m_2 + 1$ values of P (as another m_2 values are already specified by the publicly known $P(j) = \sum_i r_i$ for each j) and we rely on the check with the challenge R to capture any cheating. With those $2m_2 + 1$ shares, the parties can locally interpolate a share of P and then compute and reveal their shares of $P(R)$. In summary, each party $\mathcal{P}_k$ performs the following operations.

1. Lift $s_1^{(k)}, \dots, s_m^{(k)}$ and $t_1^{(k)}, \dots, t_m^{(k)}$ to $\mathbb{F}_{2^{8\rho}}$.

2. Receive challenges $r_i \in \mathbb{F}_{2^{8\rho}}$ for $i \in \{1, \dots, m_1\}$.

3. Compute $s'^{(k)}_{ij} = r_i \cdot s^{(k)}_{i+m_1 j}$ and write $t'^{(k)}_{ij} = t^{(k)}_{i+m_1 j}$ for $i \in \{1, \dots, m_1\}, j \in \{0, \dots, m_2 - 1\}$.

4. For $i \in \{1, \dots, m_1\}$ sample $\bar{s}_i^{(k)}, \bar{t}_i^{(k)} \leftarrow_{\$} \mathbb{F}_{2^{8\rho}}$. Interpolate polynomials $S_i^{(k)}$ and $T_i^{(k)}$ such that $S_i^{(k)}(m_2) = \bar{s}_i^{(k)}$, $T_i^{(k)}(m_2) = \bar{t}_i^{(k)}$ and $S_i^{(k)}(j) = \bar{s}_{ij}'^{(k)}$, $T_i^{(k)}(j) = \bar{t}_{ij}'^{(k)}$ for $j \in \{0, \dots, m_2 - 1\}$.

5. Receive $m_2 + 1$ shares $P^{(k)}(j)$ for $j \in \{m_2, \dots, 2m_2\}$ ($j = m_2$ is included as parties cannot compute $\sum_i \bar{s}_i \cdot \bar{t}_i$ locally). For $j \in \{0, \dots, m_2 - 1\}$, if $k = 1$, set $P^{(k)}(j) = \sum_i r_i$; if $k \neq 1$, set $P^{(k)}(j) = 0$. Interpolate $P^{(k)}$ from those $2m_2 + 1$ points.

6. Receive challenge $R \in \mathbb{F}_{2^{8\rho}} \setminus \{0, \dots, m_2 - 1\}$.

7. Compute $P^{(k)}(R), S_i^{(k)}(R)$ and $T_i^{(k)}(R)$ and open all $2m_1 + 1$ values.

Soundness. Assume that each party honestly follows the protocol but there exists $\ell \in [m]$ such that $s_\ell \cdot t_\ell \neq 1$. Since the embedding into $\mathbb{F}_{2^{8\rho}}$ is an injective homomorphism, it must then also hold that $s_i \cdot t_i \neq 1$. Assume that the above protocol succeeds, then one of the following conditions must hold:

1. In Step 2, values $r_1, \dots, r_{m_1}$ were sampled such that the equations from Step 3 hold, despite $s_\ell \cdot t_\ell \neq 1$.

2. In Step 6 a value R was chosen such that $P(R) = \sum_i S_i(R) \cdot T_i(R)$ while $P \neq \sum_i S_i \cdot T_i$.

For the first case, by assumption we have that $\exists i, j$ such that $\sum_i r_i = \sum_i s_{ij}' \cdot t_{ij}'$ while $r_i \neq s_{ij}' \cdot t_{ij}'$. By the choice of r_i this will happen with probability at most $2^{-8\rho}$.

For the second, the polynomials on both sides are of degree $2m_2$ and can have at most $2m_2$ points in common. Following from the Schwartz–Zippel Lemma, the probability of such a value of R being sampled is then at most $2m_2/(2^{8\rho} - m_2)$.

This discussion illustrates the soudness of each check in isolation. We next describe the final protocol and analyse its security in full.

10.2.2 Signature scheme from polynomial-based S-box verification

Notation. In this section, we use e to index executions, i to index parties, ℓ to index S-boxes and $j, k, \dots$ when further indexing is required. Let M denote the number of parallel execution, N denote the total number of parties and m denote the number of S-boxes.

```
Sig(params, sk, m)—part 1
Phase 1: Committing to the seeds and the execution views of the parties.
1: Sample a random salt $\mathsf{st} \leftarrow_\$ \{0,1\}^{2\lambda}$.
2: for each parallel execution $e$ do
3:     Sample a root seed: $\mathsf{sd}_e \leftarrow_\$ \{0,1\}^{\lambda}$.
4:     Compute parties' seeds $\mathsf{sd}_e^{(1)}, \ldots, \mathsf{sd}_e^{(N)}$ as leaves of binary tree with root $\mathsf{sd}_e$.
5:     for each party $i$ do
6:         Commit to seed: $\mathsf{C}_e^{(i)} \leftarrow \mathtt{Commit}(\mathsf{st}, e, i, \mathsf{sd}_e^{(i)})$.
7:         Sample witness share: $\mathsf{sk}_e^{(i)} \leftarrow \mathtt{Expand}(\mathsf{sd}_e^{(i)})$.
8:     Compute witness offset: $\Delta\mathsf{sk}_e \leftarrow \mathsf{sk} - \sum_i \mathsf{sk}_e^{(i)}$.
9:     Adjust first share: $\mathsf{sk}_e^{(1)} \leftarrow \mathsf{sk}_e^{(1)} + \Delta\mathsf{sk}_e$.
10:    for each S-box $\ell$ do
11:        For each party $i$, locally compute the share $s_{e,\ell}^{(i)}$ of the S-box input $s_{e,\ell}$.
12:        Compute the S-box output: $t_{e,\ell} = \left(\sum_i s_{e,\ell}^{(i)}\right)^{-1}$.
13:        For each party $i$, sample the share of the output: $t_{e,\ell}^{(i)} \leftarrow \mathtt{Expand}(\mathsf{sd}_e^{(i)})$.
14:        Compute output offset: $\Delta t_{e,\ell} = t_{e,\ell} - \sum_i t_{e,\ell}^{(i)}$.
15:        Adjust first share: $t_{e,\ell}^{(1)} \leftarrow t_{e,\ell}^{(1)} + \Delta t_{e,\ell}$.
16: Set $\sigma_1 \leftarrow ((\mathsf{C}_e^{(i)})_{i\in[N]}, \Delta\mathsf{sk}_e, (\Delta t_{e,\ell})_{\ell\in[m]})_{e\in[M]}$.
```

Figure 10.4: Phase 1: commitment to executions of AES.

Protocol description. We describe the protocol in phases and give the rationale behind each one. The MPC-in-the-Head computation is divided into two steps: the AES *execution* and then the *verification* of the inverse injections.

Figure 10.4 describes the computation of the first signature component: the commitment to the M parallel executions.

- Line 1 introduces a random salt to add additional randomness into the hash function calls.
- On line 12, we assume that the input is non-zero for every S-box.
- On line 16, the prover commits to the seeds, the injection of sk and the injection of the m inverse values $t_{e,\ell}$ for each execution e. These values suffice to determine uniquely the output values of the distributed AES circuit.

Figure 10.5 describes the computation of the challenge for the randomized inner products and the commitments to the checking polynomial P for each execution.

Figure 10.6 describes the computation of the challenge for the committed checking polynomial and the commitments to the views of the check openings for each execution.

```
$\mathsf{Sig}(\mathsf{params}, \mathsf{sk}, m)$ - part 2
Phase 2: Challenging the multiplications.
1: Compute challenge hash: $h_1 \leftarrow H_1(m, \mathsf{st}, \sigma_1)$.
2: Expand hash: $(r_{e,j})_{e\in[M],j\in[m_1]} \leftarrow \mathtt{Expand}(h_1)$ where $r_{e,j} \in \mathbb{F}_{2^{8\rho}}$.
Phase 3: Committing to the checking polynomials.
1: for each execution $e$ do
2:     for each party $i$ do
3:         Lift $s^{(i)}_{e,\ell}, t^{(i)}_{e,\ell} \hookrightarrow \mathbb{F}_{2^{8\rho}}$ for $\ell \in [m]$.
4:         for $j \in [m_1]$ do
5:             Rearrange shares: $s'^{(i)}_{e,j,k} \leftarrow r_{e,j} \cdot s^{(i)}_{e,j+m_1k}$ and $t'^{(i)}_{e,j,k} \leftarrow t^{(i)}_{e,j+m_1k}$ for $k \in \{0,\ldots,m_2-1\}$.
6:             Sample additional random points: $\bar{s}^{(i)}_{e,j}, \bar{t}^{(i)}_{e,j} \leftarrow \mathtt{Expand}(\mathsf{sd}^{(i)}_e)$.
7:             Define $S^{(i)}_{e,j}(k) = s'^{(i)}_{e,j,k}$ and $T^{(i)}_{e,j}(k) = t'^{(i)}_{e,j,k}$ for $k \in \{0,\ldots,m_2-1\}$, $S^{(i)}_{e,j}(m_2) = \bar{s}^{(i)}_{e,j}$ and $T^{(i)}_{e,j}(m_2) = \bar{t}^{(i)}_{e,j}$.
8:             Interpolate polynomials $S^{(i)}_{e,j}$ and $T^{(i)}_{e,j}$ of degree $m_2$ using defined $m_2+1$ points.
9:     Compute product polynomial: $P_e \leftarrow \sum_j \left(\sum_i S^{(i)}_{e,j}\right) \cdot \left(\sum_i T^{(i)}_{e,j}\right)$.
10:    for each party $i$ do
11:        For $k \in \{0,\ldots,m_2-1\}$, if $i=1$, set $P^{(i)}_e(k) = \sum_j r_{e,j}$; if $i \neq 1$, set $P^{(i)}_e(k) = 0$.
12:        For $k \in \{m_2,\ldots,2m_2\}$, sample $P^{(i)}_e(k) \leftarrow \mathtt{Expand}(\mathsf{sd}^{(i)}_e)$.
13:    for $k \in \{m_2,\ldots,2m_2\}$ do
14:        Compute offset: $\Delta P_e(k) = P_e(k) - \sum_i P^{(i)}_e(k)$.
15:        Adjust first share: $P^{(1)}_e(k) \leftarrow P^{(1)}_e(k) + \Delta P_e(k)$.
16:    For each party $i$, interpolate $P^{(i)}_e$ of degree $2m_2$ using defined $2m_2+1$ points.
17: Set $\sigma_2 \leftarrow \left((\Delta P_e(k))_{k\in\{m_2,\ldots,2m_2\}}\right)_{e\in[M]}$.
```

Figure 10.5: Phases 2 and 3: Computation of randomized inner product checking polynomials.

Figure 10.7 describes the computation of the challenge for and the opening of the views of the verification protocol.

Finally, Figure 10.8 on p. 193 describes the verification of the signature by re-computation of the polynomial shares and checking of the relations.

Security proof

Following the technique used for the signature scheme of Section 10.1, we prove that the scheme presented above achieves EUF-CMA security. We first show that it achieves EUF-KO security and then build a reduction between the two notions.

Theorem 10.5. *In the classical random oracle model, the signature scheme described in Figures 10.4, 10.5, 10.6, 10.7 and 10.8 is EUF-CMA secure under the assumption*

```
Sig(params, sk, m)—part 3
Phase 4: Challenging the checking polynomials.
1: Compute challenge hash: $h_2 \leftarrow H_2(h_1, \sigma_2)$.
2: Expand hash: $(R_e)_{e\in[M]} \leftarrow \mathtt{Expand}(h_2)$ where $R_e \in \mathbb{F}_{2^{s_\rho}} \setminus \{0,\dots,m_2-1\}$.
Phase 5: Committing to the views of the checking protocol.
1: for each execution $e$ do
2:     for each party $i$ do
3:         For $j \in [m_1]$, compute: $a^{(i)}_{e,j} \leftarrow S^{(i)}_{e,j}(R_e)$ and $b^{(i)}_{e,j} \leftarrow T^{(i)}_{e,j}(R_e)$.
4:         Compute: $c^{(i)}_e \leftarrow P^{(i)}_e(R_e)$.
5:     Open $c_e$, and $a_{e,j}$, $b_{e,j}$ for $j \in [m_1]$.
6: Set $\sigma_3 \leftarrow (c_e, (c^{(i)}_e)_{i\in[N]}, (a_{e,j}, b_{e,j}, (a^{(i)}_{e,j}, b^{(i)}_{e,j})_{i\in[N]})_{j\in[m_1]})_{e\in[M]}$.
```

Figure 10.6: Phases 4 and 5: Computation of the views of the randomized check openings.

```
Sig(params, sk, m)—part 4
Phase 6: Challenging the views of the checking protocol.
1: Compute challenge hash: $h_3 \leftarrow H_3(h_2, \sigma_3)$.
2: Expand hash: $(\bar{i}_e)_{e\in[M]} \leftarrow \mathtt{Expand}(h_3)$ where $\bar{i}_e \in [N]$.
Phase 7: Opening the views of the checking protocol.
1: for each execution $e$ do
2:     $\mathsf{sds}_e \leftarrow \{\log_2(N)$ nodes needed to compute $\mathsf{sd}_{e,i}$ for $i \in [N] \setminus \{\bar{i}_e\}\}$.
3: Set $\sigma \leftarrow (\mathsf{st}, h_1, h_3, (\mathsf{sds}_e, \mathsf{C}^{(\bar{i}_e)}_e, \Delta\mathsf{sk}_e, (\Delta t_{e,\ell})_{\ell\in[m]}, (\Delta P_e(k))_{k\in\{m_2,\dots,2m_2\}}, P_e(R_e),$
   $(S_{e,j}(R_e), T_{e,j}(R_e))_{j\in[m_1]})_{e\in[M]})$.
```

Figure 10.7: Phases 6 and 7: Challenging and opening of the views of the checking protocol.

that the chosen AES circuit is a OWF on its key-space.

Proof. The proof follows from Lemma 10.6, for the reduction of the EUF-KO security to the OWF security of $f_{\boldsymbol{x}}(\boldsymbol{k}) = \mathsf{AES}_{\boldsymbol{k}}(\boldsymbol{x})$, and Lemma 10.7, for the reduction of EUF-CMA security to EUF-KO security based on the simulatability of the MPC protocol. □

Lemma 10.6. *Let* Commit, H_1, H_2 *and* H_3 *be modeled as random oracles. If there exists a probabilistic* $\mathsf{poly}(\lambda)$*-time adversary* $\mathcal{A}$ *against the EUF-KO security of the signature scheme that makes* Q_c, Q_1, Q_2 *and* Q_3 *queries to the respective oracles, then there exists a probabilistic* $\mathsf{poly}(\lambda)$*-time algorithm* $\mathcal{B}$ *which, given* $\mathsf{pk} = (\boldsymbol{x}, \boldsymbol{y})$*, outputs a pre-image for* $f_{\boldsymbol{x}}(\boldsymbol{k}) = \boldsymbol{y}$ *with probability at least* $\mathbf{Adv}^{\mathsf{EUF\text{-}KO}}_{\mathcal{A}}(1^\lambda) - \varepsilon(Q_\mathsf{c}, Q_1, Q_2, Q_3)$,

```
$\texttt{Vfy}(\mathsf{params}, \mathsf{pk}, m, \sigma)$
1: Parse $\sigma \leftarrow (\mathsf{st}, h_1, h_3, (\mathsf{sds}_e, \mathsf{C}_e^{(\bar{i}_e)}, \Delta\mathsf{sk}_e, (\Delta t_{e,\ell})_{\ell\in[m]}, (\Delta P_e(k))_{k\in\{m_2,\ldots,2m_2\}}, P_e(R_e), (S_{e,j}(R_e), T_{e,j}(R_e))_{j\in[m_1]})_{e\in[M]})$.
2: Compute $h_2 \leftarrow H_2(h_1, ((\Delta P_e(k))_{k\in\{m_2,\ldots,2m_2\}})_{e\in[M]})$.
3: Expand challenge hash 1: $(r_{e,j})_{e\in[M],j\in[m_1]} \leftarrow \texttt{Expand}(h_1)$.
4: Expand challenge hash 2: $(R_e)_{e\in[M]} \leftarrow \texttt{Expand}(h_3)$.
5: Expand challenge hash 3: $(\bar{i}_e)_{e\in[M]} \leftarrow \texttt{Expand}(h_3)$.
6: for each execution $e$ do
7:     Use $\mathsf{sds}_e$ to compute $\mathsf{sd}_e^{(i)}$ for $i \in [N] \setminus \bar{i}_e$.
8:     for each party $i \in [N] \setminus \bar{i}_e$ do
9:         Recompute commitment: $\mathsf{C}_e^{(i)\prime} \leftarrow \texttt{Commit}(\mathsf{st}, e, i, \mathsf{sd}_e^{(i)})$.
10:        Sample witness share: $sk_e^{(i)} \leftarrow \texttt{Expand}(\mathsf{sd}_e^{(i)})$.
11:        if $i \overset{?}{=} 1$ then
12:            Adjust first share: $\mathsf{sk}_e^{(i)} \leftarrow \mathsf{sk}_e^{(i)} + \Delta\mathsf{sk}_e$.
13:        for each S-box $\ell$ do
14:            Compute local linear operations to obtain $s_{e,\ell}^{(i)}$.
15:            Sample output share: $t_{e,\ell}^{(i)} \leftarrow \texttt{Expand}(\mathsf{sd}_e^{(i)})$.
16:            if $i \overset{?}{=} 1$ then
17:                Adjust first share: $t_{e,\ell}^{(i)} \leftarrow t_{e,\ell}^{(i)} + \Delta t_{e,\ell}$.
18:        Do as in Phase 3, lines 3–8 to interpolate $S_{e,j}^{(i)\prime}, T_{e,j}^{(i)\prime}$ for $j \in [m_1]$.
19:        for $k$ from 0 to $m_2 - 1$ do
20:            If $i \overset{?}{=} 1$, set $P_e^{(i)}(k)' = \sum_j r_{e,j}$; otherwise set $P_e^{(i)}(k)' = 0$.
21:        for $k$ from $m_2$ to $2m_2$ do
22:            Sample share: $P_e^{(i)}(k)' \leftarrow \texttt{Expand}(\mathsf{sd}_e^{(i)})$.
23:            if $i \overset{?}{=} 1$ then
24:                Adjust first share: $P_e^{(i)}(k)' \leftarrow P_e^{(i)}(k)' + \Delta P_e(k)$.
25:        Interpolate $P_e^{(i)\prime}$.
26:        For $j \in [m_1]$, compute $a_{e,j}^{(i)\prime} \leftarrow S_{e,j}^{(i)\prime}(R_e)$ and $b_{e,j}^{(i)\prime} \leftarrow T_{e,j}^{(i)\prime}(R_e)$.
27:        Compute: $c_e^{(i)\prime} \leftarrow P_e^{(i)\prime}(R_e)$.
28:    Compute missing shares:
           $$a_{e,j}^{(\bar{i}_e)\prime} \leftarrow S_{e,j}(R_e) - \sum_{i\neq\bar{i}_e} a_{e,j}^{(i)\prime}, \quad b_{e,j}^{(\bar{i}_e)\prime} \leftarrow T_{e,j}(R_e) - \sum_{i\neq\bar{i}_e} b_{e,j}^{(i)\prime}, \text{for } j \in [m_1]$$
           $$c_e^{(\bar{i}_e)\prime} \leftarrow P_e(R_e) - \sum_{i\neq\bar{i}_e} c_e^{(i)\prime}$$
29: Set $h_1' \leftarrow H_1(\mathsf{st}, ((\mathsf{C}_e^{(i)\prime})_{i\in[N]}, \Delta\mathsf{sk}_e, (\Delta t_{e,\ell})_{\ell\in[m]})_{e\in[M]})$, with $\mathsf{C}_e^{(\bar{i}_e)\prime} = \mathsf{C}_e^{(\bar{i}_e)}$.
30: Set $h_3' \leftarrow H_3(h_2, (P_e(R_e), (c_e^{(i)})_{i\in[N]}, (S_{e,j}(R_e), T_{e,j}(R_e), (a_{e,j}^{(i)}, b_{e,j}^{(i)})_{i\in[N]})_{j\in[m_1]})_{e\in[M]})$.
31: Check $h_1' \overset{?}{=} h_1$, $h_3' \overset{?}{=} h_3$ and
           $$P_e(R_e) \overset{?}{=} \sum_j S_{e,j}(R_e) \cdot T_{e,j}(R_e).$$
32: Output accept if all three checks pass.
```

Figure 10.8: Verification algorithm.

Algorithm 5 $\texttt{Commit}(q_{\mathsf{c}} = (\mathsf{st}, e, i, \mathsf{sd}))$:

```
1: $x \leftarrow_{\$} \{0,1\}^{2\lambda}$.
2: if $x \in \mathsf{Bad}$ then abort.                    ▷ Check if $x$ is fresh.
3: $x \rightarrow \mathsf{Bad}$.
4: $(q_{\mathsf{c}}, x) \rightarrow \mathcal{Q}_{\mathsf{c}}$.
5: Return $x$.
```

with

$$\varepsilon(Q_{\mathsf{c}}, Q_1, Q_2, Q_3) = \frac{(MN+1)(Q_{\mathsf{c}} + Q_1 + Q_2 + Q_3)^2}{2^{2\lambda}} + \Pr[X + Y + Z = M]$$

where $X = \max_{q_1 \in \mathcal{Q}_1}\{X_{q_1}\}$ *with* $X_{q_1} \sim \mathfrak{B}(M, 1/2^{8\rho})$, $Y = \max_{q_2 \in \mathcal{Q}_2}\{Y_{q_2}\}$ *with* $Y_{q_2} \sim \mathfrak{B}(M - X, 2m_2/(2^{8\rho} - m_2))$ *and* $Z = \max_{q_3 \in \mathcal{Q}_3}\{Z_{q_3}\}$ *with* $Z_{q_3} \sim \mathfrak{B}(M - X - Y, 1/N)$, *where* $\mathfrak{B}(n, p)$ *denotes the binomial probability distribution with* n *samples each with probability* p *of success.*

Proof. We build a probabilistic $\mathsf{poly}(\lambda)$-time algorithm $\mathcal{B}$ which uses a EUF-KO adversary $\mathcal{A}$ to compute a pre-image for $f_{\boldsymbol{x}}$.

The reduction $\mathcal{B}$ simulates the EUF-KO game using the random oracles $\texttt{Commit}$, H_1, H_2 and H_3 and query lists $\mathcal{Q}_{\mathsf{c}}$, $\mathcal{Q}_1$, $\mathcal{Q}_2$ and $\mathcal{Q}_3$. In addition, $\mathcal{B}$ also maintains three tables $T_{\mathsf{sh}}, T_{\mathsf{in}}$ and T_{op} to store the shares, inputs and openings that it is able to recover from $\mathcal{A}$'s queries to the random oracles.

Behaviour of the reduction. The reduction $\mathcal{B}$ receives a OWF challenge $(\boldsymbol{x}, \boldsymbol{y})$ and forwards it to $\mathcal{A}$ as the public key of the signature scheme. It lets $\mathcal{A}$ run and answers its random oracle queries in the following way.

- $\texttt{Commit}$: When $\mathcal{A}$ queries the commitment random oracle, $\mathcal{B}$ records the query to learn which commitment corresponds to which seed. See Algorithm 5.
- H_1 : When $\mathcal{A}$ commits to seeds and sends the offsets for the secret key and the inverse values, $\mathcal{B}$ checks whether the commitments were output by its simulation of $\texttt{Commit}$. If any were for some e and i, then $\mathcal{B}$ is able to reconstruct the shares for party i in execution e. If $\mathcal{B}$ was able to reconstruct every party's share for any e, then it can use the offsets included in σ_1 to extract the values used by $\mathcal{A}$ in that execution. See Algorithm 6.
- H_2 : When $\mathcal{A}$ commits to the checking polynomials, $\mathcal{B}$ checks whether the challenge h_1 that $\mathcal{A}$ uses is one output by its simulation of H_1. If it is, then $\mathcal{B}$ uses

Algorithm 6 $H_1(q_1 = (\mathsf{st}, \sigma_1))$:

```
    Read $\sigma_1$ as $((C_e^{(i)})_i, \Delta\mathsf{sk}_e, (\Delta t_{e,\ell})_\ell)_e$.
 1: for $e \in [M], i \in [N]$ do $\mathsf{C}_e^{(i)} \to \mathsf{Bad}$.
 2: for $(e,i) \in [M] \times [N] : \exists \mathsf{sd}_e^{(i)} : ((\mathsf{st}, e, i, \mathsf{sd}_e^{(i)}), \mathsf{C}_e^{(i)}) \in \mathcal{Q}_\mathsf{c}$ do
 3:     $\mathsf{sk}_e^{(i)}, (t_{e,\ell}^{(i)})_\ell \leftarrow \texttt{Expand}(\mathsf{sd}_e^{(i)})$.
 4:     if $i \overset{?}{=} 1$ then $\mathsf{sk}_e^{(i)} \leftarrow \mathsf{sk}_e^{(i)} + \Delta\mathsf{sk}_e$ and $(t_{e,\ell}^{(i)})_\ell \leftarrow (t_{e,\ell}^{(i)} + \Delta t_{e,\ell})_\ell$.
 5:     $(\mathsf{sk}_e^{(i)}, (t_{e,\ell}^{(i)})_\ell) \to T_\mathsf{sh}[q_1, e, i]$.
 6: for each $e : \forall i, T_\mathsf{sh}[q_1, e, i] \neq \emptyset$ do
 7:     $\mathsf{sk}_e \leftarrow \sum_i \mathsf{sk}_e^{(i)}$ and $(t_{e,\ell})_\ell \leftarrow (\sum_i t_{e,\ell}^{(i)})_\ell$.
 8:     $(\mathsf{sk}_e, (t_{e,\ell})_\ell) \to T_\mathsf{in}[q_1, e]$.
 9: $x \leftarrow_\$ \{0,1\}^{2\lambda}$.
10: if $x \in \mathsf{Bad}$ then abort.
11: $x \to \mathsf{Bad}$.
12: $(q_1, x) \to \mathcal{Q}_1$.
13: Return $x$.
```

Algorithm 7 $H_2(q_2 = (h_1, \sigma_2))$:

```
    Read $\sigma_2$ as $(\Delta P_e(k))_{k,e}$.
 1: $h_1 \to \mathsf{Bad}$.
 2: $x \leftarrow_\$ \{0,1\}^{2\lambda}$.
 3: if $x \in \mathsf{Bad}$ then abort.
 4: $x \to \mathsf{Bad}$.
 5: if $\exists q_1^* : (q_1^*, h_1) \in \mathcal{Q}_1$ then
 6:     $(r_{e,j})_{e,j} \leftarrow \texttt{Expand}(h_1)$ and $(R_e)_e \leftarrow \texttt{Expand}(x)$.
 7:     for each $e : T_\mathsf{in}[q_1^*, e] \neq \emptyset$ do
 8:         $(P_e(R_e), (S_{e,j}(R_e), T_{e,j}(R_e))_j)_e \to T_\mathsf{op}[q_2, e]$.
 9: $(q_2, x) \to \mathcal{Q}_2$.
10: Return $x$.
```

h_1 and the newly sampled H_2 response to expand the challenges and extract the checking polynomials. See Algorithm 7.

- H_3 : No extraction takes place during this simulation. See Algorithm 8.

When $\mathcal{A}$ terminates, $\mathcal{B}$ checks the T_in table for any entry where the extracted sk_e is a valid secret key for the public key that it received. If one is, $\mathcal{B}$ wins the secret key recovery game; if no entry is satisfactory, $\mathcal{B}$ outputs $\perp$.

Advantage of the reduction. Given the behaviour presented above, we have the fol-

Algorithm 8 $H_3(q_3 = (h_2, \sigma_3))$:

```
1: $h_2 \to \mathsf{Bad}$.
2: $x \leftarrow_\$ \{0,1\}^{2\lambda}$.
3: if $x \in \mathsf{Bad}$ then abort.
4: $x \to \mathsf{Bad}$.
5: $(q_3, x) \to \mathcal{Q}_3$.
6: Return $x$.
```

lowing by the law of total probability:

$$\begin{aligned}\Pr[\mathcal{A} \text{ wins}] =& \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{B} \text{ aborts}] + \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{B} \text{ outputs } \bot] \\ &+ \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{B} \text{ outputs witness}] \\ \leq& \Pr[\mathcal{B} \text{ aborts}] + \Pr[\mathcal{A} \text{ wins} \mid \bot] + \Pr[\mathcal{B} \text{ outputs witness}]. \qquad (10.2)\end{aligned}$$

Given the way in which values are added to Bad, we have:

$$\begin{aligned}\Pr[\mathcal{B} \text{ aborts}] &= (\#\text{times an } x \text{ is sampled}) \cdot \Pr[\mathcal{B} \text{ aborts at that sample}] \\ &\leq (Q_\mathsf{c} + Q_1 + Q_2 + Q_3) \cdot \frac{\max |\mathsf{Bad}|}{2^{2\lambda}} \\ &= (Q_\mathsf{c} + Q_1 + Q_2 + Q_3) \cdot \frac{Q_\mathsf{c} + (MN+1)Q_1 + 2Q_2 + 2Q_3}{2^{2\lambda}} \\ &\leq \frac{(MN+1)(Q_\mathsf{c} + Q_1 + Q_2 + Q_3)^2}{2^{2\lambda}}. \qquad (10.3)\end{aligned}$$

Where Q_c, Q_1, Q_2 and Q_3 denote the number of queries made by $\mathcal{A}$ to each respective random oracle.

We now analyze the probability of $\mathcal{A}$ winning the EUF-KO experiment conditionning on the event that $\mathcal{B}$ ouputs $\bot$, i.e. that no suitable witness (that is, pre-image to $f_{\boldsymbol{x}}(\cdot) = \boldsymbol{y}$) was found on the query lists.

Cheating in the first round. For any query $q_1 \in \mathcal{Q}_1$, and its corresponding answer $h_1 = (r_{e,j})_{e\in[M],j\in[m_1]}$, let $G_1(q_1, h_1)$ be the set of indices $e \in [M]$ of "good executions" where both $T_\mathsf{in}[q_1, e] = (\mathsf{sk}_e, (t_{e,\ell})_{\ell\in[m]})$ is non-empty and the equations

$$\begin{pmatrix} s'_{e,1k} \\ \vdots \\ s'_{e,m_1k} \end{pmatrix} \begin{pmatrix} t'_{e,1k} & \cdots & t'_{e,m_1k} \end{pmatrix} = \sum_j r_{e,j} \quad \text{for } k \in \{0, \ldots, m_2 - 1\} \qquad (10.4)$$

hold, where the s'_e and t'_e values are derived from sk_e the extracted $t_{e,\ell}$ values and the challenge $r_{e,j}$ values. For any such good execution $e \in G_1(q_1, h_1)$, since $\mathcal{B}$ outputs $\perp$, then sk_e cannot be a valid pre-image for $f_{\boldsymbol{x}}$. However, since $\mathcal{A}$ wins nonetheless, then there must exist at least one $\ell \in [m]$ such that $s_{e,\ell} \cdot t_{e,\ell} \neq 1$; this then implies that the challenge values $r_{e,j}$ were sampled such that Equation (10.4) held. This happens with probability at most $1/2^{8\rho}$.

As the response $h_1 = (r_{e,j})_{e\in[M],j\in[m_1]}$ is sampled uniformly at random, each $e \in [M]$ has the same independent probability of being in $G_1(q_1, h_1)$, given that $\mathcal{B}$ outputs $\perp$. We therefore have that $\#G_1(q_1, h_1) \mid_\perp \sim X_{q_1}$ where $X_{q_1} = \mathfrak{B}(M, 1/2^{8\rho})$. Letting $(q_{\mathsf{best}_1}, h_{\mathsf{best}_1})$ denote the query-response pair which maximizes $\#G_1(q_1, h_1)$, we then have that $\#G_1(q_{\mathsf{best}_1}, h_{\mathsf{best}_1}) \mid_\perp \sim X = \max_{q_1\in\mathcal{Q}_1}\{X_{q_1}\}$.

Cheating in the second round. For any query $q_2 = (h_1, \sigma_2) \in \mathcal{Q}_2$, and its corresponding answer $h_2 = (R_e)_{e\in[M]}$, if there exists q_1 such that $(q_1, h_1) \in \mathcal{Q}_1$, let $G_2(q_2, h_2)$ be the set of indices $e \in [M]$ of "good executions" where $T_{\mathsf{op}}[q_2, e] = (P_e(R_e), (S_{e,j}(R_e), T_{e,j}(R_e))_{j\in[m_1]})$ is non-empty, and it holds that

$$P_e(R_e) = \sum_{j=1}^{m_1} S_{e,j}(R_e) \cdot T_{e,j}(R_e). \tag{10.5}$$

If there does not exist such a q_1, let $G_2(q_2, h_2) = \emptyset$. Once again, for any such good execution $e \in G_2(q_2, h_2)$, since $\mathcal{B}$ outputs $\perp$ but $\mathcal{A}$ wins nonetheless, this implies that either the challenges in the first round were such that Equation (10.4) held (in which case any value of R_e passes the check), or the challenge R_e was sampled such that Equation (10.5) held. Conditioning on the first event not happening, the Schwartz–Zippel Lemma gives us that the second happens with probability at most $\frac{2m_2}{2^{8\rho}-m_2}$, given that h_2 is sampled uniformily at random.

We therefore have that $\#G_2(q_2, h_1) \mid_{\perp\wedge G_1(q_1,h_1)=M_1} \sim M_1 + Y_{q_2}$ where $Y_{q_2} = \mathfrak{B}(M - M_1, 2m_2/(2^{8\rho} - m_2))$. By taking M_1 as a random variable, maximized by $(q_{\mathsf{best}_1}, h_{\mathsf{best}_1})$ and letting $(q_{\mathsf{best}_2}, h_{\mathsf{best}_2})$ denote the query-response pair which maximizes $\#G_2(q_2, h_2) \mid \perp \wedge \#G_1(q_{\mathsf{best}_1}, h_{\mathsf{best}_1})$, we then have that

$$\#G_2(q_{\mathsf{best}_2}, h_{\mathsf{best}_2}) \mid_\perp \sim X + Y,$$

where X is as above, $Y = \max_{q_2\in\mathcal{Q}_2} Y_{q_2}$ and the Y_{q_2} are independently and identically distributed as $\mathfrak{B}(M - X, 2m_2/(2^{8\rho} - m_2))$.

Cheating in the third round. Similarly to the proof of the same stage in [19], each third round query $q_3 = (h_2, \sigma_3)$ that $\mathcal{A}$ makes to H_3 can only be used in a winning signature if there exists a corresponding query $(q_2, h_2) \in \mathcal{Q}_2$. Then for each "bad" second-round execution $e \in [M] \setminus G_2(q_2, h_2)$, either the verification protocol failed, in which case $\mathcal{A}$ couldn't have won, or the verification protocol passed, despite Equation (10.5) not being satisfied. This implies that exactly one of the parties must have cheated during the MPC execution of the verification protocol. Less than one and the verification protocol would have failed; more than one and the verification of the signature would have failed.

Since the third-round challenge $h_3 \in [N]^M$ is chosen uniformly at random, the probability that this happens for all such "bad" second-round executions e is

$$\left(\frac{1}{N}\right)^{M-\#G_2(q_2,h_2)} \le \left(\frac{1}{N}\right)^{M-\#G_2(q_{\mathsf{best}_2},h_{\mathsf{best}_2})}.$$

The probability that this happens for at least one of the Q_3 queries made to H_3 is

$$\Pr[\mathcal{A} \text{ wins} \mid \#G_2(q_{\mathsf{best}_2}, h_{\mathsf{best}_2}) = M_2] \le 1 - \left(1 - \left(\frac{1}{N}\right)^{M-M_2}\right)^{Q_3}.$$

Finally conditioning on $\mathcal{B}$ outputting $\perp$ and summing over all values of M_2, we have that

$$\Pr[\mathcal{A} \text{ wins} \mid \perp] \le \Pr[X + Y + Z = M] \tag{10.6}$$

where X and Y are as before and $Z = \max_{q_3 \in \mathcal{Q}_3}\{Z_{q_3}\}$ where the Z_{q_3} variables are independently and identically distributed as $\mathfrak{B}(M - X - Y, 1/N)$.

Conclusion. Bringing (10.2), (10.3) and (10.6) together, we obtain the following.

$$\begin{aligned}\Pr[\mathcal{A} \text{ wins}] \le\ & \frac{(MN+1)(Q_{\mathsf{c}} + Q_1 + Q_2 + Q_3)^2}{2^{2\lambda}} + \Pr[X + Y + Z = M] \\ & + \Pr[\mathcal{B} \text{ outputs witness}]\end{aligned}$$

This finally gives us the relation

$$\mathbf{Adv}^{\mathsf{OWF}}_{\mathcal{B},(f,\boldsymbol{x},\boldsymbol{y})}(1^\lambda) \ge \mathbf{Adv}^{\mathsf{EUF\text{-}KO}}_{\mathcal{A}}(1^\lambda) - \varepsilon(Q_{\mathsf{c}}, Q_1, Q_2, Q_3),$$

which concludes the proof. □

Lemma 10.7. *Modelling* `Commit` *as a random oracle, if the signature scheme presented above is EUF-KO-secure, then it is EUF-CMA secure.*

Proof. (This proof is modelled on the proof of [19, Lemma 2].) Let $\mathcal{A}$ be an adversary against the EUF-CMA security of the scheme, we construct an adversary $\mathcal{B}$ against its EUF-KO security. When $\mathcal{B}$ is run on input pk, it starts $\mathcal{A}$ also on input pk. We first describe how $\mathcal{B}$ deals with random oracle queries and signature queries, then argue that its signature simulations are indistinguishable from real ones, and finally show that EUF-KO security implies EUF-CMA security.

Random oracle responses. When $\mathcal{A}$ queries one of its random oracles, $\mathcal{B}$ first checks if that query has been recorded before. If it has, it responds to $\mathcal{A}$ with the corresponding recorded answer. If not, $\mathcal{B}$ forwards the query to its corresponding random oracle, records the query and the answer it receives and forwards the answer back to $\mathcal{A}$.

Signature simulation. When $\mathcal{A}$ queries the signing oracle, $\mathcal{B}$ simulates a signature σ by sampling a random witness and cheating in the MPC verification phase to hide the fact it has sampled the witness as random. It then programs the last random oracle to always hide the party for which it has cheated. Formally, the reduction $\mathcal{B}$ proceeds as follows:

1. Phase 1: to simulate σ_1, $\mathcal{B}$ follows Phase 1 as in the scheme with two differences. First, for each $e \in [M]$, it samples $\Delta\mathsf{sk}_e$ uniformly, effectively sampling sk_e at random. Second, when committing to the seeds using `Commit`, $\mathcal{B}$ aborts if any of its queries has already been made to the commitment oracle.

2. $\mathcal{B}$ receives $h_1 \leftarrow H_1(m, \mathsf{st}, \sigma_1)$, expands it, and continues.

3. To simulate the other phases, $\mathcal{B}$ first samples h_2 at random, providing it with challenges $(R_e)_{e\in[M]}$ for Phase 5. It will use these values to cheat on the sampling of the ΔP_e values

4. For each execution e, $\mathcal{B}$ simulates the interpolation of $S_{e,j}^{(i)}$ and $T_{e,j}^{(i)}$ as in the protocol.

5. Instead of computing the honest product polynomial, $\mathcal{B}$ first computes $P_e(k) = \sum_i P_e^{(i)}(k)$ as in the protocol only for $k \in \{0, \dots, m_2 - 1\}$. For $k \in \{m_2, \dots, 2m_2 - 1\}$, the reduction samples the shares as in the protocol, and the offsets at random. Intead of sampling the final value $P_e(2m_2)$ at random, $\mathcal{B}$ first computes $P_e(R) = \sum_j \left(\sum_i S_{e,j}^{(i)}(R)\right) \cdot \left(\sum_i T_{e,j}^{(i)}(R)\right)$. Using this as the last interpolation value, $\mathcal{B}$ now computes P_e and evaluates $P_e(2m_2)$. It then computes the offset $\Delta P_e(2m_2) \leftarrow P_e(2m_2) - \sum_i P_e^{(i)}(2m_2)$. In summary, P_e determined by the

offsets is a random polynomial with the restriction that it statisfies both checking relations.

6. Given the simulated $\tilde{\sigma}_2$ computed as above, $\mathcal{B}$ now programs the random oracle H_2 to output the value of h_2 it sampled as answer to the query $(h_1, \tilde{\sigma}_2)$. If that query has already been made, $\mathcal{B}$ aborts.

7. Since the offsets where computed such that the relation between P_e, $S_{e,j}$ and $T_{e,j}$ is always satisfied, no cheating is required in the third round and $\mathcal{B}$ computes Phases 5 to 7 as in the protocol to produce a simulated signature.

Finally, when $\mathcal{A}$ outputs a forgery for its EUF-CMA game, $\mathcal{B}$ forwards it as its forgery for the EUF-KO game.

Simulation indistinguishability. We next show that $\mathcal{B}$'s simulation of a signature is indistinguishable from a real one. When verifying a signature, $\mathcal{A}$ will see all but one seeds and therefore the distribution of sk_e, remains perfectly uniform *as long as* $\mathcal{A}$ obtains no information regarding the last seed $\mathsf{sd}_e^{(\bar{i}_e)}$.

As we model the commitment scheme as a random oracle the commitment $\mathsf{C}_e^{(\bar{i}_e)}$ included in the signature does not reveal any information about $\mathsf{sd}_e^{(\bar{i}_e)}$ to $\mathcal{A}$. This implies that the real product polynomial P_e is indistinguishable from random in the view of $\mathcal{A}$ since it cannot compute the real polynomials $S_{e,j}$ and $T_{e,j}$. Furthermore, since the distribution of the sampled R_e is uniformly random, so is the distribution of the computed offset $\Delta P_e(2m_2)$. Therefore $\mathcal{B}$'s simulation of phases 1 and 3, by producing uniformly random $\Delta\mathsf{sk}_e$ and $(\Delta P_e(k))_{k\in\{m_2,\ldots,2m_2\}}$, produce σ_1 and σ_2 that are indistinguishable from the output of a real signer. Finally, σ_3 is computed honestly exactly as in the scheme and contains no information about $\mathsf{sd}_e^{(\bar{i}_e)}$ as argued above. We can conclude that $\mathcal{B}$'s simulation of the signing oracle is indistinguishable and that therefore $\mathcal{A}$ behaves exactly as in a normal EUF-CMA game.

EUF-KO security implies EUF-CMA security. Finally, we establish $\mathcal{B}$'s advantage against the EUF-KO security game. There are two moments at which $\mathcal{B}$ could abort: phases 1 and 4. For each of these, $\mathcal{B}$ aborts only if the query it makes to the respective random oracle has been made previously. Let Q_{c} and Q_2 denote the total number of queries made to `Commit` and H_2 respectively by $\mathcal{A}$ during its executions, including those made by $\mathcal{B}$ as part of $\mathcal{A}$'s signing queries. Each time $\mathcal{B}$ attempts a query to `Commit`, it has a maximum probability of $Q_{\mathsf{c}}/2^{2\lambda}MN$ of aborting. Similarly for H_2, $\mathcal{B}$ has a maximum probability of $Q_2/2^{2\lambda}$ of aborting. Given that $\mathcal{B}$ makes $M \cdot N$ queries to `Commit` and one query to H_2 for each signature query, we have that

$$\begin{aligned}\Pr[\mathcal{B} \text{ aborts}] &\leq Q_s \cdot \left(\frac{MNQ_{\mathsf{c}}}{2^{2\lambda}MN} + \left(1 - \frac{MNQ_{\mathsf{c}}}{2^{2\lambda}MN}\right)\frac{Q_2}{2^{2\lambda}}\right) \\ &\leq \frac{Q_s\left(Q_{\mathsf{c}} + Q_2\right)}{2^{2\lambda}}\end{aligned}$$

where Q_s denotes the number of signing queries made by $\mathcal{A}$. As $\mathcal{A}$ can only make a total amount of queries polynomial in λ, we have that $\Pr[\mathcal{B} \text{ aborts}] \leq \mathsf{negl}(\lambda)$ asymptotically.

Given that the winning conditions for the EUF-KO and EUF-CMA games are identical, if $\mathcal{A}$ is a successful EUF-CMA adversary, then $\mathcal{B}$ is a successful EUF-KO adversary as long as it does not abort. This implies that

$$\begin{aligned}\mathbf{Adv}_{\mathcal{B}}^{\mathsf{EUF\text{-}KO}} &\geq \Pr[\mathcal{B} \text{ wins} | \neg\mathcal{B} \text{ aborts}] \cdot \Pr[\neg\mathcal{B} \text{ aborts}] \\ &= \Pr[\mathcal{A} \text{ wins } \mathsf{EUF\text{-}CMA}] \cdot (1 - \Pr[\mathcal{B} \text{ aborts}]) \\ &\geq \mathbf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}} \cdot \left(1 - \frac{Q_s(Q_{\mathsf{c}} + Q_2)}{2^{2\lambda}}\right)\end{aligned}$$

where Q_s denotes the number of signing queries made by $\mathcal{A}$. Therefore, if $\mathcal{A}$ has a non-negligible advantage against the EUF-CMA game, then $\mathcal{B}$ has non-negligible advantage against the EUF-KO game as long as $\frac{Q_s(Q_{\mathsf{c}}+Q_2)}{2^{2\lambda}}$ remains negligible in λ, contradicting the assumption that the scheme is EUF-KO-secure; hence it is also EUF-CMA secure. □

Parameters and signature size estimates

Finally we present parameters and proof size estimates which we compare to the estimates for the first method of Chapter 9.

After fixing the security parameter λ, the first consideration is to ensure that the cheating probability of a dishonest prover is negligible. The first summand of the definition of $\varepsilon(Q_{\mathsf{c}}, Q_1, Q_2, Q_3)$ in Lemma 10.6 will always be negligible as long as the number of queries made to each of the random oracles remains polynomial in λ. For the second summand, we must compute the appropriate parameters M, N, m_2 and ρ such that $\Pr[X + Y + Z = M] < 2^{-\lambda}$.

To decide m_2 (recall that $m = m_2 \cdot m_1$ is the number of S-boxes in the function $f_{\boldsymbol{x}}(\boldsymbol{k})$), we first state the formula for estimating the size of a signature. Looking at

AES-	128	192	256
m	200	416	500
m_2	20	26	25
m_1	10	16	20

Table 10.1: Chosen factorizations of the number m of S-boxes.

the signature σ output by Phase 7, we have

$$|\sigma| = 6\lambda + M \cdot (\lambda(\log_2(N) + 3) + 8m + 8\rho(m_2 + 2m_1 + 2)) \text{ bits.}$$

From this we see that the factor m_1 weighs twice as much as m_2 in the final proof size. Taking into account this imbalance, we therefore factorize m as shown in Table 10.1 (note that for AES-192 and AES-256 we use the same ECB mode combination as was presented in Chapter 9).

We initially set $N = 16$ to remain consistent with the latest version of the Picnic algorithm[5] but then also consider $N = 64$ and $N = 256$ to observe the effect on the number of repetitions required and on the final signature size estimate.

To set M and ρ, we follow the strategy of [84] where we consider that an attacker can re-randomize the first two challenge computations in the hope of obtaining good challenge values for which it does not have to cheat. For the first round, we have

$$P_1(M_1) = \Pr[\text{at least } M_1 \text{ good out of } M] = \sum_{i=M_1}^{M} \left(\frac{1}{2^{8\rho}}\right)^i \left(1 - \frac{1}{2^{8\rho}}\right)^{M-i} \binom{M}{i};$$

for the second round we have

$$\begin{aligned} P_2(M_1, M_2) &= \Pr[\text{at least } M_2 \text{ good out of } M - M_1] \\ &= \sum_{i=M_2}^{M-M_1} \left(\frac{2m_2}{2^{8\rho} - m_2}\right)^i \left(1 - \frac{2m_2}{2^{8\rho} - m_2}\right)^{M-M_1-i} \binom{M - M_1}{i}. \end{aligned}$$

For the third round, the prover then has to cheat on one of the parties when opening the shares, this will not be discovered if exactly these parties are selected for the

[5]The `Picnic3` specification document is available at `https://raw.githubusercontent.com/microsoft/Picnic/master/spec/spec-v3.0.pdf`, last retrieved September 7, 2020.

Security	N	AES-λ	m_2	ρ	M	AES verif.	AES exec.	Picnic3
L1	16	128	20	4	41	19.31	40.0	12.56
	64			4	31	15.59	31.6	11.41
	256			6	21	13.28	–	–
L3	16	192	26	4	62	50.03	110.1	27.55
	64			6	40	38.89	86.9	–
	256			6	32	33.42	–	–
L5	16	256	25	4	84	81.56	171.9	48.72
	64			6	54	62.94	133.7	–
	256			6	43	52.85	–	–

Table 10.2: Parameters and corresponding proof sizes (in kB) for the verification-based AES circuits (estimates) with comparison with Picnic3 [85, Table 7] (averages) and execution-based AES circuits (estimates).

remaining executions. This happens with probability

$$P_3(M_1, M_2) = \left(\frac{1}{n}\right)^{M-M_1-M_2}.$$

If we consider that each re-randomization has unit cost, then the total cost of the attack is

$$\mathsf{cost} = \frac{1}{P_1(M_1)} + \frac{1}{P_2(M_1, M_2)} + \frac{1}{P_3(M_1, M_2)}$$

Fixing a value of ρ, for each value of M there exist values M_1^* and M_2^* which minimize this cost. We therefore present in Table 10.2 our computed values of M and ρ for which the minimum possible cost is greater than 2^λ and for which the signature size is as small as possible. Without specifying other parameters, other than the number N of parties, we also include figures for the execution-based AES circuit of Chapter 9 and for the `Picnic3` scheme. We see that the new verification-based proofs are consistenly less than half the size of the execution-based ones, accross different security levels and number of parties. Table 10.2 also shows that the AES-based proofs now reduce the gap with the state-of-the-art Picnic3 ones, and almost closes it when using $N = 256$ parties, albeit at an (anticipated) significant cost to computing time.

Chapter 11

Conclusion and future work

As illustrated by the diversity of content of this thesis, the field of post-quantum cryptography has cast a wide net in search of new techniques and constructions. Here we discussed three distinct directions and presented contributions in each under the umbrella of post-quantum security. We now summarize these and discuss potential future research questions for each of them.

Authenticated key-exchange

In Part I of this work we identified gaps in previous definitions of authentication security for AKE protocols, proposed remedies, and presented a new transformation for key-exchange (KE) protocols. Chapter 3 presented a formulation of the Bellare–Rogaway (BR) model augmented to handle the new definitions of several authentication notions given in Chapter 4. The latter also formalized and proved not only certain relations between these notions but also the folklore idea that making use of a secret key established with implicit authentication was sufficient to obtain explicit guarantees. In Chapter 5 we turned to questions of protocol design by studying three existing protocol in light of the new definitions: the Diffie–Hellman, HMQV and TLS v1.3 protocols. We also proposed a new transformation for KE protocols designed with post-quantum security considerations for the selection of its building blocks.

With the increased popularity of messaging applications on personal devices and the growing concern for individual privacy, the last couple of years have seen an emergence of multi-key key-exchange protocols [109, 17, 45]. Specifically designed for securing long-live messaging conversations, these protocols derive and update one or multiple keys over the course of a single session. As before, the focus in the development of security models for such protocols has been on key secrecy, with less concern for authentication properties. An interesting extension of our work would be to develop comparable definitions for such protocols and study whether authentication is necessarily maintained along the multiple key updates or whether security gaps could exist. Similarly, these multi-key protocols have been constructed from pre-quantum hardness assumptions given their advanced stage of deployment. A natural continuation of the post-quantum effort would be to design either transformations, or protocols from scratch, for the same purpose using post-quantum techniques and primitives.

Isogeny-based cryptography

Part II described contributions in the field of isogeny-based cryptography, another contender for post-quantum assumptions and constructions. (The only isogeny-based construction, SIKE [6], submitted to the NIST project has been selected as an alternate candidate for the third round.) Chapter 6 described preliminaries and existing constructions before presenting a new PKE scheme achieving IND-CCA security. With the concern of facilitating protocol design from isogeny assumptions, Chapter 7 defined semi-commutative masking structures and described their instantiations in different isogeny-based settings. Chapter 8 then made use of these structures to build two new OT protocols, one of which could be made secure against active adversaries using a recent transformation [63].

Since our work of this last chapter was concluded, others have proposed further, more efficient, constructions for OT protocols in the setting of $\mathbb{F}_p$-restricted isogenies [95, 2]. There have also been attempts at constructing mechanisms to obfuscate the images of torsion points in order to contruct more complicated primitives from new assumptions derived from the SIDH variants of computational problems [121]. It is yet unknown whether these new mechanisms could provide sufficient flexibility in the setting of supersingular isogenies over $\mathbb{F}_{p^2}$ to build more efficient OT protocols. Concerning our new framework of semi-commutative masking structures, a natural direction would be to expand it to capture principles of protocol design other than only exponentiation-based. This could permit the adaptation of more complex protocols to the isogeny settings that we instantiated. An interesting direction would also be to broaden the avalaible instantiations of our framework, such as using Kummer varieties as done by Vitse in their work on isogeny-based OT [125].

Signatures from MPC-in-the-head

In Part III we finally turned to the design and improvement of signature schemes based on the MPCitH paradigm. Chapter 9 presented the mechanisms of the Picnic submission to the NIST project [38] and explored the efficiency of replacing the original LowMC block cipher with AES. In Chapter 10 we then made use of a verification technique by Baum and Nof and studied the use of both AES and the Legendre PRF in this setting.

The ability of MPCitH proof systems to prove any relation for which an MPC protocol exists is promising, but further work is required to understand which technique would best suit each relation. In Chapter 10 we demonstrated efficient verification of Legendre symbols (based on multiplications) and of inverse computations; this could now be extended to other frequently used computations to broaden the potential of this technique. In order to make use of existing efficient MPC protocols, MPCitH proof systems could also be developed to make use of other paradigms, such as garbled circuits, and potentially offer the possibility of mixing-and-matching to obtain tailor-made protocols.

Bibliography

[1] G. Adj, O. Ahmadi, and A. Menezes. On isogeny graphs of supersingular elliptic curves over finite fields. *Finite Fields Their Appl.*, 55:268–283, 2019.

[2] N. Alamati, L. De Feo, H. Montgomery, and S. Patranabis. Cryptographic group actions and applications. In *ASIACRYPT*, Lecture Notes in Computer Science. Springer, 2020. To appear.

[3] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454. Springer, 2015.

[4] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - A new hope. In *USENIX Security Symposium*, pages 327–343. USENIX Association, 2016.

[5] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM Conference on Computer and Communications Security*, pages 2087–2104. ACM, 2017.

[6] R. Azarderakhsh, M. Campagna, C. Costello, L. Feo, B. Hess, A. Jalali, D. Jao, B. Koziel, B. LaMacchia, P. Longa, et al. Supersingular isogeny key encapsulation. *Submission to the NIST Post-Quantum Cryptography Standardization project*, 2017.

[7] R. Azarderakhsh, D. Jao, K. Kalach, B. Koziel, and C. Leonardi. Key compression for isogeny-based cryptosystems. In *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography*, pages 1–10. ACM, 2016.

[8] P. Barreto, B. David, R. Dowsley, K. Morozov, and A. C. A. Nascimento. A framework for efficient adaptively secure composable oblivious transfer in the ROM. *IACR Cryptol. ePrint Arch.*, 2017:993, 2017.

[9] P. Barreto, G. Oliveira, and W. Benits. Supersingular isogeny oblivious transfer. *IACR Cryptol. ePrint Arch.*, 2018:459, 2018.

[10] C. Baum and A. Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In *Public Key Cryptography (1)*, volume 12110 of *Lecture Notes in Computer Science*, pages 495–526. Springer, 2020.

[11] D. Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.

[12] M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 259–274. Springer, 2000.

[13] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *STOC*, pages 419–428. ACM, 1998.

[14] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer, 2000.

[15] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993.

[16] M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *STOC*, pages 57–66. ACM, 1995.

[17] M. Bellare, A. C. Singh, J. Jaeger, M. Nyayapati, and I. Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In *CRYPTO (3)*, volume 10403 of *Lecture Notes in Computer Science*, pages 619–650. Springer, 2017.

[18] W. Beullens, T. Beyne, A. Udovenko, and G. Vitto. Cryptanalysis of the legendre PRF and generalizations. *IACR Trans. Symmetric Cryptol.*, 2020(1):313–330, 2020.

[19] W. Beullens and C. Delpech de Saint Guilhem. Legroast: Efficient post-quantum signatures from the legendre PRF. In *PQCrypto*, volume 12100 of *Lecture Notes in Computer Science*, pages 130–150. Springer, 2020.

[20] W. Beullens, T. Kleinjung, and F. Vercauteren. CSI-FiSh: efficient isogeny based signatures through class group computations. In *ASIACRYPT (1)*, volume 11921 of *Lecture Notes in Computer Science*, pages 227–247. Springer, 2019.

[21] J. Biasse, D. Jao, and A. Sankar. A quantum algorithm for computing isogenies between supersingular elliptic curves. In *INDOCRYPT*, volume 8885 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2014.

[22] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[23] S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. In *IMACC*, volume 1355 of *Lecture Notes in Computer Science*, pages 30–45. Springer, 1997.

[24] S. Blake-Wilson and A. Menezes. Unknown key-share attacks on the station-to-station (STS) protocol. In *Public Key Cryptography*, volume 1560 of *Lecture Notes in Computer Science*, pages 154–170. Springer, 1999.

[25] X. Bonnetain and A. Schrottenloher. Quantum security analysis of CSIDH and ordinary isogeny-based schemes. *IACR Cryptol. ePrint Arch.*, 2018:537, 2018.

[26] J. W. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In *ACM Conference on Computer and Communications Security*, pages 1006–1018. ACM, 2016.

[27] J. Boyar, P. Matthews, and R. Peralta. Logic minimization techniques with applications to cryptology. *J. Cryptology*, 26(2):280–312, 2013.

[28] C. Boyd, Y. Cliff, J. M. G. Nieto, and K. G. Paterson. Efficient one-round key exchange in the standard model. In *ACISP*, volume 5107 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2008.

[29] Z. Brakerski and N. Döttling. Two-message statistically sender-private OT from LWE. In *TCC (2)*, volume 11240 of *Lecture Notes in Computer Science*, pages 370–390. Springer, 2018.

[30] P. Branco, J. Ding, M. Goulão, and P. Mateus. A framework for universally composable oblivious transfer from one-round key-exchange. In *IMACC*, volume 11929 of *Lecture Notes in Computer Science*, pages 78–101. Springer, 2019.

[31] E. Bresson, J. Monnerat, and D. Vergnaud. Separation results on the "one-more" computational problems. In *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 71–87. Springer, 2008.

[32] C. Brzuska, M. Fischlin, B. Warinschi, and S. C. Williams. Composability of bellare-rogaway key exchange protocols. In *ACM Conference on Computer and Communications Security*, pages 51–62. ACM, 2011.

[33] C. Brzuska, N. P. Smart, B. Warinschi, and G. J. Watson. An analysis of the EMV channel establishment protocol. In *ACM Conference on Computer and Communications Security*, pages 373–386. ACM, 2013.

[34] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[35] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2001.

[36] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes. CSIDH: an efficient post-quantum commutative group action. In *ASIACRYPT (3)*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, 2018.

[37] D. X. Charles, K. E. Lauter, and E. Z. Goren. Cryptographic hash functions from expander graphs. *J. Cryptology*, 22(1):93–113, 2009.

[38] M. Chase, D. Derler, S. Goldfeder, J. Katz, V. Kolesnikov, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. The Picnic signature

scheme. *Submission to the NIST Post-Quantum Cryptography Standardization project*, 2019.

[39] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM Conference on Computer and Communications Security*, pages 1825–1842. ACM, 2017.

[40] A. M. Childs, D. Jao, and V. Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *J. Math. Cryptol.*, 8(1):1–29, 2014.

[41] K. R. Choo, C. Boyd, and Y. Hitchcock. Examining indistinguishability-based proof models for key establishment protocols. In *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 585–604. Springer, 2005.

[42] K. R. Choo, C. Boyd, and Y. Hitchcock. On session key construction in provably-secure key establishment protocols. In *Mycrypt*, volume 3715 of *Lecture Notes in Computer Science*, pages 116–131. Springer, 2005.

[43] T. Chou and C. Orlandi. The simplest protocol for oblivious transfer. In *LATINCRYPT*, volume 9230 of *Lecture Notes in Computer Science*, pages 40–58. Springer, 2015.

[44] T. Chou and C. Orlandi. The simplest protocol for oblivious transfer. *IACR Cryptol. ePrint Arch.*, 2015:267, 2015.

[45] K. Cohn-Gordon, C. J. F. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the signal messaging protocol. In *EuroS&P*, pages 451–466. IEEE, 2017.

[46] C. Costello, D. Jao, P. Longa, M. Naehrig, J. Renes, and D. Urbanik. Efficient compression of SIDH public keys. In *EUROCRYPT (1)*, volume 10210 of *Lecture Notes in Computer Science*, pages 679–706, 2017.

[47] C. Costello, P. Longa, and M. Naehrig. Efficient algorithms for supersingular isogeny diffie-hellman. In *CRYPTO (1)*, volume 9814 of *Lecture Notes in Computer Science*, pages 572–601. Springer, 2016.

[48] J. M. Couveignes. Hard homogeneous spaces. *IACR Cryptol. ePrint Arch.*, 2006:291, 2006.

[49] C. Cremers. Examining indistinguishability-based security models for key exchange protocols: the case of CK, CK-HMQV, and eCK. In *AsiaCCS*, pages 80–91. ACM, 2011.

[50] C. J. F. Cremers and M. Feltz. Beyond eck: perfect forward secrecy under actor compromise and ephemeral-key reveal. *Des. Codes Cryptogr.*, 74(1):183–218, 2015.

[51] J. Daemen and V. Rijmen. AES Proposal: Rijndael, 1999.

[52] I. Damgård. On the randomness of legendre and jacobi sequences. In *CRYPTO*, volume 403 of *Lecture Notes in Computer Science*, pages 163–172. Springer, 1988.

[53] I. Damgård and M. Keller. Secure multiparty AES. In *Financial Cryptography*, volume 6052 of *Lecture Notes in Computer Science*, pages 367–374. Springer, 2010.

[54] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[55] C. Delpech de Saint Guilhem, L. De Meyer, E. Orsini, and N. P. Smart. BBQ: using AES in picnic signatures. In K. G. Paterson and D. Stebila, editors, *SAC*, volume 11959 of *Lecture Notes in Computer Science*, pages 669–692. Springer, 2019.

[56] C. Delpech de Saint Guilhem, M. Fischlin, and B. Warinschi. Authentication in key-exchange: Definitions, relations and composition. In *CSF*, pages 288–303. IEEE, 2020.

[57] C. Delpech de Saint Guilhem, P. Kutas, C. Petit, and J. Silva. Séta: Supersingular encryption from torsion attacks. *IACR Cryptol. ePrint Arch.*, 2019:1291, 2019.

[58] C. Delpech de Saint Guilhem, E. Orsini, C. Petit, and N. P. Smart. Semi-commutative masking: A framework for isogeny-based protocols, with an application to fully secure two round isogeny-based OT. In *Cryptology and Network Security - 19th International Conference, CANS 2020*, Lecture Notes in Computer Science. Springer, 2020. To appear.

[59] C. Delpech de Saint Guilhem, N. P. Smart, and B. Warinschi. Generic forward-secure key agreement without signatures. In *ISC*, volume 10599 of *Lecture Notes in Computer Science*, pages 114–133. Springer, 2017.

[60] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

[61] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Des. Codes Cryptogr.*, 2(2):107–125, 1992.

[62] I. Dinur, D. Kales, A. Promitzer, S. Ramacher, and C. Rechberger. Linear equivalence of block ciphers with partial non-linear layers: Application to LowMC. In *EUROCRYPT (1)*, volume 11476 of *Lecture Notes in Computer Science*, pages 343–372. Springer, 2019.

[63] N. Döttling, S. Garg, M. Hajiabadi, D. Masny, and D. Wichs. Two-round oblivious transfer from CDH or LPN. In *EUROCRYPT (2)*, volume 12106 of *Lecture Notes in Computer Science*, pages 768–797. Springer, 2020.

[64] B. Dowling, M. Fischlin, F. Günther, and D. Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In *ACM Conference on Computer and Communications Security*, pages 1197–1210. ACM, 2015.

[65] K. Eisenträger, S. Hallgren, K. E. Lauter, T. Morrison, and C. Petit. Supersingular isogeny graphs and endomorphism rings: Reductions and solutions. In *EUROCRYPT (3)*, volume 10822 of *Lecture Notes in Computer Science*, pages 329–368. Springer, 2018.

[66] L. D. Feo, D. Jao, and J. Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Math. Cryptol.*, 8(3):209–247, 2014.

[67] L. D. Feo, J. Kieffer, and B. Smith. Towards practical key exchange from ordinary isogeny graphs. In *ASIACRYPT (3)*, volume 11274 of *Lecture Notes in Computer Science*, pages 365–394. Springer, 2018.

[68] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[69] M. Fischlin, F. Günther, B. Schmidt, and B. Warinschi. Key confirmation in key exchange: A formal treatment and implications for TLS 1.3. In *IEEE*

Symposium on Security and Privacy, pages 452–469. IEEE Computer Society, 2016.

[70] S. Galbraith. Isogeny crypto. Blog post from ellipticnews, 2019. `https://ellipticnews.wordpress.com/2019/11/09/isogeny-crypto/`, last accessed Aug 18, 2020.

[71] S. D. Galbraith, C. Petit, and J. Silva. Identification protocols and signature schemes based on supersingular isogeny problems. *J. Cryptology*, 33(1):130–175, 2020.

[72] S. D. Galbraith and F. Vercauteren. Computational problems in supersingular elliptic curve isogenies. *Quantum Inf. Process.*, 17(10):265, 2018.

[73] C. Gentry and M. Szydlo. Cryptanalysis of the revised NTRU signature scheme. In *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 299–320. Springer, 2002.

[74] I. Giacomelli, J. Madsen, and C. Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX Security Symposium*, pages 1069–1083. USENIX Association, 2016.

[75] O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. In *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 1997.

[76] L. K. Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219. ACM, 1996.

[77] F. Günther, B. Hale, T. Jager, and S. Lauer. 0-rtt key exchange with full forward secrecy. In *EUROCRYPT (3)*, volume 10212 of *Lecture Notes in Computer Science*, pages 519–548, 2017.

[78] J. Hoffstein, N. Howgrave-Graham, J. Pipher, J. H. Silverman, and W. Whyte. NTRUSIGN: digital signatures using the NTRU lattice. In *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 122–140. Springer, 2003.

[79] J. Hoffstein, J. Pipher, and J. H. Silverman. NSS: an NTRU lattice-based signature scheme. In *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2001.

[80] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009.

[81] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. Generic compilers for authenticated key exchange. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 2010.

[82] D. Jao and L. D. Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *PQCrypto*, volume 7071 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2011.

[83] D. Kales, S. Ramacher, C. Rechberger, R. Walch, and M. Werner. Efficient FPGA implementations of lowmc and picnic. In *CT-RSA*, volume 12006 of *Lecture Notes in Computer Science*, pages 417–441. Springer, 2020.

[84] D. Kales and G. Zaverucha. An attack on some signature schemes constructed from five-pass identification schemes. *IACR Cryptol. ePrint Arch.*, 2020:837, 2020.

[85] D. Kales and G. Zaverucha. Improving the performance of the picnic signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):154–188, 2020.

[86] J. Katz, V. Kolesnikov, and X. Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *ACM Conference on Computer and Communications Security*, pages 525–537. ACM, 2018.

[87] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition.* CRC Press, 2014.

[88] J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. *J. Cryptology*, 20(1):85–113, 2007.

[89] M. Keller, E. Orsini, and P. Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *ACM Conference on Computer and Communications Security*, pages 830–842. ACM, 2016.

[90] D. Khovratovich. Key recovery attacks on the legendre prfs within the birthday bound. *IACR Cryptol. ePrint Arch.*, 2019:862, 2019.

[91] J. Kieffer. Étude et accélération du protocole d'échange de clés de couveignes-rostovtsev-stolbunov. Master's thesis, Université Paris VI, 2017. Mémoire du Master 2, `https://arxiv.org/abs/1804.10128`.

[92] H. Krawczyk. HMQV: A high-performance secure diffie-hellman protocol. In *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer, 2005.

[93] C. J. Kudla. *Special Signature Schemes and Key Agreement Protocols*. PhD thesis, Royal Holloway University of London, 2006.

[94] P. Kutas, C. Martindale, L. Panny, C. Petit, and K. E. Stange. Weak instances of SIDH variants under improved torsion-point attacks. *IACR Cryptol. ePrint Arch.*, 2020:633, 2020.

[95] Y. Lai, S. D. Galbraith, and C. Delpech de Saint Guilhem. Compact, efficient and uc-secure isogeny-based oblivious transfer. *IACR Cryptol. ePrint Arch.*, 2020:1012, 2020.

[96] B. A. LaMacchia, K. E. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In *ProvSec*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2007.

[97] Y. Li and S. Schäge. No-match attacks and robust partnering definitions: Defining trivial attacks for security protocols is not trivial. In *ACM Conference on Computer and Communications Security*, pages 1343–1360. ACM, 2017.

[98] Y. Li, S. Schäge, Z. Yang, C. Bader, and J. Schwenk. New modular compilers for authenticated key exchange. In *ACNS*, volume 8479 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2014.

[99] R. Lindner and C. Peikert. Better key sizes (and attacks) for lwe-based encryption. In *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2011.

[100] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43:1–43:35, 2013.

[101] C. Martindale and L. Panny. How to not break SIDH. *IACR Cryptol. ePrint Arch.*, 2019:558, 2019.

[102] J. L. Massey and J. K. Omura. Method and apparatus for maintaining the privacy of digital messages conveyed by public transmission, 1986. US Patent 4,567,600.

[103] A. Menezes and B. Ustaoglu. Comparing the pre- and post-specified peer models for key agreement. *Int. J. Appl. Cryptogr.*, 1(3):236–250, 2009.

[104] A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[105] P. Morrissey, N. P. Smart, and B. Warinschi. The TLS handshake protocol: A modular analysis. *J. Cryptology*, 23(2):187–223, 2010.

[106] P. Q. Nguyen and O. Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. *J. Cryptology*, 22(2):139–160, 2009.

[107] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 681–700. Springer, 2012.

[108] C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2008.

[109] T. Perrin and M. Marlinspike. The double ratchet algorithm. `www.signal.org/docs/specifications/doubleratchet/doubleratchet.pdf`, November 2016.

[110] C. Petit. Faster algorithms for isogeny problems using torsion point images. In *ASIACRYPT (2)*, volume 10625 of *Lecture Notes in Computer Science*, pages 330–353. Springer, 2017.

[111] C. Petit and K. E. Lauter. Hard and easy problems for supersingular isogeny graphs. *IACR Cryptol. ePrint Arch.*, 2017:962, 2017.

[112] M. O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptol. ePrint Arch.*, 2005:187, 2005.

[113] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[114] A. Rostovtsev and A. Stolbunov. Public-key cryptosystem based on isogenies. *IACR Cryptol. ePrint Arch.*, 2006:145, 2006.

[115] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[116] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *FOCS*, pages 124–134. IEEE Computer Society, 1994.

[117] J. H. Silverman. *Advanced topics in the arithmetic of elliptic curves*, volume 151 of *Graduate Texts in Mathematics.* Springer Science & Business Media, 1994.

[118] J. H. Silverman. *The arithmetic of elliptic curves*, volume 106 of *Graduate Texts in Mathematics.* Springer Science & Business Media, 2nd edition, 2009.

[119] A. Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Adv. Math. Commun.*, 4(2):215–235, 2010.

[120] A. Stolbunov. *Cryptographic Schemes Based on Isogenies.* PhD thesis, Norwegian University of Science and Technology, 2012.

[121] O. Taraskin, V. Soukharev, D. Jao, and J. T. LeGrow. Towards isogeny-based password-authenticated key establishment. *IACR Cryptol. ePrint Arch.*, 2018:886, 2018.

[122] E. E. Targhi and D. Unruh. Post-quantum security of the fujisaki-okamoto and OAEP transforms. In *TCC (B2)*, volume 9986 of *Lecture Notes in Computer Science*, pages 192–216, 2016.

[123] D. Urbanik and D. Jao. Sok: The problem landscape of SIDH. In *Proceedings of the 5th ACM International Workshop on ASIA Public-Key Cryptography Workshop*, pages 53–60. ACM, 2018.

[124] B. Ustaoglu. Comparing *SessionStateReveal* and *EphemeralKeyReveal* for diffie-hellman protocols. In *ProvSec*, volume 5848 of *Lecture Notes in Computer Science*, pages 183–197. Springer, 2009.

[125] V. Vitse. Simple oblivious transfer protocols compatible with supersingular isogenies. In *AFRICACRYPT*, volume 11627 of *Lecture Notes in Computer Science*, pages 56–78. Springer, 2019.

[126] X. Wang, S. Ranellucci, and J. Katz. Global-scale secure multiparty computation. In *ACM Conference on Computer and Communications Security*, pages 39–56. ACM, 2017.

[127] Q. Wu, J. Zhang, and Y. Wang. Practical t-out-n oblivious transfer and its applications. In *ICICS*, volume 2836 of *Lecture Notes in Computer Science*, pages 226–237. Springer, 2003.

[128] X. Xu, H. Xue, K. Wang, M. H. Au, and S. Tian. Strongly secure authenticated key exchange from supersingular isogenies. In *ASIACRYPT (1)*, volume 11921 of *Lecture Notes in Computer Science*, pages 278–308. Springer, 2019.

[129] R. Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.