The GOOD the BAD and the (not so) UGLY

Many facets of cryptographic backdoors

Bernardo Magri

Ph.D. Thesis

Sapienza University of Rome

SAPIENZA
UNIVERSITÀ DI ROMA

The Good, the Bad, and the (not so) Ugly: Many Facets of Cryptographic Backdoors

PhD School of the Department of Computer Science
at Sapienza, University of Rome.

Dottorato di Ricerca in Informatica – XXIX Ciclo

Candidate

Bernardo Magri
ID number 1590940

Thesis Advisor

Prof. Giuseppe Ateniese

A thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Informatica

October 2016

Thesis defended on 13 February 2017
in front of a Board of Examiners composed by:

Prof. Enrico Puppo (chairman)

Prof. Geppino Pucci

Prof. Valeria Cardellini

The Good, the Bad, and the (not so) Ugly: Many Facets of Cryptographic Backdoors

Ph.D. thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: March 2, 2017

Author's email: magri@di.uniroma1.it

To Anna

Acknowledgments

The three years that I spent on my Ph.D working very hard to be able to write this thesis were amazing. First, because I had the opportunity to live in a wonderful city like Rome, and second because of the incredible people that I met during the way.

The list of people that I need to thank is huge! I feel I should start with my thesis advisor Prof. Giuseppe Ateniese, that helped me a lot during all the Ph.D. and made this thesis possible. I'm very grateful for being his student and being able to work side by side with him.

I also need to give a special thanks to Daniele Venturi, that from colleagues we became good friends, and without his guidance, ideas and insights this work would not have been possible. I'm gonna miss all the fruitful discussions and chats that we always had, that led to many of the contents on this thesis. I'm also gonna miss all the overnights spent working together in the department to meet some crazy conference deadline.

I also owe a big thanks to all the members (and former members) of the security group of Sapienza that helped me directly (and indirectly) along the way. Thanks to Alberto Compagno, Briland Hitaj, Fabio De Gaspari, Giorgio Mancini, Gabriele Gualandi, Nino Verde, Antonio Villani, Antonio Faonio, Agostino Panico and Prof. Luigi Mancini.

A list of people that contributed with this work, and with my crypto formation in general, and that I'm very grateful for includes: Giuseppe Ateniese, Daniele Venturi, Aggelos Kiayias, Antonio Faonio, Breno De Medeiros, Yiannis Tselekounis, Ewerton Andrade and Abhishek Jain.

I can't forget to thank my master's thesis advisor Prof. Routo Terada, that helped ignite the crypto sparkle in me a long time ago. Thanks also to all the members of the data security lab from University of São Paulo (IME).

I also owe a big thanks to Dario Fiore and Giuseppe Persiano for taking the time to review this thesis and for giving me valuable comments and suggestions.

Lastly, a big thanks to my beautiful wife Anna, for putting up with me all this time, and to my entire family for giving me the support and the strength to pursue great things!

Rome, October 2016 *Bernardo Magri*

Abstract

Modern cryptography is concerned with the feasibility or infeasibility of securely realizing a task. The answer to whether or not a task can be securely realized depends on the assumed power of the adversary. The obtained security holds in a strong mathematical sense, meaning that breaking a secure cryptographic protocol is either impossible or it would require to solve some computational problem which is believed to be hard to solve. The most basic computational assumption made in cryptography is the existence of one-way functions. One-way functions are functions that are easy to compute but are hard to invert on almost all inputs.

By just assuming the existence of one-way functions it is already possible to realize many cryptographic tasks, but there are some tasks that can only be realized by assuming the existence of trapdoor functions (also called backdoored functions). Trapdoor functions are easy to compute in one direction but hard to invert; the difference is the existence of a special value that allows the function to be easily inverted by any party that has knowledge of this special value.

In this thesis we have contributions in three different facets of backdoors in cryptography, that we describe next.

- Backdoors for legitimate usage: We define a new security goal for chameleon hash functions that we call enhanced collision resistance. This security definition informally says that a chameleon hash function must remain collision-resistant even after an adversary sees polynomially many collisions for the function. We also present a novel transformation from standard chameleon hash functions to enhanced ones; a full-fledged "redactable blockchain" application, that leverages the power of enhanced collision-resistant chameleon hashes, is built to show that this new security goal for chameleon hashes makes sense in practice.

- Backdoors for malicious usage: Motivated by the revelations of Edward Snowden about intelligence agencies intentionally undermining the security of cryptographic schemes, we cover the case where backdoors are used to perform attacks against signature schemes. In this vein, we define what is an undetectable subversion attack against a signature scheme and we also present two devastating attacks against randomized signature schemes that allows the complete extraction of the users' signing key.

- Immunization techniques against malicious backdoors: Our contributions in this vein are twofold. We first show positive results for signature schemes against subversion attacks; we prove that unique signatures are secure against an undetectable class of subversion attacks and that re-randomizable signatures are secure against *all* classes of subversion attacks (by employing a cryptographic reverse firewall introduced in this thesis). In the other contribution on the immunization direction we define a secure model for the outsourcing of circuit fabrication, where the outsourcing facilities are untrusted and can potentially embed malicious hardware trojans in the circuit. We construct three compilers that upon input an arbitrary circuit it produces an outsourcing-secure version of the circuit.

Contents

CHAPTER 1

Introduction

When we talk about cryptography, the first thing that comes to mind is secret communication. Before the last few decades of the 20th century that perception was indeed accurate, and cryptography was considered more of an art than a science. After that period the landscape drastically changed, and modern cryptography was born, encompassing much more than just secret communication, and allowing many marvellous possibilities that were unimaginable before, such as play a fair game of poker over the telephone [SRA81].

Modern cryptography is concerned with the feasibility or infeasibility of securely realizing a task. The answer to whether or not a task can be securely realized often depends on the assumed power of the adversary. The obtained security holds in a strong mathematical sense, meaning that breaking a secure cryptographic protocol is either impossible or it would require to solve some computational problem which is believed to be hard to solve. Those mathematical guarantees only became possible by leveraging the tools of computational complexity theory and by defining formal models and methods when analyzing cryptographic schemes.

Computational complexity theory is the study of "how hard" it is for a computer to find a solution to a problem; if there exists an algorithm that solves the problem in polynomial-time,[1] we say that the problem is easy to solve. It is highly believed (although not proved) that there exists some types of problems that are not easy to solve, but it is easy to verify that a given solution to the problem is correct (these problems are in the complexity class called NP). One well-known example is the integer factorization problem; given a positive integer n compute its prime factors. No polynomial-time algorithm for such a problem is known, and if we take n to be a huge number (e.g. $>$ 1024-bits) it would take a lifetime for a computer to find the answer using the fastest known algorithm.[2] On the other hand, given the prime

[1] A polynomial-time algorithm is an algorithm whose running time (number of steps) can be expressed as a polynomial of any degree.

[2] The fastest algorithm known for integer factorization is the general number field sieve, and it runs in sub-exponential time [LJMP90] on the size of the input.

factors of n we can simply multiply them and see if the result is in fact n.

In modern cryptography we use those types of assumptions (such as integer factorization) to build cryptographic schemes and to give some mathematical assurances of how long it would take for an adversary[3] to violate the security guarantees of the scheme (or more informally, to "break" the scheme). This is usually accomplished with a technique borrowed from computational complexity theory, called reduction; the intuition behind it is as follows. We first assume that an adversary that breaks the cryptographic scheme exists, and then we build a (polynomial-time) algorithm that runs the adversary and in the end outputs a solution to the underlying hard-problem. But wait a minute! how can this polynomial-time algorithm solve the hard problem if there is no known polynomial-time algorithm that solves the problem? That is exactly the point! therefore, the adversary that breaks the scheme cannot exist, otherwise the adversary can solve the hard-problem, which is a contradiction since we assume that the problem is hard to solve.

The most basic computational assumption in cryptography is the existence of one-way functions (OWF). One-way functions are the type of functions that are easy to compute but are hard to invert on almost all inputs. The integer factorization problem previously described is believed to be a one-way function. We call it an assumption because so far nobody has been able to prove this to be true or false. In fact, proving the existence of one-way functions would solve the long-lasting open problem in computer science P vs NP [Sip92].

By just assuming the existence of one-way functions it is possible to realize many cryptographic tasks such as, symmetric encryption, message authentication codes, pseudorandom number generators, pseudorandom functions, digital signatures [Lam79] and more. This is already very exciting and useful in many scenarios, but there are some situations in which this is not enough. If we consider the symmetric encryption for instance, one may ask how the secret key used in this scheme is agreed between the parties; unfortunately this is a problem that cannot be solved by just assuming the existence of one-way functions [IR88]. To solve this problem we need to add another assumption into the mix which is the existence of trapdoor functions. Similar to one-way functions, trapdoor functions are easy to compute in one direction but hard to invert; the difference is the existence of a special value (i.e. the trapdoor) that allows the function to be easily inverted by any party that has knowledge of this special value. Trapdoors can also be called backdoors, and in this thesis we use both names interchangeably.

To show a little better how the assumptions considered can completely rule out (or make it possible) some exciting cryptographic applications we briefly discuss the seminal paper of Impagliazzo [Imp95]. In [Imp95], five different worlds are presented, each considering a different assumption as true and discussing the implications of it. We give a brief description of each world next. (i) Algorithmica is the world where $\mathsf{P} = \mathsf{NP}$. This is a bad world for cryptography because any problem such that its solution can be verified in polynomial-time can also be solved in polynomial-time. (ii) In Heuristica, there are hard instances of NP problems but it is also hard to find such instances. This is also a bad world for crypto. (iii) Pessiland is the world where there are hard average-case problems but there are no one-way functions. This

[3] An adversary is considered to be any third party trying to undermine the cryptographic scheme.

means that any function can be inverted for almost all values of input, therefore all crypto based on the existence of one-way functions would be broken. (iv) In Minicrypt one-way functions do exist, but there are no trapdoor functions. In this world things start to get more interesting for cryptography, as it is now possible to have all cryptographic primitives that rely only on one-way functions, such as symmetric encryption, signature schemes, and etc. (v) In Cryptomania there exist trapdoor functions (and one-way functions). This is the world that we believe to live in; we can realize all the tasks of Minicrypt in addition to every primitive that depends on trapdoor functions, such as key-agreement, public key encryption and more.

1.1 Backdoors & Cryptography

Trapdoor functions have many fundamental usages in cryptography, ranging from public key encryption to chameleon hash functions.

Most of the times trapdoors are used with a good intent, such as in public-key encryption, although there are some situations where the use of trapdoors are targeted at gaining some unwarranted advantage over the users of the scheme; the knowledge of the trapdoor may allow, for instance, to recover the secret signing key in a (backdoored) signature scheme. The dangers of such backdoored cryptographic primitives are evident and devastating for the security of the schemes, therefore some immunization techniques might be applied to those backdoored primitives in order to still maintain some level of security.

In this section we present concrete examples of all three scenarios, where trapdoors are used for the good, for the bad, and also some (not so ugly) immunization techniques.

1.1.1 Good Intentions

The use of backdoors in cryptography is quite widespread, and many primitives could not exist without it. Although there are many legitimate uses of backdoors, such as in public key encryption or key-agreement, in this thesis we focus more on primitives that do not depend on the backdoor directly, but its use gives the primitive an extra functionality that is essential to some application (e.g. chameleon hash functions).

Trapdoor generators. Vazirani and Vazirani [VV83] started in 1983 the study of backdoored psedurandom number generators (PRG) to build a secure 1-bit disclosure protocol. This type of protocol is used when one party wants to disclose a single bit of information to some other party in exchange of a receipt.

A backdoored PRG behaves exactly like a standard PRG when one does not know the backdoor (i.e. the output sequence of the PRG is indistinguishable from a random sequence), but the knowledge of the backdoor allows to easily predict the output sequence.

Chameleon hash functions. The concept of chameleon hashing was put forward by Krawczyk and Rabin [KR00], building on the notion of chameleon commitments [BCC88]. Informally, a chameleon hash is a cryptographic hash function that contains a trapdoor: Without the trapdoor it should be hard to find collisions, but knowledge of the trapdoor information allows to efficiently generate collisions for the hash function.

1.1.2 Bad Intentions

In this section we describe previous works on the topic of backdoored cryptographic primitives that are built with the intention of subverting the security of the underlying scheme. The most extreme case is when the backdoor allows for the extraction of the users' secret key (i.e. a total break).

Subliminal channels. Subliminal channel attacks were introduced in 1983 by Simmons [Sim83]. The prisoners' problem described in [Sim83] is related to two prisoners kept in different cells that need to coordinate an escape plan. The prisoners can only communicate through the prison warden, that relays unencrypted messages back and forth among the prisoners. Simmons shows how the prisoners can successfully coordinate an escape plan without the warden noticing, by cleverly substituting parameters in the signature algorithm used to authenticate the prisoners.

After its introduction, the potential of subliminal channels has been explored in several works (e.g., [Des88a, Des88b, BDI+99]).

Kleptography. The ability of substituting a cryptographic algorithm with an altered version was first considered formally by Young and Yung (extending previous works of Simmons on subliminal channels [Sim83, Sim84]), who termed this field kleptography [YY96, YY97]. The idea is that the attacker surreptitiously modifies a cryptographic scheme with the intent of subverting its security. Kleptography tries to leak the secret information from cryptographic protocols that do not have an explicit subliminal channel. The leakage instead occurs by using "implicit channels" created through repetitive correlated usage of the protocol.

Subversion attacks. Inspired by the recent revelations of Edward Snowden [PLS13, BBG13, Gre14], research in the kleptography area has recently been revitalized by Bellare *et al.* [BPR14] who considered encryption algorithms with the possibility of mass surveillance under the algorithm-substitution attack. They analyzed the possibility of an intelligence agency substituting an encryption algorithm with the code of an alternative version that undetectably reveals the secret key or the plaintext. What they uncovered is that any randomized and stateless encryption scheme would fall to generic algorithm-substitution attacks. The only way to achieve a meaningful security guarantee (CPA-security) is to use a nonce-based encryption that must keep state. Unfortunately, only stateless schemes are deployable effectively with the current network technology and indeed all deployed encryption algorithms are in this class. After that, many results followed [DFP15, DGG+15, AMV15] constructing several concrete primitives resisting large classes of subversion attacks.

1.1.3 Immunization Techniques

In this section we describe some previous works targeted at immunizing backdoored primitives, in order to maintain some level of security, even in the presence of a malicious backdoor.

Backdoored PRG. Many years after the introduction of backdoored PRGs [VV83], Dodis *et al.* [DGG+15] formalized the notion and showed how backdoored PRGs (BPRGs) can be used in a malicious way. They also presented new constructions and immunization techniques for BPRGs. The immunization is a post-processing of the BPRG output, in a way that the adversary can no longer predict the output of the BPRG, even knowing the trapdoor. This immunization technique allows for the user to maintain the security of the BPRG when the backdoor was inserted by a malicious adversary.

Cryptographic reverse firewall. A cryptographic reverse firewall [MS15,DMS16] is a piece of hardware (or software) that sits between the user and the external world. The concept was introduced by Mironov and Stephens-Davidowitz [MS15] and its goal is to avoid the leakage of secret information in compromised cryptographic protocols. The firewall "sanitizes" all the messages exchanged between the users' computer and the outside world. A secure cryptographic reverse firewall contains no secret information in itself and needs to satisfy the following properties. (i) Maintain the functionality of the cryptographic protocol being used, (ii) preserve the security of the underlying protocol, and (iii) resist exfiltration of any secret to the outside world.

1.2 Contributions

In this section we give a concise description of the contributions of this thesis. We have contributions in all three use cases of backdoors in cryptography that we describe next.

1.2.1 The Good

For our contribution on the use of backdoors for good purposes we focus on chameleon hash (CH) functions. Informally, a chameleon hash is a cryptographic hash function that contains a trapdoor: Without the trapdoor it should be hard to find collisions, but knowledge of the trapdoor allows to efficiently generate collisions for the hash function.

Unfortunately, as observed by Ateniese and de Medeiros [AdM04], collision resistance is not sufficient for most of the applications of chameleon hash. The reason is that, while the hash function is indeed collision resistant, any party seeing a collision for the hash function would be able to find other collisions or even recover the secret trapdoor information. This "key exposure" problem makes chameleon hashes not applicable in many contexts. We introduce and formalize the notion of *enhanced* collision resistance for chameleon hash functions, where collisions are hard to compute by an adversary even after seeing polynomially many collisions.

To the best of our knowledge, the only chameleon hash function satisfying the enhanced collision resistance property is due to Ateniese and de Medeiros [AdM04]; this construction is ad-hoc and relies on the Nyberg-Rueppel signature scheme [NR94]. We present a generic transformation that turns (almost) any collision resistant chameleon hash function into an enhanced collision resistant one; this transformation holds in the standard model and relies only on CPA-secure public-key encryption (PKE) and tSE non-interactive zero-knowledge (NIZK) (cf. Section 3.2). Prior to our work it was unknown whether enhanced collision resistance can be achieved in a non ad-hoc fashion in the standard model. We answer this question in the affirmative.

Lastly, as an application for the enhanced collision chameleon hash, we present a new design for a redactable blockchain which is compatible with all popular blockchain proposals. The main feature of our system is that it is compatible with current blockchain designs, i.e., it can be implemented right now and requires only minimal changes to the way current client software interprets information stored in the blockchain, and to the current blockchain, block, or transaction structures.

A blockchain is just a sequence of blocks $B_i = \langle s_i, x_i, ctr_i \rangle$, where $s \in \{0,1\}^\kappa$, $x \in \{0,1\}^*$ and $ctr \in \mathbb{N}$. The blocks are linked through the relation $s_i = H(G(s_{i-1}, x_{i-1}), ctr_i)$, where $H : \{0,1\}^* \to \{0,1\}^\kappa$ and $G : \{0,1\}^* \to \{0,1\}^\kappa$ are collision-resistant hash functions. If any of the previous blocks are changed the chain breaks down. The main idea behind our approach is to set the inner hash function (i.e., the function G), used to chain the blocks in the blockchain, to be an enhanced collision resistant chameleon hash function. Intuitively, re-writing the content of each block is possible by finding collisions in the hash function (without modifying the outer hash function H). This application is described in details in Section 3.3.

1.2.2 The Bad

For our contribution on the malicious use of backdoors, we analyze digital signature schemes under the so-called *subversion attacks* (SAs), that in particular include algorithm-substitution and kleptographic attacks as a special case, but additionally cover more general malware and virus attacks. We stress that our intention is not to propose schemes that can be abused by criminals to avoid monitoring. We are motivated by pure scientific curiosity and aspire to contribute to an active field of research.

On the setting of signature schemes, we define what it means for a class $\mathcal{A}$ of SAs to be (efficiently) undetectable; roughly this means that a user, given polynomially many queries, cannot distinguish the output of the genuine signature algorithm from the output of the subverted algorithm. See Section 4.2 for a precise definition. Our definitions of undetectability are similar in spirit to the ones put forward by [BPR14] for the setting of symmetric encryption. Importantly we distinguish the case where the user (trying to detect the attack) knows only public or private information (i.e., it knows the secret key).[4]

[4] As we show, secret and public undetectability are *not* equivalent, in that there exist natural classes of SAs that are publicly undetectable but secretly detectable.

Next, we explore the possibility of designing classes of SAs that are (even secretly) undetectable and yet allow for complete security breaches. This direction was already pursued by Bellare *et al.*, who showed that it is possible to stealthily bias the random coins of sufficiently randomized symmetric encryption schemes in a way that allows to extract the secret key after observing a sufficient number of (subverted) ciphertexts. As a first contribution in this direction, we explain how to adapt the "biased randomness attack" of [BPR14] to the case of signature schemes.

The above generic attack requires that the signature scheme uses a minimal amount of randomness (say, 7 bits). This leaves the interesting possibility that less randomized schemes (such as the Katz-Wang signature scheme [KW03], using only one bit of randomness) might be secure. In Section 4.3.2, we present a new attack showing that this possibility is vacuous: Our attack allows to stealthily bias the randomness in a way that later allows to extract the signing key—regardless of the number of random bits required by the scheme—assuming that the targeted signature scheme is *coin-extractable*. The latter roughly means that the random coins used for generating signatures can be extracted efficiently from the signature itself; as we discuss in more detail in Section 4.3.2 many real schemes (including Katz-Wang) are coin-extractable. A subversion attack against signature schemes that are *not* coin-extractable, and with short randomness is still unknown.

1.2.3 The (not so) Ugly

We have two main contributions in immunization against backdoors in cryptography. The first is concerned with signature schemes, where we show how to sign securely even when the signing algorithm is subverted by a malicious adversary. For the second contribution we put forward a formal framework for assessing security of a cryptographic circuit whose production has been, in part, outsourced to a set of manufacturers that are not trusted.

Immunization for Signature Schemes.

Following from our subversion attacks on signature schemes (cf. Section 4.3), we present our first positive result for immunization against such type of attacks, where we show that fully deterministic schemes with *unique*[5] signatures are existentially unforgeable under chosen-message attacks against the class of SAs that satisfies the so-called verifiability condition.[6] This means that—*for all values in the message space*—signatures produced by the subverted signature algorithm should (almost always) verify correctly under the target verification key (note that both attacks mentioned above fall into this category).

Clearly, the assumption that the verifiability condition should hold for all messages is quite a strong one. Hence, we also relax the verifiability condition to hold for all but a negligible fraction of the messages. However, we are not able to prove that unique signatures achieve existential unforgeability under chosen-message attacks against the

[5] A signature scheme is unique if for an honestly generated verification key there is a single valid signature for each message.

[6] One might ask whether a similar result holds for all deterministic schemes where signatures are not unique; the answer to this question is negative as our attacks also apply to certain types of deterministic schemes (e.g., de-randomized schemes—see the proof of Theorem 4 in Section 4.4.2).

class of SAs that satisfies relaxed verifiability.[7] Instead, as our second positive result, we show that unique signatures are existentially unforgeable under random-message attacks (where the adversary can only see potentially subverted signatures of random messages) against the class of SAs that satisfies relaxed verifiability. Interestingly, this weaker security flavor is still useful for applications. In Section 5.1.4 we show how to construct subversion-resilient identification schemes from such unique signature schemes.

As our third positive result, we provide a way how to achieve the ambitious goal of protecting signature schemes against *arbitrary* SAs, relying on a cryptographic reverse firewall. We prove that every signature scheme that is re-randomizable (as defined in [HJK12]) admits a RF that preserves unforgeability against arbitrary SAs. Re-randomizable signatures admit an efficient algorithm ReRand that takes as input a tuple (m, σ, VK) and outputs a signature σ' that is distributed uniformly over the set of all valid signatures on message m (under VK); unique signatures, for instance, are re-randomizable. Upon input a pair (m, σ) our firewall uses the public state to verify (m, σ) is valid under VK, and, in case the test passes, it runs ReRand on (m, σ) and outputs the result. Otherwise the firewall simply returns an invalid symbol $\perp$ and *self-destructs*, i.e., it stops processing any further query.[8] The latter is a requirement that we prove to be unavoidable: No RF can at the same time maintain functionality and preserve unforgeability of a signature scheme without the self-destruct capability.

We remark that our results and techniques for the setting of RFs are incomparable to the ones in [MS15]. The main result of Mironov and Stephens-Davidowitz is a compiler that takes as input an arbitrary two-party protocol and outputs a functionally equivalent (but different) protocol that admits a RF preserving both functionality and security. Instead, we model directly security of RFs for signatures schemes in the game-based setting; while our goal is more restricted (in that we only design RFs for signatures), our approach results in much more efficient and practical solutions.

Lastly, we give a concrete construction of a new tightly secure unique signature scheme based on the quadratic residuosity problem. The scheme is based on a lossy function from [FGK$^+$13], and its security proof follows from the surprising result of Kakvi and Kiltz [KK12], where they show that a unique signature scheme can have a tight security proof if the function is lossy. The scheme is the first of its kind to have a tight security proof based on a standard assumption (in the random oracle model).

Secure circuit fabrication.

We put forward a formal framework for assessing security of a circuit whose production has been, in part, outsourced to a set of manufacturers that are not trusted. With

[7] In fact, as shown very recently by Degabriele *et al.* [DFP15] for the case of symmetric encryption, it is not hard to show that such limitation is inherent: No (even deterministic) scheme can achieve security under chosen-message attacks against the class of SAs that meets relaxed verifiability. See Section 1.3 for more details.

[8] This can be implemented, for instance, by having the public state include a single one-time writable bit used to signal a self-destruct took place.

such a framework in hand, we give two design methodologies. Our first methodology borrows ideas from the setting of verifiable computing (see, e.g., [GGP10]), while the second one relies on secure multiparty computation (see, e.g., [GMW87]) in the client-server model [Bea97]. A more detailed explanation of our main contributions in this direction follows below.

Let Γ be the original circuit to be produced. Instead of producing Γ directly, we first "compile" it into a different circuit $\widehat{\Gamma}$ using an efficient, possibly randomized, procedure Φ that we call an *outsourcing compiler*. The compiler Φ takes as input a description of Γ and returns a description of $\widehat{\Gamma}$, together with some auxiliary information specifying how $\widehat{\Gamma}$ can be divided into sub-components, and which of these components can be produced off-shore; the remaining components will be instead built in-house. After all components have been produced, the circuit designer re-assembles the circuit $\widehat{\Gamma}$ (by combining the outsourced components and the components built in-house), which is then initialized with some initial secret memory M_1, and used in the wild.

In order to make sense, the above approach needs to satisfy a few important requirements. The first requirement is that Φ needs to be functionality preserving, meaning that the compiled circuit $\widehat{\Gamma}$ should compute the same functionality as the original circuit Γ (for all possible initial memories M_1, and for all possible inputs). The second requirement is that the effort needed to fabricate the trusted sub-components should be (much) less compared to the effort required to build the original circuit Γ. The third requirement is that Φ should be secure, meaning that, under an acceptable assumption about the manufacturers who construct the outsourced components, the produced circuit $\widehat{\Gamma}$ can be safely used in real-life applications.

Our security definition follows the simulation paradigm, and is inspired by similar definitions in the setting of tamper-proof circuit compilers [IPSW06, FPV11]. We refer the reader to Section 1.3 for a more detailed comparison between the two approaches. In a nutshell, security of Φ is defined by requiring that whatever an adversary can learn by interacting with the fabricated circuit $\widehat{\Gamma}$ (produced following the steps outlined above), can be simulated given only black-box access to the original circuit Γ. This essentially means that, no matter how the outsourced components are maliciously modified (e.g., by inserting a hardware Trojan), using circuit $\widehat{\Gamma}$ is as secure as using the original circuit Γ, and thus, in particular, does not leak sensitive information on the secret memory. See Section 5.3.1 for a precise definition.

We also consider a weakening of the above definition, in which the simulator is allowed to receive a short advice (or leakage) on the secret memory M_1. This models a setting where the adversary might be able to learn a short amount of information on the secret memory, but still yields a meaningful security guarantee provided that the original circuit is resilient to such a short leakage. An appealing advantage of this weaker definition is that it might allow for significantly more efficient circuit compilers.

A solution using VC. In Section 5.3.2, we show how to construct secure outsourcing compilers that work for *arbitrary* circuits Γ in the setting where *all* outsourcing manufacturers are corrupted. Our compilers generically leverage a verifiable com-

putation (VC) scheme for the function $\mathcal{F}$ implemented by Γ. Verifiable computing (see, e.g., [GGP10]) is a recent paradigm by which resource-constrained clients can delegate the computation of some function $\mathcal{F}$, on (possibly private) input X, to an untrusted (and computationally powerful) server, without the server being able to cheat about the outcome of the computation, and with the property that verifying the server's answer is much more efficient than computing the function from scratch.

Recent breakthrough research on verifiable computing led to nearly practical schemes that work for any function [PHGR13, CFH+15]; some schemes additionally preserve the privacy of the inputs on which the function is being computed on [FGP14]. VC schemes satisfying the latter property are called *input-private*.

The main idea of how to use verifiable computing in order to build secure outsourcing compilers is simple enough to describe it here. The entire bulk of the computation will be outsourced to the untrusted fabrication facility, whereas the only circuit components that need to be built in-house are: (i) the component corresponding to the algorithm for encoding the inputs (in case of input-private VC), (ii) the component corresponding to the algorithm run by the client in order to verify correctness of the server's computation, and (iii) the component used to generate fresh random coins as needed for computing the function (in case of randomized functions). Thanks to the nature of VC, the size of the components in (i) and (ii) is independent of the size of the original circuit computing the function. As for the component in (iii), we can use any existing (and trusted) circuitry for generating true random numbers (RNG). A good example is the Intel on-chip hardware random number generator which can be accessed through the RDRAND instruction available on all modern processors [JK99, HKM12].[9]

Hence, the effort needed to fabricate the components built in-house is much less (and, in fact, independent) of the effort needed to fabricate the original circuit.

We implement the above idea in two ways, depending on the properties satisfied by the underlying VC scheme, as explained below.

- Our first compiler relies on VC schemes with input-privacy, and achieves our strongest security notion (i.e., no leakage required for the simulation).

- Our second compiler relies on VC schemes without input-privacy, and achieves security provided the original primitive (implemented by the circuit Γ) is resilient against a logarithmic amount of leakage on the private memory.

  Remarkably, any public-key encryption or signature scheme is resilient to such an amount of leakage at the price of a polynomial loss in the concrete security, and recently many leakage-resilient schemes [DP08, Pie09, KV09, DHLW10, NS12, BSW13, NVZ14, FNV15] have been constructed, where the concrete security does not degrade with the total amount of tolerated leakage.

The second compiler additionally relies on a special "self-destruct" feature (which is implemented in one of the components built in-house), meaning that after the first invalid output is ever processed, the entire memory is overwritten. As we show, for

[9] Intel's generator relies on unpredictable thermal noise to generates bits that are fed to a cryptographic "conditioner" (AES in CBC-MAC mode) which produces a 256-bit seed that is then passed through a NIST SP800-90A-based pseudorandom generator.

this compiler, this is an inherent requirement, in that, without such a property, there exist generic attacks that allow to recover the entire private memory. Moreover, such attacks are undetectable by all polynomial-time (black-box) tests. Our definition of undetectability (see Section 5.3.1) is similar in spirit to analogous definitions in the context of subversion-resilient cryptography 4.2.

A solution using MPC. In Section 5.3.3, we show how to construct secure outsourcing compilers for arbitrary circuits Γ in the setting where $m \geq 2$ outsourcing manufacturers are available, and a certain unknown subset of them is untrustworthy. This is a strictly stronger assumption compared to the VC setting, nevertheless, opens the possibility for more efficient constructions and stronger availability guarantees. Our compiler utilizes a general client-server secure multiparty computation (MPC) protocol, i.e., a protocol that for any function enables a set of clients to privately communicate their inputs to a set of servers that will perform a computation and return the output to a single designated recipient.

We stress that many MPC protocols follow this paradigm (e.g., [DI05]), while others, as we comment later, can be easily adapted to it.

Given such a protocol, the compiler operates as follows. For a given circuit Γ it produces the MPC protocol implementing it, isolates the client and recipient computation for manufacturing in-house, and outsources each of the other components (representing a server in the MPC protocol) to the untrusted manufacturers. The key points of this compiler construction are as follows: (i) The client and recipient computation are typically quite lightweight; the client, in many protocols, simply performs an encryption or a secret-sharing operation, and the recipient a secret-reconstruction protocol; in either case, the computation is independent of the circuit that is outsourced. (ii) There are MPC protocols that can tolerate up to $m - 1$ malicious servers, something we can leverage to argue that if at least one of the outsourcing manufacturer is honest the compiled circuit would be safe for use. Additional properties of the underlying MPC protocol can also be very valuable by our compiler: for instance, if the underlying MPC protocol supports guaranteed output delivery, we can use this guarantee to argue that the final circuit will be resilient to a certain faulty outsourced sub-component. Moreover, if the underlying protocol satisfies the identifiable abort property, cf. [IOZ14], we can enable our compiled circuit to partially shutdown an outsourced sub-component that is discovered to be faulty, thus reducing energy consumption. By adapting an efficient protocol, e.g, [DPSZ12], in the client-server model, we can provide a very efficient compiled circuit (with an overhead linear in the size of the original circuit $|\Gamma|$).

A remark. Both our approaches require a partitioning and assembly procedure that must be performed in-house. To lower the cost, trusted and untrusted sub-circuits will likely be diced in their own wafers and packaged independently. Fortunately, recent advanced designs and emerging technologies in the electronics assembly and packaging industry are making the entire process of interconnecting different components more affordable, reliable, and automatic.

1.3 Related Work

In this section we discuss related works concerning all the three previously described usages of backdoor in cryptography.

Stateless subversion attacks. Bellare *et al.* [BJK15] introduced a stronger definition of undetectability for subversion attacks where the user trying to detect a subverted scheme can query messages to the signing oracle and receive as output a pair (σ, τ) containing the signature and the state; since the original signature scheme is stateless the state τ will always be empty (represented by the empty string ϵ) when querying the original signing algorithm. This way, any stateful subversion can be easily detected by checking whether $\tau = \epsilon$. Moreover, the "biased randomness attack" from [BPR14] is modified into a stateless version, so it can remain undetectable under this stronger undetectability definition.

Input-triggered subversions. In a very recent paper, Degabriele, Farshim and Poettering (DFP) [DFP15] pointed out some shortcomings of the Bellare-Patterson-Rogaway (BPR) [BPR14] security model for subversion resilience of symmetric encryption schemes. Consider the class of SAs that upon input a secret (trapdoor) message $\bar{m}$ outputs the secret key, but otherwise behaves like the genuine signature algorithm. Clearly this class of SAs will be undetectable by the users, as without knowing the trapdoor there is only a negligible chance to query the secret message $\bar{m}$ and check if the signature algorithm was subverted (at least if the message space is large enough). Yet, an adversary mounting a chosen-message attack can recover the signing key by asking a signature for message $\bar{m}$.

As a consequence, it is impossible to prove existential unforgeability under-chosen message attacks against such "input-triggered" subversions (in the BPR model). Note however that, for the case of signatures, one can still prove a positive result by restricting the adversary to only see signatures of random messages (i.e., in case of a random-message attack). Indeed, input-triggered subversions meet our notion of relaxed verifiability (see Section 5.1) and thus our positive results for unique signatures apply to such case.

The solution proposed by DFP is to modify the definition of undetectability so that the adversary (and not the user) specifies the input messages to the (potentially subverted) encryption algorithm, whereas the goal of the user is to detect the attack given access to the transcript of all queries made by the adversary (and answers to these queries). Hence, a scheme is said to be subversion-resilient if there exists a fixed polynomial-time test algorithm such that either a subversion attack cannot be detected efficiently but it does not leak any useful information, or it is possible to efficiently detect that the system was subverted.[10]

It is possible to make a similar change as in [DFP15] and adapt the DFP model to signature schemes in order to achieve security under chosen-message attacks. The end result would share some similarities with our approach using cryptographic

[10] For instance, in case of the attack outlined above, the polynomial-time test could simply decrypt the ciphertext and check the outcome matches the input message.

RFs;[11] however, our framework provides notable advantages. First, note that the DFP model does not provide any guarantee against SAs that are efficiently detectable, whereas our RF model explicitly accounts for the actions to be taken after an attack is detected; this is particularly relevant for signature schemes where our generic attack uncovered the necessity of a self-destruct capability. Second, the polynomial-time detection test in DFP is performed directly by the user since it requires knowledge of the secret key. This is problematic in practice since often the user's machine is completely compromised; instead, in our framework, a cryptographic RF for a signature scheme relies only on public information and could easily be located on a (untrusted) external proxy.

Tampering attacks. A related line of research analyzes the security of cryptosystems against tampering attacks. Most of these works are restricted to the simpler setting of memory tampering (sometimes known as related-key security), where only the secret key of a targeted cryptoscheme is subject to modification. By now we know several concrete primitives that remain secure against different classes of memory-tampering attacks, including pseudorandom functions and permutations [BK03, Luc04, BC10, AFPW11, BCM11], pseudorandom generators and hard-core bits [GL10], hash functions [GOR11], public-key encryption [AHI11, Wee12], identification and digital signature schemes [KKS11, DFMV13]. Elegant generic compilers are also available, relying on so-called tamper-resilient encodings and non-malleable codes (see, among others, [GLM+04, DPW10, LL12, FMNV14, FMVW14, ADL14, JW15, DLSZ15, AGM+15, FMNV15, DFMV15]).

The setting of randomness tampering, where the random coins of a cryptographic algorithm are subject to tampering, has also been considered. For instance Austrin *et al.* [ACM+14] consider so-called p-tampering attacks, that can efficiently tamper with each bit of the random tape with probability p. In this setting they show that some cryptographic tasks (including commitment schemes and zero-knowledge protocols) are impossible to achieve, while other tasks (in particular signature and identification schemes) can be securely realized.

Yet another related setting is that of tampering attacks against gates and wires in the computation of a cryptographic circuit, and the design of tamper-proof circuit compilers [IPSW06, FPV11, DK12, KT13, DK14, GIP+14].

Hardware Trojans. Prevention of hardware Trojans in integrated circuits (ICs) is a common practice, that might take place during the design, manufacturing, and post-manufacturing stage [Pot10, LJM11].

However, since it is not always possible to efficiently prevent Trojans insertion, Trojans *detection* has also been vastly explored [BR15]; once a Trojan is detected, the circuit can be disposed and not used. Common methodologies used to perform Trojans detection vary from invasive ones (that destroy the IC to examine it inside), to non-invasive ones (where the circuit is executed and compared against a trusted copy of the circuit or against some expected output values). Trojan detection is

[11] On a high level, one can interpret the polynomial-time test as playing the role of the reverse firewall.

typically a very expensive and unreliable process, therefore the best practice is usually not to rely on any kind of testing to protect against Trojans.

Explicit *countermeasures* against Trojans also exist, where the objective is to guarantee the functionality or security of the circuit even in the presence of some unknown Trojan. For instance, the so-called "data guards" are designed to prevent a Trojan from being activated and/or to access sensitive data [WS11]. Another approach is the duplication of logic elements and the division of the sensitive data to independent parts of the circuit [MWPB09, WS11].

To the best of our knowledge, our work is the first introducing a formal model for assessing security of ICs in the presence of arbitrary hardware Trojans. The only exception is [SB15] that considered an even stronger definition than ours (where the output of the produced circuit must always be the same as that of the original circuit), but could only achieve security for very limited classes of Trojans (i.e., the adversary is allowed to "corrupt" only a small fraction of the gates in each layer of the IC, and a small fraction of the wires connecting different layers).

Very recently, Wahby *et al.* [WHG+16] also explored the idea of using VC to address the issue of hardware Trojans in fabless circuit manufacturing. That paper, however, greatly differ in scope and techniques from this, and can be regarded as complementary as explained below.

First, our goal is to make sure the secret memory of the rebuilt circuit cannot be leaked to an outsider (even in the presence of arbitrary hardware Trojans), whereas [WHG+16] aims at the incomparable goal of ensuring *correctness* of the computation (which makes sense also for non-cryptographic functionalities); note that, as we prove, correctness implies security in our sense using self-destruct (up to logarithmic leakage on the memory). Second, our main focus is on precise definitions and proofs in the style of provable security (characterizing which properties are needed for the VC scheme in order for our approach to go through), while [WHG+16] addresses the orthogonal question of how to implement the idea of using VC for the problem at hand in practice (providing a concrete ASIC implementation based on an optimized existing VC scheme, and measuring its performances in terms of energy consumption, circuit area, and throughput). Lastly, we also provide a solution to the problem based on MPC.

In [DFS16] the authors show how to protect against hardware Trojans using testing-based mechanisms. Their work is based on two existing techniques for Trojan detection, called "input scrambling" and "split manufacturing", for which the authors provide formal models and they use them to construct a compiler satisfying a quantitative notion of security: They present a generic compiler that transforms any circuit into another one that satisfies certain guarantees with respect to the number of correct executions. The trusted verification mechanism of [DFS16] is simpler than the trusted components employed by our constructions, still our approach provides stronger security guarantees against broader classes of attackers, that are allowed to interact with the circuit for an arbitrary (polynomially many) number of executions.

Tamper-proof circuits. Our main security definition for circuits outsourcing shares similarities with analogous definitions in the context of protecting circuit implementations against tampering attacks. This line of research received considerable

attention in the past few years [IPSW06, FPV11, DK12, KT13, DK14].

The main difference between this setting and the one considered in this thesis is that tamper-proof circuit compilers are typically used to protect against fault injection [Ott06] and tampering attacks at run-time; such attacks are usually carried out in an adaptive manner, depending on the outcome of previous attempts. Outsourcing compilers, instead, only protect against (non-adaptive) tampering taking place during the circuit fabrication process. Importantly, the latter restriction allows to obtain security against arbitrary modifications, whereas in circuit tampering one has to consider very restricted attacks (e.g., wire tampering [IPSW06] or gate tampering [KT13]).

1.4 Roadmap

In Chapter 2 we introduce the notation used in this thesis and some basic cryptographic concepts and definitions needed for the subsequent chapters.

Next, in Chapter 3 we start the discussion on the uses of backdoors in cryptography for good and legitimate purposes; in Section 3.1 we introduce a new definition of enhanced collision resistance for chameleon hashes, in Section 3.2 we detail our transformation from a standard collision resistant CH to an enhanced collision resistant CH, presenting the proofs and constructions in the standard model and in the random oracle model. In Section 3.3 we give an overview of the blockchain technology, and discuss the motivations of the chameleon blockchain application (that is powered by an enhanced collision-resistant CH function) that we introduce later in Section 3.3.4.

In Chapter 4 we start in Section 4.2 by introducing the definitions of subversion attacks and undetectability on signature schemes. In Section 4.3 we present two subversion attacks against signature schemes; the first is an adaption of the attack from Bellare *et al.* to signatures that is successful against signature schemes with at least 7-bits of randomness. The second is a novel attack against coin-extractable schemes, with even a single bit of randmoness. Later, in Section 4.4 we extend our definitions and attacks to the multi-user setting, and we show the relations among the undetectability definition.

In Chapter 5 we start introducing in Section 5.1 the security definitions for subversion-resilient signature schemes, and we show that unique signature schemes satisfy this definition. In Section 5.1.6 we present the construction of a unique signature scheme with a tight security reduction to the quadratic residuosity problem. Later, in Section 5.1.4 we show how to construct subversion-resilient identification schemes from subversion-resilient signatures. Closing on the subversion-resilient signatures topic, in Section 5.2 we provide a way how to achieve the ambitious goal of protecting signature schemes against arbitrary subversion attacks, relying on a cryptographic reverse firewall. In Section 5.3.1 we introduce the problem of secure cryptographic circuit outsourcing, giving motivations and formal definitions. Later on, in Section 5.3.2 and Section 5.3.3 we detail our two outsourcing circuit compilers, one based on verifiable computation (VC), and the other based on client-server secure multiparty computation (MPC). The conclusion remarks in Chapter 6 discuss the impact of this thesis and future works on the topic.

CHAPTER 2

Preliminaries

2.1 Notation

For a string x, we denote its length by $|x|$; if $\mathcal{X}$ is a set, $|\mathcal{X}|$ represents the number of elements in $\mathcal{X}$. When x is chosen randomly in $\mathcal{X}$, we write $x \leftarrow\!\!\$\ \mathcal{X}$. When A is an algorithm, we write $y \leftarrow \mathsf{A}(x)$ to denote a run of A on input x and output y; if A is randomized, then y is a random variable and $\mathsf{A}(x;r)$ denotes a run of A on input x and randomness r. An algorithm A is *probabilistic polynomial-time* (PPT) if A is randomized and for any input $x, r \in \{0,1\}^*$ the computation of $\mathsf{A}(x;r)$ terminates in at most $poly(|x|)$ steps.

We denote with $\kappa \in \mathbb{N}$ the security parameter. A function $\nu : \mathbb{N} \to \mathbb{R}$ is negligible in the security parameter (or simply negligible) if it vanishes faster than the inverse of any polynomial in κ, i.e. $\nu(\kappa) = \kappa^{-\omega(1)}$.

The statistical distance between two random variables $\mathbf{A}$ and $\mathbf{B}$ defined over the same domain $\mathcal{D}$ is defined as $\Delta(\mathbf{A};\mathbf{B}) = \frac{1}{2}\sum_{x\in\mathcal{D}} |\mathbb{P}[\mathbf{A}=x] - \mathbb{P}[\mathbf{B}=x]|$.

We rely on the following lemma (which follows directly from the definition of statistical distance):

Lemma 1. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be a pair of random variables, and* E *be an event defined over the probability space of* $\mathbf{A}$ *and* $\mathbf{B}$*. Then,*

$$\Delta(\mathbf{A};\mathbf{B}) \leq \Delta(\mathbf{A};\mathbf{B}|\neg E) + \mathbb{P}[E].$$

For two ensembles $\mathbf{Z} := \{Z_\kappa\}_{\kappa\in\mathbb{N}}$ and $\mathbf{Z}' := \{Z'_\kappa\}_{\kappa\in\mathbb{N}}$, we write $\mathbf{Z} \equiv \mathbf{Z}'$ to denote that the two ensembles are identically distributed. We also write $\mathbf{Z} \approx_c \mathbf{Z}'$ to denote that the ensembles are computationally indistinguishable, i.e. for all PPT distinguishers D there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that

$$\Delta^{\mathsf{D}}(\mathbf{Z};\mathbf{Z}') := \left|\mathbb{P}[\mathsf{D}(z)=1 :\ z \leftarrow\!\!\$\ \mathbf{Z}] - \mathbb{P}[\mathsf{D}(z)=1 :\ z \leftarrow\!\!\$\ \mathbf{Z}']\right| \leq \nu(\kappa).$$

Definition 1 (Jacobi Symbol [Sho06])**.** *Let a, N be integers, where N is positive and odd, so that $N = q_1 \cdots q_k$, where q_i are odd primes, not necessarily distinct. Then the jacobi symbol $\boldsymbol{J}_N(a)$ is defined as $\boldsymbol{J}_N(a) = (a|q_i) \cdot (a|q_k)$, where $(a|q_i)$ is 1 if a is a quadratic residue modulo q_i and -1 otherwise.*

We denote with $\mathbf{J}_N^+$ the values $x \in \mathbb{Z}_N$ such that $\mathbf{J}_N(x) = 1$. The following lemma is needed for describing the signature scheme of Section 5.1.6.

Lemma 2. *Let $N = pq$ s.t. $p \equiv q \equiv 3 \mod 4$. We choose $y \in \mathbb{Z}_N^*$ to be a quadratic residue. The equation $y \equiv x^2 \mod N$ takes four distinct values, namely $\{x_0, -x_0, x_1, -x_1\}$ where $\boldsymbol{J}_N(x_0) = \boldsymbol{J}_N(-x_0)$, $\boldsymbol{J}_N(x_1) = \boldsymbol{J}_N(-x_1)$ and $\boldsymbol{J}_N(x_0) = -\boldsymbol{J}_N(x_1)$.*

Proof. When $p \equiv q \equiv 3 \mod 4$ then $\mathbf{J}_p(-1) = -1$, and $\mathbf{J}_q(-1) = -1$. Taken modulo N, we get $\mathbf{J}_N(-1) = 1$. For $i \in \{0, 1\}$, it is easy to see that $\mathbf{J}_N(-x_i) = \mathbf{J}_N(-1) \cdot \mathbf{J}_N(x_i) = 1 \cdot \mathbf{J}_N(x_i)$. □

2.2 Public-Key Cryptography

In this section we introduce some basic aspects and definitions of public-key cryptography.

2.2.1 Public-Key Encryption

A Public-Key Encryption (PKE) scheme is a tuple of efficient algorithms $\mathcal{PKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ defined as follows. (i) The probabilistic algorithm KGen takes as input the security parameter $\kappa \in \mathbb{N}$, and outputs a public/secret key pair (PK, SK). (ii) The probabilistic algorithm Enc takes as input the public key PK, a message $m \in \mathcal{M}$, and implicit randomness $\rho \in \mathcal{R}_{\text{pke}}$, and outputs a ciphertext $c = \mathsf{Enc}(PK, m; \rho)$. the set of all ciphertexts is denoted by $\mathcal{C}$. (iii) The deterministic algorithm Dec takes as input the secret key SK and a ciphertext $c \in \mathcal{C}$ and outputs $m = \mathsf{Dec}(SK, c)$ which is either equal to some message $m \in \mathcal{M}$ or to an error symbol $\perp$.

Correctness. A PKE scheme meets the correctness property if the decryption of a ciphertext encrypting a given plaintext yields the plaintext.

Definition 2 (Correctness for PKE)**.** *We say that $\mathcal{PKE}$ satisfies correctness if for all $(PK, SK) \leftarrow\!\!\$\ \mathsf{KGen}(1^\kappa)$ there exists a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that $\mathbb{P}[\mathsf{Dec}(SK, \mathsf{Enc}(PK, m)) = m] \geq 1 - \nu(\kappa)$ (where the randomness is taken over the internal coin tosses of algorithm Enc).*

Semantic security. The standard security notion for PKE schemes goes under the name of semantic security, and informally states that no efficient adversary given the public key can distinguish the encryption of two (possibly known) messages [GM84]. We note that the notions of semantic security and security against chosen-plaintext attacks (CPA), presented below, are equivalent.

Definition 3 (CPA security)**.** *Let $\mathcal{PKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be a PKE scheme. We say that $\mathcal{PKE}$ is CPA-secure if for all PPT adversaries A the following quantity is negligible:*

$$\mathbb{P}\left[b' = b : \begin{array}{c} b' \leftarrow\!\!\$\ \mathsf{A}(PK, c); c \leftarrow\!\!\$\ \mathsf{Enc}(PK, m_b); b \leftarrow\!\!\$\ \{0,1\} \\ (m_0, m_1) \leftarrow\!\!\$\ \mathsf{A}(PK); (PK, SK) \leftarrow\!\!\$\ \mathsf{KGen}(1^\kappa) \end{array}\right].$$

2.2.2 Signature Schemes

A signature scheme is a triple of algorithms $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ specified as follows: (i) KGen takes as input the security parameter κ and outputs a verification/signing key pair $(VK, SK) \in \mathcal{VK} \times \mathcal{SK}$, where $\mathcal{VK} := \mathcal{VK}_\kappa$ and $\mathcal{SK} := \mathcal{SK}_\kappa$ denote the sets of all verification and secret keys produced by $\mathsf{KGen}(1^\kappa)$; (ii) Sign takes as input the signing key $SK \in \mathcal{SK}$, a message $m \in \mathcal{M}$ and random coins $r \in \mathcal{R}$, and outputs a signature $\sigma \in \Sigma$; (iii) Vrfy takes as input the verification key $VK \in \mathcal{VK}$ and a pair (m, σ), and outputs a decision bit that equals 1 iff σ is a valid signature for message m under key VK.

Informally, correctness of a signature scheme says that verifying honestly generated signatures always works (with overwhelming probability over the randomness of all involved algorithms).

Definition 4 (Correctness for Signatures)**.** *Let $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme. We say that $\mathcal{SS}$ satisfies ν_c-correctness if for all $m \in \mathcal{M}$*

$$\mathbb{P}\left[\mathsf{Vrfy}(VK, (m, \mathsf{Sign}(SK, m))) = 1 : (VK, SK) \leftarrow \mathsf{KGen}(1^\kappa)\right] \geq 1 - \nu_c,$$

where the probability is taken over the randomness of KGen*,* Sign*, and* Vrfy*.*

The standard notion of security for a signature scheme demands that no PPT adversary given access to a signing oracle returning signatures for arbitrary messages, can forge a signature on a "fresh" message (not asked to the signing oracle).

Definition 5 (Existential Unforgeability)**.** *Let $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme. We say that $\mathcal{SS}$ is (t, q, ε)-existentially unforgeable under chosen-message attacks ((t, q, ε)-EUF-CMA in short) if for all PPT adversaries A running in time t it holds:*

$$\mathbb{P}\left[\mathsf{Vrfy}(VK, (m^*, \sigma^*)) = 1 \wedge m^* \notin \mathcal{Q} : \begin{array}{c} (VK, SK) \leftarrow \mathsf{KGen}(1^\kappa); \\ (m^*, \sigma^*) \leftarrow \mathsf{A}^{\mathsf{Sign}(SK, \cdot)}(VK) \end{array}\right] \leq \varepsilon,$$

where $\mathcal{Q} = \{m_1, \ldots, m_q\}$ denotes the set of queries to the signing oracle. Whenever $\varepsilon(\kappa) = \nu(\kappa)$ and $q = poly(\kappa)$, we simply say that $\mathcal{SS}$ is EUF-CMA.

Unique signatures. For our results in Section 5.1 we rely on so-called *unique* signatures, that we define next. Informally a signature scheme is unique if for any message there is a single signature that verifies w.r.t. an honestly generated verification key.

Definition 6 (Uniqueness)**.** *Let $\mathcal{SS}$ be a signature scheme. We say that $\mathcal{SS}$ satisfies ν_u-uniqueness if $\forall m \in \mathcal{M}$ and $\forall \sigma_1, \sigma_2$ s.t. $\sigma_1 \neq \sigma_2$*

$$\mathbb{P}\left[\mathsf{Vrfy}(VK, (m, \sigma_1)) = \mathsf{Vrfy}(VK, (m, \sigma_2)) = 1 : (VK, SK) \leftarrow \mathsf{KGen}(1^\kappa)\right] \leq \nu_u,$$

where the probability is taken over the randomness of the verification and key generation algorithms.

Full Domain Hash signatures with trapdoor permutations, for instance RSA-FDH [BR96], are unique. Sometimes unique signatures are also known under the name of *verifiable unpredictable functions* (VUFs).[1] Known constructions of VUFs exist based on strong RSA [MRV99], and on several variants of the Diffie-Hellman assumption in bilinear groups [Lys02, Dod03, DY05, ACF14, Jag15].

It is easy to see that a unique signature must be deterministic; however the converse is not true (i.e., a signature scheme can be deterministic without satisfying the uniqueness condition). One such example is the Katz-Wang signature scheme [KW03].

2.3 Pseudorandom Functions

Let $F : \{0,1\}^\kappa \times \mathcal{X} \to \mathcal{Y}$ be an efficient keyed function, where $\mathcal{X}$ and $\mathcal{Y}$ denote the domain and the range of F. Denote by $\mathcal{F}$ the set of all functions mapping $\mathcal{X}$ into $\mathcal{Y}$.

Definition 7 (Pseudorandom function)**.** *A function $F : \{0,1\}^\kappa \times \mathcal{X} \to \mathcal{Y}$ is a (t, q, ε)-secure pseudorandom function (PRF), if for all adversaries D running in time at most t we have*

$$\left| \mathop{\mathbb{P}}_{s \leftarrow\$ \{0,1\}^\kappa} \left[\mathsf{D}^{F_s(\cdot)}(1^\kappa) = 1\right] - \mathop{\mathbb{P}}_{f \leftarrow\$ \mathcal{F}} \left[\mathsf{D}^{f(\cdot)}(1^\kappa) = 1\right] \right| \leq \varepsilon,$$

where D asks at most q queries to its oracle.

We note that the existence of PRFs depends on the existence of one-way functions. We point the reader to [Gol01] for more details.

2.4 Trapdoor Permutations

A Trapdoor Permutation (TDP) is a family of polynomial-time algorithms $\mathcal{TDP} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Invert})$ defined as follows. (i) On input the security parameter κ the probabilistic algorithm Gen outputs a public description of a permutation α, an efficiently sampleable domain $\mathcal{X}_\alpha$ and a trapdoor tk. (ii) The deterministic algorithm Eval on input α and $x \in \mathcal{X}_\alpha$, outputs $y \in \mathcal{X}_\alpha$. (iii) The deterministic algorithm Invert on input tk and $y \in \mathcal{X}_\alpha$, outputs $x \in \mathcal{X}_\alpha$.

[1] Strictly speaking, VUFs satisfy a stronger requirement—namely the uniqueness property holds even for maliciously generated verification keys; the weak variant above is sufficient for the results of this thesis.

Definition 8 (Correctness for TDP)**.** *Let $\mathcal{TDP} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Invert})$ be a trapdoor permutation. We say that $\mathcal{TDP}$ satisfies correctness if for all $(\alpha, \mathcal{X}_\alpha, tk) \leftarrow \mathsf{Gen}(1^\kappa)$ and for all $x \in \mathcal{X}_\alpha$ there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that $\mathbb{P}[\mathsf{Invert}(tk, \mathsf{Eval}(\alpha, x)) = x] \geq 1 - \nu(\kappa)$.*

Definition 9 (One-way trapdoor permutation)**.** *Let $\mathcal{TDP} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Invert})$ be a trapdoor permutation. We say that $\mathcal{TDP}$ is one-way if for all PPT adversaries D there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that*

$$\mathbb{P}\left[(\alpha, tk) \leftarrow \mathsf{Gen}(1^\kappa); x \leftarrow \mathsf{D}(\alpha, y) : y \leftarrow \mathsf{Eval}(\alpha, x)\right] \leq \nu(\kappa).$$

2.5 Non-Interactive Zero-Knowledge

Let $R : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ be an *NP* relation on pairs (x, y), with corresponding language $L := \{y : \exists x \text{ s.t. } R(x, y) = 1\}$. A non-interactive argument for R allows a prover P to convince a verifier V that a common element y belongs to the language L (where both P and V are modeled as PPT algorithms); the prover P is facilitated by knowing a witness x for $y \in L$. A formal definition follows.

Definition 10 (Non-interactive argument)**.** *A non-interactive argument for an NP relation R is a tuple of efficient algorithms $\mathcal{NIA} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ specified as follows.*

- *$\omega \leftarrow\!\!\$\ \mathsf{Setup}(1^\kappa)$: The probabilistic algorithm Setup takes as input the security parameter $\kappa \in \mathbb{N}$, and outputs the public common reference string (CRS) ω.*
- *$\pi \leftarrow\!\!\$\ \mathsf{P}(\omega, x, y)$: The probabilistic algorithm P takes as input the CRS ω and a pair x, y such that $R(x, y) = 1$, and returns a proof π for membership of $y \in L$.*
- *$d = \mathsf{V}(\omega, y, \pi)$: The deterministic algorithm V takes as input the CRS ω and a pair (y, π), and returns a decision bit $d \in \{0,1\}$.*

Non-interactive arguments typically satisfy three properties known as completeness, zero-knowledge, and soundness, which we review below. We remark that the CRS is necessary for achieving non-interactive zero-knowledge (see, e.g., [Gol01]).

Completeness. The completeness property states that a honest prover (holding a valid witness x) should always be able to convince the verifier that $y \in L$.

Definition 11 (Completeness for arguments)**.** *Let $\mathcal{NIA} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be a non-interactive argument for an NP relation R. We say that $\mathcal{NIA}$ satisfies completeness if for all pairs (x, y) such that $R(x, y) = 1$, the following probability is overwhelming:*

$$\mathbb{P}\left[\mathsf{V}(\omega, y, \pi) = 1 : \ \pi \leftarrow\!\!\$\ \mathsf{P}(\omega, x, y); \omega \leftarrow\!\!\$\ \mathsf{Setup}(1^\kappa)\right].$$

Zero-knowledge. The zero-knowledge property informally says that a possibly malicious verifier cannot acquire any knowledge on the witness that it couldn't acquire by itself. Non-interactive zero-knowledge (NIZK) was first formalized by Blum, Feldman and Micali [BFM88].

Definition 12 (Zero-knowledge)**.** *Let $\mathcal{NIA} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be a non-interactive argument for an NP relation R. We say that $\mathcal{NIA}$ satisfies zero-knowledge if there exists a PPT simulator $\mathsf{S} := (\mathsf{S}_1, \mathsf{S}_2)$ such that for all adversaries A the following quantity is negligible:*

$$\left| \mathbb{P}\left[b = b' : \begin{array}{c} b' \leftarrow\!\!\$\ \mathsf{A}(\omega, \tau, \pi_b); \pi_0 \leftarrow\!\!\$\ \mathsf{P}(\omega, x, y); \pi_1 \leftarrow\!\!\$\ \mathsf{S}_2(\tau, y) \\ b \leftarrow\!\!\$\ \{0,1\}; (x, y) \leftarrow\!\!\$\ \mathsf{A}(\omega, \tau); (\omega, \tau) \leftarrow\!\!\$\ \mathsf{S}_1(1^\kappa) \end{array} \right] - \frac{1}{2} \right|.$$

Simulation extractability. The soundness property states that it is hard for a malicious prover to generate an accepting proof π for an element $y \notin L$. Below, we review a strictly stronger formulation of the soundness requirement which is known as simulation extractability, and informally says that soundness still holds even if the malicious prover can access simulated proofs for *true* statements.

This leads to the concept of true-simulation extractable (tSE) NIZK, as defined by Dodis, Haralambiev, Lòpez-Alt, and Wichs [DHLW10].

Definition 13 (True-simulation extractability)**.** *Let $\mathcal{NIA} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be a NIZK for an NP relation R, with zero-knowledge simulator $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$, and let f be an efficiently computable function. We say that $\mathcal{NIA}$ satisfies true-simulation f-extractability (f-tSE for short) if there exists a PPT extractor Ext such that for all PPT adversaries A the following quantity is negligible:*

$$\mathbb{P}\left[\begin{array}{c} y^* \notin \mathcal{Q} \wedge (\mathsf{V}(\omega, y^*, \pi^*) = 1) \\ \wedge \forall x^* \ s.t.\ f(x^*) = z^*(R(x^*, y^*) = 0) \end{array} : \begin{array}{c} z^* \leftarrow\!\!\$\ \mathsf{Ext}(\tau, y^*, \pi^*) \\ (y^*, \pi^*) \leftarrow\!\!\$\ \mathsf{A}^{\mathcal{O}_\tau(\cdot,\cdot)}(\omega) \\ (\omega, \tau) \leftarrow\!\!\$\ \mathsf{S}_1(1^\kappa) \end{array} \right],$$

where oracle $\mathcal{O}_\tau$ takes as input pairs (x_i, y_i) and returns the same as $\mathsf{S}_2(\tau, y_i)$ as long as $R(x_i, y_i) = 1$ (and $\perp$ otherwise), and $\mathcal{Q}$ is the set of all values y_i asked to oracle $\mathcal{O}_\tau$.

Note that in the above definition the adversary is only allowed to see simulated proof for *true* statements. A stronger variant (which is not needed in this thesis) requires that simulation extractability holds even if the adversary is allowed to see simulated proofs for possibly *false* statements. The latter property is also known under the name of robust NIZK [SCO+01, Gro06].

As noted in [DHLW10] tSE NIZK are significantly more efficient to construct, indeed they can be generically obtained combining any standard NIZK (such as the powerful Groth-Sahai NIZK [GS08]) with a CCA-secure PKE scheme.

2.6 Circuits

A (Boolean) circuit $\Gamma = (V, E)$ is a directed acyclic graph. The vertices V are logical gates, and the edges E are wires connecting the gates. For the case of *deterministic* circuits, the gates can be of type `AND`, `XOR` and `copy`, where `AND` (resp. `XOR`) have fan-in two and fan-out one, and output the AND (resp. XOR) operation on the input bits; a copy gate, denoted `copy`, simply forwards the input bit into two output wires.

The depth of a circuit is defined as the longest path from an input to an output; the size of a circuit is defined as its total number of gates. Sometimes we explicitly write $\langle \Gamma \rangle$ for the description of the circuit Γ.

A circuit is clocked if it evolves in clock cycles (or rounds). The input and output values of the circuit Γ in clock cycle i are denoted by X_i and Y_i, respectively. A circuit is *probabilistic* if it uses internal randomness as part of its logic. We call such probabilistic logic *randomness gates* and denote them with \$. In each clock cycle \$ outputs a fresh random bit. Additionally, a circuit may contain memory gates. Memory gates, which have a single incoming edge and any number of outgoing edges, maintain state: at any clock cycle, a memory gate sends its current state down its outgoing edges and updates it according to the value of its incoming edge. Any cycle in the circuit graph must contain at least one memory gate. The state of all memory gates at clock cycle i is denoted by M_i, with M_1 denoting the initial state. When a circuit is run in state M_i on input X_i, the circuit will output Y_i and the memory gates will be in a new state M_{i+1}. We will denote this by $(Y_i, M_{i+1}) \leftarrow \Gamma[M_i](X_i)$.

CHAPTER 3

The Good

In this chapter we discuss the beneficial use of backdoors in cryptographic primitives. We start by formally defining collision resistance of public/secret coin chameleon hash functions in Section 3.1. In Section 3.2 we introduce the main ingredients required by our generic chameleon hash transformation, which is described and analyzed in full details later in Section 3.2.

Finally, later in Section 3.2, we instantiate our transformation under standard complexity assumptions, both in the standard and in the random oracle model.

3.1 Chameleon Hash Functions

The concept of chameleon hashing was put forward by Krawczyk and Rabin [KR00], building on the notion of chameleon commitments [BCC88]. Informally, a chameleon hash is a cryptographic hash function that contains a trapdoor: Without the trapdoor it should be hard to find collisions, but knowledge of the trapdoor information allows to efficiently generate collisions for the hash function.

3.1.1 Secret-coin Hashing

We start by introducing a generalization of the standard concept of chameleon hashing to make it more relevant in practice. Our generalization is referred to as "secret-coin" and includes standard chameleon hashes as a special case (now referred to as "public-coin").

Definition 14 (Secret-coin chameleon hash)**.** *A* secret-coin *chameleon hash function is a tuple of efficient algorithms* $\mathcal{CH} = (\mathsf{HGen}, \mathsf{Hash}, \mathsf{HVer}, \mathsf{HCol})$ *specified as follows.*

- $(hk, tk) \leftarrow\!\!\$\ \mathsf{HGen}(1^\kappa)$*: The probabilistic key generation algorithm* HGen *takes as input the security parameter* $\kappa \in \mathbb{N}$*, and outputs a public hash key* hk *and a secret trapdoor key* tk*.*

- $(h, \xi) \leftarrow\!\!\$\ \mathsf{Hash}(hk, m)$*: The probabilistic hashing algorithm* Hash *takes as input the hash key* hk*, a message* $m \in \mathcal{M}$*, and implicit random coins* $r \in \mathcal{R}_{\text{hash}}$*, and outputs a pair* (h, ξ) *that consists of the hash value* h *and a check string* ξ*.*

- $d = \mathsf{HVer}(hk, m, (h, \xi))$*: The deterministic verification algorithm* HVer *takes as input a message* $m \in \mathcal{M}$*, a candidate hash value* h*, and a check string* ξ*, and returns a bit* d *that equals* 1 *if* (h, ξ) *is a valid hash/check pair for the message* m *(otherwise* d *equals* 0*).*

- $\pi' \leftarrow\!\!\$\ \mathsf{HCol}(tk, (h, m, \xi), m')$*: The probabilistic collision finding algorithm* HCol *takes as input the trapdoor key* tk*, a valid tuple* (h, m, ξ)*, and a new message* $m' \in \mathcal{M}$*, and returns a new check string* ξ' *such that* $\mathsf{HVer}(hk, m, (h, \xi)) = \mathsf{HVer}(hk, m', (h, \xi')) = 1$*. If* (h, ξ) *is not a valid hash/check pair for message* m *then the algorithm returns* $\perp$*.*

Correctness informally says that a pair (h, ξ), computed by running the hashing algorithm, verifies with overwhelming probability.

Definition 15 (Correctness for chameleon hashing)**.** *Let* $\mathcal{CH} = (\mathsf{HGen}, \mathsf{Hash}, \mathsf{HVer}, \mathsf{HCol})$ *be a secret-coin chameleon hash function with message space* $\mathcal{M}$*. We say that* $\mathcal{CH}$ *satisfies correctness if for all* $m \in \mathcal{M}$ *there exists a negligible function* $\nu : \mathbb{N} \to [0, 1]$ *such that*

$$\mathbb{P}\left[\mathsf{HVer}(hk, m, (h, \xi)) = 1 : (h, \xi) \leftarrow\!\!\$\ \mathsf{Hash}(hk, m); (hk, tk) \leftarrow\!\!\$\ \mathsf{HGen}(1^\kappa)\right] \geq 1 - \nu(\kappa).$$

3.1.2 Public-coin Hashing

In the definition above the hashing algorithm is randomized, and, upon input some message m, it produces a hash value h together with a check value ξ that helps verifying the correct computation of the hash given the public hash key. The random coins of the hashing algorithm are, however, secret. A particular case is the one where the check value ξ consists of the random coins r used to generate h, as the hash computation becomes completely deterministic once m and r are fixed; we call such a chameleon hash function *public-coin* and we define it formally below. Since the verification algorithm simply re-runs the hashing algorithm, we typically drop the verification algorithm from $\mathcal{CH}$ in the case of public-coin chameleon hashing.

Definition 16 (Public-coin chameleon hash)**.** *A* public-coin *chameleon hash is a collection of efficient algorithms* $\mathcal{CH} = (\mathsf{HGen}, \mathsf{Hash}, \mathsf{HVer}, \mathsf{HCol})$ *specified as in Definition 14, with the following differences:*

- *The hashing algorithm* Hash*, upon input the hash key* hk *and message* $m \in \mathcal{M}$*, returns a pair* (h, r)*, where* $r \in \mathcal{R}_{\text{hash}}$ *denote the implicit random coins used to generate the hash value.*

- *The verification algorithm* HVer*, given as input the hash key* hk*, message* m*, and a pair* (h, r)*, returns* 1 *if and only if* $\mathsf{Hash}(hk, m; r) = h$*.*

Collision resistance. The main security property satisfied by a secret/public-coin chameleon hash function is that of collision resistance: No PPT algorithm, given the public hash key hk, can find two pairs (m, ξ) and (m', ξ') that are valid under hk and such that $m \neq m'$, with all but a negligible probability. Furthermore, for the application that we devise in Section 3.3, it is important that the above still holds even after seeing arbitrary collisions generated using the trapdoor key tk corresponding to hk.

3.1.3 Enhanced Collision Resistance

A *collision* for a secret-coin or public-coin hash function is a tuple $h, (m, \xi), (m', \xi')$ such that $m \neq m'$, and (h, ξ) and (h, ξ') are valid hash/check pairs for m and m' (respectively). For a chameleon hash function we require the following security property, which intuitively says that it should be hard to find collisions for the hash function even given access to the collision finding algorithm (returning collisions for adaptively chosen hash values). We call such a property *enhanced* collision resistance, and we define it formally below.

Definition 17 (Enhanced collision resistance)**.** *Let $\mathcal{CH} = (\mathsf{HGen}, \mathsf{Hash}, \mathsf{HVer}, \mathsf{HCol})$ be a (secret-coin or public-coin) chameleon hash function. We say that $\mathcal{CH}$ satisfies enhanced collision resistance if for all PPT breakers B, the following quantity is negligible in the security parameter:*

$$\mathbb{P}\left[\begin{array}{c} (\mathsf{HVer}(hk, m, (h, \xi)) = \\ \mathsf{HVer}(hk, m', (h, \xi')) = 1) \\ \wedge (m \neq m') \wedge (h \notin \mathcal{Q}) \end{array} : \begin{array}{c} (h, (m, \xi), (m', \xi')) \leftarrow_{\$} \mathsf{B}^{\mathcal{O}_{hk,tk}(\cdot)}(hk) \\ (hk, tk) \leftarrow_{\$} \mathsf{HGen}(1^{\kappa}) \end{array}\right],$$

where the set $\mathcal{Q}$ is the set of all hash values queried by B to its oracle, and oracle $\mathcal{O}_{hk,tk}$ is defined as follows: Upon input a collision query of the form $((h, m, \xi), m')$ run $\mathsf{HVer}(hk, m, (h, \xi)) := d$; if $d = 1$ return the output of $\mathsf{HCol}(tk, (h, m, \xi), m')$, otherwise return $\perp$. In case B is not allowed to query oracle $\mathcal{O}_{hk,tk}$, we simply say that $\mathcal{CH}$ is collision-resistant.

Discussion. Any standard chameleon hash (e.g., the ones considered in [Dam87, KR00, ST01, BR14]) is easily seen to imply a public-coin collision-resistant chameleon hash as specified in Definition 16. Let us stress, however, that secret-coin chameleon hash functions can be used for the very same applications as public-coin ones, in particular for constructing chameleon signatures [KR00] and online/offline signatures [EGM96, ST01, BCR+13]; the only difference is that one needs to store the check value ξ (instead of the randomness r) in order to verify a hash value, and the hash verification does not in general consist of re-computing the hash.

Unfortunately, as observed by Ateniese and de Medeiros [AdM04], collision resistance is not sufficient for most of the applications of chameleon hash. The reason is that, while the hash function is indeed collision-resistant, any party seeing a collision for the hash function would be able to find other collisions or even recover the secret trapdoor information. This "key exposure" problem makes chameleon hashes not applicable in many contexts. Enhanced collision resistance, as defined

above, precisely addresses such issue as it requires that it should be hard to find collisions even after seeing (polynomially many) collisions.

Yet another flavor of chameleon hashing consists of so-called "labeled" hash functions, where the hash algorithm takes as input an additional value λ called the label. Some of these constructions, e.g. the ones in [AdM04, CZK04, CTZD10, CZS+10], do not suffer from the key exposure problem, as they satisfy the property that it should be unfeasible to find collisions for a "fresh" label λ^*, even given access to an oracle that outputs collisions for arbitrary other labels $\lambda \neq \lambda^*$. [1] However, labeled chameleon hash functions are not useful for constructing online/offline signatures and for the type of application developed in Section 3.3.

3.2 Chameleon Hash Transformation

To the best of our knowledge, the only construction of a chameleon hash function satisfying *enhanced* collision resistance is due to [AdM04]; the construction is ad-hoc and relies on the Nyberg-Rueppel signature scheme [NR94] (whose security can be shown under the Discrete Logarithm assumption in the generic group model [Sho97]).

Previously to our work it was unknown whether enhanced collision resistance can be achieved in a non ad-hoc fashion, based on different complexity assumptions in the standard model. We answer this open question in the affirmative, by exhibiting a generic transformation from any public-coin collision-resistant chameleon hash to a secret-coin chameleon hash satisfying the stronger *enhanced* collision resistance requirement. The transformation is based on a CPA-secure PKE scheme (cf. Section 2.2.1) and on a tSE NIZK [DHLW10] (cf. Section 2.5), and is presented in detail below.

3.2.1 The Generic Transformation

Let $\mathcal{CH} = (\mathsf{HGen}, \mathsf{Hash}, \mathsf{HCol})$ be a public-coin chameleon hash function (with message space $\mathcal{M}_{\text{hash}}$ and randomness space $\mathcal{R}_{\text{hash}}$), let $\mathcal{PKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be a PKE scheme (with message space $\mathcal{R}_{\text{hash}}$ and randomness space $\mathcal{R}_{\text{pke}}$), and let $\mathcal{NIA} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be a non-interactive argument system for the language

$$L_{\mathcal{CH}} = \left\{ (PK, c, hk, h, m) : \exists (r, \rho) \text{ s.t. } \begin{array}{l} h = \mathsf{Hash}(hk, m; r) \wedge \\ c = \mathsf{Enc}(PK, r; \rho) \end{array} \right\} \tag{3.1}$$

Consider the secret-coin chameleon hash function $\mathcal{CH}^* = (\mathsf{HGen}^*, \mathsf{Hash}^*, \mathsf{HVer}^*, \mathsf{HCol}^*)$ specified as follows.

- $\mathsf{HGen}^*(1^\kappa)$: Run $(hk, tk) \leftarrow\!\!\$\ \mathsf{HGen}(1^\kappa)$, sample $(PK, SK) \leftarrow\!\!\$\ \mathsf{KGen}(1^\kappa)$, and $\omega \leftarrow\!\!\$\ \mathsf{Setup}(1^\kappa)$. Return the pair (hk^*, tk^*), such that $hk^* := (hk, \omega, PK)$, and $tk^* := (tk, SK)$.
- $\mathsf{Hash}^*(hk^*, m)$: Sample a random value $r \in \mathcal{R}_{\text{hash}}$ and run $\mathsf{Hash}(hk, m; r) := h$. Sample a random value $\rho \in \mathcal{R}_{\text{pke}}$ and run $c := \mathsf{Enc}(PK, r; \rho)$. Compute the

[1] Identity-based chameleon hash functions [AdM04, CZT+11, CZS+14] also partially address the key exposure problem, but they require a trusted party and thus only offer a partial solution.

Game $\mathbf{G}_{0\text{-}3}$, $\mathbf{G}_{1\text{-}3}$:
$(PK, SK) \leftarrow\$ \mathsf{KGen}(1^\kappa)$
$\omega \leftarrow\$ \mathsf{Setup}(1^\kappa)$; $(\omega, \tau) \leftarrow\$ \mathsf{S}_1(1^\kappa)$
$(hk, tk) \leftarrow\$ \mathsf{HGen}(1^\kappa)$
$hk^* := (hk, PK, \omega)$
$tk^* := (tk, SK)$; $tk^* = (tk, SK, \tau)$
$(h, m, \xi, m', \xi') \leftarrow\$ \mathsf{B}^{*\mathcal{O}_{hk^*, tk^*}(\cdot)}(hk^*)$

Oracle $\mathcal{O}_{hk^*, tk^*}((h, m, \xi), m')$:
$//\mathbf{G}_{0\text{-}3}$, $\mathbf{G}_{1\text{-}3}$, $\mathbf{G}_{2\text{-}3}$, $\mathbf{G}_3$
Parse $\xi := (c, \pi)$; $y = (PK, c, hk, h, m)$
If $\mathsf{V}(\omega, y, \pi) = 0$
  Return $\perp$
$r := \mathsf{Dec}(SK, c)$; $r \leftarrow\$ \mathsf{Ext}(\tau, y, \pi)$
$r' \leftarrow\$ \mathsf{HCol}(tk, (h, m, r), m')$
$\rho' \leftarrow\$ \mathcal{R}_{\mathrm{pke}}$
$c' = \mathsf{Enc}(PK, r'; \rho')$; $c' = \mathsf{Enc}(PK, 0; \rho)$
$y' := (PK, c', hk, h, m')$
$x' := (r', \rho')$
$\pi' \leftarrow\$ \mathsf{P}(\omega, x', y')$; $\pi' \leftarrow\$ \mathsf{S}_2(\tau, y')$
Return $\xi' := (c', \pi')$.

Figure 3.1. Games in the proof of Theorem 1.

proof $\pi \leftarrow\$ \mathsf{P}(\omega, x, y)$, where $x := (r, \rho)$ and $y := (PK, c, hk, h, m)$, and return (h, ξ) such that $\xi := (c, \pi)$.

- $\mathsf{HVer}^*(hk^*, m, (h, \xi))$: Parse $\xi = (c, \pi)$ and return the output of $\mathsf{V}(\omega, y, \pi)$ where $y = (PK, c, hk, h, m)$.
- $\mathsf{HCol}^*(tk^*, (h, m, \xi), m')$: First run $\mathsf{HVer}(hk^*, m, (h, \xi)) := d$; if $d = 0$ then output $\perp$, otherwise, decrypt the randomness $r := \mathsf{Dec}(SK, c)$, compute a collision $r' \leftarrow\$ \mathsf{HCol}(tk, (h, m, r), m')$, sample a random $\rho' \in \mathcal{R}_{\mathrm{pke}}$ and encrypt the new randomness $c' := \mathsf{Enc}(PK, r'; \rho')$. Compute the proof $\pi' \leftarrow\$ \mathsf{P}(\omega, x', y')$, such that $x' = (r', \rho')$ and $y' := (PK, c', hk, h, m')$, and return $\xi' := (c', \pi')$.

The correctness property follows readily from the correctness of the underlying building blocks. As for security, we show the following result.

Theorem 1. *Assume that $\mathcal{CH}$ is a public-coin collision-resistant chameleon hash function, that $\mathcal{PKE}$ is a CPA-secure PKE scheme, and that $\mathcal{NIA}$ is an f-tSE-NIZK for the language of Eq. (3.1), where for any witness (r, ρ) we define $f(r, \rho) = r$. Then the above defined secret-coin chameleon hash function $\mathcal{CH}^*$ satisfies enhanced collision resistance.*

Proof. The proof is by game hopping. We define a series of games, starting with the original game for enhanced collision resistance of our construction $\mathcal{CH}^*$. Next, we argue that each pair of adjacent games is computationally indistinguishable and additionally that any PPT breaker has only a negligible advantage in the last game; this yields the theorem.

Below we give a concise description of the games, focusing only on the incremental changes between each game and the previous one; a full description of the games appears in Fig. 3.1.

Game $\mathbf{G}_0$: This is the original experiment of Definition 17, running with our secret-coin chameleon hash function $\mathcal{CH}^*$ and a PPT breaker B^*.

Game $\mathbf{G}_1$: We change the way collision queries are answered. In particular, we compute the proof π' by running the zero-knowledge simulator $\mathsf{S} = (\mathsf{S}_1, \mathsf{S}_2)$ instead of running the real prover P. In order to do so, we first set-up the CRS by running $(\omega, \tau) \leftarrow\!\!\$\ \mathsf{S}_1(1^\kappa)$ and later generate π' by running $\mathsf{S}_2(\tau, \cdot)$.

Game $\mathbf{G}_2$: We change the way collision queries are answered. In particular, instead of recovering the randomness r by decrypting the ciphertext c, we now compute r by extracting the proof π. In order to do so, we first set-up the CRS by running $(\omega, \tau) \leftarrow\!\!\$\ \mathsf{S}_1(1^\kappa)$ and later recover r by running $\mathsf{Ext}(\tau, \cdot, \cdot)$.

Game $\mathbf{G}_3$: We change the way collision queries are answered. In particular, instead of first equivocating the hash value h yielding some new randomness r' (corresponding to message m') and then computing c' as an encryption of r', we simply let c' be an encryption of zero.

In each game $\mathbf{G}_i$, for $i \in [0, 3]$, we define the event $\mathbf{G}_i = 1$ to be the event that B^* wins in the corresponding game, namely that the tuple (h, m, ξ, m', ξ') returned by B^* is such that $m \neq m'$ and both proofs π and π' contained in ξ and ξ' are accepting. Next, we analyze the computational distance between each pair of adjacent games.

Claim 1.1. *For all PPT distinguishers* D *there exists a negligible function* $\nu_{0,1} : \mathbb{N} \to [0, 1]$ *such that* $|\mathbb{P}[\mathsf{D}(\mathbf{G}_0(\kappa)) = 1] - \mathbb{P}[\mathsf{D}(\mathbf{G}_1(\kappa)) = 1]| \leq \nu_{0,1}(\kappa)$.

Proof of claim. Assume that there exists a PPT distinguisher D and a polynomial $p_{0,1}(\cdot)$ such that, for infinitely many values of $\kappa \in \mathbb{N}$, we have that D distinguishes between game $\mathbf{G}_0$ and game $\mathbf{G}_1$ with probability at least $1/p_{0,1}(\kappa)$. Let $q \in poly(\kappa)$ be the number of queries that D is allowed to ask to its oracle. For an index $i \in [0, q]$ consider the hybrid game $\mathbf{H}_i$ that answers the first i queries as in game $\mathbf{G}_0$ and all the subsequent queries as in game $\mathbf{G}_1$. Note that $\mathbf{H}_0 \equiv \mathbf{G}_1$ and $\mathbf{H}_q \equiv \mathbf{G}_0$.

By a standard hybrid argument, we have that there exists an index $i \in [0, q]$ such that D tells apart $\mathbf{H}_{i-1}$ and $\mathbf{H}_i$ with non-negligible probability $1/q \cdot 1/p_{0,1}(\kappa)$. We build a PPT adversary A that (using distinguisher D) breaks the non-interactive zero-knowledge property of $\mathcal{NIA}$. A formal description of A follows.

Adversary A:

- The challenger runs $(\omega, \tau) \leftarrow\!\!\$\ \mathsf{S}_1(1^\kappa)$ and forwards (ω, τ) to A.
- Run $(hk, tk) \leftarrow\!\!\$\ \mathsf{HGen}(1^\kappa)$, sample $(PK, SK) \leftarrow\!\!\$\ \mathsf{KGen}(1^\kappa)$, and send $hk^* := (hk, PK, \omega)$ to D.
- Upon input a collision query of type $((h_j, m_j, \xi_j), m'_j)$ from D, such that $\xi_j = (c_j, \pi_j)$, first check whether $\mathsf{V}(\omega, (PK, c_j, hk, h_j, m), \pi_j) = 0$. In case this happens return $\perp$, otherwise: Decrypt the randomness $r_j := \mathsf{Dec}(SK, c_j)$, find a collision $r'_j \leftarrow\!\!\$\ \mathsf{HCol}(tk, (h_j, m_j, r_j), m'_j)$, sample $\rho'_j \leftarrow\!\!\$\ \mathcal{R}_{\text{pke}}$, and let $c'_j = \mathsf{Enc}(PK, r'_j; \rho'_j)$. Hence:
  - If $j \leq i-1$, compute $\pi' \leftarrow\!\!\$\ \mathsf{P}(\omega, (PK, c'_j, hk, h_j m'_j), (r'_j, \rho'_j))$ and return $\xi'_j := (c'_j, \pi'_j)$ to D.

  - If $j = i$, forward $((r'_j, \rho'_j), (PK, c'_j, hk, h_j, m'_j))$ to the challenger obtaining a proof π'_j; return $\xi'_j := (c'_j, \pi'_j)$ to D.
  - If $j \geq i{+}1$, compute $\pi'_j \leftarrow\!\!\$\ \mathsf{S}_2(\tau, (PK, c'_j, hk, h_j, m'_j))$ and return $\xi' := (c'_j, \pi'_j)$.
- Output whatever D outputs.

For the analysis, note that the only difference between game $\mathbf{H}_{i-1}$ and game $\mathbf{H}_i$ is on how the i-th query is answered. In particular, in case the hidden bit b in the definition of non-interactive zero-knowledge equals zero A's simulation produces exactly the same distribution as in $\mathbf{H}_{i-1}$, and otherwise A's simulation produces exactly the same distribution as in $\mathbf{H}_i$. Hence, A breaks the NIZK property with non-negligible advantage $1/q \cdot 1/p_{0,1}(\kappa)$, a contradiction. This concludes the proof. □

Claim 1.2. *For all PPT distinguishers* D *there exists a negligible function* $\nu_{1,2} : \mathbb{N} \to [0,1]$ *such that* $|\mathbb{P}[\mathsf{D}(\mathbf{G}_1(\kappa)) = 1] - \mathbb{P}[\mathsf{D}(\mathbf{G}_2(\kappa)) = 1]| \leq \nu_{1,2}(\kappa)$.

Proof of claim. Let $q \in poly(\kappa)$ be the number of collision queries that the adversary is allowed to ask to its oracle, where each query has a type $((h_j, m_j, \xi_j), m'_j)$ for some $\xi'_j = (c_j, \pi_j)$. Define the following "bad event" E, in the probability space of game $\mathbf{G}_1$: The event becomes true if there exists an index $i \in [q]$ such that the proof π_i is accepting for $(PK, c_i, hk, h_i, m_i) \in L_{\mathcal{CH}^*}$, but running the extractor $\mathsf{Ext}(\tau, \cdot, \cdot)$ on (y_i, π_i) yields a value r_i such that $h_i \neq \mathsf{Hash}(hk, m_i; r_i)$, whereas this does not happen if r_i is computed as in $\mathbf{G}_1$.

Notice that $\mathbf{G}_1(\kappa)$ and $\mathbf{G}_2(\kappa)$ are identically distributed conditioning on E not happening. Hence, by a standard argument, it suffices to bound the probability of provoking event E by all PPT adversaries D. Assume that there exists a PPT distinguisher D and a polynomial $p_{1,2}(\cdot)$ such that, for infinitely many values of $\kappa \in \mathbb{N}$, we have that D provokes event E with probability at least $1/p_{1,2}(\kappa)$.

We build an adversary A that (using distinguisher D) breaks true-simulation extractability of $\mathcal{NIA}$ (for the function f defined in the theorem statement). A formal description of A follows.

Adversary A:

- The challenger runs $(\omega, \tau) \leftarrow\!\!\$\ \mathsf{S}_1(1^\kappa)$ and forwards ω to A.
- Run $(hk, tk) \leftarrow\!\!\$\ \mathsf{HGen}(1^\kappa)$, sample $(PK, SK) \leftarrow\!\!\$\ \mathsf{KGen}(1^\kappa)$, and send $hk^* := (hk, PK, \omega)$ to D.
- Upon input a collision query of type $((h_j, m_j, \xi_j), m'_j)$ from D, such that $\xi_j = (c_j, \pi_j)$, first check whether $\mathsf{V}(\omega, (PK, c_j, hk, h_j, m_j), \pi_j) = 0$. In case this happens return $\perp$, otherwise: Decrypt the randomness $r_j := \mathsf{Dec}(SK, c_j)$, find a collision $r'_j \leftarrow\!\!\$\ \mathsf{HCol}(tk, (h_j, m_j, r_j), m'_j)$, sample $\rho'_j \leftarrow\!\!\$\ \mathcal{R}_{\text{pke}}$ and encrypt $c'_j := \mathsf{Enc}(PK, r'_j; \rho'_j)$. Forward (x'_j, y'_j) to the target oracle, where $x'_j := (r'_j, \rho'_j)$ and $y'_j := (PK, c'_j, hk, h_j, m'_j)$, obtaining a simulated proof π'_j and forward $\xi'_j := (c'_j, \pi'_j)$.

- After D is done with its queries, sample a random index $i \leftarrow\!\!\$\, [q]$ and forward (y^*, π^*) to the challenger, where the values $y^* := (PK, c_i, hk, h_i, m_i)$ and $\pi^* := \pi_i$ are taken from D's i-th query to the collision oracle.

For the analysis, we note that A's simulation is perfect as the answer to D's queries to the collision oracle are distributed exactly as in $\mathbf{G}_1$. Thus, D provokes event E with probability $1/p_{1,2}(\kappa)$ and so the pair (y^*, π^*) violates the f-tSE property of the non-interactive argument with non-negligible probability $1/q \cdot 1/p_{1,2}(\kappa)$. The claim follows. □

Claim 1.3. *For all PPT distinguishers* D *there exists a negligible function* $\nu_{2,3} : \mathbb{N} \to [0,1]$ *such that* $|\mathbb{P}[\mathsf{D}(\mathbf{G}_2(\kappa)) = 1] - \mathbb{P}[\mathsf{D}(\mathbf{G}_3(\kappa)) = 1]| \leq \nu_{2,3}(\kappa)$.

Proof of claim. Assume that there exists a PPT distinguisher D and a polynomial $p_{2,3}(\cdot)$ such that, for infinitely many values of $\kappa \in \mathbb{N}$, we have that D distinguishes between game $\mathbf{G}_2$ and game $\mathbf{G}_3$ with probability at least $1/p_{2,3}(\kappa)$. Let $q \in poly(\kappa)$ be the number of queries that D is allowed to ask to its oracle. For an index $i \in [0, q]$ consider the hybrid game $\mathbf{H}_i$ that answers the first i queries as in game $\mathbf{G}_2$ and all the subsequent queries as in game $\mathbf{G}_3$. Note that $\mathbf{H}_0 \equiv \mathbf{G}_3$ and $\mathbf{H}_q \equiv \mathbf{G}_2$.

By a standard hybrid argument, we have that there exists an index $i \in [0, q]$ such that D tells apart $\mathbf{H}_{i-1}$ and $\mathbf{H}_i$ with non-negligible probability $1/q \cdot 1/p_{2,3}(\kappa)$. We build a PPT adversary A that (using distinguisher D) breaks CPA security of $\mathcal{PKE}$. A formal description of A follows.

Adversary A:

- Receive PK from the challenger, where $(PK, SK) \leftarrow\!\!\$\, \mathsf{KGen}(1^\kappa)$.
- Run $(hk, tk) \leftarrow\!\!\$\, \mathsf{HGen}(1^\kappa)$, $(\omega, \tau) \leftarrow\!\!\$\, \mathsf{S}_1(1^\kappa)$, and send $hk^* := (hk, PK, \omega)$ to D.
- Upon input a collision query of type $((h_j, m_j, \xi_j), m'_j)$ from D, such that $\xi_j = (c_j, \pi_j)$, first check whether $\mathsf{V}(\omega, (PK, c_j, hk, h_j, m), \pi_j) = 0$. In case this happens return $\perp$, otherwise: Extract the randomness $r_j := \mathsf{Ext}(\tau, (PK, c_j, hk, h_j, m), \pi_j)$ and find a collision $r'_j \leftarrow\!\!\$\, \mathsf{HCol}(tk, (h_j, m_j, r_j), m'_j)$. Hence:
  - If $j \leq i-1$, sample a random $\rho'_j \in \mathcal{R}_{\text{pke}}$, encrypt $c'_j := \mathsf{Enc}(PK, r'_j; \rho'_j)$, simulate a proof $\pi'_j \leftarrow\!\!\$\, \mathsf{S}_2(\tau, (PK, c'_j, hk, h_j, m'))$, and return $\xi'_j := (c'_j, \pi'_j)$ to D.
  - If $j = i$, forward $(r'_j, 0)$ for the challenger receiving back a ciphertext c'_j; simulate a proof $\pi'_j \leftarrow\!\!\$\, \mathsf{S}_2(\tau, (PK, c'_j, hk, h_j, m'))$, and return $\xi'_j := (c'_j, \pi'_j)$ to D.
  - If $j \geq i+1$, sample a random $\rho'_j \in \mathcal{R}_{\text{pke}}$, encrypt $c'_j := \mathsf{Enc}(PK, 0; \rho'_j)$, simulate a proof $\pi'_j \leftarrow\!\!\$\, \mathsf{S}_2(\tau, (PK, c'_j, hk, h_j, m'))$, and return $\xi'_j := (c'_j, \pi'_j)$ to D.
- Output whatever D outputs.

For the analysis, note that the only difference between game $\mathbf{H}_{i-1}$ and game $\mathbf{H}_i$ is on how the i-th collision query is answered. In particular, in case the hidden bit b in the definition of CPA security equals zero, A's simulation produces exactly the same distribution as in $\mathbf{H}_{i-1}$, and otherwise A's simulation produces exactly the same distribution as in $\mathbf{H}_i$. Hence, A breaks CPA security with non-negligible advantage $1/q \cdot 1/p_{2,3}(\kappa)$, a contradiction. This concludes the proof. □

Finally, we show that any PPT adversary has a negligible success probability in game $\mathbf{G}_3$, which concludes the proof of the theorem.

Claim 1.4. *For all PPT breakers* B^* *there exists a negligible function* $\nu_3 : \mathbb{N} \to [0,1]$ *such that* $\mathbb{P}[\mathbf{G}_3(\kappa) = 1] \leq \nu_3(\kappa)$.

Proof of claim. The proof is down to collision resistance of the underlying public-coin chameleon hash function $\mathcal{CH}$. Namely, assume that there is a PPT breaker B^* and a polynomial $p_3(\cdot)$ such that, for infinitely many values of $\kappa \in \mathbb{N}$, we have $\mathbb{P}[\mathbf{G}_3(\kappa) = 1] \geq 1/p_3(\kappa)$. We build a PPT breaker B that (using breaker B^*) breaks collision resistance of $\mathcal{CH}$ as follows.

Adversary B:

- Receive hk from the challenger, where $(hk, tk) \leftarrow\!\!\$\ \mathsf{HGen}(1^\kappa)$.
- Run $(\omega, \tau) \leftarrow\!\!\$\ \mathsf{S}_1(1^\kappa)$, $(PK, SK) \leftarrow\!\!\$\ \mathsf{KGen}(1^\kappa)$, and send $hk^* := (hk, PK, \omega)$ to B^*.
- Upon input a collision query of type $((h_j, m_j, \xi_j), m'_j)$ from B^*, such that $\xi_j = (c_j, \pi_j)$, answer as this would be done in game $\mathbf{G}_3$. Note that this can be done because the way collision queries are treated in $\mathbf{G}_3$ is completely independent on the trapdoor information tk.

  In particular, first check whether $\mathsf{V}(\omega, (PK, c_j, hk, h_j, m), \pi_j) = 0$. In case this happens return $\perp$, otherwise: Extract the randomness $r_j := \mathsf{Ext}(\tau, (PK, c_j, hk, h_j, m), \pi_j)$, sample a random $\rho'_j \in \mathcal{R}_{\mathrm{pke}}$, encrypt $c'_j := \mathsf{Enc}(PK, 0; \rho'_j)$, and simulate $\pi'_j \leftarrow\!\!\$\ \mathsf{S}_2(\tau, (PK, c'_j, hk, h_j, m'_j))$; return $\xi'_j := (c'_j, \pi'_j)$ to B^*.
- Eventually B^* outputs the tuple (h, m, ξ, m', ξ'). When this happens, let $r := \mathsf{Dec}(SK, r)$ and $r' := \mathsf{Dec}(SK, c')$, and output (h, m, r, m', r') to the challenger.

For the analysis, note that B perfectly simulates B^*'s queries to the collision oracle. It follows that, with probability at least $1/p_3(\kappa)$, adversary B^* outputs a collision for $\mathcal{CH}^*$. This implies that the output of B constitutes a valid collision for $\mathcal{CH}$, with the same probability, and thus concludes the proof of the claim. □

□

3.2.2 Concrete Instantiations

We now explain how to instantiate our generic transformation from the previous section using standard complexity assumptions. We need three main ingredients: (i) A public-coin chameleon hash function $\mathcal{CH} = (\mathsf{HGen}, \mathsf{Hash}, \mathsf{HCol})$ with randomness space $\mathcal{R}_{\text{hash}}$; (ii) A CPA-secure PKE scheme $\mathcal{PKE}^1 = (\mathsf{KGen}^1, \mathsf{Enc}^1, \mathsf{Dec}^1)$ with message space $\mathcal{M}^1_{\text{pke}} := \mathcal{R}_{\text{hash}}$ and randomness space $\mathcal{R}^1_{\text{pke}}$; (iii) An f-tSE NIZK for the language of Eq. (3.1), where the function $f : \mathcal{R}_{\text{hash}} \times \mathcal{R}^1_{\text{pke}} \to \mathcal{R}_{\text{hash}}$ has a type $f(r, \rho) = r$. For the latter component, we rely on the construction due to Dodis *et al.* [DHLW10] that allows to obtain an f-tSE NIZK for any efficiently computable function f and for any language L, based on a standard (non-extractable) NIZK for that language and a CCA-secure PKE scheme.

Let $\mathcal{PKE}^2 = (\mathsf{KGen}^2, \mathsf{Enc}^2, \mathsf{Dec}^2)$ be a CCA-secure PKE scheme with message space $\mathcal{M}^2_{\text{pke}} := \mathcal{R}_{\text{hash}}$. Plugging in the construction from [DHLW10] the check value ξ in our construction has the form $\xi := (c_1, c_2, \pi)$, where π is a standard NIZK argument for $((PK_1, c_1), (hk, h, m), (PK_2, c_2)) \in L_{\mathcal{CH}}$, with language $L_{\mathcal{CH}}$ being defined as follows:

$$L_{\mathcal{CH}} = \left\{ ((PK_1, c_1), (hk, h, m), (PK_2, c_2)) : \begin{array}{c} \exists (r, \rho_1, \rho_2) \text{ s.t.} \\ h = \mathsf{Hash}(hk, m; r), \\ c_1 = \mathsf{Enc}^1(PK_1, r; \rho_1), \\ c_2 = \mathsf{Enc}^2(PK_2, r; \rho_2) \end{array} \right\}, \tag{3.2}$$

and where PK_1 and PK_2 are public keys generated via KGen^1 and KGen^2 (respectively), and where hk is generated via HGen.

As for the public-coin chameleon hash function, we use the framework of Bellare and Ristov [BR14] which is based on so-called Sigma-protocols. Below, we first define the complexity assumptions on which we build, and later detail two concrete instantiations (the first one in the random oracle model and the second one in the standard model).

Hardness Assumptions

We review the main complexity assumptions on which our instantiations are based. In what follows, let $\mathbb{G}$ be a group with prime order q and with generator $\mathbb{G}$.

Discrete Logarithm assumption. Let $g \leftarrow\!\!\$\, \mathbb{G}$ and $x \leftarrow\!\!\$\, \mathbb{Z}_q$. We say that the Discrete Logarithm (DL) assumption holds in $\mathbb{G}$ if it is computationally hard to find $x \in \mathbb{Z}_q$ given $y = g^x \in \mathbb{G}$.

Decisional Diffie-Hellman assumption. Let $g_1, g_2 \leftarrow\!\!\$\, \mathbb{G}$ and $x_1, x_2, x \leftarrow\!\!\$\, \mathbb{Z}_q$. We say that the Decisional Diffie-Hellman (DDH) assumption holds in $\mathbb{G}$ if the following distributions are computationally indistinguishable: $(\mathbb{G}, g_1, g_2, g_1^{x_1}, g_2^{x_2})$ and $(\mathbb{G}, g_1, g_2, g_1^{x}, g_2^{x})$.

Symmetric External Diffie-Hellman assumption. Let $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ be groups of prime order q and let $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a non-degenerate, efficiently computable, bilinear map. The Symmetric External Diffie-Hellman (SXDH) assumption states that the DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$. Such an assumption is not satisfied in case $\mathbb{G}_1 = \mathbb{G}_2$, but it is believed to hold in case there is no efficiently computable mapping between $\mathbb{G}_1$ and $\mathbb{G}_2$ [Sco02, BBS04].

K-Linear assumption [Sha07a, HK07]. Let $K \geq 1$ be a constant, and let $g_1, \ldots, g_{K+1} \leftarrow\!\!\$\, \mathbb{G}$ and $x_1, \ldots, x_K \leftarrow\!\!\$\, \mathbb{Z}_q$. The K-linear assumption holds in $\mathbb{G}$ if the following distributions are computationally indistinguishable: $(\mathbb{G}, g_1^{x_1}, \ldots, g_K^{x_K}, g_{K+1}^{x_{K+1}})$ and $(\mathbb{G}, g_1^{x_1}, \ldots, g_K^{x_K}, g_{K+1}^{\sum_{i=1}^{K} x_i})$. Note that for $K = 1$ we obtain the DDH assumption, and for $K = 2$ we obtain the so-called Linear assumption [Sha07a].

In what follows we assume that the K-linear assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$, which is the case for symmetric pairings provided that $K \geq 2$. For $K = 1$ we slightly abuse notation, and assume that the SXDH assumption holds instead (although that requires asymmetric pairings).

Random Oracle Model Instantiation

For concreteness, we focus here on a specific construction relying on the DDH assumption and on the Sigma-protocol due to Schnorr [Sch91]; similar constructions can be obtained based on the RSA assumption, on Quadratic Residuosity, and on Factoring, using the Sigma protocols due to Guillou-Quisquater [GQ88], Fiat-Shamir [FS86], Ong-Schnorr [OS90], Okamoto [Oka92], and Fischlin and Fischlin [FF02].

Public-Coin Hash: Let $\mathbb{G}$ be a group with prime order q and with generator g, where the Discrete Logarithm problem is believed to be hard. [2] Algorithm HGen picks a random $x \leftarrow\!\!\$\, \mathbb{Z}_q$ and defines $hk := g^x = y$ and $tk = x$. In order to hash a message $m \in \mathbb{Z}_q$, algorithm $\mathsf{Hash}(hk, m; r)$ picks a random $r \leftarrow\!\!\$\, \mathbb{Z}_q$ and returns $h := g^r \cdot y^{-m}$. In order to compute a collision, algorithm $\mathsf{HCol}(tk, (h, m, r), m')$ returns $r' = r - x \cdot (m - m') \bmod q$.[3] Notice that $\mathcal{R}_{\text{hash}} = \mathcal{M}_{\text{hash}} := \mathbb{Z}_q$.

CPA PKE: We use the ElGamal PKE scheme [Gam85]. In order to encrypt messages in $\mathbb{Z}_q$ we rely on a public, onto, invertible mapping $\Omega : \mathbb{Z}_q \to \mathbb{G}$.[4] Let $\hat{g}$ be a generator in $\mathbb{G}$. The public-key is $PK_1 = \hat{g}^{\hat{x}} = \hat{y}$ for a random secret key $\hat{x} \leftarrow\!\!\$\, \mathbb{Z}_q$, and the encryption of $r \in \mathbb{Z}_q$ is equal to $c_1 := (c_1^1, c_1^2) = (\hat{g}^{\rho_1}, \hat{y}^{\rho_1} \cdot \Omega(r))$ for a random $\rho_1 \leftarrow\!\!\$\, \mathbb{Z}_q$.

CCA PKE: We use the PKE scheme due to Cramer and Shoup [CS98]. In order to encrypt messages in $\mathbb{Z}_q$ we rely on the same mapping Ω described above. Let

[2] Examples of such groups include elliptic curve groups and the subgroup $\mathbb{QR}_p$ of $\mathbb{Z}_p^*$ containing the quadratic residues modulo a prime $p = 2q + 1$.

[3] For readers familiar with [BR14], the hashing algorithm corresponds to the (strong) HVZK simulator and the collision-finding algorithm corresponds to the strong prover of the underlying Sigma-protocol.

[4] As observed in [CPP06] relying on such an encoding might be computationally expensive; we can adopt the same technique as in [CPP06] for obtaining an encoding-free solution.

g_1, g_2 be generators in $\mathbb{G}$, and let $H : \mathbb{G}^3 \to \mathbb{Z}_q$ be a standard collision-resistant hash function. The public-key is $PK_2 = (g_1^{x_1^1} g_2^{x_2^1}, g_1^{x_1^2} g_2^{x_2^2}, g_1^{x_1^3} g_2^{x_2^3}) = (y_1, y_2, y_3)$ for a random secret key $(x_1^1, x_2^1, x_1^2, x_2^2, x_1^3, x_2^3) \leftarrow\!\!\$\ \mathbb{Z}_q^6$, and the encryption of $r \in \mathbb{Z}_q$ is equal to $c_2 := (c_2^1, c_2^2, c_2^3, c_2^4) = (g_1^{\rho_2}, g_2^{\rho_2}, y_3^{\rho_2} \cdot \Omega(r), y_1^{\rho_2} \cdot y_2^{\rho_2 \cdot t})$ for a random $\rho_2 \leftarrow\!\!\$\ \mathbb{Z}_q$ and with $t = H(c_2^1, c_2^2, c_2^3)$.

NIZK: We use the Fiat-Shamir heuristic [FS86].[5] Let $G : \{0,1\}^* \to \mathbb{Z}_q$ be a hash function modeled as a random oracle. The language of Eq. (3.2) boils down to prove knowledge of $(r, \rho_1, \rho_2) \in \mathbb{Z}_q^3$ such that: (i) $r = \log_g(h \cdot y^m)$; (ii) $\rho_1 = \log_{\hat{g}} c_1^1$; (iii) $\log_{g_1} c_2^1 = \rho_2 = \log_{g_2} c_2^2$; (iv) $c_2^4 = y_1^{\rho_2} \cdot y_2^{\rho_2 \cdot t}$; (v) $c_1^2 / c_2^3 = \hat{y}^{\rho_1} \cdot y_3^{-\rho_2}$. Notice that this is indeed sufficient, as proving knowledge of ρ_1, ρ_2 implies knowledge of $\Omega(r) = c_1^2 / \hat{y}_1^{\rho} = c_2^3 / y_3^{\rho_2}$ (and, in turn, $\Omega(r)$ uniquely determines r).

The proofs in (i) and (ii) can be obtained from a Sigma-protocol for showing knowledge of a discrete logarithm [Sch91]. The proof in (iii) can be obtained from a Sigma-protocol for showing equality of two discrete logarithms [CP92]. The proof in (iv) and (v) can be obtained from a Sigma-protocol for showing knowledge of a representation [Oka92].

Hence, the NIZK π has a type $\pi := (\pi_0, \pi_1, \pi_2, \pi_3, \pi_4)$ where: (i) $\pi_0 = (\alpha, \gamma) = (g^a, \beta r + a)$ for random $a \leftarrow\!\!\$\ \mathbb{Z}_q$ and $\beta =: G(h \cdot y^m || \alpha)$; (ii) $\pi_1 = (\alpha_1, \gamma_1) = (\hat{g}^{a_1}, \beta_1 \rho_1 + a_1)$ for random $a_1 \leftarrow\!\!\$\ \mathbb{Z}_q$ and $\beta_1 =: G(c_1^1 || \alpha_1)$; (iii) $\pi_2 = (\alpha_2^1, \alpha_2^2, \gamma_2) = (g_1^{a_2}, g_2^{a_2}, \beta_2 \cdot \rho_2 + a_2)$ for random $a_2 \leftarrow\!\!\$\ \mathbb{Z}_q$ and $\beta_2 =: G(c_2^1 || c_2^2 || \alpha_2^1 || \alpha_2^2)$; (iv) $\pi_3 = (\alpha_3, \gamma_3^1, \gamma_3^2) = (y_1^{a_3^1} \cdot y_2^{a_3^2}, \beta_3 \cdot \rho_2 + a_3^1, \beta_3 \cdot \rho_2 \cdot t + a_3^2)$, for random $a_3^1, a_3^2 \leftarrow\!\!\$\ \mathbb{Z}_q$ and $\beta_3 = (c_2^4 || y_1 || y_2 || \alpha_3)$; (v) $\pi_4 = (\alpha_4, \gamma_4^1, \gamma_4^2) = (\hat{y}^{a_4^1} \cdot y_3^{a_4^2}, \beta_4 \cdot \rho_1 + a_4^1, -\beta_4 \cdot \rho_2 + a_4^2)$, for random $a_4^1, a_4^2 \leftarrow\!\!\$\ \mathbb{Z}_q$ and $\beta_4 = (c_1^2 / c_2^3 || \hat{y} || y_3 || \alpha_4)$.[6]

Putting together the above constructions we obtain the following result.

Corollary 1. *Let $\mathbb{G}$ be a group with prime order q. Under the DDH assumption in $\mathbb{G}$ there exists a secret-coin chameleon hash function satisfying enhanced collision resistance in the ROM, such that the hash value consists of a single element of $\mathbb{G}$, whereas the check value consists of* 12 *elements of $\mathbb{G}$ plus* 7 *elements of $\mathbb{Z}_q$.*

Standard Model Instantiation

We give an instantiation based on the K-Linear assumption.

Let $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ be groups of prime order q and let $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a non-degenerate, efficiently computable, bilinear map. Before describing the ingredients, we briefly recall the Groth-Sahai [GS08] proof system for showing multi-exponentiation equations. The CRS consists of vectors $\vec{u}_1, \ldots \vec{u}_K, \vec{u}$ such that

[5] It might seem that using the construction of Dodis *et al.* [DHLW10] for obtaining a tSE NIZK in the ROM is an overkill, as the Fiat-Shamir heuristic directly yields a full-fledged simulation-sound extractable NIZK. However, the Fiat-Shamir transform is only known to satisfy a weaker form of extractability [FKMV12] which is insufficient for our application. Alternatively, we could use Fischlin's transformation [Fis05], but this would probably result in longer proofs.

[6] For simplicity, we omit the description of the verification algorithm.

$\vec{u}_i = (u_0, 1, \ldots, 1, u_i, 1, \ldots, 1) \in \mathbb{G}_1^{K+1}$, where $u_0, u_1, \ldots, u_K$ are randomly chosen in $\mathbb{G}_1^*$ and $\vec{u}$ is randomly chosen in the span of $(\vec{u}_1, \ldots, \vec{u}_K)$.

Consider an equation of type $\tilde{g}_0 = \tilde{g}_1^{\varphi_1} \cdot, \cdots, \cdot \tilde{g}_N^{\varphi_N}$, where $\tilde{g}_0, \tilde{g}_1, \ldots, \tilde{g}_N \in \mathbb{G}_2$ are constants and $\varphi_1, \ldots, \varphi_N \in \mathbb{Z}_q$ are variables. To generate a proof one first commits to all the variables one by one; in particular, to commit to $\varphi_i \in \mathbb{Z}_q$, we sample $\vec{s}_i = (s_i^1, \ldots, s_i^K) \leftarrow_\$ \mathbb{Z}_q^K$ and compute $\vec{\psi}_i := \vec{u}^{\varphi} \prod_{j=1}^K \vec{u}_j^{s_i^j} \in \mathbb{G}_1^{K+1}$ (where vector multiplication is defined component-wise). Hence, we return the proof elements $\pi_j = \prod_{i=1}^N \tilde{g}_i^{s_i^j} \in \mathbb{G}_2$ for $j \in [K]$. In order to verify a proof $\pi = (\vec{\psi}_1, \ldots, \vec{\psi}_N, \pi_1, \ldots, \pi_K)$, we check that

$$\prod_{i=1}^N \hat{\mathsf{e}}(\vec{\psi}_i, \tilde{g}_i) = \hat{\mathsf{e}}(\vec{u}, \tilde{g}_0) \prod_{j=1}^K \vec{u}_j \cdot \pi_j,$$

where $\hat{\mathsf{e}} : \mathbb{G}_1^{K+1} \times \mathbb{G}_2 \to \mathbb{G}_T^{K+1}$, such that $\hat{\mathsf{e}}((a_1, \ldots, a_{K+1}), b) := (\mathsf{e}(a_1, b), \ldots, \mathsf{e}(a_{K+1}, b))$, is a bilinear map.

Public-Coin Hash: We use the same public-coin chameleon hash function described in the previous section, based on the DL assumption in $\mathbb{G}_2$ (which is implied by both the SXDH assumption and the DLIN assumption). Recall that $\mathcal{R}_{\text{hash}} = \mathcal{M}_{\text{hash}} := \mathbb{Z}_q$, whereas $\mathbb{G}_2$ is the output range of the hash function.

CPA PKE: We use the Linear ElGamal PKE scheme, introduced by Boneh, Boyen and Shacham [BBS04], which is based on the K-Linear assumption. Let Ω be as above. At key generation we sample a random generator $\hat{g}_{K+1} \in \mathbb{G}_2$, and, for all $i \in [K]$, define $\hat{g}_i := \hat{g}_{K+1}^{1/\hat{x}_i}$ for random $\hat{x}_i \in \mathbb{Z}_q$; the public key is $PK_1 = (\hat{g}_1, \ldots, \hat{g}_{K+1})$, and the secret key is $(\hat{x}_1, \ldots, \hat{x}_K)$. The encryption of $r \in \mathbb{Z}_q$ is equal to $c_1 = (c_1^1, \ldots, c_1^K, c_1^{K+1})$ such that $c_1^i := \hat{g}_i^{\rho_1^i}$ for random $\rho_1^i \leftarrow_\$ \mathbb{Z}_q$ and for all $i \in [K]$, and $c_1^{K+1} := \Omega(r) \cdot \hat{g}_{K+1}^{\sum_{i=1}^K \rho_1^i}$.

CCA PKE: We use the Linear Cramer-Shoup PKE scheme, introduced in [Sha07a], which is based on the K-Linear assumption. Let $H : \mathbb{G}_2^{K+2} \to \mathbb{Z}_q$ be a collision-resistant hash function. At key generation we sample random generators $g_1, \ldots, g_{K+1} \in \mathbb{G}_2$, and random exponents $x_i^j \in \mathbb{Z}_q$ for all $i \in [K+1]$ and for all $j \in [3]$. We then compute:

$$\begin{array}{lll}
y_1^1 := g_1^{x_1^1} g_2^{x_2^1} & y_2^1 := g_1^{x_1^2} g_2^{x_2^2} & y_3^1 := g_1^{x_1^3} g_2^{x_2^3} \\
y_1^2 := g_1^{x_1^1} g_3^{x_3^1} & y_2^2 := g_1^{x_1^2} g_3^{x_3^2} & y_3^2 := g_1^{x_1^3} g_3^{x_3^3} \\
\vdots & \vdots & \vdots \\
y_1^K := g_1^{x_1^1} g_{K+1}^{x_{K+1}^1} & y_2^K := g_1^{x_1^2} g_{K+1}^{x_{K+1}^2} & y_3^K := g_1^{x_1^3} g_{K+1}^{x_{K+1}^3},
\end{array}$$

and return public key $PK_2 := (g_1, \ldots, g_{K+1}, y_1^1, \ldots, y_K^1, y_1^2, \ldots, y_K^2, y_1^3, \ldots, y_K^3)$ with corresponding secret key $(x_1^1, \ldots, x_{K+1}^1, x_1^2, \ldots, x_{K+1}^2, x_1^3, \ldots, x_{K+1}^3) \in \mathbb{Z}_q^{3K+3}$. In order to encrypt $r \in \mathbb{Z}_q$ one samples $\rho_2^1, \ldots, \rho_2^K \leftarrow_\$ \mathbb{Z}_q$ and returns

a ciphertext:

$$\begin{aligned} c_2 &:= (c_2^1, \ldots, c_2^K, c_2^{K+1}, c_2^{K+2}, c_2^{K+3}) \\ &:= \left(g_1^{\rho_2^1}, \ldots, g_K^{\rho_2^K}, g_{K+1}^{\sum_{i=1}^K \rho_2^i}, \Omega(r) \cdot \prod_{i=1}^K (y_3^i)^{\rho_2^i}, \prod_{i=1}^K (y_1^i \cdot (y_2^i)^t)^{\rho_2^i} \right), \end{aligned}$$

with $t := H(c_2^1, \ldots, c_2^{K+2})$. Observe that for $K = 1$ we obtain exactly the Cramer-Shoup PKE scheme described in the previous section.

NIZK: We use the Groth-Sahai proof system [GS08]. In order to prove knowledge of a witness (r, ρ_1, ρ_2) for $((c_1, PK_1), (hk, h, m), (c_2, PK_2)) \in L_{\mathcal{CH}}$ we use a system of multi-exponentiation equations:

$$\begin{aligned} h \cdot y^m &= g^r \\ c_1^i &= \hat{g}^{\rho_1^i} \qquad \forall i \in [K] \\ c_2^i &= g_i^{\rho_2^i} \qquad \forall i \in [K] \\ c_2^{K+1} &= g_{K+1}^{\sum_{i=1}^K \rho_2^i} \\ c_2^{K+3} &= \prod_{i=1}^K (y_1^i \cdot (y_2^i)^t)^{\rho_2^i} \\ c_1^{K+1}/c_2^{K+2} &= \hat{g}_{K+1}^{\sum_{i=1}^K \rho_1^i} \cdot \prod_{i=1}^K (y_3^i)^{-\rho_2^i}. \end{aligned}$$

This corresponds to a system of $2K + 4$ equations with witness $(r, \rho_1^1, \ldots, \rho_1^K, \rho_2^1, \ldots, \rho_2^K)$, and hence using the Groth-Sahai proof system we obtain that the proof π consists of $2K + 1$ commitments (each containing $K + 1$ elements of $\mathbb{G}_1$) and $2K^2 + 4K$ proof elements (in $\mathbb{G}_2$).

Putting together the above constructions we obtain the following result.

Corollary 2. *Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be pairing based groups, and let $K \geq 1$. Under the K-Linear assumption there exists a secret-coin chameleon hash function satisfying enhanced collision resistance in the standard model, such that the hash value consists of a single group element, whereas the check value consists of $4K^2 + 9K + 5$ group elements. In particular, the size of the check value is 18 group elements under the SXDH assumption and 39 group elements under the DLIN assumption.*

3.3 Blockchain Application

In this Section we discuss the blockchain technology and the motivations for the creation of a redactable blockchain, our application based on an enhanced collision-resistant chameleon hash functions.

3.3.1 Overview

The cost of the bankruptcy of Lehman Brothers in 2008 to the United States is estimated in trillions [U.S13] and triggered a chain of events that sent several countries into economic recession or depression. One contributor to the crisis was the centralized payment and monetary system based on clearinghouses that act as intermediaries between buyers and sellers and take on the risk of defaults. Unfortunately, clearinghouses add a significant cost to any interbank transactions and do not always operate transparently.

Bitcoin is an innovative technology that may allow banks to settle accounts between themselves without relying on centralized entities. It is considered the first decentralized currency system that works on a global scale. It relies on cryptographic proofs of work, digital signatures, and peer-to-peer networking to provide a distributed ledger (called the blockchain) containing transactions. Digital currency is, however, the simplest application of the blockchain technology. Bitcoin includes a scripting language that can be used to build more expressive "smart contracts", basically cryptographically-locked boxes that can be opened if certain conditions are verified. In addition, transactions can store arbitrary data via the OP_RETURN mechanism.

The blockchain technology promises to revolutionize the way we conduct business. Blockchain startups have received more than $1bn [Coi] of venture capital money to exploit this technology for applications such as voting, record keeping, contracts, etc. Conventional services are centralized and do not scale well. The blockchain allows services to be completely decentralized. There is no need to rely on, or trust, a single organization. It is a disruptive technology that will change the way money, assets and securities are currently managed. Business agreements can be encoded as smart contracts which in turn can handle automatically their executions along with the arbitration of disputes, thus reducing cost and providing more transparency. From a technology point of view, the blockchain is equally revolutionary. It provides for the first time a probabilistic solution to the Byzantine generals problem, where consensus is reached over time (after confirmations), and makes use of economic incentives to secure the overall infrastructure.

Two approaches have emerged to facilitate the use of the blockchain technology to implement decentralized services and applications (what is referred to as Bitcoin 2.0). The first "overlay" approach is to rely on the existing Bitcoin blockchain and build a new framework on top of it. This is done through transactions with OP_RETURN outputs which are unspendable and do not need to be stored in the UTXO database. This is the only mechanism to add data in the blockchain approved by the Bitcoin dev community as it does not cause UTXO "bloatin" and waste of precious computing resources. The rationale of this approach is that the Bitcoin blockchain already exists and is adopted by many, which makes it inherently more secure and resilient. However, certain constraints and constants set by the creator of Bitcoin (Satoshi) impede some (but not all) applications. For instance, blocks are mined every 10 minutes on average and the Bitcoin scripting language is not Turing-complete. This works perfectly for the currency, but forces other applications to get around these limitations through cumbersome hacks. The second approach is to build an alternative blockchain with all the desired features. This approach is

gathering momentum (see, e.g, Ethereum [Eth]), and promises full decentralization. It enables very expressive smart contracts that achieve a high degree of automation.

3.3.2 Motivation

The append-only nature of the blockchain is essential to the security of the Bitcoin ecosystem. Transactions are stored in the ledger forever and are immutable. This fits perfectly with the currency system. However, we argue that an immutable ledger is not appropriate for all new applications that are being envisaged for the blockchain. Whether the blockchain is used to store data or code (smart contracts), there must be a way to redact its content in specific and exceptional circumstances. Redactions should be performed only by trusted entities, under strict constraints, and with full transparency and accountability. Some examples where a redactable blockchain is desirable are outlined below.

(i) The ability to store arbitrary messages has already been abused, and now the Bitcoin blockchain contains child pornography, improper content, and material that infringes on intellectual rights (see e.g., [Hop13, Pea15, HC13]). The intent of these abuses is to disrupt the Bitcoin system, since users may not be willing to participate and download the blockchain for fear of being prosecuted for possession of illegal or improper content on their computers. There are currently only 8-10K full nodes that store the entire blockchain and if this number declines, the Bitcoin ecosystem may be severely disrupted. In addition, improper content (gossip, pictures, etc.) may affect the life of people forever if it is not removed from the blockchain. Thus, appending new information is not an option in these cases.

(ii) Bitcoin 2.0 applications require re-writable storage. Smart contracts and overlay applications may not work or scale if the blockchain is immutable. A smart contract is essentially a sequence of instructions that a miner is going to run in exchange for a compensation. Amending a contract or patching code, by appending a new version of it, does not scale and wastes precious resources.

(iii) Is our society ready for permanent storage or perfect accountability? We believe it is not and indeed much effort is spent to promote the "right to be forgotten". New blockchain applications promise to store files, notarize documents, manage health records, coordinate IoT devices, administer assets, etc. But records should be expunged in case they contain errors or sensitive information, or when it is required by law. Even encryption may not help as keys are notoriously difficult to manage and are often leaked.

(iv) Several financial institutions are exploring the benefits of blockchain-based solutions to reduce cost and increase trust in interbank interactions. Budgets, transactions, and financial results are routinely consolidated to create meaningful reports while allowing entities to maintain distinct accounting structures. Consolidation is difficult to achieve with immutable blockchains, since it is impossible to consolidate past transactions without affecting any subsequent blocks.

3.3.3 Remarks

Proposing to affect the immutability of the blockchain may seem an ill-conceived concept given the importance of the append-only nature of the blockchain. However,

hard forks exist that can be used to undo recent transactions. As for hard forks, we expect redactions to occur in rare and exceptional circumstances.

Why not applying a hard fork in the past? Hard forks can be seen as the Undo operation and thus they make sense only for recently mined blocks. Imagine making a hard fork for a block added to the blockchain, say, 5 years ago. All subsequent blocks will be rendered invalid and all transactions from 5 years ago till now will have to be reprocessed. Thus, regenerating the blockchain will take another 5 years assuming similar mining power.

Would this redaction mechanism make sense for Bitcoin? We target Bitcoin 2.0 applications but we believe Bitcoin can also benefit from our solution. Consider this trivial but effective attack against Bitcoin. (i) Divide objectionable content (e.g., child or revenge pornography, sensitive or private information, etc.) in packets as it is done with TCP/IP. (ii) Store each packet within the OP_RETURN field of several Bitcoin transactions. (iii) After several blocks are mined, release a simple script or provide a web page where the improper content can be reconstructed as with TCP/IP packets. (iv) Wait for a lawsuit to be filed. If (when) this happens, then Bitcoin could be legally shut down and the blockchain removed for good. Notice that access to content on the Internet can be controlled, filtered out, or made it hard to find. On the other end, content in the blockchain must always be available and stored locally at each node.

Who can make redactions? We show how to make redactions given the knowledge of a secret key. This key could be in the hands of miners, a centralized auditor, or shares of the key could be distributed among several authorities. The actual way the trapdoor key is managed depends upon the requirements of specific applications; while we provide some examples (see Section 3.4.4), we stress that those are just a few possibilities out of many.

Why can't the blockchain be edited "by fiat", relying on meta-transactions? It is possible to create a block revocation list that miners are instructed to check and avoid. The problem however is that old blocks will still be there with the information that was supposed to be redacted. Thus, this approach is pointless. Another variant is to actually remove blocks, creating "holes" in the blockchain, and instruct miners to ignore those blocks. This approach is even worse since the blockchain is not valid anymore and exceptions must be hardcoded in the software of each miner or made available as an authenticated blacklist.

Couldn't the set of miners "vote" by their power, "sign" the new block and insert it into the correct position? No, because this is essentially a hard fork and all subsequent blocks will be invalid. Punching the blockchain makes it invalid and can only be handled as described in the previous point.

If trusted authorities can redact the blockchain, can't you get rid of PoW-based consensus? Redactions, as hard forks, are supposed to happen

very rarely, in case of emergencies (e.g., the DAO attack [dC16]) or when sensitive information is leaked (e.g., revenge porn). Editors do not operate daily but only in exceptional circumstances. They do not have the ability to run or maintain a blockchain. Trusted authorities could be individuals, such as judges or arbitrators, or/and organizations, such as the International Monetary Fund (IMF), the World Trade Organization (WTO), Electronic Frontier Foundation (EFF), INTERPOL, etc. They are not meant to operate blockchain infrastructures (as a simple analogy, consider that the Securities and Exchange Commission (SEC) is not meant to run stock and options exchanges or electronic securities markets but to intervene to enforce federal securities laws). Thus, we must rely on PoW-based consensus to run the blockchain.

Are these trusted authorities the same as in permissioned blockchain? Not necessarily. In permissioned blockchain only specified actors (banks, financial operators, individuals, etc.) can participate and post transactions. Some or all of these actors could be allowed to redact the blockchain if they collaborate. Shares of the chameleon hash key could be distributed among them so that the key is reconstructed when shares are pooled together according to some access control structure.

Could redactions have helped with the DAO attack? The DAO attack was resolved with a hard fork. Technically there is no substantial difference between hard forks and redactions for recent events. Our solution would help in case frauds or unintended errors are discovered much later, when it is too late to apply a hard fork and efficiently rebuild the blockchain.

Is blockchain immutability a chimera? The aftermath of the DAO attack shows that immutability is contentious (DAO is dead, lawsuits are looming, two parallel chains ETH/ETC were created, the future of Ethereum is in question, etc.). Other than affecting privacy (see [Ten]), immutability also affects scalability (see [DeR]). Our primary intent is to provide a technical answer to the question: "How can I make a redactable blockchain?". However, we do believe immutability of the blockchain should be reconsidered if Bitcoin 2.0 applications are to be turned from lab experiments to real deployments.

3.3.4 The Chameleon Blockchain

We propose an approach to make the blockchain redactable; by redaction we mean one of the following actions (and any combination of those): re-writing one or more blocks, compressing any number of blocks into a smaller number of blocks, and inserting one or more new blocks. Redactions can be made only by authorized entities and under specific constraints; moreover redactions are publicly auditable by existing miners, since they must approve the new blockchain and have access to its old copies. However, new miners are *oblivious*, given that the blockchain in our design is implemented as a history-independent data structure in the sense introduced by Naor and Teague [NT01]. That is, no information can be deduced

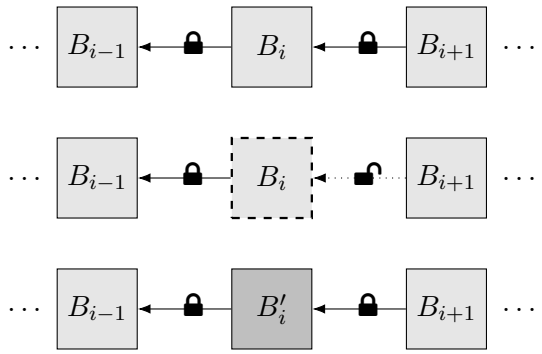

Figure 3.2. Redaction operations on a chameleon blockchain. In the top blockchain, all padlocks are locked resulting in an immutable blockchain. In the middle blockchain, the padlock from block B_{i+1} to block B_i is open, meaning that the content of block B_i can be redacted. In the bottom blockchain, the block B_i was redacted (resulting in block B'_i) and all the padlocks are once again locked, making the blockchain immutable.

about the past from the current view of the blockchain (also called *anti-persistence* in [NT01]).

All blockchain designs rely on a hash chain that connects each block to the previous one, to create an immutable sequence. The immutability comes from the collision resistance property of the hash function. The best way to grasp the concept of a redactable blockchain is to think of adding a lock to each link of the hash chain (see Figure 3.2): Without the lock key it is hard to find collisions and the chain is immutable, but given the lock key it is possible to efficiently find collisions and thus replace the content of any block in the chain. With the knowledge of the key, any redaction is then possible: deletion, modification, and insertion of any number of blocks. Note that if the lock key is lost or destroyed, then a chameleon blockchain reverts to an immutable one.

The main idea of our design is to employ a chameleon hash function that is collision-resistant unless the trapdoor is known. In a standard chameleon hash, collisions must be kept private since the trapdoor can be extracted from a single collision. In our enhanced collision resistant chameleon hash design, it is safe to reveal any number of collisions.

3.4 Redacting the Blockchain

In this section we introduce our framework, explaining how to modify current blockchain technologies in order to obtain a redactable blockchain. We start with a brief description of a blockchain abstraction, due to Garay, Kiayias and Leonardos [GKL15], in Section 3.4.1. We then put forward two new algorithms that can be used to re-write the content of the blockchain, both in the centralized setting where a trusted party is in charge of rewriting the blocks (in Section 3.4.2) and in the decentralized setting where no such trusted party is available (in Section 3.4.3). Finally, in Section 3.4.4, we comment on how the chameleon hash keys can be managed in a few concrete scenarios.

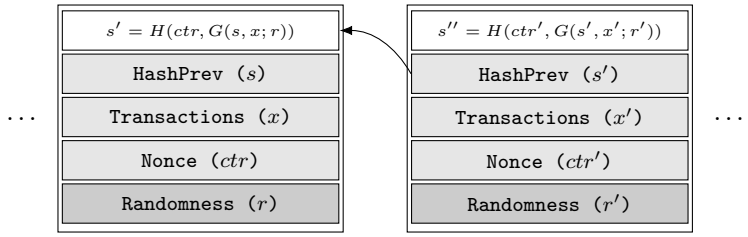

Figure 3.3. The redactable blockchain structure (using a public-coin chameleon hash). The field s of a block stores the value shown in the top white field of the previous block. We note that the top white field is not stored in the block. The bottom darker field (`Randomness`) is updated when the block is redacted (i.e., a collision is computed).

3.4.1 Blockchain Basics

We make use of the notation of [GKL15] to describe the blockchain. A block is a triple of the form $B = \langle s, x, ctr \rangle$, where $s \in \{0,1\}^\kappa$, $x \in \{0,1\}^*$ and $ctr \in \mathbb{N}$. Block B is *valid* if

$$\mathbf{validblock}_q^D(B) := (H(ctr, G(s,x)) < D) \wedge (ctr < q) = 1.$$

Here, $H : \{0,1\}^* \to \{0,1\}^\kappa$ and $G : \{0,1\}^* \to \{0,1\}^\kappa$ are collision-resistant hash functions, and the parameters $D \in \mathbb{N}$ and $q \in \mathbb{N}$ are the block's difficulty level and the maximum number of hash queries that a user is allowed to make in any given round of the protocol, respectively.

The *blockchain* is simply a chain (or sequence) of blocks, that we call $\mathcal{C}$. The rightmost block is called the head of the chain, denoted by $\mathsf{Head}(\mathcal{C})$. Any chain $\mathcal{C}$ with a head $\mathsf{Head}(\mathcal{C}) := \langle s, x, ctr \rangle$ can be extended to a new longer chain $\mathcal{C}' := \mathcal{C} || B'$ by attaching a (valid) block $B' := \langle s', x', ctr' \rangle$ such that $s' = H(ctr, G(s,x))$; the head of the new chain $\mathcal{C}'$ is $\mathsf{Head}(\mathcal{C}') = B'$. A chain $\mathcal{C}$ can also be empty, and in such a case we let $\mathcal{C} = \varepsilon$. The function $\mathsf{len}(\mathcal{C})$ denotes the length of a chain $\mathcal{C}$ (i.e., its number of blocks). For a chain $\mathcal{C}$ of length n and any $k \geq 0$, we denote by $\mathcal{C}^{\lceil k}$ the chain resulting from removing the k rightmost blocks of $\mathcal{C}$, and analogously we denote by ${}^{k\rceil}\mathcal{C}$ the chain resulting in removing the k *leftmost* blocks of $\mathcal{C}$; note that if $k \geq n$ then $\mathcal{C}^{\lceil k} = \varepsilon$ and ${}^{k\rceil}\mathcal{C} = \varepsilon$. If $\mathcal{C}$ is a prefix of $\mathcal{C}'$ we write $\mathcal{C} \prec \mathcal{C}'$. We also note that the difficulty level D can be different among blocks in a chain.

The work of [GKL15] models the Bitcoin protocol in a setting where the number of participants is always fixed and the network in synchronized. They show that the protocol satisfies *consistency* in this model, meaning that all honest participants have the same chain prefix of the blockchain. A more recent work by Pass, Seeman and shelat [PSas16] analyses the case where the network is asynchronous and the number of participants can dynamically change. We point that our framework is independent of the network type in these models.

3.4.2 Centralized Setting

The main idea behind our approach is to set the inner hash function (i.e., the function G), used to chain the different blocks in the blockchain, to be a chameleon hash function. Intuitively, re-writing the content of each block is possible by finding

collisions in the hash function (without modifying the outer hash function H). Below, we detail this idea in the simple setting where only a single (trusted) central authority is able to redact the blockchain; see Section 3.4.4 for concrete examples where this case applies.

In order for the above to work, we require some modifications to the previously defined block. A block is now a tuple $B := \langle s, x, ctr, (h, \xi)\rangle$, where the components s, x and ctr are the same as before, and the new component (h, ξ) is the hash/check pair for a chameleon hash. The function G is defined to be a secret-coin chameleon hash $\mathcal{CH} = (\mathsf{HGen}, \mathsf{Hash}, \mathsf{HVer}, \mathsf{HCol})$, and the validation predicate for a block is now equal to

$$\mathbf{validblock}_q^D(B) := (H(ctr, h) < D) \wedge (\mathsf{HVer}(hk, (s, x), (h, \xi))) \wedge (ctr < q) = 1.$$

Given a chain $\mathcal{C}$ with head $\mathsf{Head}(\mathcal{C}) := \langle s, x, ctr, (h, \xi)\rangle$, we can extend it to a longer chain by attaching a (valid) block $B' := \langle s', x', ctr', (h', \xi')\rangle$ such that $s' = H(ctr, h)$. Notice that the domain of the chameleon hash can be easily adjusted to the proper size by first hashing the input of Hash with a regular collision-resistant hash of the desired output size. We also stress that the verification of a chameleon hash value needs to be computed by its own verification function (i.e., by running HVer), and not simply by recomputing the hash like it is done with standard (deterministic) hash functions.

The case where the chameleon hash is public-coin can be cast as a special case of the above. However, note that there is no need for storing the hash value h, as this value can be computed as a deterministic function of the chameleon hash function's input and randomness. Thus, in this case, a block has a type $B := \langle s, x, ctr, r\rangle$, where r is the randomness for the chameleon hash. The validation predicate for a block becomes

$$\mathbf{validblock}_q^D(B) := (H(ctr, \mathsf{Hash}(hk, (s, x); r)) < D) \wedge (ctr < q) = 1.$$

Finally, given a chain $\mathcal{C}$ with head $\mathsf{Head}(\mathcal{C}) := \langle s, x, ctr, r\rangle$, we can extend it to a longer chain by attaching a (valid) block $B' := \langle s', x', ctr', r'\rangle$ such that $s' = H(ctr, \mathsf{Hash}(hk, (s, x); r))$. See Fig. 3.3 for a pictorial representation.

Rewriting blocks. Next, we define a chain redacting algorithm (see Algorithm 1) that takes as input a chain $\mathcal{C}$ to be redacted, a set of indices that represents the positions (in the chain $\mathcal{C}$) of the blocks that are going to be redacted, and another set with the new x''s values for each of the blocks to be redacted. The algorithm also takes as input the chameleon hash trapdoor key tk. The intuition behind it is that, for each block to be redacted, we compute a collision for the hash of the block with its new content x'. A new chain $\mathcal{C}'$ is created by replacing the original block with its modified counterpart. We note that at the end of the execution of Algorithm 1, the central authority should broadcast the new redacted chain as a *special* chain, meaning that every user of the system should adopt this new redacted chain in favor of any other chain, even longer ones. The way this is achieved depends on the actual system in use.

Note that each time a block is redacted using Algorithm 1, a collision for the underlying chameleon hash function is exposed. Hence, it is important that

Algorithm 1: Chain Redact

input : The input chain $\mathcal{C}$ of length n, a set of block indices $\mathcal{I} \subseteq [n]$, a set of values $\{x'_i\}_{i\in\mathcal{I}}$, and the chameleon hash trapdoor key tk.
output : The redacted chain $\mathcal{C}'$ of length n.

$\mathcal{C}' \leftarrow \mathcal{C}$;
Parse the chain $\mathcal{C}'$ as $(B_1, \cdots, B_n)$;
for $i := 1, \ldots, n$ **do**
  if $i \in \mathcal{I}$ **then**
    Parse the i-th block of $\mathcal{C}'$ as $B_i := \langle s_i, x_i, ctr_i, (h_i, \xi_i)\rangle$;
    $\xi'_i \leftarrow \mathsf{HCol}(tk, (h_i, s_i||x_i, \xi_i), (s_i||x'_i))$;
    $B'_i := \langle s_i, x'_i, ctr_i, (h_i, \xi'_i)\rangle$;
    $\mathcal{C}' \leftarrow \mathcal{C}'^{\lceil n-i+1}||B'_i||^{i\rceil}\mathcal{C}'$;
return $\mathcal{C}'$

the ability to see arbitrary collisions does not expose the secret trapdoor key, as otherwise unauthorized users might be able to rewrite arbitrary blocks in the chain. In Section 3.2 we explain how to generically leverage any standard collision-resistant chameleon hash function into one additionally meeting such a key-exposure freeness requirement.

Compressing the chain. Another possibility with chameleon blockchains is to completely *remove* entire blocks from a chain. This can be essential for scalability purposes, such as saving disk space and computational power necessary when handling larger chains. We present an algorithm (see Algorithm 2) for such a "chain compressing" functionality. The intuition behind it is that in order to remove the block B_i it is necessary to redact the block B_{i+1} by assigning $s_{i+1} \leftarrow s_i$. A collision then needs to be computed for B_{i+1} producing the new block B'_{i+1} that is inserted in the chain in place of the B_{i+1} block, leaving the chain in a consistent state. As in Algorithm 1, we also note that at the end of the execution of Algorithm 2, the central authority should broadcast the new compressed chain as a *special* chain, meaning that every user of the system should adopt this new redacted chain in favor of any other chain, even longer ones.

We note that in Algorithm 2, if the set $\mathcal{I}$ contains only indexes to successive blocks, the execution can be optimized to essentially one execution of the *for* loop. This is because in order to remove blocks B_k to B_{k+j}, it is sufficient to redact only block B_{k+j+1} (i.e., the next remaining block).

3.4.3 Decentralized Setting

Below, we explain how to adapt our framework to the decentralized setting, where there is no central trusted authority. The main idea is to have the trapdoor key be secretly shared among some fixed set of users that are in charge of redacting the blockchain. When a block needs to be redacted, the users from this set engage in a secure multiparty computation (MPC) protocol to compute Algorithm 1 and

Algorithm 2: Chain Compression

input : The input chain $\mathcal{C}$ of length n, a set of block indices $\mathcal{I} \subseteq [n]$ and the chameleon hash trapdoor key tk.

output : The new chain $\mathcal{C}'$ of length $n - |\mathcal{I}|$.

$\mathcal{C}' \leftarrow \mathcal{C}$;
Parse the chain $\mathcal{C}'$ as $(B_1, \cdots, B_n)$;
for $i := 1, \ldots, n$ **do**
  if $i \in \mathcal{I}$ **then**
    Parse the i-th block of $\mathcal{C}'$ as $B_i := \langle s_i, x_i, ctr_i, (h_i, \xi_i)\rangle$;
    Parse the $i+1$-th block of $\mathcal{C}'$ as $B_{i+1} := \langle s_{i+1}, x_{i+1}, ctr_{i+1}, (h_{i+1}, \xi_{i+1})\rangle$;
    $\xi'_{i+1} \leftarrow \mathsf{HCol}(tk, (h_{i+1}, s_{i+1}||x_{i+1}, \xi_{i+1}), (s_i||x_{i+1}))$;
    $B'_{i+1} := \langle s_i, x_{i+1}, ctr_{i+1}, (h_{i+1}, \xi'_{i+1})\rangle$;
    $\mathcal{C}' \leftarrow \mathcal{C}'^{\lceil n-i}||B'_{i+1}||^{i+1]}\mathcal{C}'$;
return $\mathcal{C}'$

Algorithm 2 in a fully distributed manner.

Ideal Functionalities

During the set up of the system, we fix a subset $\mathcal{U}$ of cardinality n, containing the users that will be in charge of redacting the blockchain content. We remark that the actual choice of the subset $\mathcal{U}$ can be completely dependent on the application and on the system requirements; we discuss some examples in Section 3.4.4.

Following the common practice in the setting of MPC, we now define two ideal functionalities that aim at capturing the security requirements for generating the hash keys and for redacting the blockchain in the decentralized setting. These functionalities will later be realized by concrete MPC protocols, in both cases of semi-honest and fully malicious corruptions.

Key generation. When the system is set-up for the first time, we need to run the key generation algorithm HGen for the underlying chameleon hash function, obtaining a public hash key hk and a secret trapdoor key tk. Since no user is allowed to know the trapdoor key, the idea is to have each player P_i in the set $\mathcal{U}$ obtain a share τ_i of tk. This is the purpose of the ideal functionality described in Fig. 3.4, which is parametrized by a secret sharing scheme $(\mathsf{Share}, \mathsf{Rec})$.

Recall that a t-out-of-n secret sharing scheme $(\mathsf{Share}, \mathsf{Rec})$ consists of a pair of algorithms such that: (i) The randomized algorithm Share takes as input a target value x and returns a sequence of n shares $\tau_1, \ldots, \tau_n$; (ii) The deterministic algorithm Rec takes as input n shares $\tau_1, \ldots, \tau_n$ and returns a value x or an incorrect output symbol $\perp$. The main security guarantee is that any subset of t shares (a.k.a. an unqualified set) reveals no information on the shared value x; on the other hand, any subset of $t+1$ (or more) shares allows to efficiently recover x. We refer the reader, e.g., to [Bei11] for more details on secret sharing; some examples are also

Key Generation Functionality:

1. After receiving the "start" signal from all honest parties, run $(hk, tk) \leftarrow\!\!\$\ \mathsf{HGen}(1^\kappa)$ and send hk to the adversary.
2. We assume a secret sharing scheme $(\mathsf{Share}, \mathsf{Rec})$ is given, with which the trapdoor key tk can be secret-shared. For each dishonest party P_j, receive a share τ_j from the adversary.
3. Construct a complete set of shares $(\tau_1, \cdots, \tau_n)$ for the trapdoor key tk taking into consideration all the dishonest shares sent by the adversary. We note that it is always possible to construct such a set of shares since all the dishonest parties form an unqualified set for the secret sharing scheme. Send τ_i to each honest party P_i.

Figure 3.4. The ideal functionality for the distributed key generation

discussed below.

Chain redaction. When a block $B := \langle s, x, ctr, (h, \xi)\rangle$ needs to be redacted into a modified block $B' := \langle s, x', ctr, (h, \xi')\rangle$, each user in the set $\mathcal{U}$ needs to inspect its own blockchain and find block B. Hence, the players need to execute Algorithm 1 in a distributed manner. In particular, each player P_i is given as input its own share τ_i of the chameleon hash trapdoor key, and they all need to run the collision-finding algorithm HCol on common input $((h, s||x, \xi), s||x')$, in order to obtain the modified check value ξ'.

This is the purpose of the ideal functionality described in Fig. 3.5, which is again parametrized by a secret sharing scheme $(\mathsf{Share}, \mathsf{Rec})$ (in fact, the same secret sharing scheme as for the functionality of Fig. 3.4). For simplicity, we described the functionality for the general case where the goal is to find collisions between arbitrary messages m and m'. Note that actively corrupted players might submit incorrect shares, and the secret sharing scheme needs to cope with such a possibility. Also note that after each player receives the modified value ξ' for the new block B', each of the users in $\mathcal{U}$ constructs a new chain by replacing block B with block B'. Thus, the redacted chain is broadcast to all users in the system as a new *special* chain that should replace any other chain, even longer ones. Although the latter needs to be done in an application-specific manner, we recall that in practice the redact operation is not going to be performed very often, but only in case there is a need to redact undesirable content from a given block.

The decentralized version of Algorithm 2 is similar to the one described above, the only difference being that instead of redacting a block B_i, a new chain is built without the block B_i in it. To keep the chain valid the block B_{i+1} needs to be redacted, as detailed in Algorithm 2. The latter can be achieved using the same ideal functionality as in Fig. 3.5, by simply adjusting the input messages from the users.

Collision-Finding Functionality:

1. Receive the shares τ_i from each party P_i and reconstruct the trapdoor key $tk := \mathsf{Rec}(\tau_1, \cdots, \tau_n)$. Note that the shares of the dishonest parties are chosen by the adversary.
2. Upon receiving a "compute collision" signal for the pair $((h, m, \xi), m')$ from all honest parties, compute $\xi' \leftarrow \mathsf{HCol}(tk, (h, m, \xi), m')$ and send (h, m, ξ) and ξ' to the adversary.
3. Upon receiving an "OK" signal from the adversary forward the value ξ' to all honest parties, otherwise forward $\perp$ to all honest parties.

Figure 3.5. The ideal functionality for the distributed collision-finding algorithm

Concrete Instantiations

We now present concrete protocols for securely realizing the ideal functionalities described in the previous section. For the sake of concreteness and practicality, we chose to work with the (public-coin) chameleon hash function introduced by Ateniese and de Medeiros [AdM04]; this construction satisfies enhanced collision resistance (cf. Definition 17) in the generic group model, based on the Discrete Logarithm assumption. After presenting the hash function, we deal separately with the setting in which the corrupted players within the set $\mathcal{U}$ are assumed to be semi-honest (i.e., they always follow the protocol but try to learn additional information from the transcript) and fully malicious (i.e., they can arbitrarily deviate from the protocol description).

The hash function. Let p, q be prime such that $p = 2q + 1$, and let g be a generator for the subgroup of quadratic residues $\mathbb{QR}_p$ of $\mathbb{Z}_p^*$. Consider the following public-coin chameleon hash function $(\mathsf{HGen}, \mathsf{Hash}, \mathsf{HCol})$.

- $(y, x) \leftarrow\!\!\$\ \mathsf{HGen}(1^\kappa)$: The trapdoor key tk is a random value $x \in [1, q-1]$, and the hash key hk is equal to $y = g^x$.
- $h := \mathsf{Hash}(y, m; r, s)$: To hash a message $m \in \{0,1\}^*$, pick random $r, s \leftarrow\!\!\$\ \mathbb{Z}_q$, and return $h := r - (y^{H(m||r)} \cdot g^s \bmod p) \bmod q$ where $H : \{0,1\}^* \to \mathbb{Z}_q$ is a standard-collision resistant hash function.
- $(r', s') \leftarrow\!\!\$\ \mathsf{HCol}(x, (h, m, r, s), m')$: To compute a collision for message m', pick a random $k \in [1, q-1]$ and compute $r' := h + (g^k \bmod p) \bmod q$ and $s' := k - H(m'||r') \cdot x \bmod q$. Return (r', s').

Semi-honest setting. As a warm up we consider the case of passive corruption, where up to t players in the set $\mathcal{U}$ are semi-honest. For this setting, we will rely on the following simple secret sharing scheme $(\mathsf{Share}, \mathsf{Rec})$: (i) Upon input a value $x \in \mathbb{Z}_q$, algorithm Share samples random $\tau_1, \ldots \tau_{n-1} \leftarrow\!\!\$\ \mathbb{Z}_q$, sets $\tau_n := x - \sum_{i=1}^{n-1} \tau_i \bmod q$, and returns $(\tau_1, \ldots, \tau_n)$; (ii) Algorithm Rec takes as input $\tau_1, \ldots, \tau_n \in \mathbb{Z}_q$ and returns

$x = \sum_{i=1}^{n} \tau_i \bmod q$. The above is easily seen to be an $(n-1)$-out-of-n secret sharing scheme.

Next, we describe two simple MPC protocols Π_{sh}^1 and Π_{sh}^2 for securely realizing the functionality of Fig. 3.4 and Fig. 3.5 (respectively).

- Consider the following n-party protocol Π_{sh}^1. Each player P_i picks a random $\tau_i \in \mathbb{Z}_q$ and then all players engage into a semi-honest MPC protocol for computing $y = \prod_{i=1}^{n} g^{\tau_i} \bmod p$; each player outputs (y, τ_i). This protocol is easily seen to realize the functionality of Fig. 3.4 under semi-honest corruption of up to $n-1$ players. Indeed, as long as one of the players is honest, the value y (with corresponding trapdoor $x := \sum_{i=1}^{n} \tau_i$) will be uniformly distributed, as required.

- Consider the following n-party protocol Π_{sh}^2, on common input $((h, m, r, s), m')$. First, each player P_i chooses a random $k_i \leftarrow\!\!\$\ \mathbb{Z}_q$ and then all players engage into a semi-honest MPC protocol for computing $r' := h + \left(\prod_{i=1}^{n} g^{k_i} \bmod p\right) \bmod q$. Second, the players engage into a semi-honest MPC protocol for computing $\sum_{i=1}^{n} k_i - H(m'||r') \cdot \sum_{i=1}^{n} \tau_i \bmod q$, where the private input of P_i is defined to be (k_i, τ_i). Finally, each player outputs (r', s').

  The above protocol can be easily seen to securely realize the functionality of Fig. 3.5 under semi-honest corruptions. The number of tolerated corruptions depends on the semi-honest MPC protocols for performing the computations described above. Suitable protocols, for the setting where at least half of the players are honest, are described, e.g., in [BGW88, AL11].

Malicious setting. We briefly explain how to extend the previous protocol to the setting of active corruptions. The main difficulty here is that malicious players can now use incorrect shares. In order to ensure that the correct trapdoor is reconstructed, we rely on so-called *robust* secret sharing. Informally, a secret sharing scheme $(\mathsf{Share}, \mathsf{Rec})$ is δ-robust if an adversary adaptively modifying at most t shares computed via Share can cause the output of Rec to be wrong with probability at most δ. See, e.g., [RB07, BPRW15] for a formal definition.

Before adapting the protocols, we recall the standard secret sharing scheme due to Shamir [Sha79]: (i) Upon input a value $x \in \mathbb{Z}_q$, algorithm Share picks random coefficients $\alpha_1, \ldots, \alpha_{t-1} \in \mathbb{Z}_q$ and defines $\tau_i := x + \alpha_1 \cdot i + \cdots + \alpha_{t-1} \cdot i^{t-1} \bmod q$ for all $i \in [n]$; (ii) Upon input t shares $(\tau_1, \ldots, \tau_t)$, algorithm Rec interpolates the polynomial $\alpha(X) = \alpha_0 + \alpha_1 \cdot X + \cdots + \alpha_{t-1} \cdot X^{t-1}$ and returns $\alpha(0)$. Shamir's secret sharing can be made robust against corruption of at most $t < n/3$ shares when used in tandem with Reed-Solomon decoding during the reconstruction procedure. Alternatively, for tolerating a higher threshold $t < n/2$ (which is also the maximal threshold for such schemes [Man11]) one could use Shamir's secret sharing in conjunction with information-theoretic message authentication codes, as proposed by Rabin and Ben-Or [RB89]. This results in shares of sub-optimal size $\mu + \tilde{O}(n \cdot \kappa)$, where μ is the bit-size of the message, and κ is the security parameter ensuring $\delta = 2^{-\kappa}$; robust schemes with almost optimal share size have recently been designed in [BPRW15].

Below is a sketch of how the protocols Π^1_{sh} and Π^2_{sh} described above can be adapted to the malicious setting, where for simplicity we use Shamir's secret sharing in tandem with Reed-Solomon decoding.

- Consider the following n-party protocol Π^1_{mal}. Each player P_i samples uniformly at random $x_i := (\alpha^i_0, \alpha^i_1, \ldots, \alpha^i_{t-1}) \in \mathbb{Z}^t_q$. Hence, the players engage in an MPC protocol for computing the function $(x_1, \ldots, x_n) \mapsto ((y, \alpha(1)), \ldots, (y, \alpha(n)))$, where $y = g^{\alpha(0)}$ and

$$\alpha(X) := \sum_{i=1}^{n} \alpha^i_0 + \sum_{i=1}^{n} \alpha^i_1 \cdot X + \cdots + \sum_{i=1}^{n} \alpha^i_{t-1} \cdot X^{t-1}.$$

- Consider the following n-party protocol Π^2_{mal}. The protocol proceeds similarly to Π^2_{sh} with the following differences. In the first step the random value k is shared among the players using Shamir's secret sharing (as done in Π^1_{mal}); denote by $\beta(X)$ the corresponding polynomial, and by $\beta(i)$ the share obtained by player P_i. In the second step the players engage in an MPC protocol for computing the value $r' = h + (g^{\hat{\beta}(0)} \bmod p) \bmod q$, where the polynomial $\hat{\beta}(X)$ is reconstructed by using the (possibly corrupted) shares $\beta(i)$ from the players, via the Berlekamp-Welch [WB86] algorithm. In the third step the players engage in another MPC protocol for computing $s' = \hat{\beta}(0) - H(m'||r'') \cdot \hat{\alpha}(0) \bmod q$, where the private input of each player P_i is $(\alpha(i), \beta(i))$ and $\hat{\alpha}(X)$, $\hat{\beta}(X)$ are again reconstructed by using the (possibly corrupted) shares $\alpha(i)$, $\beta(i)$ from the players, via the Berlekamp-Welch algorithm.

Note that the above protocols rely on auxiliary MPC protocols with malicious security, for computing arithmetic functions in $\mathbb{Z}_q$. Suitable MPC protocols for the above tasks, for the setting where at least two thirds of the players are honest, are described, e.g., in [BGW88, AL11, DFK+06].

3.4.4 On Key Management

Although we view the technical tools that make redactions possible as the main feature of this application, a natural question that may arise is how the trapdoor key for the chameleon hash function is managed. We stress that the answer to this question is completely application dependent, but we still provide some examples.

Below we briefly describe three types of blockchains that occur in real-world applications [But15], and clarify how the trapdoor key could be managed in each case.

- **Private (or permissioned) blockchain**: In this type of blockchain, which is widely used by the financial sector [PB16], the write permissions are only given to a central authority, and the read permissions may be public or restricted. In this case the key management becomes simple; the trapdoor key could be given to the central authority that has the power to compute collisions and therefore redact blocks. This scenario is described in Section 3.4.2.

- **Consortium blockchain**: In this type of blockchain the consensus is controlled by a predetermined set of parties (i.e., a consortium). In this case

the trapdoor key can be shared among all the parties of the consortium, and redactions can be realized using MPC, as described in Section 3.4.3.

- **Public (or permissionless) blockchain**: This type of blockchain is completely decentralized, and any party is allowed to send transactions to the network and have them included in the blockchain (as long as the transactions are valid). The consensus process is decentralized and not controlled by any party. The best example of a public blockchain is Bitcoin. In this case we have two options to manage the trapdoor key (both using MPC, as described in Section 3.4.3).

  1. The trapdoor key can be distributed among all the parties (full miners) of the network. The drawback of this solution is that, if the number of parties in the network is too big (e.g., > 200), it might not be very efficient due to performance issues of the MPC protocol.
  2. The trapdoor key can be distributed among a carefully chosen subset of the parties. For example, in Bitcoin it is well known that the majority of the network hashing power is actually controlled by a small number of parties (e.g., the top 7 mining pools control almost 70% of the network total hashing power [Inf]).

     Although we acknowledge that the concentration of hashing power to a small number of parties can be unhealthy to the system, this solution does not change the existing Bitcoin trust assumption (i.e., Bitcoin already assumes trusted majority).

CHAPTER 4

The Bad

In this chapter we cover the use of backdoors with the intention of disrupting the security of cryptographic primitives. For that purpose, we cover subversion attacks against signature schemes. Starting in Section 4.2 we introduce the definitions of subversion attacks and undetectability. In Section 4.3 we present two subversion attacks against signature schemes; the first is an adaptation of the attack from Bellare *et al.* [BPR14] to signatures that is successful against signature schemes with at least 7-bits or randomness. The second is a novel attack against *coin-extractable* schemes, with even a single bit of randomness.

4.1 Background

Balancing national security interests with the rights to privacy of lawful citizen is always a daunting task. It has been particularly so in the last couple of years after the revelations of Edward Snowden [PLS13,BBG13,Gre14] that have evidenced a massive collection of metadata and other information perpetrated by several intelligence agencies. It is now clear that intelligence operators were not just interested in collecting and mining information but they also actively deployed malware, exploited zero-day vulnerabilities, and carried out active attacks against standard protocols. In addition, it appears some cryptographic protocol specifications were modified to embed backdoors.

Snowden also disclosed the subversion of the NIST standard for pseudorandom number generator (PRG), called Dual_EC_DRBG [Per13]. It was revealed that it contained a backdoor embedded in its public parameters that made it possible to predict the outputs of the PRG with the knowledge of the backdoor.

Whether this activity was effective or even allowed by the constitution is open to debate and it is indeed being furiously discussed among policy makers, the public, and the intelligence community. Ultimately, a balance between security and privacy must be found for a free and functioning society.

The ability of substituting a cryptographic algorithm with an altered version was first considered formally by Young and Yung (extending previous works of Simmons on subliminal channels [Sim83, Sim84]), who termed this field *kleptography* [YY96, YY97]. The idea is that the attacker surreptitiously modifies a cryptographic scheme with the intent of subverting its security. This research area has recently been revitalized by Bellare *et al.* [BPR14] who considered encryption algorithms with the possibility of mass surveillance under the algorithm-substitution attack. They analyzed the possibility of an intelligence agency substituting an encryption algorithm with the code of an alternative version that undetectably reveals the secret key or the plaintext. What they uncovered is that any randomized and stateless encryption scheme would fall to generic algorithm-substitution attacks. The only way to achieve a meaningful security guarantee (CPA-security) is to use a nonce-based encryption that must keep state. Unfortunately, only stateless schemes are deployable effectively with the current network technology and indeed all deployed encryption algorithms are in this class.

In this Chapter we analyze digital signature schemes under the so-called *subversion attacks* (SAs), that in particular include algorithm-substitution and kleptographic attacks as a special case, but additionally cover more general malware and virus attacks. We stress that our intention is not to propose schemes that can be abused by criminals to avoid monitoring. We are motivated by pure scientific curiosity and aspire to contribute to an active field of research.

4.2 Subverting Signatures

We introduce a new and generic framework and definitions for subversions of digital signatures. In the standard black-box setting, a signature scheme should remain unforgeable even against an adversary able to obtain signatures on (polynomially many) chosen messages. Our security definitions empower the adversary with the ability of *continuously* subverting the signing algorithm within a class $\mathcal{A}$ of allowed SAs. For each chosen subversion in the class, the adversary can access an oracle that answers (polynomially many) signature queries using the subverted signature algorithm. Importantly, the different subversions can be chosen in a fully-adaptive manner possibly depending on the target verification key of the user.

We believe our model is very general and flexible, as it nicely generalizes previous models and definitions. First off, when the class $\mathcal{A}$ consists of a set of algorithms containing a secretly embedded backdoor, and in case the adversary is restricted to non-adaptively choose only a single subversion algorithm from this class, we obtain the setting of algorithm-substitution and kleptographic attacks as a special case. However, we note that the above definition is far more general as it covers (fully-adaptive and continuous) *tampering with the computation* performed by the signing algorithm (within the class $\mathcal{A}$). This models, for instance, a machine running a signature software infected by a malware (e.g., via a buffer overflow attack [One96, Fry00, PB04]); we also obtain memory and randomness tampering (see Section 1.3) as a special case.

Clearly, without making any restriction on the class $\mathcal{A}$ (or without making additional assumptions) there is no hope for security: An arbitrary subverted

signature algorithm could, for instance, just ignore all inputs and output the secret key.

We proceed to define what it means for an adversary B to subvert a signature scheme $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$. We model subversion as the ability of the adversary to replace the genuine signing algorithm with a different algorithm within a certain class $\mathcal{A}$ of Subversion Attacks (SAs). A subversion of $\mathcal{SS}$ is an algorithm $\widetilde{\mathsf{A}} \in \mathcal{A}$, specified as follows.

- Algorithm $\widetilde{\mathsf{A}}(\cdot, \cdot; \cdot)$ takes as input a signing key $SK \in \mathcal{SK}$, a message $m \in \mathcal{M}$, random coins $r \in \mathcal{R}$, and outputs a subverted signature $\widetilde{\sigma} \in \Sigma$, where $\widetilde{\sigma} := \widetilde{\mathsf{A}}(SK, m; r)$. Notice that algorithm $\widetilde{\mathsf{A}}$ is completely arbitrary, with the only restriction that it maintains the same input-output interfaces as the original signing algorithm.

In particular, algorithm $\widetilde{\mathsf{A}}$ can hard-wire arbitrary auxiliary information chosen by the adversary, which we denote by a string $\alpha \in \{0,1\}^*$. In general we also allow algorithm $\widetilde{\mathsf{A}}$ to be *stateful*, even in case the original signing algorithm is not, and we denote the corresponding state by $\tau \in \{0,1\}^*$; the state is only used internally by the subverted algorithm and never outputted to the outside.

In the next section we define what it means for a class of SAs to be *undetectable* by a user. Some of our definitions are similar in spirit to the ones put forward in [BPR14], except that our modelling of subversion is more general (see below for a more detailed comparison).

Public/Secret Undetectability

By undetectability, we mean the inability of ordinary users to tell whether signatures are computed using the subverted or the genuine signing algorithm. We will distinguish between the case where a subversion is *publicly* or *secretly* undetectable. Roughly speaking, public undetectability means that no user can detect subversions using the verification key VK only (i.e., without knowing the signing key SK); secret undetectability means that no user, even with knowledge of the signing key SK, can detect subversions.

A formal definition follows. While reading it, bear in mind that the challenger plays the role of the "bad guy" trying to sabotage the signature scheme without being detected.

Definition 18 (Public/Secret Undetectability)**.** *Let $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme, and $\mathcal{A}$ be some class of SAs for $\mathcal{SS}$. We say that $\mathcal{A}$ is* secretly (t, q, ε)*-undetectable w.r.t. $\mathcal{SS}$ if for all PPT users U running in time t, there exists an efficient challenger such that $\left|\mathbb{P}\left[\mathsf{U} \text{ wins}\right] - \frac{1}{2}\right| \leq \varepsilon(\kappa)$ in the following game:*

1. *The challenger runs $(VK, SK) \leftarrow \mathsf{KGen}(1^\kappa)$, chooses an algorithm $\widetilde{\mathsf{A}} \in \mathcal{A}$ (possibly depending on VK), samples $b \leftarrow\!\!\$\ \{0,1\}$ and gives (VK, SK) to U.*

2. *The user U can ask queries $m_i \in \mathcal{M}$, for all $i \in [q]$. The answer to each query depends on the secret bit b. In particular, if $b = 1$, the challenger returns $\sigma_i \leftarrow \mathsf{Sign}(SK, m_i)$; if $b = 0$, the challenger returns $\widetilde{\sigma}_i \leftarrow \widetilde{\mathsf{A}}(SK, m_i)$.*

3. Finally, U *outputs a value* $b' \in \{0,1\}$*; we say that* U *wins iff* $b' = b$*.*

We say that $\mathcal{A}$ *is* publicly *undetectable w.r.t.* $\mathcal{SS}$ *if in step 1. of the above game,* U *is only given the verification key. Moreover, whenever* $\varepsilon(\kappa) = \nu(\kappa)$ *and* $t, q = poly(\kappa)$ *we simply say that* $\mathcal{A}$ *is secretly/publicly undetectable w.r.t.* $\mathcal{SS}$*.*

Our definition of undetectability is similar to the corresponding definition considered by Bellare *et al.* [BPR14] for the case of symmetric encryption. One key difference is that, in the definition above, the challenger is allowed to choose the subversion algorithm possibly depending on the verification key of the user.[1]

We note that our definition does not capture other means of detection, such as measuring the running time of algorithms, or other types of side channel detection.

Note that it could be potentially easier for user U to detect $\widetilde{\mathsf{A}}$ in practice. User U could, for example, promptly detect an anomaly if the running time of $\widetilde{\mathsf{A}}$ is significantly different from the running time of the signing algorithm Sign.

While one could in principle define even stronger forms of undetectability, e.g. by requiring that continuous and fully-adaptive SAs remain undetectable, we do not pursue this direction here. The reason for this is that the attacks we analyze in Section 4.3 are non-adaptive, with similar running time, and only require to use a single subversion.

Secret vs. public undetectability. While secret undetectability clearly implies public undetectability, the converse is not true. In particular, in Section 4.4.2 we show that there exists a signature scheme $\mathcal{SS}$ and a set of subversions $\mathcal{A}$ of it such that $\mathcal{A}$ is publicly undetectable w.r.t. $\mathcal{SS}$ but it is secretly detectable w.r.t. $\mathcal{SS}$. Moreover, in Section 4.3.1 we show that it is possible to subvert (in a strong sense) any sufficiently randomized signature scheme in a way that is secretly undetectable.

Multi-user setting. For simplicity Definition 18 considers a single user. We provide an extension to the more general setting with $u \geq 2$ users, together with a complete picture of the relationships between different notions, in Section 4.4.

4.3 Mounting Subversion Attacks

In Section 4.3.1 we show that the biased-randomness attack of [BPR14] (adapted to the case of signatures), satisfies secret undetectability as per Definition 18 while allowing to recover the user's signing key with overwhelming probability. This attack allows to break all signature schemes using a sufficient amount of randomness; in Section 4.3.2 we present a new attack allowing to surreptitiously subvert even signature schemes using only little randomness (say 1 bit), provided that the targeted scheme satisfies an additional property.

[1] Looking ahead, our new attack (cf. Section 4.3.2) will rely on this feature in the multi-user setting.

SA class $\mathcal{A}^F_{\mathsf{bias}}$

Let $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a randomized signature scheme with randomness space $\mathcal{R}$, and $F : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}$ be a pseudorandom function. The class $\mathcal{A}^F_{\mathsf{bias}}$ consists of a set of algorithms $\{\widetilde{\mathsf{A}}_{s,\tau}\}_{s\in\{0,1\}^\kappa,\tau=1}$, where each algorithm in the class behaves as follows:

$\underline{\widetilde{\mathsf{A}}_{s,\tau}(SK, m)}$:

- For $|SK| = \ell$, let $i := \tau \mod \ell$.
- Define the function $g(\cdot) := \mathsf{Sign}(SK, m; \cdot)||\tau$ and sample a random element $\widetilde{r}$ from the distribution

$$\widetilde{\mathcal{R}}^{F(s,\cdot),g(\cdot)}(SK[i], \mathcal{R}) := \{r \in \mathcal{R} : F(s, g(r)) = SK[i]\}. \tag{4.2}$$

- Return the signature $\sigma := \mathsf{Sign}(SK, m; \widetilde{r})$, and update the state $\tau \leftarrow \tau + 1$.

Extracting the signing key. Given as input a vector of signatures $\vec{\sigma} = (\sigma_1, ..., \sigma_\ell)$, for each signature $\sigma_i \in \vec{\sigma}$ try to extract the i-th bit of the signing key by defining $SK'[i] := F(s, \sigma_i||i)$. Return the signing key $SK' := (SK'[1], \ldots, SK'[\ell])$.

Figure 4.1. Attacking coin-injective schemes

4.3.1 Attacking Coin-Injective Schemes

We start by recalling an information-theoretic lemma from [BPR14]. Suppose $g : \mathcal{R} \to \mathcal{R}'$ where $\mathcal{R}, \mathcal{R}' \subseteq \{0,1\}^*$, $f : \{0,1\}^* \to \{0,1\}$, and $\rho = |\mathcal{R}|$. For $b \in \{0,1\}$ consider the following *biased* distribution:

$$\widetilde{\mathcal{R}}^{f,g}(b, \mathcal{R}) = \{r \in \mathcal{R} : f(g(r)) = b\}. \tag{4.1}$$

The lemma below roughly says that if a value r is chosen at random from the real distribution $\mathcal{R}$, the probability that r is also in the biased distribution $\widetilde{\mathcal{R}}$ is high if $|\mathcal{R}|$ is large enough.

Lemma 3 (Lemma 1 of [BPR14])**.** *Let f, g, b, $\mathcal{R}$, and $\widetilde{\mathcal{R}} = \widetilde{\mathcal{R}}^{f,g}(b, \mathcal{R})$ be as defined above. Then, if g is injective and f is drawn at random, for all $r \in \mathcal{R}$ we have*

$$\mathop{\mathbb{P}}_{\widetilde{r} \leftarrow\$ \widetilde{\mathcal{R}}}[r = \widetilde{r}] = (1 - 2^{-\rho})/\rho.$$

The following attack is based on the biased-randomness attack from [BPR14]. Roughly, what it does is to embed a trapdoor—a key for a pseudorandom function—in the subverted signing algorithm and to "bias" the randomness in a way that it becomes possible to any party that knows the trapdoor to leak one bit of the signing key for each signed messaged under that signing key. Hence, if the adversary can obtain at least $|SK|$ signed messages then it can later extract the entire signing key in full.

For the analysis, which relies on Lemma 3, we will need to assume the signing function is injective w.r.t. its random coins—a notion which we define below.

Definition 19 (Coin-injective)**.** *We say that $\mathcal{SS}$ is* coin-injective *if for all $m \in \mathcal{M}$, and for all $(VK, SK) \leftarrow \mathsf{KGen}(1^\kappa)$, we have that $\mathsf{Sign}(SK, m; \cdot)$ is injective.*

Theorem 2. *Let $F : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}$ be a $(t_{\mathsf{prf}}, q_{\mathsf{prf}}, \varepsilon_{\mathsf{prf}})$-secure PRF. For a randomized, coin-injective signature scheme $\mathcal{SS}$ with randomness space of size $\rho = |\mathcal{R}|$, consider the class of SAs $\mathcal{A}^F_{\mathsf{bias}}$ described in Fig. 4.1. Then,*

(i) $\mathcal{A}^F_{\mathsf{bias}}$ is secretly (t, q, ε)-undetectable for $t \approx t_{\mathsf{prf}}$, $q \approx q_{\mathsf{prf}}$ and $\varepsilon \leq q \cdot 2^{-(\rho+1)} + \varepsilon_{\mathsf{prf}}$.

(ii) A random $\widetilde{\mathsf{A}} \in \mathcal{A}^F_{\mathsf{bias}}$ recovers the signing key of the user with probability at least $(1 - (1/2 + \varepsilon_{\mathsf{prf}})^\rho)^\ell$ where ℓ is the size of the signing key.

Proof. (i) Let $\mathbf{G}$ be the game described in Definition 18, where the challenger picks $\widetilde{\mathsf{A}} \leftarrow\!\!\$\ \mathcal{A}^F_{\mathsf{bias}}$ (independently of the user's verification key). Consider the game $\mathbf{G}_0$, an identical copy of game $\mathbf{G}$ when $b = 0$, and consider the game $\mathbf{G}_1$, an identical copy of game $\mathbf{G}$ when $b = 1$. For the first part of the proof the objective is to show that $\mathbf{G}_0 \approx \mathbf{G}_1$.

Now consider game $\mathbf{G}'_0$ an identical copy of game $\mathbf{G}_0$ except that $\mathbf{G}'_0$ utilizes the distribution from Eq. (4.1) instead of the distribution from Eq. (4.2).

Claim 2.1. $|\mathbb{P}[\mathsf{U} \text{ wins in } \mathbf{G}_0] - \mathbb{P}[\mathsf{U} \text{ wins in } \mathbf{G}'_0]| \leq \varepsilon_{\mathsf{prf}}$.

Proof. We assume that there exists a user U that distinguishes between games $\mathbf{G}_0$ and $\mathbf{G}'_0$, and we build a distinguisher D (using U) that breaks the pseudorandomness of the PRF F. Distinguisher D is described below below.

Distinguisher D:

- Run $(VK, SK) \leftarrow \mathsf{KGen}(1^\kappa)$, and return (VK, SK) to U.
- For each query $m_i \in \mathcal{M}$ asked by U, do:
  1. Pick a random $r \leftarrow\!\!\$\ \mathcal{R}$ and compute $x_i = \mathsf{Sign}(SK, m_i; r)||\tau$.
  2. Forward x_i to the target oracle, which answers with $y_i = f(x_i)$ if $b = 0$ or with $y_i = F(s, x_i)$ if $b = 1$ (for a hidden bit b).
  3. If $y_i = SK[i]$, then forward $\sigma_i = \mathsf{Sign}(SK, m_i; r)$ as an answer to the query of U, otherwise return to step (1).[2]
- Output whatever U outputs.

Notice that the probability that D aborts in step (3) of the reduction is the same probability that in game $\mathbf{G}_0$ and $\mathbf{G}'_0$ the subverted signing algorithm fails to sample from the set $\widetilde{\mathcal{R}}$. It follows that in case $b = 0$ distinguisher D perfectly emulates the distribution of $\mathbf{G}_0$, whereas in case $b = 1$ it perfectly emulates the distribution of $\mathbf{G}'_0$. The claim follows. □

Claim 2.2. $|\mathbb{P}[\mathsf{U} \text{ wins in } \mathbf{G}'_0] - \mathbb{P}[\mathsf{U} \text{ wins in } \mathbf{G}_1]| \leq q \cdot 2^{-(\rho+1)}$.

[2] In case $|\mathcal{R}|$ is exponential D simply aborts after polynomially many trials.

Proof. Abusing notation, let us write $\mathbf{G}'_0$ and $\mathbf{G}_1$ for the distribution of the random variables corresponding to U's view in games $\mathbf{G}'_0$ and $\mathbf{G}_1$ respectively. For an index $i \in [0, q]$ consider the hybrid game $\mathbf{H}_i$ that answers the first i signature queries as in game $\mathbf{G}'_0$ while all the subsequent queries are answered as in $\mathbf{G}_1$. We note that $\mathbf{H}_0 = \mathbf{G}_1$ and $\mathbf{H}_q = \mathbf{G}'_0$.

We claim that for all $i \in [q]$, we have $\Delta(\mathbf{H}_{i-1}, \mathbf{H}_i) \leq 2^{-(\rho+1)}$. To see this, fix some $i \in [q]$ and denote with $\mathbf{R}$ (resp. $\widetilde{\mathbf{R}}$) the random variable defined by sampling an element from $\mathcal{R}$ (resp. $\widetilde{\mathcal{R}}$) uniformly at random. Clearly,

$$\begin{aligned}\Delta(\mathbf{H}_{i-1}, \mathbf{H}_i) \leq \Delta(\mathbf{R}, \widetilde{\mathbf{R}}) &= \frac{1}{2} \cdot \sum_{r \in \mathcal{R}} \left| \mathbb{P}[\mathbf{R} = r] - \mathbb{P}[\widetilde{\mathbf{R}} = r] \right| \\ &= \frac{1}{2} \cdot \sum_{r \in \mathcal{R}} \left| \frac{1}{\rho} - \frac{1 - 2^{-\rho}}{\rho} \right| \qquad (4.3) \\ &= \frac{1}{2} \cdot 2^{-\rho} = 2^{-(\rho+1)},\end{aligned}$$

where Eq. (4.3) follows by Lemma 3.

The claim now follows by the triangle inequality, as

$$\Delta(\mathbf{G}_1, \mathbf{G}'_0) \leq \sum_{i=1}^{q} \Delta(\mathbf{H}_{i-1}, \mathbf{H}_i) \leq q \cdot 2^{-(\rho+1)}.$$

□

The two claims above finish the proof of statement (i).

(ii) For the second part of the proof we show that the attack of Fig. 4.1 fails to recover the secret key with probability at most $e_1 + e_2 + \ldots + e_\ell$, where $e_j := \mathbb{P}[SK'[j] \neq SK[j]]$. In the analysis, we replace for simplicity the function F with a truly random function f; a generalization accounting for the negligible error due to the use of a pseudorandom function is straightforward. Note that all applications of f are independent because we append the value τ to each query.

Now if g is injective and f is a random function that outputs one bit, then for each element $r \in \mathcal{R}$ we have $\mathbb{P}[f(g(r)) = SK[j]] = 1/2$. Extending to the entire set $\mathcal{R}$ of size ρ we have that

$$e_j := \mathbb{P}\left[\widetilde{\mathcal{R}}^{f,g}(SK[j], \mathcal{R}) = \emptyset\right] = 2^{-\rho},$$

is the error probability for each bit of the secret key. Therefore the probability of recovering the key is at least $(1 - 2^{-\rho})^\ell$. □

Notice that for the attack to be undetectable with high probability, the underlying signature scheme needs to rely on a *minimal* amount of randomness, say $\rho \geq 2^7$.

Making the attack stateless Note that the attack of Fig. 4.1 requires the subverted signature algorithm to maintain a state of logarithmic size (the counter τ). At first sight this might seem a strong assumption, since the original signing algorithm is typically stateless.

However, if we assume that the adversary can control the input messages, one can easily adapt the attack to be completely stateless by letting the input message play the role of the counter τ. Namely, algorithm $\widetilde{\mathsf{A}}$ interprets the input message m as an integer $i \in [\ell]$ and proceeds as before. The above adaptation still allows to recover the signing key and it is undetectable under the *strong undetectability* definition put forward by [BJK15].

4.3.2 Attacking Coin-Extractable Schemes

The attack on Section 4.3.1 allows to break all sufficiently randomized schemes. This leaves the interesting possibility to show a positive result for schemes using less randomness, e.g., the Katz-Wang signature scheme [KW03] that uses a single bit of randomness. In this section we present a simple attack (cf. Fig. 4.2) ruling out the above possibility for all signature schemes that are *coin-extractable*, a notion which we define next.

Definition 20 (Coin-extractable)**.** *Let $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme. We say that $\mathcal{SS}$ is ν_{ext}-coin-extractable if there exists a PPT algorithm CExt such that for all $m \in \mathcal{M}$*

$$\mathbb{P}\left[\sigma = \mathsf{Sign}(SK, m; r) : \begin{array}{c}(VK, SK) \leftarrow \mathsf{KGen}(1^\kappa) \\ \sigma = \mathsf{Sign}(SK, m); r \leftarrow \mathsf{CExt}(VK, m, \sigma)\end{array}\right] \geq 1 - \nu_{ext}.$$

We point that many existing signature schemes are coin-extractable:

- All *public-coin* signature schemes [Sch12], where the random coins used to generate a signature are included as part of the signature. Concretely, the schemes in [GHR99, CS00, NPS01, CL02, Fis03, CL04, BB08, HW09a, HW09b, HK12], and the Unstructured Rabin-Williams scheme [Ber08], are all public-coin.
- The Katz-Wang scheme [KW03], where the signature on a message m is computed as $\sigma = f^{-1}(H(m||r))$ such that f is a trapdoor permutation, H is a hash function, and r is random bit. Given a pair (m, σ) the extractor simply sets $r = 1$ iff $f(\sigma) = H(m||1)$.
- The PSS signature scheme [BR96, Cor02].

Theorem 3. *For a randomized, ν_{ext}-coin-extractable, signature scheme $\mathcal{SS}$ with randomness space $\mathcal{R}$ of size $\rho = 2^d$, consider the class of SAs $\mathcal{A}_{\mathsf{cext}}$ described in Fig. 4.2. Then,*

(i) $\mathcal{A}_{\mathsf{cext}}$ is secretly $(t, q, 0)$-undetectable for $t, q \in \mathbb{N}$.

(ii) Each $\widetilde{\mathsf{A}} \in \mathcal{A}_{\mathsf{cext}}$ recovers the signing key of the user with probability at least $(1 - \nu_{ext})^{\ell/d}$, where ℓ is the size of the key.

Proof. (i) Let $\mathbf{G}$ be the game described in Definition 18, where the challenger picks $\widetilde{\mathsf{A}} \leftarrow\!\!\$\, \mathcal{A}_{\mathsf{cext}}$ uniformly at random (and independently of the user's verification key). Consider the game $\mathbf{G}_0$, an identical copy of game $\mathbf{G}$ when $b = 0$, and consider the game $\mathbf{G}_1$, an identical copy of game $\mathbf{G}$ when $b = 1$. For the first part of the proof the objective is to show that $\mathbf{G}_0 \approx \mathbf{G}_1$.

SA class $\mathcal{A}_{\mathsf{cext}}$

Let $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a coin-extractable, randomized signature scheme with randomness space $\mathcal{R}$ of size $\rho = 2^d$. For simplicity assume that $d|\ell$, where ℓ is the size of the signing key (a generalization is straightforward). The class $\mathcal{A}_{\mathsf{cext}}$ consists of a set of algorithms $\{\widetilde{\mathsf{A}}_{s,\tau}\}_{s\in\{0,1\}^\ell,\tau=0}$, where each algorithm in the class behaves as follows:

$\underline{\widetilde{\mathsf{A}}_{s,\tau}(SK, m)}$:

- If $\tau \geq \ell$ output an honestly generated signature $\sigma := \mathsf{Sign}(SK, m; r)$.
- Else,
  - for each value $j \in [d]$ compute the biased random bit $\widetilde{r}[j] := s[\tau + j] \oplus SK[\tau + j]$;
  - return the signature $\sigma := \mathsf{Sign}(SK, m; \widetilde{r})$, and update the state $\tau \leftarrow \tau + d$.

Extracting the signing key. Given as input a vector of signatures $\vec{\sigma} = (\sigma_1, \ldots, \sigma_{\ell/d})$, parse the trapdoor s as ℓ/d chunks of d bits $s = \{s_1, \ldots, s_{\ell/d}\}$. For each signature $\sigma_i \in \vec{\sigma}$ try to extract the d-bit chunk sk'_i of the signing key as follows.

- Extract the randomness from the i-th signature $\widetilde{r} \leftarrow \mathsf{CExt}(VK, m_i, \sigma_i)$.
- For each value $j \in [d]$ compute the secret key bit $sk'_i[j] := \widetilde{r}[j] \oplus s_i[j]$.

Return the signing key $SK' := (SK'_i, \ldots, SK'_{\ell/d})$.

Figure 4.2. Attacking coin-extractable schemes

Claim 3.1. $|\mathbb{P}[\mathsf{U} \text{ wins in } \mathbf{G}_0] - \mathbb{P}[\mathsf{U} \text{ wins in } \mathbf{G}_1]| = 0$.

Proof. Abusing notation, let us write $\mathbf{G}_0$ and $\mathbf{G}_1$ for the distribution of the random variables corresponding to U's view in games $\mathbf{G}_0$ and $\mathbf{G}_1$ respectively. For an index $i \in [0, q]$ consider the hybrid game $\mathbf{H}_i$ that answers the first i signature queries as in game $\mathbf{G}_0$ while all the subsequent queries are answered as in $\mathbf{G}_1$. We note that $\mathbf{H}_0 \equiv \mathbf{G}_1$ and $\mathbf{H}_q \equiv \mathbf{G}_0$.

We claim that for all $i \in [q]$, we have $\mathbf{H}_{i-1} \equiv \mathbf{H}_i$. To see this, fix some $i \in [q]$ and denote with $\mathbf{R}$ (resp. $\widetilde{\mathbf{R}}$) the random variable defined by sampling an element from $\mathcal{R}$ (resp. $\widetilde{\mathcal{R}}$) uniformly at random. It is easy to see that $\mathbf{R}$ and $\widetilde{\mathbf{R}}$ are identically distributed, as the biased distribution consists of a one-time pad encryption of (part of) the signing key with a uniform key. The claim follows. □

(ii) For the second part of the proof we note that the attack of Fig. 4.2 successfully recovers the biased randomness $\widetilde{r}$ of each $\sigma_i \in \{\sigma_1, \ldots, \sigma_{\ell/d}\}$ and computes the chunk sk_i of the signing key with probability at least $1 - \nu_{ext}$. This gives a total probability of recovering the entire signing key of at least $(1 - \nu_{ext})^{\ell/d}$. □

4.4 The Multi-User Setting

In this section we consider the multi-user setting for the undetectability definition. We also provide a complete picture of relationships between the definition, as shown Fig. 4.3.

4.4.1 Multi-User Public/Secret Undetectability

In the undetectability definition for the multi-user setting user U now receives $u \geq 1$ key pairs from the challenger (only the verification keys for public undetectability) and is allowed to make polynomially many signature queries for all users (key pairs). The answer to these queries are either computed using the real signature algorithm or a subverted algorithm previously chosen by the challenger possibly depending on the verification keys of the users. A formal definition follows.

Definition 21 (Public/Secret Undetectability—Multi-User)**.** *Let $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme, and $\mathcal{A}$ be some class of SAs for $\mathcal{SS}$. We say that $\mathcal{A}$ is u-users* secretly *undetectable w.r.t. $\mathcal{SS}$ if for all PPT users U, there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ and an efficient challenger such that $\left|\mathbb{P}\left[\mathsf{U} \text{ wins}\right] - \frac{1}{2}\right| \leq \nu(\kappa)$ in the following game:*

1. *The challenger samples $b \leftarrow\!\!\$\, \{0,1\}$, generates $(VK_i, SK_i) \leftarrow \mathsf{KGen}(1^\kappa)$ for $i \in [u]$, chooses $\widetilde{\mathsf{A}} \leftarrow \mathcal{A}$ (possibly depending on $VK_1, \ldots, VK_u$), and gives $(VK_1, SK_1, \ldots, VK_u, SK_u)$ to B.*
2. *The user U can ask polynomially many queries of the form (i, m), where $i \in [u]$ and $m \in \mathcal{M}$. The answer to each query depends on the secret bit b. In particular, if $b = 1$, the challenger returns $\sigma \leftarrow \mathsf{Sign}(SK_i, m)$; if $b = 0$, the challenger returns $\widetilde{\sigma} \leftarrow \widetilde{\mathsf{A}}(SK_i, m)$.*
3. *Finally, U outputs a value $b' \in \{0,1\}$; we say that U wins iff $b' = b$.*

We say that $\mathcal{A}$ is u-users publicly *undetectable w.r.t. $\mathcal{SS}$ if in step 1. of the above game, U is only given the verification keys of the u-users.*

4.4.2 Undetectability Relations

The following theorem (4) formalizes the relations depicted in Fig. 4.3.

Theorem 4. *Let $\mathcal{SS}$ be a signature scheme. The following relations hold.*

1. *For all SA classes $\mathcal{A}$ against $\mathcal{SS}$, if $\mathcal{A}$ is u-sUND then $\mathcal{A}$ is also u-pUND.*

   Proof. Let C^* be the (efficient) challenger that exists by the assumption that $\mathcal{A}$ is secretly undetectable. We claim that $\mathcal{A}$ is also publicly undetectable for the same choice of the challenger C^*. Towards contradiction, consider a user U that wins the public undetectability game described in Definition 21 against C^*. We build a user U' (using U) that wins the secret undetectability game described in Definition 21 against C^*.

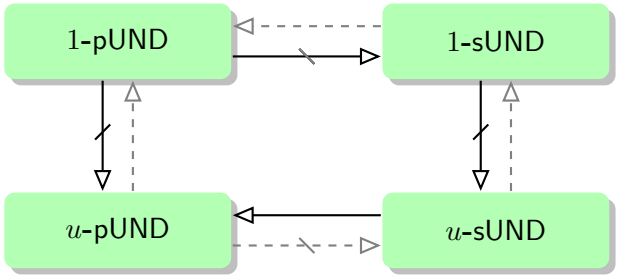

Figure 4.3. Diagram of the relationships between the undetectability notions considered in this paper. $X \to Y$ means that X implies Y; $X \nrightarrow Y$ indicates a separation between X and Y. The lighter arrows indicates trivial implications (or implications that follow from Theorem 4). Public undetectability (Definition 21) is represented by u-pUND and the secret undetectability (Definition 21) is represented by u-sUND.

User U':

(a) Receive $(VK_i, SK_i) \leftarrow \mathsf{KGen}(1^\kappa)$, for $i \in [u]$, from the challenger C^* and forward it to user U.

(b) User U asks polynomially many queries of the type (i, m) which are forwarded to the challenger C^*.

(c) Output whatever U outputs.

We note that the simulation performed by user U' is perfect, therefore U' wins the secret undetectability game with the same probability that user U wins the public undetectability game. □

2. *There exist a SA class $\mathcal{A}$ against $\mathcal{SS}$, such that $\mathcal{A}$ is* 1-pUND *but $\mathcal{A}$ is* not 1-sUND.

Proof. Consider $\mathcal{SS}$ to be a randomized signature scheme (with only two valid signatures for each message $m \in \mathcal{M}$), and let $\mathcal{SS}'$ be its derandomized implementation s.t. $SK' := (SK, s)$, $VK' := VK$, and $\sigma' := \mathsf{Sign}(SK, m; r')$ with $r' := F'_s(m)$ for the PRF $F' : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}$ that outputs a single bit. For concreteness, one can think of $\mathcal{SS}'$ as the Katz-Wang signature scheme [KW03]. Let $\mathcal{A} = \{\widetilde{\mathsf{A}}\}$ be the class of SAs for $\mathcal{SS}'$ described next.

$\widetilde{\mathsf{A}}(SK, m)$:

(a) Upon signature of message m, compute $r' := F'_s(m)$ and set $r'' := \bar{r'}$ (complement bit of r').

(b) Output $\sigma \leftarrow \mathsf{Sign}(SK, m; r'')$.

Every signature generated by $\widetilde{\mathsf{A}}$ verifies correctly[3], therefore, a user U has only a negligible probability of winning at the public undetectability game described in Definition 18. On the other hand, a user U playing the secret undetectability game (has knowledge of SK') can easily detect the subversion of $\widetilde{\mathsf{A}} \in \mathcal{A}$ by simply recomputing $r := F_s(m)$ and signing the message $\sigma \leftarrow \mathsf{Sign}(SK, m; r)$.

[3]This follows by the correcteness of $\mathcal{SS}'$.

A simple comparison can tell if the oracle is the original signing algorithm or the subversion.

□

3. *There exist a SA class $\mathcal{A}$ against $\mathcal{SS}$, such that $\mathcal{A}$ is* 1-pUND *but $\mathcal{A}$ is* not u-pUND*.*

Proof. Consider $\mathcal{SS}$ to be a contrived signature scheme such that the signature of a message $m \in \mathcal{M}$ is $\sigma = \mathsf{Sign}(SK, m; r)||r$, where $r \leftarrow\!\!\$\ \{0,1\}^\kappa$. Let $\mathcal{A} = \{\widetilde{\mathsf{A}}_{\tau,\bar{r}}\}_{\tau=0,\bar{r}\in\{0,1\}^\kappa}$ to be class of SAs for $\mathcal{SS}$ described next.

$\underline{\widetilde{\mathsf{A}}_{\tau,\bar{r}}(SK, m)}$:
(a) If $\tau = 0$ then let $r := \bar{r}$, else let $r \leftarrow\!\!\$\ \{0,1\}^\kappa$.
(b) Output $\sigma \leftarrow \mathsf{Sign}(SK, m; r)||r$ and update $\tau = \tau + 1$.

Clearly, the class $\mathcal{A}$ is publicly undetectable for a single user because the output of the subverted signature algorithm is indistinguishable from that of the real signing algorithm, even for the first query (when $\tau = 0$). However, the class $\mathcal{A}$ is clearly 2-users publicly *detectable* since (no matter the strategy of the challenger) it suffices to ask one query for each user and compare the last κ bits of the signatures to distinguish between real and subverted signatures. □

4. *There exist a SA class $\mathcal{A}$ against $\mathcal{SS}$, such that $\mathcal{A}$ is* 1-sUND *but $\mathcal{A}$ is* not u-sUND*.*

Proof. Consider $\mathcal{SS}$ to be a randomized, coin-extractable signature scheme, with randomness size of ℓ-bits, where $\ell = |SK|$, and $\mathcal{A}_{\mathsf{cext}}$ to be the class of SAs for $\mathcal{SS}$ described in Fig. 4.2. We already showed in Theorem 3 that (for the challenger C^* that chooses $\widetilde{\mathsf{A}}$ at random from $\mathcal{A}_{\mathsf{cext}}$) any PPT user U playing the secret undetectability game described in Definition 18 has a negligible advantage. Now consider the same user U playing the 2-users secret undetectability game described in Definition 21; user U now has 2 key pairs that can be used to detect the attack in the following way.

$\underline{\text{User U:}}$
(a) Receive $(VK_i, SK_i) \leftarrow \mathsf{KGen}(1^\kappa)$, for $i = 1, 2$.
(b) Fix a message $\bar{m} \in \mathcal{M}$ and query $(1, \bar{m})$ and $(2, \bar{m})$ to the challenger, that replies with σ_1 and σ_2.
(c) Use CExt to extract the randomness from σ_1 and σ_2 to get $r_1 \leftarrow \mathsf{CExt}(VK_1, \bar{m}, \sigma_1)$ and $r_2 \leftarrow \mathsf{CExt}(VK_2, \bar{m}, \sigma_2)$.
(d) Compute $SK_1 \oplus SK_2$ and return 0 iff the result equals $r_1 \oplus r_2$.

We note that the above detection strategy works regardless what strategy the challenger uses to select an algorithm from the class $\mathcal{A}_{\mathsf{cext}}$. We conclude that user U has an overwhelming probability of distinguishing between real and subverted signatures. □

4.4.3 Mounting Multi-User SAs

In this section we extend the attacks of Fig. 4.1 and Fig. 4.2 to the multi-user setting.

Attacking Coin-Injective Schemes (Multi-User Version)

The attack described in Fig. 4.1 can be extended to the multi-user setting with minor modifications. We create an SA class $\mathcal{A}^{F,u}_{\mathsf{bias}}$ from the class $\mathcal{A}^{F}_{\mathsf{bias}}$ of Fig. 4.1 by just appending the index j, that represents each user, to the function $g(\cdot) = \mathsf{Sign}(SK_j, m)||\tau||j$, so that each application of the random function $f(g(\cdot))$ remains independent.

The two lemmas below (Lemma 4 and Lemma 5) are needed for the proof of undetectability in the multi-user setting. The two lemmas combined roughly state that the statistical distance of a joint distribution of u random variables is *at most* u times the statistical distance of each pair of the random variables.

Lemma 4 ([Rey11])**.** *Let* $\mathbf{X}$ *and* $\mathbf{Y}$ *be two random variables over some finite domain, and let* G *be a randomized function. Then* $\Delta\left(G(\mathbf{X}), G(\mathbf{Y})\right) \leq \Delta\left(\mathbf{X}, \mathbf{Y}\right)$*.*

Lemma 5. *Let* $\mathbf{X}$ *and* $\mathbf{Y}$ *be two random variables over some finite domain, and let* $(\mathbf{X}_1, \ldots, \mathbf{X}_u)$ *and* $(\mathbf{Y}_1, \ldots, \mathbf{Y}_u)$ *be* u *independent copies of the random variables* $\mathbf{X}$ *and* $\mathbf{Y}$*. Then*

$$\Delta\left((\mathbf{X}_1, \ldots, \mathbf{X}_u), (\mathbf{Y}_1, \ldots, \mathbf{Y}_u)\right) \leq u \cdot \Delta\left(\mathbf{X}, \mathbf{Y}\right).$$

Proof. We prove this lemma by induction. First we consider the basis case where $i = 1$, which trivially holds as $\Delta\left(\mathbf{X}_1, \mathbf{Y}_1\right) \leq \Delta\left(\mathbf{X}, \mathbf{Y}\right)$.

For the induction step we define the random functions $G_1(\cdot) = (\mathbf{X}_1, \ldots, \mathbf{X}_{i-1}, \cdot)$ and $G_2(\cdot) = (\cdot, \mathbf{Y}_i)$. We assume that the statement holds up to $i-1$ random variables and then we proceed to show that it also holds for i random variables.

$$\begin{aligned}
&\Delta\left((\mathbf{X}_1, \ldots, \mathbf{X}_i), (\mathbf{Y}_1, \ldots, \mathbf{Y}_i)\right) \\
&\leq \Delta\left((\mathbf{X}_1, \ldots, \mathbf{X}_i), (\mathbf{X}_1, \ldots, \mathbf{X}_{i-1}, \mathbf{Y}_i)\right) + \Delta\left((\mathbf{X}_1, \ldots, \mathbf{X}_{i-1}, \mathbf{Y}_i), (\mathbf{Y}_1, \ldots, \mathbf{Y}_i)\right) \\
&= \Delta\left(G_1(\mathbf{X}_i), G_1(\mathbf{Y}_i)\right) + \Delta\left(G_2(\mathbf{X}_1, \ldots, \mathbf{X}_{i-1}), G_2(\mathbf{Y}_1, \ldots, \mathbf{Y}_{i-1})\right) \\
&\leq \Delta\left(\mathbf{X}, \mathbf{Y}\right) + \Delta\left((\mathbf{X}_1, \ldots, \mathbf{X}_{i-1}), (\mathbf{Y}, \ldots, \mathbf{Y}_{i-1})\right) \\
&\leq i \cdot \Delta\left(\mathbf{X}, \mathbf{Y}\right),
\end{aligned}$$

where the first inequality follows by the triangle inequality, the second inequality follows by Lemma 4, and the third inequality follows by the induction hypothesis. □

The theorem below quantifies the effectiveness of the attack of Fig. 4.1 in the multi-user setting.

Theorem 5. *Let* $F : \{0,1\}^{\kappa} \times \{0,1\}^{*} \to \{0,1\}$ *be a secure PRF. For a randomized, coin-injective signature scheme* $\mathcal{SS}$ *with randomness space of size* $\rho = |\mathcal{R}|$*, consider the class of SAs* $\mathcal{A}^{F,u}_{\mathsf{bias}}$ *described above. Then,*

(i) $\mathcal{A}^{F,u}_{\mathsf{bias}}$ *is* u*-users secretly undetectable.*

(ii) Each $\widetilde{\mathsf{A}} \in \mathcal{A}^{F,u}_{\mathsf{bias}}$ *recovers the signing key of any of the users with probability at least* $(1-(1/2+\varepsilon_{\mathsf{prf}})^{\rho})^{\ell}$*, where* ℓ *is the size of the signing key.*

Proof. (i) Let $\mathbf{G}$ be the game described in Definition 21. Consider the game $\mathbf{G}_0$, an identical copy of game $\mathbf{G}$ when $b=0$, and consider the game $\mathbf{G}_1$, an identical copy of game $\mathbf{G}$ when $b=1$. For the first part of the proof the objective is to show that $\mathbf{G}_0 \approx \mathbf{G}_1$.

Now consider game $\mathbf{G}'_0$ an identical copy of game $\mathbf{G}_0$ except that $\mathbf{G}'_0$ utilizes the distribution from the random function f (analogous to Eq. (4.1) in the single user attack) instead of the distribution from the PRF F (analogous to Eq. (4.2) in the single user attack).

Claim 5.1. $|\mathbb{P}[\mathsf{U} \text{ wins in } \mathbf{G}_0] - \mathbb{P}[\mathsf{U} \text{ wins in } \mathbf{G}'_0]| \leq \nu(\kappa)$.

The above claim follows by a standard reduction argument to the hardness of the PRF F to distinguishing games $\mathbf{G}_0$ and $\mathbf{G}'_0$. The proof is similar to the one in Theorem 2 and is therefore ommited.

Claim 5.2. $|\mathbb{P}[\mathsf{U} \text{ wins in } \mathbf{G}'_0] - \mathbb{P}[\mathsf{U} \text{ wins in } \mathbf{G}_1]| \leq \nu(\kappa)$.

Proof. Abusing notation, let us write $\mathbf{G}'_0$ and $\mathbf{G}_1$ for the distribution of the random variables corresponding to U's view in games $\mathbf{G}'_0$ and $\mathbf{G}_1$ respectively. For an index $i \in [0,q]$ consider the hybrid game $\mathbf{H}_i$ that answers the first i signature queries as in game $\mathbf{G}'_0$ while all the subsequent queries are answered as in $\mathbf{G}_1$. We note that $\mathbf{H}_0 = \mathbf{G}_1$ and $\mathbf{H}_q = \mathbf{G}'_0$.

We claim that for all $i \in [q]$, we have $\Delta(\mathbf{H}_{i-1}, \mathbf{H}_i) \leq 2^{-(\rho+1)}$. To see this, fix some $i \in [q]$ and denote with $\mathbf{R}_1, \ldots, \mathbf{R}_u$ (resp. $\widetilde{\mathbf{R}}_1, \ldots, \widetilde{\mathbf{R}}_u$) the random variables defined by sampling an element from $\mathcal{R}$ (resp. $\widetilde{\mathcal{R}}$) uniformly at random. Clearly,

$$\begin{aligned}
\Delta(\mathbf{H}_{i-1}, \mathbf{H}_i) &\leq \Delta((\mathbf{R}_1, \ldots, \mathbf{R}_u), (\widetilde{\mathbf{R}}_1, \ldots, \widetilde{\mathbf{R}}_u)) \\
&\leq u \cdot \Delta(\mathbf{R}, \widetilde{\mathbf{R}}) \qquad (4.4) \\
&= u \cdot 2^{-(\rho+1)}, \qquad (4.5)
\end{aligned}$$

where Eq. (4.4) follows by Lemma 5 and Eq. (4.5) follows by Eq. (4.3).

The claim now follows by the triangle inequality, as

$$\Delta(\mathbf{G}_1, \mathbf{G}'_0) \leq \sum_{i=1}^{q} \Delta(\mathbf{H}_{i-1}, \mathbf{H}_i) \leq qu \cdot 2^{-(\rho+1)}$$

and the last term becomes negligible for $u, q = \mathit{poly}(\kappa)$ and for ρ large enough. □

The two claims above finish the proof of statement (i).

(ii) For the second part of the proof we proceed as in Theorem 2. We note that the specified class of SAs $\mathcal{A}^{F,u}_{\mathsf{bias}}$ maintains each application of the random function f independent by appending the index j, that represents each user, to the function g, obtaining $g(\cdot) = \mathsf{Sign}(SK_j, m)||\tau||j$. The statement follows. □

Attacking Coin-Extractable Schemes (Multi-User Version)

The attack against coin-extractable schemes described in Fig. 4.2 becomes easily detectable in the presence of two or more users (see the proof of Theorem 4, item (iv)). An easy solution is to modify the SA class such that each algorithm in the class uses a different one-time pad key for each target user. We describe this class of SAs in Fig 4.4.

SA class $\mathcal{A}^u_{\mathsf{cext}}$

Let $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a coin-extractable randomized signature scheme with randomness space $\mathcal{R}$ of size $\rho = 2^d$. The class $\mathcal{A}^u_{\mathsf{cext}}$ consists of a set of algorithms $\{\widetilde{\mathsf{A}}_{\vec{s},\vec{VK},\vec{\tau}}\}_{\vec{s}\in\{0,1\}^{\ell\cdot u},\vec{VK}\in\mathcal{VK}^u,\vec{\tau}=0^u}$, where $\ell = |SK|$, and where each algorithm in the class behaves as follows:

$\underline{\widetilde{\mathsf{A}}_{\vec{s},\vec{VK},\vec{\tau}}(SK_i, m)}$:

- Parse $\vec{s}$ as $(s_1, \ldots, s_u)$, $\vec{VK}$ as $(VK_1, \ldots, VK_u)$, and $\vec{\tau}$ as $(\tau_1, \ldots, \tau_u)$.
- Find the index i such that $\mathsf{Vrfy}(VK_i, m, \mathsf{Sign}(SK_i, m)) = 1$.
- If $\tau_i \geq \ell$ output a real signature $\sigma \leftarrow \mathsf{Sign}(SK_i, m)$.
- Else,
  - For each value $j \in [d]$ compute the biased random bit $\widetilde{r}[j] = s_i[\tau_i + j] \oplus SK_i[\tau_i + j]$.
  - Return the signature $\sigma := \mathsf{Sign}(SK_i, m; \widetilde{r})$, and update the state $\tau_i \leftarrow \tau_i + d$.

Extracting the signing key. Given as input a vector of signatures $\vec{\sigma} = (\sigma_1, ..., \sigma_{\ell/d})$ of user i, represent the trapdoor s_i as ℓ/d chunks of d bits $s_i = \{s_{i,1}, \ldots, s_{i,\ell/d}\}$. For each signature $\sigma_k \in \vec{\sigma}$ try to extract the d-bit chunk $sk'_{i,k}$ of the signing key as follows.

- Extract the randomness from the k-th signature $\widetilde{r} \leftarrow \mathsf{CExt}(VK_i, m_k, \sigma_k)$.
- For each value $j \in [d]$ compute the secret key bit $sk'_{i,k}[j] = \widetilde{r}[j] \oplus s_{i,k}[j]$.

Return the signing key $SK'_i := (SK'_{i,k}, \ldots, SK'_{i,\ell/d})$.

Figure 4.4. Attacking coin-extractable schemes in the multi-user setting

Theorem 6. *For a randomized, ν_{ext}-coin-extractable, signature scheme $\mathcal{SS}$ with randomness space $\mathcal{R}$ of size $\rho = 2^d$, consider the class of SAs $\mathcal{A}^u_{\mathsf{cext}}$ described in Fig. 4.4. Then,*

(i) $\mathcal{A}^u_{\mathsf{cext}}$ is u-users secretly undetectable.

(ii) Each $\widetilde{\mathsf{A}} \in \mathcal{A}^u_{\mathsf{cext}}$ recovers the signing key of any of the users with probability at least $(1 - \nu_{ext})^{\ell/d}$.

Proof. (i) Let $\mathbf{G}$ be the game described in Definition 21, where the challenger first generates all key pairs (VK_i, SK_i) (for $i \in [u]$) and afterwards selects the algorithm $\widetilde{\mathsf{A}} \leftarrow \mathcal{A}^u_{\mathsf{cext}}$ such that $\vec{VK} := (VK_1, \ldots, VK_u)$. Consider the game $\mathbf{G}_0$, an identical copy of game $\mathbf{G}$ when $b = 0$, and consider the game $\mathbf{G}_1$, an identical copy of game $\mathbf{G}$ when $b = 1$. For the first part of the proof the objective is to show that $\mathbf{G}_0 \approx \mathbf{G}_1$.

Claim 6.1. $\mathbf{G}_0 \equiv \mathbf{G}_1$.

Proof. Abusing notation, let us write $\mathbf{G}_0$ and $\mathbf{G}_1$ for the distribution of the random variables corresponding to U's view in games $\mathbf{G}_0$ and $\mathbf{G}_1$ respectively. For an index $i \in [0, q]$ consider the hybrid game $\mathbf{H}_i$ that answers the first i signature queries as in game $\mathbf{G}_0$ while all the subsequent queries are answered as in $\mathbf{G}_1$. We note that $\mathbf{H}_0 \equiv \mathbf{G}_1$ and $\mathbf{H}_q \equiv \mathbf{G}_0$.

We claim that for all $i \in [q]$, we have $\mathbf{H}_{i-1} \equiv \mathbf{H}_i$. To see this, fix some $i \in [q]$ and denote with $\mathbf{R}_1, \ldots, \mathbf{R}_u$ the random variables defined by sampling an element from $\mathcal{R}$ uniformly at random and with $\widetilde{\mathbf{R}}_1, \ldots, \widetilde{\mathbf{R}}_u$ the random variables defined by sampling an element from the biased distribution $\widetilde{\mathcal{R}}$ also uniformly at random. It is easy to see that $\mathbf{R}_i$ and $\widetilde{\mathbf{R}}_i$, for $i \in [q]$, are identically distributed, as the biased distribution consists of a one-time pad encryption of (part of) the signing key with a uniform key (a different key for each user). The claim follows. □

(ii) For the second part of the proof we note that the attack of Fig. 4.4 successfully recovers the biased randomness $\widetilde{r}$ of each $\sigma_i \in \{\sigma_1, \ldots, \sigma_{\ell/d}\}$ and computes the chunk $sk_{j,i}$ of the signing key of a user j with probability at least $1 - \nu_{ext}$. This gives a total probability of recovering an entire signing key of at least $(1 - \nu_{ext})^{\ell/d}$. □

CHAPTER 5

The (not so) Ugly

In this chapter we cover immunization techniques against malicious backdoors. We focus on two scenarios for immunization; the first is the security of signature schemes against subversion attacks.[1] In that direction we give security definitions for signature schemes under the subversion attack model and in Section 5.1 we show that unique signature schemes are secure against all subversion attacks that conforms to a basic form of undetectability. In Section 5.2 we present a secure solution for any re-randomizable signature scheme (of which unique signatures are a special case) against *all* subversion attacks by using an un-tamperable hardware device, called Cryptographic Reverse Firewall [MS15]. Later, in Section 5.1.5 we extend our definitions to the multi-user scenario, and we show all the relations among the different security definitions. Lastly, in Section 5.1.6 we present the first efficient unique signature scheme with a tight security proof based on a standard cryptographic assumption, namely the Quadratic Residuosity assumption.

The second scenario of immunization techniques that we cover is concerned with cryptographic circuits. In Section 5.3.1, we propose a new approach to the untrusted circuit fabrication problem. We assume that the circuit specification and design are trusted but the fabrication facility is not. This leaves the possibility of a malicious foundry to inject malicious circuit in the design before building the circuit. Rather than testing or reverse engineering the circuit hardware received, we employ it in a controlled environment and continuously verify its operations. The idea is to compile the circuit design before the production phase in a way that the malicious circuit inserted by the foundry is not able to cause any harm to the operations of the compiled circuit. We introduce three compilers; two are based on techniques of verifiable computation (Section 5.3.2) and the last one is based on multiparty computation (Section 5.3.3).

[1] Subversion attacks are a generalization of backdoor attacks, therefore they are even more powerful.

5.1 Security of Unique Signatures

In this section we give security definitions for signature schemes against subversion attacks (SAs) and we prove that signature schemes with unique signatures are subversion-resilient against SAs that meet a basic undetectability requirement, which we call the verifiability condition.

5.1.1 Impersonation

We consider two security definitions, corresponding to different adversarial goals.

Indistinguishability. In the first definition, it is required that an adversary B having access to polynomially many subversion oracles chosen adaptively (possibly depending on the user's verification key), cannot distinguish signatures produced via the standard signing algorithm from subverted signatures.

Definition 22 (Indistinguishability against SAs)**.** *Let $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme, and $\mathcal{A}$ be some class of SAs for $\mathcal{SS}$. We say that $\mathcal{SS}$ is (t, n, q, ε)-indistinguishable w.r.t* continuous *$\mathcal{A}$-SAs if for all PPT adversaries B running in time t, we have $\left|\mathbb{P}\left[\mathsf{B} \text{ wins}\right] - \frac{1}{2}\right| \leq \varepsilon(\kappa)$ in the following game:*

1. *The challenger runs $(VK, SK) \leftarrow \mathsf{KGen}(1^\kappa)$, samples $b \leftarrow\!\!\$\ \{0,1\}$, and gives VK to B.*
2. *The adversary B can ask the following two types of queries; the queries can be specified adaptively and in an arbitrary order:*
   - *Choose an algorithm $\widetilde{\mathsf{A}}_j \in \mathcal{A}$, for $j \in [n]$, and give it to the challenger.*
   - *Forward a pair $(j, m_{i,j})$ to the challenger, where $i \in [q]$ and $j \in [n]$. The answer to each query depends on the value of the secret bit b. In particular, if $b = 1$, the output is $\sigma_{i,j} \leftarrow \mathsf{Sign}(SK, m_{i,j})$; if $b = 0$, the output is $\widetilde{\sigma}_{i,j} \leftarrow \widetilde{\mathsf{A}}_j(SK, m_{i,j})$.*
3. *Finally, B outputs a value $b' \in \{0,1\}$; we say that B wins iff $b' = b$.*

Whenever $\varepsilon(\kappa) = negl(\kappa), q = poly(\kappa)$, and $n = poly(\kappa)$ we simply say that $\mathcal{SS}$ is indistinguishable against continuous $\mathcal{A}$-SAs.

Impersonation under chosen-message attacks. We also consider an alternative (strictly weaker—cf. Section 5.1.5) definition, where the goal of the adversary is now to forge a signature on a "fresh" message (not asked to any of the oracles).

Definition 23 (EUF-CMA against SAs)**.** *Let $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme, and $\mathcal{A}$ be some class of SAs for $\mathcal{SS}$. We say that $\mathcal{SS}$ is (t, n, q, ε)-EUF-CMA w.r.t.* continuous *$\mathcal{A}$-SAs if for all PPT adversaries B running in time t, we have $\mathbb{P}\left[\mathsf{B} \text{ wins}\right] \leq \varepsilon(\kappa)$ in the following game:*

1. *The challenger runs $(VK, SK) \leftarrow \mathsf{KGen}(1^\kappa)$, and gives VK to B.*

2. *The adversary* B *is given oracle access to* $\mathsf{Sign}(SK, \cdot)$. *Upon input the* i*-th query* m_i*, this oracle returns* $\sigma_i \leftarrow \mathsf{Sign}(SK, m_i)$*; let* $\mathcal{Q} = \{m_1, \ldots, m_q\}$ *be the set of all queried messages.*

3. *For each* $j \in [n]$*, the adversary* B *can adaptively choose an algorithm* $\widetilde{\mathsf{A}}_j \in \mathcal{A}$. *For each algorithm,* B *is given oracle access to* $\widetilde{\mathsf{A}}_j(SK, \cdot)$. *Upon input a message* $\widetilde{m}_{i,j}$*, the oracle returns* $\widetilde{\sigma}_{i,j} \leftarrow \widetilde{\mathsf{A}}_j(SK, \widetilde{m}_{i,j})$*; let* $\widetilde{\mathcal{Q}}_j = \{\widetilde{m}_{1,j}, \ldots, \widetilde{m}_{q,j}\}$ *be the set of all queried messages to the oracle* $\widetilde{\mathsf{A}}_j$.

4. *Finally,* B *outputs a pair* (m^*, σ^*)*; we say that* B *wins iff* $\mathsf{Vrfy}(VK, (m^*, \sigma^*)) = 1$ *and* $m^* \notin \mathcal{Q} \cup \widetilde{\mathcal{Q}}$*, where* $\widetilde{\mathcal{Q}} := \bigcup_{j=1}^{n} \widetilde{\mathcal{Q}}_j$.

Whenever $\varepsilon(\kappa) = negl(\kappa)$*,* $q = poly(\kappa)$*, and* $n = poly(\kappa)$ *we simply say that* $\mathcal{SS}$ *is EUF-CMA against continuous* $\mathcal{A}$*-SAs.*

Remarks. Some remarks on the above definitions are in order.

- First, note that it is impossible to prove that a signature scheme $\mathcal{SS}$ satisfies Definition 22 (and consequently Definition 23) for an *arbitrary* class $\mathcal{A}$, without making further assumptions.[2] To see this, consider the simple algorithm that ignores all inputs and outputs the secret key.[3]

- We observe that continuous $\mathcal{A}$-SAs security, implies security against continuous tampering attacks with the secret key. This can be seen by considering a class of algorithms $\mathcal{A}_{\mathsf{key}} = \{\widetilde{\mathsf{A}}_f\}_{f \in \mathcal{F}}$, where $\mathcal{F}$ is a class of functions such that each $f \in \mathcal{F}$ has a type $f : \mathcal{SK} \to \mathcal{SK}$, and for all $f \in \mathcal{F}$, $m \in \mathcal{M}$ and $r \in \mathcal{R}$ we have that $\widetilde{\mathsf{A}}_f(\cdot, m; r) := \mathsf{Sign}(f(\cdot), m; r)$.[4]

- It is useful to compare Definition 22 to the security definition against algorithm-substitution attacks given in [BPR14] (for the case of symmetric encryption). In the language of Bellare *et al.* [BPR14], a subversion of a signature scheme would be a triple of algorithms $\widetilde{\mathcal{SS}} = (\widetilde{\mathsf{KGen}}, \widetilde{\mathsf{Sign}}, \widetilde{\mathsf{Vrfy}})$, where in the security game $\widetilde{\mathsf{KGen}}$ is run by the challenger in order to obtain a trapdoor $\alpha \in \{0,1\}^*$ and some initial state $\tau \in \{0,1\}^*$ which are both hard-wired in the algorithm $\widetilde{\mathsf{Sign}} := \widetilde{\mathsf{Sign}}_{\alpha,\tau}$ (and given to B).[5]

  The above setting can be cast in our framework by considering the class of SAs $\mathcal{A}_{\mathsf{BRP14}} := \{\widetilde{\mathsf{A}}_{\alpha,\tau} : (\alpha, \tau) \leftarrow \widetilde{\mathsf{KGen}}(1^\kappa)\}$, and by setting $n = 1$ in Definition 22.

[2] Looking ahead, one of our positive results achieves security w.r.t. arbitrary SAs assuming the existence of a cryptographic reverse firewall. See Section 5.2.

[3] In case the secret key is too long, one can make the algorithm stateful so that it outputs a different chunk of the key at each invocation. Alternatively, consider the class of algorithms $\{\widetilde{\mathsf{A}}_{\bar{m}}\}_{\bar{m} \in \mathcal{M}}$ that always outputs a signature $\bar{\sigma}$ on $\bar{m}$; obviously this subversion allows to forge on $\bar{m}$ without explicitly querying the message to any of the oracles.

[4] It is worth noting that already for $n = 1$ Definition 23 implies *non-adaptive* key tampering, as the subverted algorithm can hard-wire (the description of) polynomially many pre-set tampering functions.

[5] The algorithm $\widetilde{\mathsf{Vrfy}}$ is not explicitly part of the definitions in [BPR14]—in fact, a secure scheme implicitly excludes that any $\widetilde{\mathsf{Vrfy}}$ algorithm exists—and can be considered as part of the adversary itself.

Our definition is more general, as it accounts for arbitrary classes of SAs and moreover allows B to subvert a user's algorithm continuously and in a fully-adaptive fashion (possibly depending on the target verification key).

Multi-user setting. For simplicity Definition 22 and 23 consider a single user. We provide an extension to the more general setting with $u \geq 2$ users, together with a complete picture of the relationships between the different notions, in Section 5.1.5.

The Verifiability Condition

We say that $\mathcal{A}$ meets the *verifiability condition* relative to $\mathcal{SS}$ if for all $\widetilde{\mathsf{A}} \in \mathcal{A}$ and for all $m \in \mathcal{M}$ the signatures produced using the subverted signing algorithm $\widetilde{\mathsf{A}}$ (almost) always verify under the corresponding verification key VK.

Definition 24 (Verifiability)**.** *Let $\mathcal{A}$ be some class of SAs for a signature scheme $\mathcal{SS}$. We say that $\mathcal{A}$ satisfies ν_v-verifiability if for all $\widetilde{\mathsf{A}} \in \mathcal{A}$ and for all $m \in \mathcal{M}$*

$$\mathbb{P}\left[\mathsf{Vrfy}(VK, (m, \widetilde{\mathsf{A}}(SK, m))) = 1 : (VK, SK) \leftarrow \mathsf{KGen}(1^\kappa)\right] \geq 1 - \nu_v,$$

where the probability is taken over the randomness of all involved algorithms.

Public undetectability vs. verifiability. One might think that verifiability is a special case of public undetectability (Definition 18). However, this is not true and in fact Definition 24 and 18 are incomparable. To see this, consider the class of SAs $\mathcal{A}_{\mathsf{msg}} = \{\widetilde{\mathsf{A}}_{\bar{m}}\}_{\bar{m} \in \mathcal{M}}$ that behaves identically to the original signing algorithm, except that upon input $\bar{m} \in \mathcal{M}$ it outputs an invalid signature.[6] Clearly, $\mathcal{A}_{\mathsf{msg}}$ satisfies public undetectability as a user has only a negligible chance of hitting the value $\bar{m}$; yet $\mathcal{A}_{\mathsf{msg}}$ does not meet the verifiability condition as the latter is a property that must hold for *all* messages.

On the other hand, consider the class of SAs $\mathcal{A}_{\mathsf{det}}$ that is identical to the original signing algorithm, except that it behaves deterministically on repeated inputs. Clearly, $\mathcal{A}_{\mathsf{det}}$ meets the verifiability condition relative to any (even randomized) signature scheme $\mathcal{SS}$; yet $\mathcal{A}_{\mathsf{det}}$ does not satisfy public undetectability for any randomized signature scheme $\mathcal{SS}$, as a user can simply query the same message twice in order to guess the value of the hidden bit b with overwhelming probability.

Relaxed verifiability. Clearly, the assumption that the verifiability condition should hold for all values $m \in \mathcal{M}$ is quite a strong one. A natural relaxation is to require that the probability in Definition 24 is taken also over the choice of the message.

Definition 25 (Relaxed Verifiability)**.** *Let $\mathcal{A}$ be some class of SAs for a signature scheme $\mathcal{SS}$. We say that $\mathcal{A}$ satisfies* relaxed *ν_v-verifiability if for all $\widetilde{\mathsf{A}} \in \mathcal{A}$*

$$\mathbb{P}\left[\mathsf{Vrfy}(VK, (m, \widetilde{\mathsf{A}}(SK, m))) = 1 : (VK, SK) \leftarrow \mathsf{KGen}(1^\kappa); m \leftarrow\!\!\$\, \mathcal{M}\right] \geq 1 - \nu_v,$$

[6] A similar class of attacks—under the name of input-triggered subversion—has been recently considered in [DFP15] for the case of symmetric encryption.

where the probability is taken over the choice of the message and over the randomness of all involved algorithms.

We argue that relaxed verifiability is implied by public undetectability (cf. Definition 18) in many interesting cases.

- *Input-triggered subversions.* Whenever public undetectability holds for all algorithms in the class $\mathcal{A}$. This is the case, for instance, for the class $\mathcal{A}_{\mathsf{msg}}$ of input-triggered subversions described above.

  To see this, let $\mathcal{A}$ be a class of SAs that is publicly undetectable for all $\widetilde{\mathsf{A}} \in \mathcal{A}$. Towards a contradiction, assume that $\mathcal{A}$ does not satisfy relaxed verifiability. This means that there exists an $\widetilde{\mathsf{A}} \in \mathcal{A}$ and a polynomial $p(\cdot)$ such that

  $$\mathbb{P}\left[\mathsf{Vrfy}(VK, (m, \widetilde{\mathsf{A}}(SK, m))) = 0 : (VK, SK) \leftarrow \mathsf{KGen}(1^\kappa); m \leftarrow\!\!\$\ \mathcal{M}\right] \geq \frac{1}{p(\kappa)},$$

  for infinitely many values of $\kappa \in \mathbb{N}$. It follows that $\widetilde{\mathsf{A}}$ can be used to break public undetectability with probability $1/p(\kappa)$, by simply signing a random message and trying to verify the outcome.

- *Backdoored implementations.* The above implication also holds for the class $\mathcal{A}_{\mathsf{BPR14}}$ of algorithm-substitution attacks and backdoored implementations (see paragraph "Remarks" in Section 5.1.1), as long as the winning condition in Definition 18 and Definition 25 is taken also over the choice of the backdoor (i.e., over the random coins of algorithm $\widetilde{\mathsf{KGen}}$).

5.1.2 Security Against Chosen-Message Attacks

The theorem below shows that unique signature schemes (cf. Definition 6) achieve indistinguishability (and thus EUF-CMA) against the class of all SAs that meet the verifiability condition (cf. Definition 24).

Theorem 7. *Let $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme with ν_c-correctness and ν_u-uniqueness, and denote by $\mathcal{A}^{\nu_v}_{\mathsf{ver}}$ the class of all algorithms that satisfy ν_v-verifiability relative to $\mathcal{SS}$. Then $\mathcal{SS}$ is (t, n, q, ε)-indistinguishable against continuous $\mathcal{A}^{\nu_v}_{\mathsf{ver}}$-SAs, for all $n, q \in \mathbb{N}$ and for $\varepsilon \leq qn \cdot (\nu_c + \nu_v + \nu_u)$.*

Proof. Let $\mathbf{G}$ be the game described in Definition 22. Consider the game $\mathbf{G}_0$, an identical copy of game $\mathbf{G}$ when $b = 0$, and consider the game $\mathbf{G}_1$, an identical copy of game $\mathbf{G}$ when $b = 1$. The objective here is to show that $\mathbf{G}_0 \approx \mathbf{G}_1$.

For an index $k \in [0, n]$, consider the hybrid game $\mathbf{H}_k$ that answers each query $(j, m_{i,j})$ such that $j \leq k$ as in game $\mathbf{G}_0$ (i.e., by running $\mathsf{Sign}(SK, m_{i,j})$), while all queries $(j, m_{i,j})$ such that $j > k$ are answered as in $\mathbf{G}_1$ (i.e., by running $\widetilde{\mathsf{A}}_j(SK, m_{i,j})$). We note that $\mathbf{H}_0 \equiv \mathbf{G}_1$ and $\mathbf{H}_n \equiv \mathbf{G}_0$. Abusing notation, let us write $\mathbf{G}_k$ for the distribution of the random variable corresponding to B's view in games $\mathbf{G}_k$.

Fix a particular $k \in [0, n]$, and for an index $l \in [0, q]$ consider the hybrid game $\mathbf{H}_{k,l}$ that is identical to $\mathbf{H}_k$ except that queries $(k, m_{i,k})$ with $i \leq l$ are treated as in game $\mathbf{G}_0$, while queries $(k, m_{i,k})$ with $i > l$ are treated as in $\mathbf{G}_1$. Observe that $\mathbf{H}_{k,0} \equiv \mathbf{H}_{k-1}$, and $\mathbf{H}_{k,q} \equiv \mathbf{H}_k$.

Claim 7.1. *Fix some $k \in [0,n]$. For each $l \in [0,q]$, we have $\Delta(\mathbf{H}_{k,l-1}, \mathbf{H}_{k,l}) \leq \nu_c + \nu_v + \nu_u$.*

Proof. Notice that the only difference between $\mathbf{H}_{k,l-1}$ and $\mathbf{H}_{k,l}$ is how the two games answer the query $(k, m_{l,k})$: Game $\mathbf{H}_{k,l-1}$ returns $\sigma_{l,k} \leftarrow \mathsf{Sign}(SK, m_{l,k})$, whereas game $\mathbf{H}_{k,l}$ returns $\widetilde{\sigma}_{l,k} \leftarrow \widetilde{\mathsf{A}}_k(SK, m_{l,k})$. Now let $E_{l,k}$ be the event that $\sigma_{l,k} \neq \widetilde{\sigma}_{l,k}$. We can write

$$\Delta(\mathbf{H}_{k,l-1}, \mathbf{H}_{k,l}) \leq \Delta(\mathbf{H}_{k,l-1}; \mathbf{H}_{k,l} | \neg E_{l,k}) + \mathbb{P}[E_{l,k}] \tag{5.1}$$

$$\leq \nu_c + \nu_u + \nu_v. \tag{5.2}$$

Eq. (5.1) follows by Lemma 1 and Eq. (5.2) follows by the fact that $\mathbf{H}_{k,l-1}$ and $\mathbf{H}_{k,l}$ are identically distributed conditioned on $E_{l,k}$ not happening, and moreover $\mathbb{P}[E_{l,k}] \leq \nu_c + \nu_u + \nu_v$. The latter can also be seen as follows. By the correctness condition of $\mathcal{SS}$ we have that $\sigma_{l,k}$ is valid for $m_{l,k}$ under VK except with probability at most ν_c. By the assumption that $\widetilde{\mathsf{A}}_k \in \mathcal{A}^{\nu_v}_{\mathsf{ver}}$ we have that $\widetilde{\sigma}_{l,k}$ is also valid for $m_{l,k}$ under VK except with probability at most ν_v. Finally, by the uniqueness property of $\mathcal{SS}$ we have that $\sigma_{l,k}$ and $\widetilde{\sigma}_{l,k}$ must be equal except with probability at most ν_u. It follows that $\mathbb{P}[E_{l,k}] \leq \nu_c + \nu_u + \nu_v$, as desired. □

The statement now follows by the above claim and by the triangle inequality, as

$$\Delta(\mathbf{G}_0, \mathbf{G}_1) \leq \sum_{k=1}^{n} \Delta(\mathbf{H}_{k-1}, \mathbf{H}_k) \leq \sum_{k=1}^{n} \sum_{l=1}^{q} \Delta(\mathbf{H}_{k,l-1}, \mathbf{H}_{k,l}) \leq qn \cdot (\nu_c + \nu_u + \nu_v).$$

□

Unfortunately, unique signatures do not satisfy EUF-CMA against the class of all SAs that satisfy relaxed verifiability (cf. Definition 25). In fact, it is not hard to show that no signature scheme with large enough message space (no matter if randomized or deterministic) can achieve EUF-CMA against such class of SAs.

This can be seen by looking again at the class of SAs $\mathcal{A}_{\mathsf{msg}} = \{\widetilde{\mathsf{A}}_{\bar{m}}\}_{\bar{m} \in \mathcal{M}}$ that behaves identically to the original signing algorithm, except that upon input $\bar{m} \in \mathcal{M}$ it outputs the secret key. Clearly, $\mathcal{A}_{\mathsf{msg}}$ satisfies relaxed verifiability as a randomly chosen message will be different from $\bar{m}$ with high probability and the user has only a negligible chance of hitting the value $\bar{m}$; yet $\mathcal{A}_{\mathsf{msg}}$ clearly allows to break EUF-CMA for an adversary knowing $\bar{m}$.

5.1.3 Security Against Random-Message Attacks

We show that if we restrict to the case of random-message attacks (RMA), i.e. the adversary can only see signatures of randomly chosen messages, unique signatures achieve unforgeability against the class of SAs that meets relaxed verifiability (cf. Definition 25).

Definition 26 (EUF-RMA against SAs)**.** *Let $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme, and $\mathcal{A}$ be some class of SAs for $\mathcal{SS}$. We say that $\mathcal{SS}$ is (t, n, q, ε)-EUF-RMA w.r.t.* continuous *$\mathcal{A}$-SAs if for all PPT adversaries B running in time t, we have $\mathbb{P}[\mathsf{B}$ wins$] \leq \varepsilon(\kappa)$ in the game of Definition 23 with the adaptation that the messages*

in the sets $\mathcal{Q}, \widetilde{\mathcal{Q}}_1, \ldots, \widetilde{\mathcal{Q}}_n$ *are drawn uniformly at random from the message space* $\mathcal{M}$*.*

While the above definition might seem a weak guarantee, it is still useful for applications. In particular, in Section 5.1.4 we show how to use any signature scheme that is EUF-RMA against a given class of SAs, to construct an identification scheme that is subversion-resilient against the same class of SAs.

Theorem 8. *Let* $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *be a* $(t, (q+1) \cdot n, \varepsilon)$*-EUF-CMA signature scheme with* ν_c*-correctness and* ν_u*-uniqueness, and denote by* $\mathcal{A}^{\nu_v}_{\mathsf{r\_ver}}$ *the class of all algorithms that satisfy* relaxed ν_v*-verifiability relative to* $\mathcal{SS}$*. Then* $\mathcal{SS}$ *is* (t', n, q, ε')*-indistinguishable against continuous* $\mathcal{A}^{\nu_v}_{\mathsf{rel\_ver}}$*-SAs, for* $t' \approx t$*, for all* $n, q \in \mathbb{N}$*, and for* $\varepsilon' \leq \varepsilon + qn \cdot (\nu_c + \nu_v + \nu_u)$*.*

Proof. Let $\mathbf{G}$ be the game of Definition 26. Consider the modified game $\mathbf{H}$ that is identical to $\mathbf{G}$ except that queries to the subverted signing algorithms are answered as described below:

- For all $i \in [q]$, $j \in [n]$, sample $\widetilde{m}_{i,j} \leftarrow\!\!\$\, \mathcal{M}$ and return $\sigma_{i,j} \leftarrow \mathsf{Sign}(SK, \widetilde{m}_{i,j})$.

Claim 8.1. $|\mathbb{P}[\mathsf{B} \text{ wins in } \mathbf{G}] - \mathbb{P}[\mathsf{B} \text{ wins in } \mathbf{H}]| \leq qn \cdot (\nu_c + \nu_v + \nu_u)$.

Proof. For an index $k \in [0, n]$, consider the hybrid game $\mathbf{H}_k$ that answers each query to the j-th subversion oracle, such that $j \leq k$, as in game $\mathbf{G}$, while all queries with $j > k$ are answered as in $\mathbf{H}$. We note that $\mathbf{H}_0 \equiv \mathbf{H}$ and $\mathbf{H}_n \equiv \mathbf{G}$. Abusing notation, let us write $\mathbf{H}_k$ for the distribution of the random variable corresponding to B's view in game $\mathbf{H}_k$.

We will show that $\Delta(\mathbf{H}_{k-1}, \mathbf{H}_k) \leq q \cdot (\nu_c + \nu_v + \nu_u)$ for all k. Fix a particular $k \in [0, n]$, and for an index $l \in [0, q]$ consider the hybrid game $\mathbf{H}_{k,l}$ that is identical to $\mathbf{H}_k$ except that it answers queries (k, i) with $i \leq l$ as in game $\mathbf{G}$, while all queries (k, i) with $i > l$ are treated as in $\mathbf{H}$. Observe that $\mathbf{H}_{k,0} \equiv \mathbf{H}_{k-1}$, and $\mathbf{H}_{k,q} \equiv \mathbf{H}_k$.

We now argue that for each $l \in [q]$, one has that $\mathbb{SD}(\mathbf{H}_{k,l-1}, \mathbf{H}_{k,l}) \leq \nu_c + \nu_v + \nu_u$. Notice that the only difference between $\mathbf{H}_{k,l-1}$ and $\mathbf{H}_{k,l}$ is how the two games answer the query (k, l): Game $\mathbf{H}_{k,l-1}$ returns $\sigma_{l,k} \leftarrow \mathsf{Sign}(SK, \widetilde{m}_{l,k})$, whereas game $\mathbf{H}_{k,l}$ returns $\widetilde{\sigma}_{l,k} \leftarrow \widetilde{\mathsf{A}}_k(SK, \widetilde{m}_{l,k})$ (where $\widetilde{m}_{l,k} \leftarrow\!\!\$\, \mathcal{M}$). Now let $E_{l,k}$ be the event that $\sigma_{l,k} \neq \widetilde{\sigma}_{l,k}$. We can write

$$\Delta(\mathbf{H}_{k,l-1}, \mathbf{H}_{k,l}) \leq \Delta(\mathbf{H}_{k,l-1}; \mathbf{H}_{k,l} | \neg E_{l,k}) + \mathbb{P}[E_{l,k}] \tag{5.3}$$

$$\leq \nu_c + \nu_u + \nu_v. \tag{5.4}$$

Eq. (5.3) follows by Lemma 1 and Eq. (5.4) follows by the fact that $\mathbf{H}_{k,l-1}$ and $\mathbf{H}_{k,l}$ are identically distributed conditioned on $E_{l,k}$ not happening, and moreover $\mathbb{P}[E_{l,k}] \leq \nu_c + \nu_u + \nu_v$. The latter can also be seen as follows. By the correctness condition of $\mathcal{SS}$ we have that $\sigma_{l,k}$ is valid for $\widetilde{m}_{l,k}$ under VK except with probability at most ν_c. By the assumption that $\widetilde{\mathsf{A}}_k \in \mathcal{A}^{\nu_v}_{\mathsf{r\_ver}}$ we have that $\widetilde{\sigma}_{l,k}$ is also valid for $\widetilde{m}_{l,k}$ under VK except with probability at most ν_v (this is because $\widetilde{m}_{l,k}$ is chosen at random). Finally, by the uniqueness property of $\mathcal{SS}$ we have that $\sigma_{l,k}$ and $\widetilde{\sigma}_{l,k}$ must be equal except with probability at most ν_u. It follows that $\mathbb{P}[E_{l,k}] \leq \nu_c + \nu_u + \nu_v$, as desired.

The claim now follows by the above argument and by the triangle inequality, as

$$\begin{aligned}\Delta(\mathbf{G},\mathbf{H}) &\leq \sum_{k=1}^{n} \Delta(\mathbf{H}_{k-1},\mathbf{H}_k) \\ &\leq \sum_{k=1}^{n}\sum_{l=1}^{q} \Delta(\mathbf{H}_{k,l-1},\mathbf{H}_{k,l}) \\ &\leq qn\cdot(\nu_c+\nu_v+\nu_u).\end{aligned}$$

□

Claim 8.2. $\mathbb{P}[\mathsf{B} \text{ wins in } \mathbf{H}] \leq \varepsilon$.

Proof. Towards a contradiction, assume B wins in game $\mathbf{H}$ with probability larger than $qn\cdot\varepsilon$. We build an adversary B' (using B) that breaks EUF-CMA of $\mathcal{SS}$. Adversary B' is described below.

Adversary B':

- Receive the verification key VK from the challenger, and return VK to B.
- Upon input the i-th signature query, query the target oracle receiving back a signature $\sigma_i \leftarrow \mathsf{Sign}(SK, m_i)$ for $m_i \leftarrow\!\!\$\ \mathcal{M}$. Return σ_i to B.
- Upon input a query of the form (j,i), query the target oracle receiving back a signature $\sigma_{i,j} \leftarrow \mathsf{Sign}(SK, \widetilde{m}_{i,j})$ for $\widetilde{m}_{i,j} \leftarrow\!\!\$\ \mathcal{M}$. Return $\sigma_{i,j}$ to B.
- Whenever B outputs (m^*,σ^*), output (m^*,σ^*).

For the analysis, note that B' runs in time similar to that of B and asks a total of at most $q+qn$ signing queries. Moreover, the simulation is perfect and thus $\mathbb{P}[\mathsf{B}' \text{ wins}] \geq \varepsilon$, a contradiction. □

The proof follows by combining the above two claims. □

5.1.4 Subversion-Resilient Identification Schemes

We show how to apply EUF-RMA against SAs to the setting of subversion-resilient identification (ID) schemes. Similar applications already appeared in the literature for leakage and tamper resistance [ADW09, FHN+12, DFMV13, NVZ14, FNV15].

In a public-key ID scheme a prover with secret key SK attempts to prove its identity to a verifier holding the corresponding verification key VK. More formally, an ID scheme $\mathcal{ID} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{P}, \mathsf{V})$ consists of four PPT algorithms described as follows: (1) The parameters generation algorithm takes as input the security parameter and outputs public parameters $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa)$, shared by all users.[7] (2) The key generation algorithm takes as input the security parameter and outputs a verification key/secret key pair $(VK, SK) \leftarrow \mathsf{KGen}(1^\kappa)$. (3) P and V are probabilistic

[7] In what follows all algorithms take as input params, but we omit to explicitly write this for ease of notation.

Turing machines interacting in a protocol; at the end of the execution V outputs a decision bit $d \in \{0,1\}$, where $d = 1$ means that the identification was successful. We write $\langle \mathsf{P}(SK), \mathsf{V}(VK) \rangle$ for the random variable corresponding to the verifier's verdict, and $\mathsf{P}(SK) \rightleftarrows \mathsf{V}(VK)$ for the random variable corresponding to transcripts of honest protocol executions.

We now define a variant of passive security, where in a first phase the adversary is allowed to subvert the prover algorithm; in a second phase the adversary has to impersonate the prover. Similarly to the case of signature schemes subversion is modelled by considering a class $\mathcal{A}$ of SAs, where each $\widetilde{\mathsf{A}} \in \mathcal{A}$ is an algorithm replacing the prover algorithm P within the ID scheme $\mathcal{ID}$.

Definition 27 (Subversion-Resilient Identification)**.** *Let $\mathcal{ID} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be an ID scheme, and $\mathcal{A}$ be some class of SAs for $\mathcal{ID}$. We say that $\mathcal{ID}$ is (t, n, q, ε)-secure w.r.t.* continuous *$\mathcal{A}$-SAs if for all PPT adversaries B running in time t, we have $\mathbb{P}[\mathsf{B}\ wins] \leq \varepsilon(\kappa)$ in the following game:*

1. *The challenger runs $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa)$, $(VK, SK) \leftarrow \mathsf{KGen}(1^\kappa)$, and forwards (params, VK) to B.*

2. *The adversary B can observe q transcripts $\mathsf{P}(SK) \rightleftarrows \mathsf{V}(VK)$ corresponding to honest protocol executions between the prover and the verifier.*

3. *For each $j \in [n]$, the adversary B can adaptively choose an algorithm $\widetilde{\mathsf{A}}_j \in \mathcal{A}$. For each algorithm, B can observe q transcripts $\widetilde{\mathsf{A}}_j(SK) \rightleftarrows \mathsf{V}(VK)$ corresponding to protocol executions between the subverted prover and the verifier.*

4. *The adversary B loses access to all oracles and plays the role of the prover in an execution with an honest verifier $d \leftarrow \langle \mathsf{B}(VK), \mathsf{V}(VK) \rangle$; we say that B wins if and only if $d = 1$.*

Consider the following standard construction (see, e.g., [BFGM01]) of an identification scheme $\mathcal{ID}$ from a signature scheme $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$.

- *Parameters generation.* Algorithm Setup samples the public parameters params for the signature schemes (if any).

- *Key Generation.* Algorithm KGen runs the key generation algorithm of the signature scheme, obtaining $(VK, SK) \leftarrow \mathsf{KGen}(1^\kappa)$.

- *Identification protocol.* The interaction $\mathsf{P}(SK) \rightleftarrows \mathsf{V}(VK)$ is depicted in Figure 5.1.

The theorem below states that the above protocol achieve subversion resilience w.r.t. a given class $\mathcal{A}$ of SAs, provided that the underlying signature scheme is EUF-RMA w.r.t. the same class $\mathcal{A}$.

Theorem 9. *Let $\mathcal{SS}$ be a signature scheme with message space $\mathcal{M}$, and let $\mathcal{A}$ be a class of SAs for $\mathcal{SS}$. Assume that $\mathcal{SS}$ is (t, n, q, ε)-EUF-RMA w.r.t. continuous $\mathcal{A}$-SAs. Then the ID scheme $\mathcal{ID}$ from Figure 5.1 is (t', n, q, ε')-secure w.r.t. continuous $\mathcal{A}$-SAs where $t' \approx t$ and $\varepsilon' \leq \varepsilon + \frac{(n+1)q}{|\mathcal{M}|}$.*

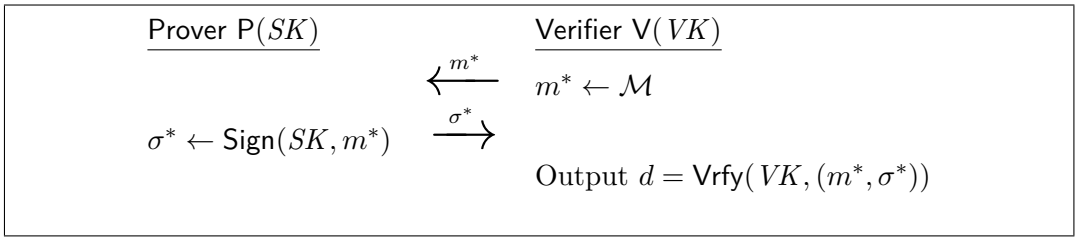

Figure 5.1. Two-round identification using a signature scheme $\mathcal{SS}$ with message space $\mathcal{M}$

Proof. For the sake of contradiction, assume that there exists an adversary B breaking security of the identification scheme with probability larger than $\varepsilon + \frac{(n+1)q}{|\mathcal{M}|}$. We construct a PPT adversary B' breaking EUF-RMA of $\mathcal{SS}$ with probability at least ε (a contradiction). Adversary B' runs in the game of Definition 26 and is described below. The main observation is that the prover's algorithm P is completely specified by algorithm Sign, and thus subverting the ID scheme is equivalent to subverting the signature scheme.

Adversary B':

1. Receive the public parameters params and the verification key VK for $\mathcal{SS}$ and forward (params, VK) to B.
2. Whenever B wants to observe a honest transcript $\mathsf{P}(SK) \rightleftarrows \mathsf{V}(VK)$, query the signing oracle obtaining a pair (m_i, σ_i) such that $\sigma_i \leftarrow \mathsf{Sign}(SK, m_i)$ and $m_i \leftarrow\!\!\$\, \mathcal{M}$. Forward (m_i, σ_i) to B.
3. Whenever B specifies an algorithm $\widetilde{\mathsf{A}}_j \in \mathcal{A}$, forward $\widetilde{\mathsf{A}}_j$ to the challenger. For each query of B to its own j-th oracle, query the target j-th oracle obtaining a pair $(\widetilde{m}_{i,j}, \widetilde{\sigma}_{i,j})$ such that $\widetilde{\sigma}_{i,j} \leftarrow \widetilde{\mathsf{A}}_j(SK, \widetilde{m}_{i,j})$ and $\widetilde{m}_{i,j} \leftarrow\!\!\$\, \mathcal{M}$. Forward $(\widetilde{m}_{i,j}, \widetilde{\sigma}_{i,j})$ to B.
4. Finally, when B is ready to start the impersonation phase, sample a random message $m^* \leftarrow\!\!\$\, \mathcal{M}$ and send it to B. Upon receiving a value σ^* from B output (m^*, σ^*) as forgery.

It is easy to see that B''s simulation of B's queries is perfect; moreover, since the message m^* in the impersonation stage is chosen at random from $\mathcal{M}$, also the simulation of this phase has the right distribution and in particular the forgery (m^*, σ^*) will be valid with probability ε.

It remains to compute the probability that B' is successful. Observe that B' is successful whenever (m^*, σ^*) is valid and $m^* \notin \mathcal{Q} \cup \widetilde{\mathcal{Q}}$. Also, note that m^* is independent from $\widetilde{\mathcal{Q}}$, so in particular

$$\mathbb{P}\left[m^* \in \mathcal{Q} \cup \widetilde{\mathcal{Q}}\right] \leq \frac{|\mathcal{Q}| + |\widetilde{\mathcal{Q}}|}{|\mathcal{M}|} = \frac{(n+1)q}{|\mathcal{M}|}.$$

Let E be the event that $m^* \notin \mathcal{Q} \cup \widetilde{\mathcal{Q}}$. We have,

$$\begin{aligned}\mathbb{P}\left[\mathsf{B}' \text{ wins}\right] &\geq \mathbb{P}\left[\mathsf{B} \text{ wins} \wedge E\right] \geq \mathbb{P}\left[\mathsf{B} \text{ wins}\right] - \mathbb{P}\left[\neg E\right] \\ &\geq \mathbb{P}\left[\mathsf{B} \text{ wins}\right] - \frac{(n+1)q}{|\mathcal{M}|} \\ &> \varepsilon,\end{aligned}$$

where the last inequality follows by our initial assumption on B's advantage. This concludes the proof. □

5.1.5 Impersonation in the Multi-User Setting

In this section we consider the multi-user setting for the impersonation definition. We also provide a complete picture of the relationships between the definition, as shown in Fig. 5.2.

Multi-User Impersonation

Analogous to the single-user setting, we consider two security definitions corresponding to different adversarial goals.

In the indistinguishability definition for the multi-user setting adversary B now receives $u \geq 1$ key pairs from the challenger and can continuously subvert each user independently. A formal definition follows.

Definition 28 (Indistinguishability against SAs—Multi-User)**.** *Let $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme, and $\mathcal{A}$ be some class of SAs for $\mathcal{SS}$. We say that $\mathcal{SS}$ is u-users indistinguishable w.r.t* continuous *$\mathcal{A}$-SAs if for all PPT adversaries B there exists a negligible function $\nu : \mathbb{N} \to [0,1]$, such that $\left|\mathbb{P}[\mathsf{B} \text{ wins}] - \frac{1}{2}\right| \leq \nu(\kappa)$ in the following game:*

1. *The challenger samples $b \leftarrow\!\!\$\, \{0,1\}$, generates $(VK_i, SK_i) \leftarrow \mathsf{KGen}(1^\kappa)$ for $i \in [u]$ and gives $VK_1, \ldots, VK_u$ to B.*
2. *The adversary B can specify polynomially many queries (adaptively and in an arbitrary order) of the form $(i, \widetilde{\mathsf{A}})$ for $i \in [u]$.*
   (a) *For each such query, B is given access to an oracle that can be queried polynomially many times on inputs $m \in \mathcal{M}$.*
   (b) *The answer to each query m depends on the value of the secret bit b. In particular, if $b = 1$, the output is $\sigma \leftarrow \mathsf{Sign}(SK_i, m)$; if $b = 0$, the output is $\widetilde{\sigma} \leftarrow \widetilde{\mathsf{A}}(SK_i, m)$.*
3. *Finally, B outputs a value $b' \in \{0,1\}$; we say that B wins iff $b' = b$.*

In the impersonation definition for the multi-user setting adversary B now receives $u \geq 1$ key pairs from the challenger and can continuously subvert each user independently; adversary B is successful if it can impersonate *any* of the users. A formal definition follows.

Definition 29 (EUF-CMA against SAs—Multi-User)**.** *Let $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme, and $\mathcal{A}$ be some class of SAs for $\mathcal{SS}$. We say that $\mathcal{SS}$ is u-users EUF-CMA w.r.t.* continuous *$\mathcal{A}$-SAs if for all PPT adversaries B there exists a negligible function $\nu : \mathbb{N} \to [0,1]$, such that $\mathbb{P}[\mathsf{B} \text{ wins}] \leq \nu(\kappa)$ in the following game:*

1. *The challenger generates $(VK_i, SK_i) \leftarrow \mathsf{KGen}(1^\kappa)$ for $i \in [u]$ and gives $VK_1, \ldots, VK_u$ to B.*

2. *Adversary* B *is given oracle access to* $\mathsf{Sign}(SK_i, \cdot)$. *Upon input query* $m \in \mathcal{M}$, *this oracle returns* $\sigma \leftarrow \mathsf{Sign}(SK_i, m)$; *let* $\mathcal{Q}_i$ *be the set of all messages queried to oracle* $\mathsf{Sign}(sk_i, m)$.

3. *The adversary* B *can specify polynomially many queries (adaptively and in an arbitrary order) of the form* $(i, \widetilde{\mathsf{A}}_j)$ *for* $i \in [u]$ *and* $j \in poly(\kappa)$.

   (a) *For each such query,* B *is given access to an oracle that can be queried polynomially many times upon inputs* $m \in \mathcal{M}$.

   (b) *The answer to each query* m *is* $\widetilde{\sigma} \leftarrow \widetilde{\mathsf{A}}_j(SK_i, m)$; *let* $\widetilde{\mathcal{Q}}_{i,j}$ *be the set containing all queried messages to oracle* $\widetilde{\mathsf{A}}_j(SK_i, m)$.

4. *Finally,* B *outputs a tuple* (m^*, σ^*, i^*); *we say that* B *wins iff* $\mathsf{Vrfy}(VK_{i^*}, (m^*, \sigma^*)) = 1$ *and* $m^* \notin \mathcal{Q}_{i^*} \bigcup_{j=1}^{poly(\kappa)} \widetilde{\mathcal{Q}}_{i^*,j}$.

Impersonation Relations

Theorem 10 formalizes the relations depicted in Fig. 5.2.

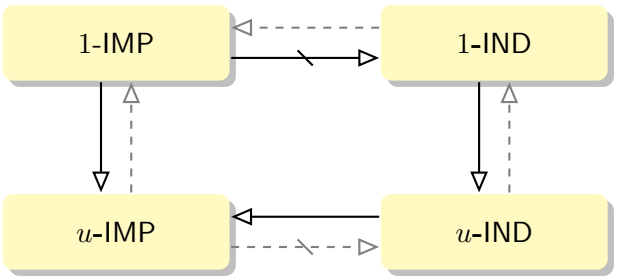

Figure 5.2. Diagram of the relationships between the subversion notions considered in this paper. $X \to Y$ means that X implies Y; $X \nrightarrow Y$ indicates a separation between X and Y. The lighter arrows indicates trivial implications (or implications that follow from Theorem 10). Definition 28 is represented by u-IND and Definition 29 is represented by u-IMP.

Theorem 10. *Let* $\mathcal{SS}$ *be a signature scheme. Then the following relations hold.*

(i) For all SA classes $\mathcal{A}$ *against* $\mathcal{SS}$, *if* $\mathcal{A}$ *is* 1-IND *then* $\mathcal{A}$ *is also* u-IND.

Proof. Towards contradiction, consider an adversary B that wins the game described in Definition 28. We build an adversary B' that (using B) wins the game described in Definition 22. Let $\mathbf{G}$ be the game described in Definition 28. Consider the game $\mathbf{G}_0$, an identical copy of game $\mathbf{G}$ when $b = 0$, and consider the game $\mathbf{G}_1$ an identical copy of game $\mathbf{G}$ when $b = 1$.

For an index $k \in [0, u]$, consider the hybrid game $\mathbf{H}_k$ where each oracle corresponding to query $(i, \widetilde{\mathsf{A}})$ such that $i \leq k$ behaves as $\widetilde{\mathsf{A}}(SK_i, \cdot)$ (i.e., as in game $\mathbf{G}_0$), while all oracles corresponding to queries $(i, \widetilde{\mathsf{A}})$ such that $i > k$ behave as $\mathsf{Sign}(SK_i, \cdot)$ (i.e., as in game $\mathbf{G}_1$). We note that $\mathbf{H}_0 \equiv \mathbf{G}_1$ and $\mathbf{H}_u \equiv \mathbf{G}_0$. By assumption, we know that B can distinguish between the extreme hybrid games $\mathbf{H}_0$ and $\mathbf{H}_u$. So there must exist a pair of hybrids $\mathbf{H}_i$,

$\mathbf{H}_{i-1}$ that B can distinguish with a non-negligible advantage. We can construct B' as follows.

Adversary B':

(a) Receive VK^* from the challenger and sample $(VK_j, SK_j) \leftarrow \mathsf{KGen}(1^\kappa)$ for all $j \in [u] \setminus \{i\}$. Define $VK_i = VK^*$ and forward $(VK_1, \ldots, VK_u)$ to adversary B.

(b) Upon input a query $(j, \widetilde{\mathsf{A}})$ from B, behave as follows.
- If $j \leq i-1$ answer all queries $m \in \mathcal{M}$ as $\widetilde{\sigma} \leftarrow \widetilde{\mathsf{A}}(SK_j, m)$;
- if $j = i$ forward all queries $m \in \mathcal{M}$ to the challenger;
- if $j \geq i$ answer all queries $m \in \mathcal{M}$ as $\sigma \leftarrow \mathsf{Sign}(SK_j, m)$.

(c) Output whatever B outputs.

We observe that adversary B' simulates perfectly the distribution of the games $\mathbf{H}_{i-1}$ (when $b = 0$) and $\mathbf{H}_i$ (when $b = 1$). Since adversary B can distinguish this pair of hybrids with non-negligible probability it follows that adversary B' wins in the single-user game with the same probability. □

(ii) There exists a SA class $\mathcal{A}$ against $\mathcal{SS}$ that is 1-IMP *but it is not* 1-IND.

Proof. Consider $\mathcal{SS}$ to be an EUF-CMA signature scheme with signature size ℓ bits, and let $\mathcal{A}$ be the class of SAs for $\mathcal{SS}$ that always outputs 0^ℓ as the signature of any message $m \in \mathcal{M}$. By $\mathcal{SS}$ being EUF-CMA, adversary B has only a negligible probability of winning at the game described in Definition 23. Consider the adversary B against the game described in Definition 22.

Adversary B:

(a) The challenger samples $b \leftarrow \{0,1\}$, runs $(VK, SK) \leftarrow \mathsf{KGen}(1^\kappa)$ and forwards VK to B.

(b) B queries the oracle for an arbitrary message m and receives σ as a reply.

(c) If $\sigma = 0^\ell$ then output 0, otherwise output 1.

Adversary B clearly has a non-negligible probability of distinguishing the real signing oracle from the subversion oracle in the game of Definition 22. □

(iii) Let $\mathcal{SS}$ be EUF-CMA. For all SA classes $\mathcal{A}$ against $\mathcal{SS}$, if $\mathcal{A}$ is u-IND *then $\mathcal{A}$ is also* u-IMP.

Proof. Let $\mathcal{A}$ be a class of SAs for $\mathcal{SS}$. The objective here is to show that if $\mathcal{A}$ is u-users indistinguishable w.r.t continuous SAs (Definition 28) then $\mathcal{A}$ is also u-users EUF-CMA w.r.t continuous SA (Definition 29). We sketch a proof by considering a modified game for Definition 29, where all oracles behave like the real signing oracle (one oracle for each signing key). Since the class $\mathcal{A}$ is u-users indistinguishable we get that the advantage of any adversary in winning the game of Definition 29 is negligibly close to the advantage of winning in the modified game. However, since $\mathcal{SS}$ is EUF-CMA no PPT adversary can win the modified game with non-negligible advantage, and so $\mathcal{SS}$ satisfies u-IMP. □

(iv) For all SA classes $\mathcal{A}$ against $\mathcal{SS}$, if $\mathcal{A}$ is 1-IMP *then $\mathcal{A}$ is also* u-IMP.

Proof. Consider an adversary B that wins the game described in Definition 29. We build an adversary B' that (using B) wins the game described in Definition 23.

Adversary B':

(a) Receive VK^* from the challenger, sample $i^* \leftarrow\!\!\$\ \{1, \ldots, u\}$ and $(VK_i, SK_i) \leftarrow \mathsf{KGen}(1^\kappa)$ for all $i \in [u] \setminus \{i^*\}$. Set $VK_{i^*} := VK^*$ and forward $(VK_1, \ldots, VK_u)$ to adversary B.

(b) Upon each query (i, m), for $i \in [u]$ and $m \in \mathcal{M}$: If $i \neq i^*$ reply with $\sigma = \mathsf{Sign}(sk_i, m)$, else forward the query to the challenger.

(c) Upon each query $(i, \widetilde{\mathsf{A}})$, with $i \in [u]$, behave as follows.

- For each $m \in \mathcal{M}$ chosen by the adversary B, if $i \neq i^*$ answer with $\widetilde{\sigma} = \widetilde{\mathsf{A}}(SK_j, m)$, else forward the query to the challenger.

(d) Eventually B outputs a forgery (i', m', σ'); adversary B' outputs (m', σ') as its own forgery.

Adversary B' is successful if adversary B outputs a forgery for user i^*. Define E, to be the event that B' guesses correctly the index $i' = i^*$; note that $\mathbb{P}[E] = 1/u$. Therefore adversary B' has a non-negligible probability of winning at the game described in Definition 23. □

5.1.6 Unique Signature Scheme Based on QR

We showed in Section 5.1 that unique signatures are the only type of signature scheme that are secure against undetectable subversion attacks[8]. The good news is that secure unique signature schemes already exist and are quite efficient. The bad news is that the most efficient ones have their security based on non-standard computational assumptions, such as Φ-Hiding. In this Section we focus on efficient unique signature schemes and we present a new unique signature scheme with a tight security proof based on quadratic residuosity. We note that this is the first such scheme with a tight security proof based on a standard and well studied assumption.

The most efficient (regarding signature verification time) unique signature schemes known are the ones that the signature is computed on the output of the hash of the message to be signed, which are called Full Domain Hash (FDH) schemes, such as RSA-FDH [BR93] and Principal Rabin-Williams (PRW) [Ber08]. All such schemes are only secure in the Random Oracle Model (ROM) (that usually are much more practical than schemes secure in the standard model [GHR99, CS00, BB04, Wat05, BSW06, CJ07]). Since we focus on efficient schemes, we consider only those secure in the ROM.

[8] To be more precise, we show that unique signatures are secure against all subversion attacks that satisfies the verifiability condition.

What is wrong with the Φ-Hiding assumption? The Φ-Hiding assumption appears to be much stronger than factoring and it does not even hold in some cases as shown in [Her11, SF08]. The fastest scheme to date, i.e., PRW [Ber08], is tightly secure under a new assumption dubbed 2-Φ/4-Hiding assumption [Seu14]. As reported by Seurin [Seu14], the 2-Φ/4-Hiding assumption is clearly stronger than quadratic residuosity on which our scheme relies instead: When $N \equiv 1 \mod 4$, the 2-Φ/4-Hiding problem is equivalent to the problem of establishing whether -1 is a square in $\mathbb{Z}_N^*$; thus, it's enough to provide $y = -x^2 \mod N$, for a random $x \in \mathbb{Z}_N^*$, to a quadratic residuosity solver to violate the 2-Φ/4-Hiding assumption.

In Table 5.1 we explore the efficient unique signature schemes and their security reductions based on its undelying hardness assumptions.

Unique Signature Schemes	**Hardness Assumption**	
	Loose Reduction	*Tight Reduction*
Principal RW [Ber08]	Factoring	2-Φ/4-Hiding
RSA-FDH [BR93]	RSA	Φ-Hiding
BLS [BLS04]	EC-CDH	No reduction
Our Scheme	Factoring	Quadratic Residuosity

Table 5.1. Unique signature schemes and their hardness assumptions.

State of Affairs. A seminal impossibility result by Coron [Cor02] states that any FDH signature scheme with unique signatures could not hope to have a tight security proof. At Eurocrypt 2012, Kakvi et al. [KK12] clarified that Coron's impossibility result only holds when the trapdoor permutation is *certified*. They also presented a tight security proof for RSA-FDH based on the Φ-Hiding assumption [CMS99].

Bernstein [Ber08] studied all variants of Rabin-Williams signatures and devised an ingenious tight proof for Rabin-Williams with derandomized output ("fixed unstructured"), where it releases systematically one of the four square roots which is initially selected at random. Bernstein provides also a loose proof for PRW and left as an open problem finding a tight proof for it. Seurin [Seu14] first showed that a variant of the Rabin function is lossy and then presented a tight security proof for PRW, but under a new assumption dubbed 2-Φ/4-Hiding assumption.

Thus, to summarize: There is no known *tight* security proof for any unique signature FDH scheme from the assumption that the underlying trapdoor permutation is one-way. All known tight proofs rely on the lossiness of the trapdoor permutation and are based on variants of the Φ-Hiding assumption. Seurin (cf. Theorem 5 in [Seu14]) noted that it is very unlikely that RSA and Rabin-Williams FDH signatures will have a tight security reduction from, respectively, inverting RSA or factoring.

It is evident that the state of affairs above is a bit confusing. RSA and Rabin-Williams FDH signatures with loose proofs were criticized as being potentially impractical due to the large size of the parameters involved. Their tight proofs however rely on new assumptions that appear to be markedly stronger than factoring [Her11, SF08]. How should these results be interpreted in practice? Should we trust these new assumptions and keep parameters short or should we use large parameters to account for possible cryptanalytic attacks on these new assumptions?

Arguably the next best assumption after factoring is quadratic residuosity which has been extensively studied, at least as much as the RSA assumption. Therefore we depart from the current approaches and look for unique signature FDH schemes with a tight security reduction directly from quadratic residuosity. Admittedly this can be achieved with little effort: It's enough to apply the transformation of Kakvi and Kiltz [KK12] to a lossy function based on quadratic residuosity from [FGK+13]. As far as we could ascertain, this is the first unique signature FDH scheme tightly secure under the quadratic residuosity assumption (and loosely secure under factoring). In addition, the reduction is tighter than the one in [Seu14], i.e., our scheme is closer to quadratic residuosity than Rabin-Williams with unique signatures is to the 2-Φ/4-Hiding assumption. The efficiency of our scheme is comparable to that of schemes in the Rabin-Williams family which are considered the fastest signature verification schemes ever devised [Ber08].

The scheme does require the computation of a Jacobi symbol but we believe such a computation carries an unfair stigma. In reality, computing Jacobi symbols can be performed very efficiently [SS93, ES98] (in particular in $O(n^2/\log n)$ as reported in [ES98]), and can be parallelized [ES98] to harness recent multicore and/or distributed platforms.

The Signature Scheme

In this section we describe the signature scheme. The scheme is derived from the lossy function of [FGK+13]. The parameters are described below.

- Security parameter κ (e.g. $\kappa = 2048$).
- $N = p \cdot q$ s.t. p, q are primes, $p \equiv q \equiv 3 \bmod 4$ and $|p| \approx |q| \approx \kappa/2$.
- Collision resistant hash function $H : \{0,1\}^* \to \mathbb{Z}_N^*$.

We define the following functions $h, j : \mathbb{Z}_N \to \{0,1\}$,

$$h(x) = \begin{cases} 1, \text{ if } x > N/2 \\ 0, \text{ otherwise} \end{cases} \tag{5.5}$$

$$j(x) = \begin{cases} 1, \text{ if } \mathbf{J}_N(x) = -1 \\ 0, \text{ otherwise} \end{cases} \tag{5.6}$$

Lemma 2 shows that the four square roots of some $y \in \mathbb{QR}_N$ take all the values of $j(x), h(x)$. We formally define the algorithms KGen (Algorithm 3), Sign (Algorithm 4) and Vrfy (Algorithm 5) in this Section.

In Algorithm 4 we need to make sure that y' is a quadratic residue before we try to compute its four square roots. Note that $j(x) = 1$ *if and only if* $\mathbf{J}_N(x) = -1$ and since $\mathbf{J}_N(r) = -1$ (by definition) it is guaranteed that $\mathbf{J}_N(y) = 1$. The next step is to determine if y is a quadratic residue or not, which is easy to compute knowing the private key. If y is not a quadratic residue, we compute $y' = y \cdot s$ to get a quadratic residue, otherwise when y already is a quadratic residue we just set $y' = y$.

Algorithm 3: (KGen) Key generation algorithm

input : Security parameter κ
output : Matching public and private keys (PK, SK)

Choose random primes p, q s.t. $p \equiv q \equiv 3 \mod 4$;
$N = pq$;
$r \leftarrow \mathbb{Z}_N^*$ s.t. $\mathbf{J}_N(r) = -1$;
$s \leftarrow \mathbb{Z}_N^*$ s.t. $\mathbf{J}_N(s) = 1$ and $s \notin \mathbb{QR}_N$;
$PK \leftarrow (N, r, s)$; $SK \leftarrow (p, q)$;
return (PK, SK)

Algorithm 4: (Sign) Signing algorithm

input : Message m, key pair (pk, sk)
output : Signature σ of message m under the key pair (pk, sk)

$b \leftarrow 0$;
$x \leftarrow H(m)$;
$y \leftarrow x \cdot r^{j(x)}$;
if $(y \notin \mathbb{QR}_N)$ **then** $b \leftarrow 1$;
$y' \leftarrow y \cdot s^b$
Let $\{y'_1, y'_2, y'_3, y'_4\}$ be the four square roots of y';
$\sigma \leftarrow y'_i$ s.t. $j(y'_i) = j(x)$ and $h(y'_i) = b$;
return σ

As the signature of message m, we choose the only square root (Lemma 2) of y' that agrees to the values of $j(x)$ and the bit b (e.g., if $\mathbf{J}_n(x) = -1$ and $b = 1$ we choose the only y'_i such that $\mathbf{J}_n(y'_i) = -1$ and $y'_i > N/2$).

To check the correctness of Algorithm 5 we note that $\sigma^2 = y'$. To obtain x it is enough to compute $x = y' \cdot r^{-j(\sigma)} \cdot s^{-h(\sigma)}$ to "cancel out" the values r and s. Now it suffices to compute $H(m)$ and compare it to x to check the validity of the signature.

Algorithm 5: (Vrfy) Signature verification algorithm

input : Candidate signature σ, message m, public key pk
output : 1 if signature is accepted, 0 otherwise

if $\sigma \notin \{1, ..., N-1\}$ **return** 0;
if $H(m) = \sigma^2 \cdot r^{-j(\sigma)} \cdot s^{-h(\sigma)}$ **then**
  return 1;
else
  return 0;

Security Analysis

In this section we analyse the security of the proposed signature scheme. We first present a loose security proof based on the hardness of factoring, and later we show

that our signature scheme has a tight security proof based on the QR assumption. To achieve the latter we use the result of [KK12].

Loose Security Proof Based on Factoring.

Theorem 11. *Assuming that the Integer Factorization Problem (IFP) is (t, ε)-secure, our scheme is $(t', q_h, q_s, \varepsilon')$-secure, with:*

$$t = t' + (q_h + q_s + 1) \cdot O(\kappa^2)$$

$$\varepsilon = \frac{\varepsilon'}{2 \cdot (q_h + q_s + 1)}$$

Proof. Let A be an adversary that $(t', q_h, q_s, \varepsilon')$-breaks our signature scheme. We build an inverter I that (t, ε)-breaks the IFP. This construction is standard and is based on the one from [BR96].

The inverter I receives as input the public key N, and its objective is to factor N. I generates the parameters r, s according to the definition of the scheme. We allow the adversary A to make two types of oracle queries, namely hash queries and sign queries. The inverter I must answer the queries itself, with the same distribution as a real oracle would. Inverter I chooses a random $j \in \{1, ..., q\}$, where $q = q_h + q_s$. We now describe how I answers the two types of queries.

- Hash queries: When A makes a hash query, I increments a counter i that is initially set to zero, and set $m_i = m$. If $i = j$, what happens with probability $1/q$, I chooses a random $y \in \mathbb{Z}_N^*$ and random values $l, z \in \{0, -1\}$ and sets $y_i = y^2 \cdot r^l \cdot s^z$ and return y_i, otherwise it chooses a random $\sigma_i \in \mathbb{Z}_N^*$, sets $y_i = f(\sigma_i)$ and return y_i.

- Sign queries: When A makes a sign query I verifies if there is a m_i in the list such that $m_i = m$. If not, I simply makes the corresponding hash query and replies with σ_i that corresponds to the message m_i.

The adversary A eventually outputs a forgery (m, x). We assume that $m = m_i$ for some i. If $i = j$ we have that both $x' = x \cdot r^{-j(\sigma)} \cdot s^{-h(\sigma)}$ and y are square roots of y^2, so with probability $1/2$, x' and y are different and we can factor N by the computing $gcd(N, x - y)$. The running time for I is the running time of adversary A plus all the oracle queries. □

The above proof is a loose security proof based on integer factorization. We note that the above security reduction can be further improved by applying the results of Coron [Cor00], where we can achieve roughly $\varepsilon = \frac{\varepsilon'}{q_s}$.

In the next paragraph we present a tight security proof based on the Quadratic Residuosity assumption (QR).

Tight Security Proof Based on Quadratic Residuosity. Our signature scheme is inspired by a lossy function described in [FGK+13]. Since the function is lossy, we can use the result of [KK12] to prove a tight security for our scheme based on its lossiness.

In the following description of the lossy function we use the functions j and h described in Section 5.1.6. We define the algorithms KGen, lossyKGen, Eval, Invert below:

- KGen: On input κ the algorithm chooses an κ bit modulus $N = pq$, such that $p \equiv q \equiv 3 \mod 4$ are $\kappa/2$ random prime numbers. Then it chooses $r \in \mathbb{Z}_N^*$ such that $\mathbf{J}_N(r) = -1$, and a random $s \in \mathbb{Z}_N^*$ such that $\mathbf{J}_N(s) = 1$ and s is *not* a quadratic residue. The function index is $i = (N, r, s)$, the trapdoor is (p, q) and the function f_i is defined on the domain $D_i = \{1, ..., N-1\}$.
- lossyKGen: On input κ the algorithm chooses an κ bit modulus $N = pq$, such that $p \equiv q \equiv 3 \mod 4$ are $\kappa/2$ random prime numbers. Then it chooses $r \in \mathbb{Z}_N^*$ such that $\mathbf{J}_N(r) = -1$, and a random $s \in \mathbb{Z}_N^*$ such that $\mathbf{J}_N(s) = 1$ and s is a quadratic residue. The function index is $i = (N, r, s)$, and the function f_i is defined on the domain $D_i = \{1, ..., N-1\}$.
- Eval: Given a function index $i = (N, r, s)$ and a $x \in D_i$, the algorithm outputs:

$$f_{N,r,s}(x) = x^2 \cdot r^{j(x)} \cdot s^{h(x)} \mod N$$

- Invert: Given an index of an injective function $i = (N, r, s)$, its trapdoor (p, q), and a value $y = f_{N,r,s}(x)$, the algorithm Invert retrieves x as follows.
  - Find $j(x)$ by computing $\mathbf{J}_N(y)$ and set $y' = y \cdot r^{-j(x)}$.
  - Find $h(x)$ by checking if y' is a quadratic residue mod N and set $y'' = y' \cdot s^{-h(x)}$.
  - Find the four square roots of y'' and return the only one that agrees with $j(x)$ and $h(x)$.

We point out that the aforementioned function is a permutation over the domain $\{1, ..., N-1\}$. The injective KGen and the lossy lossyKGen functions only differ in the s parameter, that in both cases have Jacobi symbol 1 but it is not a quadratic residue in the injective function and it is a quadratic residue in the lossy function. It is easy to see that an algorithm that is able to distinguish between the injective and the lossy function must solve the QR problem.

If we assume that the Quadratic Residuosity assumption is $(t_{\mathbb{QR}}, \varepsilon_{\mathbb{QR}})$-hard, then this function is regular $(2, t_{\mathbb{QR}}, \varepsilon_{\mathbb{QR}})$-lossy, meaning that the lossy function is 2-to-1, i.e. have two pre-images for each image.

The security of the signature scheme now follows from lossiness of the function.

Theorem 12 (Theorem 8 in [KK12]). *Let $\mathcal{TDP} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Invert})$ be a regular (l, t', ε')-lossy trapdoor function. Then for any q_h, q_s, the $\mathcal{TDP}$-FDH signature scheme is $(t, q_h, q_s, \varepsilon)$-EUF-CMA secure in the random oracle model with:*

$$t = t' - q_h \cdot \mathcal{TDP}$$

$$\varepsilon = \left(\frac{2l-1}{l-1}\right) \cdot \varepsilon'$$

where $\mathcal{TDP}$ is the time to compute the Eval *procedure of $\mathcal{TDP}$.*

Theorem 13. *Let the Quadratic Residuosity assumption be $(t_{\mathbb{QR}}, \varepsilon_{\mathbb{QR}})$-hard. Then for any q_h, q_s, our signature scheme is $(t, q_h, q_s, \varepsilon)$-EUF-CMA secure in the random oracle model with:*

$$t = t_{\mathbb{QR}} - q_h \cdot \mathcal{O}(\kappa^2)$$

$$\varepsilon = 3 \cdot \varepsilon_{\mathbb{QR}}$$

Proof. The proof immediately follows from the results of [KK12], therefore we omit it here. □

Performance

We assume p and q are safe primes, i.e., $p = 2p' + 1$ and $q = 2q' + 1$. When the factors p and q are known, calculating the Jacobi symbol of an element is very efficient since it is enough to compute two Lagrange symbols. In particular, for $x \in \mathbb{Z}_N^*$, $\mathbf{J}_N(x) = 1$ if $x^{p'} \bmod p = x^{q'} \bmod q = 1$, otherwise $\mathbf{J}_N(x) = -1$[9].

The signature σ is the unique square root y_i' of the square y' such that $j(y_i') = j(x)$ and $y_i' > N/2$ (or $y_i' \leq N/2$). Computing such a square root is very efficient thanks to the Chinese remainder theorem. It is enough to compute integers a, b such that $ap + bq = 1$ (these can be precomputed and stored once for all) and then $s_p = (y')^{(p'+1)/2} \bmod p$ and $s_q = (y')^{(q'+1)/2} \bmod q$. Suppose that the selected square root is the principal one, then the signature σ is simply equal to $(b \cdot q \cdot s_p + a \cdot p \cdot s_q) \bmod N$.

In general, when p and q are known, the computations of the Jacobi symbol and a square root share several calculations and can be optimized when performed simultaneously [Ber03]. Computing Jacobi symbols when p and q are unknown is computationally more expensive than other modular operations and should be avoided if possible.

5.2 Reverse Firewalls for Signatures

In Section 5.1 we have shown that unique signatures are secure against a restricted class of SAs, namely all SAs that meet the so-called verifiability condition. As discussed in Section 1, by removing the latter requirement (i.e., allowing for arbitrary classes of SAs in Definition 22 and 23) would require that a signature scheme $\mathcal{SS}$ remains unforgeable even against an adversary allowed *arbitrary tampering with the computation* performed by the signing algorithm. This is impossible without making further assumptions.

In this section we explore to what extent one can model signature schemes secure against arbitrary tampering with the computation, by making the extra assumption of an un-tamperable cryptographic reverse firewall (RF) [MS15]. Roughly, a RF for a signature scheme is a (possibly stateful) algorithm[10] that takes as input a message/signature pair and outputs an updated signature; importantly the firewall

[9] We do not consider cases where the Jacobi or Legendre symbols are 0 since they happen with negligible probability.

[10] The reverse firewall could be deployed in hardware or in software, as long as its requirements are met.

has to do so using only public information (in particular, without knowing the signing key). A formal definition follows.

Definition 30 (RF for signatures)**.** *Let $\mathcal{SS}$ be a signature scheme. A RF for $\mathcal{SS}$ is a pair of algorithms $\mathcal{FW} = (\mathsf{Setup}, \mathsf{Patch})$ specified as follows: (i) Setup takes as input the security parameter and a verification key $VK \in \mathcal{VK}$, and outputs some initial (public) state $\delta \in \{0,1\}^*$; (ii) Patch takes as input the current (public) state δ, and a message/signature pair (m, σ) and outputs a possibly modified signature or a special symbol $\perp$ and an updated (public) state δ'. We write this as $\sigma' \leftarrow \mathsf{Patch}_\delta(m, \sigma)$ (and omit the updated state δ' as an explicit output).*

We will typically assume that the current state δ_{cur} of the RF, can be computed efficiently given just the verification key VK, the initial state δ and the entire history of all inputs to the RF.

5.2.1 Properties

Below, we discuss the correctness and security requirements of cryptographic RF $\mathcal{FW}$ for a signature scheme $\mathcal{SS}$.

Maintaining functionality. The first basic property of a RF is that it should preserve the functionality of the underlying signature scheme, i.e. if a signature σ on a message m is computed using signing key SK, and the firewall is initialized with the corresponding verification key VK, the patched signatures σ' should (almost always) be a valid signatures for m under VK. More precisely, we say that $\mathcal{FW}$ is *functionality maintaining* for $\mathcal{SS}$, if for any polynomial $p(\kappa)$ and any vector of inputs $(m_1, \ldots, m_p) \in \mathcal{M}$, the following quantity is negligible in the security parameter

$$\mathbb{P}\left[\begin{array}{c} \exists i \in [p] \text{ s.t.} \\ \mathsf{Vrfy}(VK, (m_i, \sigma_i')) = 0 \end{array} : \begin{array}{c} (VK, SK) \leftarrow \mathsf{KGen}(1^\kappa), \delta \leftarrow \mathsf{Setup}(VK, 1^\kappa) \\ \sigma_1 \leftarrow \mathsf{Sign}(SK, m_1), \ldots, \sigma_p \leftarrow \mathsf{Sign}(SK, m_p) \\ \sigma_1' \leftarrow \mathsf{Patch}_\delta(m_1, \sigma_1), \ldots, \sigma_p' \leftarrow \mathsf{Patch}_\delta(m_p, \sigma_p) \end{array}\right],$$

where the probability is taken over the coin tosses of all involved algorithms. Recall that each invocation of algorithm Patch updates the (public) state δ of the RF.

Preserving Unforgeability. The second property of a RF is a security requirement. Note that a firewall can never "create" security (as it does not know the signing key). Below we define what it means for a RF to *preserve* unforgeability of a signature scheme against *arbitrary* tampering attacks.

Definition 31 (Unforgeability preserving RF)**.** *Let $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme with RF $\mathcal{FW} = (\mathsf{Setup}, \mathsf{Patch})$. We say that $\mathcal{FW}$ (t, n, q, ε)-preserves unforgeability for $\mathcal{SS}$ against continuous SAs if for all adversaries B running in time t we have that $\mathbb{P}[\mathsf{B} \text{ wins}] \leq \varepsilon$ in the following game:*

1. *The challenger runs $(VK, SK) \leftarrow \mathsf{KGen}(1^\kappa)$, $\delta \leftarrow \mathsf{Setup}(VK, 1^\kappa)$, and gives (VK, δ) to B.*

2. *The adversary* B *is given oracle access to* $\mathsf{Sign}(SK, \cdot)$. *Upon input the* i*-th query* m_i*, this oracle returns* $\sigma_i \leftarrow \mathsf{Sign}(SK, m_i)$. *Let* $\mathcal{Q} = \{m_1, \ldots, m_q\}$ *be the set of all signature queries.*

3. *The adversary* B *can adaptively choose an arbitrary algorithm* $\widetilde{\mathsf{A}}_j$*, and correspondingly obtain oracle access to* $\mathsf{Patch}_\delta(\cdot, \widetilde{\mathsf{A}}_j(SK, \cdot))$*:*

   - *Upon input the* i*-th query* $\widetilde{m}_{i,j}$*, for* $i \in [q]$ *and* $j \in [n]$*, the oracle returns* $\widetilde{\sigma}_{i,j} \leftarrow \mathsf{Patch}_\delta(\widetilde{m}_{i,j}, \widetilde{\mathsf{A}}_j(SK, \widetilde{m}_{i,j}))$ *and updates the public state* δ*;*
   - *Whenever* $\widetilde{\sigma}_{i,j} = \perp$ *the oracle enters a special self-destructs mode, in which the answer to all future queries is by default set to* $\perp$*.*

   Let $\widetilde{\mathcal{Q}}_j = \{\widetilde{m}_{1,j}, \ldots, \widetilde{m}_{q,j}\}$ *be the set of all queries for each* $\widetilde{\mathsf{A}}_j$*.*

4. *Finally,* B *outputs a pair* (m^*, σ^*)*; we say that* B *wins iff* $\mathsf{Vrfy}(VK, (m^*, \sigma^*)) = 1$ *and* $m^* \notin \mathcal{Q} \cup \widetilde{\mathcal{Q}}$*, where* $\widetilde{\mathcal{Q}} := \bigcup_{j=1}^{n} \widetilde{\mathcal{Q}}_j$*.*

Whenever $t = poly(\kappa)$*,* $q = poly(\kappa)$ *and* $\varepsilon = negl(\kappa)$ *we simply say that* $\mathcal{FW}$ *preserves unforgeability for* $\mathcal{SS}$*. Furthermore, in case* A *specifies all of its queries* $\{\widetilde{\mathsf{A}}_j, \widetilde{m}_{i,j}\}_{j\in[n],i\in[q]}$ *at the same time we say that* $\mathcal{FW}$ non-adaptively *preserves unforgeability.*

We observe that Definition 31 is very similar to Definition 23, except for a few crucial differences. First, note that the above definition considers arbitrary classes of SAs instead of SAs within a given class $\mathcal{A}$; this is possible because the output of each invocation of the subverted signing algorithm is patched using the firewall (which is assumed to be un-tamperable).

Second, observe that the above definition relies on the so-called self-destruct capability: Whenever the firewall returns $\perp$, all further queries to any of the oracles results in $\perp$; as we show in Section 5.2.2 this is necessary as without such a capability there exists simple generic attacks that allow for complete security breaches. We stress, however, that the assumption of the self-destruct capability does not make the problem of designing an unforgeability preserving reverse firewall trivial. In fact, the biased-randomness attacks of Section 4.3.1 allow to break all randomized scheme *without* ever provoking a self-destruct. On the positive side, in Section 5.2.3, we show how to design an unforgeability preserving RF for any *re-randomizable* signature scheme.

Impossibility of exfiltration resistance. More generally, one might require a stronger security property from a RF. Namely, we could ask that patched signatures are indistinguishable from real signatures to the eyes of an attacker. This property, which is called exfiltration resistance in [MS15], would be similar in spirit to our definition of indistinguishability w.r.t. continuous SAs (see Definition 22).

It is not hard to see that exfiltration resistance against arbitrary SAs is impossible to achieve in the case of signature schemes; this is because the attacker could simply set the subverted signing algorithm to always output the all-zero string, in which case the RF has no way to patch its input to a valid signature (and thus the adversary can easily distinguish subverted patched signatures from real signatures).[11]

[11] We note, however, that our techniques from Section 5.1 can be extended to design a RF that is

5.2.2 Necessity of Self-Destruct

We show that no RF can preserve both functionality and unforgeability, without assuming the self-destruct capability. This is achieved via a generic (non-adaptive) attack that allows to extract the secret key in case the RF does not self-destruct. The attack itself is a generalization of a similar attack by Gennaro *et al.* [GLM+04] in the context of memory tampering.

Theorem 14. *Let $\mathcal{SS}$ be an EUF-CMA signature scheme. No RF $\mathcal{FW}$ can at the same time be functionality maintaining and non-adaptively $(poly(\kappa), 1, poly(\kappa), \nu(\kappa))$-preserve unforgeability for $\mathcal{SS}$, without assuming the self-destruct capability.*

Proof sketch. Consider the following adversary B playing the game of Definition 31 (omitting the self-destruct capability).

- Upon input the verification key VK, and the initial state δ, initialize $\tau := 1$.
- Forward $\widetilde{\mathsf{A}}_\tau$ to the challenger, where algorithm $\widetilde{\mathsf{A}}_\tau$ is defined as follows: Upon input a message $\widetilde{m}_i$, set $j = \tau \bmod \ell$ (where $\ell := |SK|$) and
  - If $SK[j] = 1$, output $\widetilde{\sigma}_i \leftarrow \mathsf{Sign}(SK, \widetilde{m}_i)$.
  - Else, output $0^{|\sigma|}$.

  Update $\tau \leftarrow \tau + 1$.
- Let $(\bar{m}, \widetilde{\sigma}'_1), \ldots, (\bar{m}, \widetilde{\sigma}'_\ell)$ be the set of tampered signature queries (and answers to these queries) asked by B, where $\widetilde{\sigma}'_i \leftarrow \mathsf{Patch}_\delta(\bar{m}, \widetilde{\mathsf{A}}_\tau(SK, \bar{m}))$. Define $SK'[i] = \mathsf{Vrfy}(VK, (\bar{m}, \widetilde{\sigma}'_i))$ and return $SK' := (SK'[1], \ldots, SK'[\ell])$.

Notice that B specifies its queries non-adaptively, and moreover it only uses one subversion which is queried upon a fixed message $\bar{m} \in \mathcal{M}$. We will show that the extracted key SK' is equal to the original secret key SK with overwhelming probability, which clearly implies the statement. The proof is by induction; assume that the statement is true up to some index $i \geq 1$. We claim that $SK'[i{+}1] = SK[i{+}1]$ with all but negligible probability. To see this, define the event E_{i+1} that $SK[i{+}1] = 0$ and $\mathsf{Vrfy}(VK, (\bar{m}, \widetilde{\sigma}'_{i+1})) = 1$ *or* $SK[i+1] = 1$ and $\mathsf{Vrfy}(VK, (\bar{m}, \widetilde{\sigma}'_{i+1})) = 0$. By the assumption that the RF does not self-destruct and is functionality maintaining, we get that the latter sub-case happens only with negligible probability. On the other hand, if the former sub-case happens we get that the RF forged a signature on $\bar{m}$, which contradicts EUF-CMA security of $\mathcal{SS}$. By a union bound, we get that $\mathbb{P}[E_{i+1}]$ is negligible as desired. □

5.2.3 Patching Re-Randomizable Signatures

We design a RF preserving unforgeability of so-called *re-randomizable* signature schemes (that include unique signatures as a special case).

weakly exfiltration resistant, namely it is exfiltration resistant against restricted SAs that satisfy the verifiability condition.

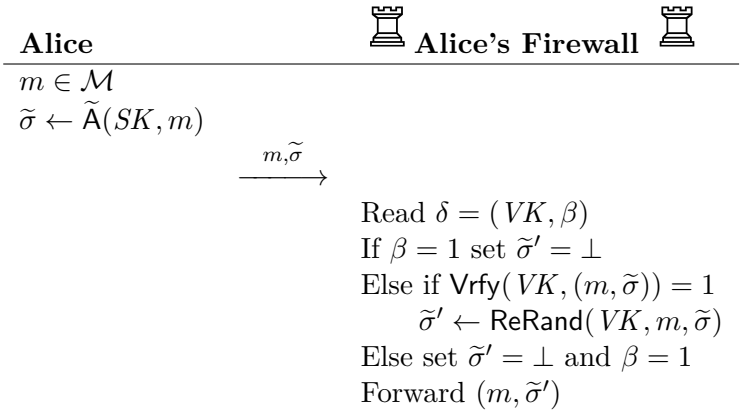

Figure 5.3. A cryptographic reverse firewall preserving unforgeability of any re-randomizable signature scheme against arbitrary SAs.

Definition 32 (Re-randomizable signatures)**.** *A signature scheme* $\mathcal{SS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is efficiently* r*-re-randomizable, if there exists a PPT algorithm* ReRand *such that for all messages* $m \in \mathcal{M}$ *and for all* $(VK, SK) \leftarrow \mathsf{KGen}(1^\kappa)$ *and* $\sigma \leftarrow \mathsf{Sign}(SK, m)$*, we have that* $\Delta(\mathsf{ReRand}(VK, m, \sigma); \mathsf{Sign}(SK, m)) \leq r$*.*

Note that unique signatures are efficiently re-randomizable, for $\mathsf{ReRand}(VK, m, \sigma) = \sigma$ and $r = 0$; Waters' signature scheme [Wat05], and its variant by Hofheinz *et al.* [HJK12], are also efficiently re-randomizable.

Our firewall, which is formally described in Fig. 5.3, first checks if σ is a valid signature on message m under key VK (provided that a self-destruct was not provoked yet). If not, it self-destructs and returns $\perp$; otherwise it re-randomizes σ and outputs the result. The self-destruct capability is implemented using a one-time writable bit β (which is included in the public state).

Theorem 15. *Let* $\mathcal{SS}$ *be a* $(t, (q+1)n, \varepsilon)$*-EUF-CMA signature scheme that is efficiently* r*-re-randomizable and that satisfies* ν_c*-correctness. Then, the RF of Fig. 5.3 maintains functionality and* (t', q, ε')*-preserves unforgeability for* $\mathcal{SS}$*, where* $t' \approx t$ *and* $\varepsilon' \leq qn \cdot (\nu_c + r + \varepsilon)$*.*

Proof. The fact that the firewall maintains functionality follows directly by ν_c-correctness of $\mathcal{SS}$. We now show the firewall preserves unforgeability. Let $\mathbf{G}$ be the game of Definition 31; we write $(i^*, j^*) \in [q] \times [n]$ for the pair of indexes in which the firewall self-destructs (if any). Consider the modified game $\mathbf{H}$ that is identical to $\mathbf{G}$ except that tampered signature queries are answered as described below:

- For all $j < j^*$, upon input $(j, \widetilde{m}_{i,j})$ return $\sigma_{i,j} \leftarrow \mathsf{Sign}(SK, \widetilde{m}_{i,j})$ for all $i \in [q]$.
- For $j = j^*$, upon input $(j, \widetilde{m}_{i,j})$ if $i < i^*$ return $\sigma_{i,j} \leftarrow \mathsf{Sign}(SK, \widetilde{m}_{i,j})$; else return $\perp$.
- For all $j > j^*$, upon input message $\widetilde{m}_{i,j}$ return $\perp$ for all $i \in [q]$.

Claim 15.1. $|\mathbb{P}[\mathsf{B} \text{ wins in } \mathbf{G}] - \mathbb{P}[\mathsf{B} \text{ wins in } \mathbf{H}]| \leq qn \cdot (\nu_c + r)$.

Proof. For an index $k \in [0, n]$, consider the hybrid game $\mathbf{H}_k$ that answers each query $(j, \widetilde{m}_{i,j})$ such that $j \leq k$ as in game $\mathbf{G}$, while all queries $(j, \widetilde{m}_{i,j})$ such that $j > k$ are answered as in $\mathbf{H}$. We note that $\mathbf{H}_0 \equiv \mathbf{H}$ and $\mathbf{H}_n \equiv \mathbf{G}$. Abusing notation, let us write $\mathbf{H}_k$ for the distribution of the random variable corresponding to B's view in game $\mathbf{H}_k$.

We will show that $\Delta(\mathbf{H}_{k-1}, \mathbf{H}_k) \leq q \cdot (\nu_c + r)$ for all k. Fix a particular $k \in [0, n]$, and for an index $l \in [0, q]$ consider the hybrid game $\mathbf{H}_{k,l}$ that is identical to $\mathbf{H}_k$ except that it answers queries $(k, \widetilde{m}_{i,k})$ with $i \leq l$ as in game $\mathbf{G}$, while all queries $(k, \widetilde{m}_{i,k})$ with $i > l$ are treated as in $\mathbf{H}$. Observe that $\mathbf{H}_{k,0} \equiv \mathbf{H}_{k-1}$, and $\mathbf{H}_{k,q} \equiv \mathbf{H}_k$.

We now argue that for each $l \in [q]$, one has that $\mathbb{SD}(\mathbf{H}_{k,l-1}, \mathbf{H}_{k,l}) \leq \nu_c + r$. Observe that, since for $k > j^*$ both games always return $\perp$, we can assume without loss of generality that $k \leq j^*$. Note that the only difference between $\mathbf{H}_{k,l-1}$ and $\mathbf{H}_{k,l}$ is how the two games answer the query $(k, \widetilde{m}_{l,k})$: $\mathbf{H}_{k,l-1}$ returns $\sigma_{l,k} \leftarrow \mathsf{Sign}(SK, \widetilde{m}_{l,k})$ whereas $\mathbf{H}_{k,l}$ returns $\widetilde{\sigma}'_{l,k} \leftarrow \mathsf{Patch}_\delta(\widetilde{m}_{l,k}, \widetilde{\sigma}_{l,k})$ where $\widetilde{\sigma}_{l,k} \leftarrow \widetilde{\mathsf{A}}_k(SK, \widetilde{m}_{l,k})$. Let $E_{l,k}$ be the event that $\mathsf{Vrfy}(VK, (\widetilde{m}_{l,k}, \sigma_{l,k})) = 0$. We have

$$\begin{aligned} \Delta(\mathbf{H}_{k,l-1}; \mathbf{H}_{k,l}) &\leq \Delta(\mathbf{H}_{k,l-1}; \mathbf{H}_{k,l} | \neg E_{l,k}) + \mathbb{P}[E_{l,k}] && (5.7) \\ &\leq r + \nu_c. && (5.8) \end{aligned}$$

Eq. (5.7) follows by Lemma 1 and Eq. (5.8) by the fact that $\mathbf{H}_{k,l-1}$ and $\mathbf{H}_{k,l}$ are statistically close (up to distance r) conditioned on $E_{l,k}$ not happening, and moreover $\mathbb{P}[E_{l,k}] \leq \nu_c$. The former is because signatures are re-randomizable, and thus (as long as the firewall did not self-destruct) the output of ReRand is statistically close (up to distance r) to the output of the original signing algorithm; the latter follows by ν_c-correctness of the signature scheme.

The statement now follows by the above argument and by the triangle inequality, as

$$\begin{aligned} \Delta(\mathbf{G}, \mathbf{H}) &\leq \sum_{k=1}^{n} \Delta(\mathbf{H}_{k-1}, \mathbf{H}_k) \\ &\leq \sum_{k=1}^{n} \sum_{l=1}^{q} \Delta(\mathbf{H}_{k,l-1}, \mathbf{H}_{k,l}) \\ &\leq qn \cdot (\nu_c + r). \end{aligned}$$

□

Claim 15.2. $\mathbb{P}[\mathsf{B} \text{ wins in } \mathbf{H}] \leq qn \cdot \varepsilon$.

Proof. Towards a contradiction, assume B wins in game $\mathbf{H}$ with probability larger than $qn \cdot \varepsilon$. Wlog. we assume that B always outputs its forgery after provoking a self-destruct.[12] We build an adversary B' (using B) that breaks EUF-CMA of $\mathcal{SS}$. Adversary B' is described below.

Adversary B':

[12] If not we can always modify B in such a way that it asks one additional query provoking a self-destruct; this clearly does not decrease B's advantage.

- Receive the verification key VK from the challenger, sample a random pair $(j^*, i^*) \leftarrow\!\!\$\ [n] \times [q]$, and return VK to B.
- Upon input the i-th signature query m_i, forward this value to the signing oracle receiving back a signature $\sigma_i \leftarrow \mathsf{Sign}(SK, m_i)$. Return σ_i to B.
- Upon input a query of the form $(j, \widetilde{m}_{i,j})$ answer as follows:
  - In case $j < j^*$, forward $\widetilde{m}_{i,j}$ to the signing oracle, obtaining $\widetilde{\sigma}_{i,j} \leftarrow \mathsf{Sign}(SK, \widetilde{m}_i)$, and return $\widetilde{\sigma}_{i,j}$ to B.
  - In case $j = j^*$, if $i < i^*$ forward $\widetilde{m}_{i,j}$ to the signing oracle, obtaining $\widetilde{\sigma}_{i,j} \leftarrow \mathsf{Sign}(SK, \widetilde{m}_i)$, and return $\widetilde{\sigma}_{i,j}$ to B. Else, return $\perp$.
  - In case $j > j^*$ answer with $\perp$.
- Whenever B outputs (m^*, σ^*), output (m^*, σ^*).

For the analysis, note that B' runs in time similar to that of B and asks a total of at most $q + qn$ signing queries. Moreover, define the event E that B' guesses correctly the query (j^*, i^*) where B provokes a self-destruct. Clearly, in case E happens we have that B' perfectly simulates the distribution of game **H**. Hence $\mathbb{P}[\mathsf{B}' \text{ wins}] \geq (qn \cdot \varepsilon)/(qn) = \varepsilon$, a contradiction. □

The proof follows by combining the above two claims. □

5.3 Secure Outsourcing of Circuit Manufacturing

The fabrication process adopted by the semiconductor industry is fundamentally global, involving several parties that may not be trusted. As a result, integrated circuits (ICs) are vulnerable to so-called hardware Trojans that can compromise or disable critical systems, or covertly leak sensitive information [LKG+09, CNB09, BRPB14]. Analogously to a software Trojan, a hardware Trojan is a *back-door* deliberately added to the circuit to disrupt its operation or disable it when certain events occur. A Trojan can be added to the circuit during the design phase, by some malicious designer, or more often during the manufacturing phase, by some malicious off-shore fabrication facility. A hardware Trojan's objectives may be to modify the functionality of the circuit (e.g., in order to compromise or disable critical systems), modify its specification (e.g., by changing its energy consumption), covertly leak sensitive information (e.g., from a secret memory), or simply disable the entire circuit when instructed to do so [BHN11]. Once the Trojan is inserted into the circuit it can stay activated the entire time, or it can be "triggered" by some event such as a special input to the circuit.

Reliably detecting compromised circuit components through testing and reverse engineering appears to be an impossible task given our current technology [BR15]. Indeed, all non-destructive testing techniques can easily be circumvented by properly obfuscating embedded Trojans. The U.S. military recognized this threat and started two programs, Trust and IRIS, with the intent of developing techniques and metrics to certify ICs going into weapon systems. The main concern is that advanced weapons may appear to work properly but then switch off in combat or when

triggered by some special events. Another stated concern is information leakage, where a malicious component is programmed to leak sensitive information [Sha07b].

The U.S. military however currently obtains trusted chips through the DOD Trusted Foundry program which is currently managed by the NSA's Trusted Access Program Office (TAPO). Within this program, a trusted design center and foundry are established through an exclusive partnership with IBM for secure semiconductor fabrication and ASIC services, along with the involvement of several Trusted Suppliers which are accredited by an accreditation authority (DMEA). The intent of the Trusted Foundry program is to provide national security and defense programs with access to ICs from trusted sources. However, a recent report by the U.S. Government Accountability Office (GAO) [Mak15], released in April 2015, found that even though the Trusted Foundry program started in 2004, IBM remained the sole-source supplier for leading-edge technologies meeting the criteria put forth by DOD. GAO's report highlights two main issues: First, it notices that IBM sold its microelectronics fabrication business to a foreign-owned entity (GlobalFoundries). Second, relying on a single source supplier for defense microelectronics hinders competition and thus innovation in this critical area.

In this Section, we propose a new approach to the untrusted fabrication problem. We assume that the IC specification and design are trusted but the fabrication facility is not. Rather than testing or reverse engineering the IC hardware received, we employ it in a controlled environment and continuously verify its operations. Our approach makes sense as long as the controlled environment can be: (i) made cheaply, and (ii) run efficiently. We show how to reach these two goals whenever the main objective is to prevent hardware Trojans from releasing sensitive information.

By allowing manufacturers to use off-shore fabrication facilities, we ensure a high degree of competition among suppliers, thus providing lower cost, improved innovation, and access to leading-edge microelectronics.

5.3.1 Secure Circuit Fabrication

In this section we put forward a formal model for assessing security of a circuit whose production is outsourced to one or more untrusted facilities. We start by recalling the standard notion of connected component of a circuit or graph.

Definition 33 (Component)**.** *A circuit $\Gamma' = (V', E')$ is a (connected) component of circuit $\Gamma = (V, E)$ if $V' \subseteq V$, $E' \subseteq E$ and for all $g_1, g_2 \in V'$ we have that $(g_1, g_2) \in E'$ iff $(g_1, g_2) \in E$.*

Next, we introduce the notion of an outsourcing circuit compiler (or simply compiler). In a nutshell a circuit compiler is an efficient algorithm Φ that takes as input (the description of) a circuit Γ, and outputs (the description of) a compiled circuit $\widehat{\Gamma}$. Additionally, Φ returns a list of sub-components $\widehat{\Gamma}_i$ of $\widehat{\Gamma}$ whose production can be outsourced to one or more external manufacturers, together with the relevant information how to connect those sub-components with the remaining ones (that need to be built in-house) in order to re-assemble the compiled circuit $\widehat{\Gamma}$.

Definition 34 (Outsourcing circuit compiler)**.** *Let Γ be an arbitrary circuit. A (ρ, m)-outsourcing compiler Φ is a* PPT *algorithm $(\widehat{\Gamma}, \alpha) \leftarrow \Phi(\Gamma)$, such that the following holds:*

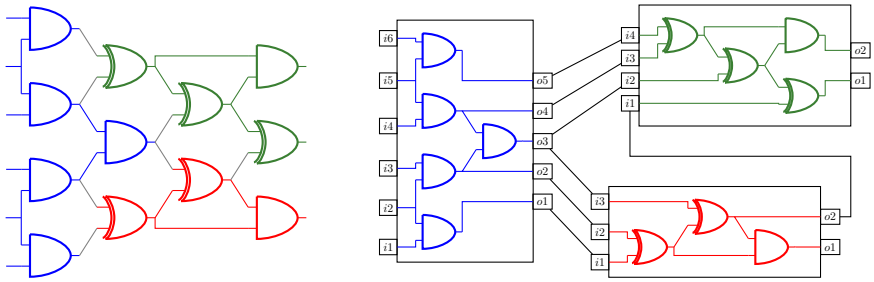

Figure 5.4. On the left side we present the description of a (compiled) circuit. On the right side the same circuit is represented as three different components. The mapping function $\mathcal{M}$ establishes the connections between the blue component and the green and red components.

- $\alpha := ((\widehat{\Gamma}_1, \ldots, \widehat{\Gamma}_n), \mathcal{M}, (I_1, \ldots, I_m))$, *with* $n \in \mathbb{N}$ *and* $I_j \subseteq [n]$, *for* $j \in [m]$, *mutually disjoint subsets.*
- $(\widehat{\Gamma}_1, \ldots, \widehat{\Gamma}_n)$ *are disjoint (connected) components of* $\widehat{\Gamma}$ *such that* $V = \bigcup_{i \in [n]} V_i$, *where* $\Gamma_i = (V_i, E_i)$.
- $\mathcal{M} : V \times V \to \{0, 1\}$ *is a function such that* $\mathcal{M}(v, v') = 1$ *iff* $v, v' \in V_i, V_j$ *for some* $i \neq j$ *and* $(v, v') \in E$.

We call $\rho := \dfrac{\sum_{i \in [n] \setminus I_1 \cup \ldots \cup I_m} |\widehat{\Gamma}_i|}{|\Gamma|}$ *the outsourcing ratio of the compiler.*

Intuitively, in the above definition, the outsourcing ratio ρ represents the fraction of the compiled circuit (w.r.t. the original circuit) that should be built in-house. Note that the sub-components $(\widehat{\Gamma}_i)_{i \in [n]}$ "cover" the entire compiled circuit $\widehat{\Gamma}$ (without overlap), and the mapping function $\mathcal{M}$ specifies how to connect the different components in order to reconstruct $\widehat{\Gamma}$. The sets of indexes $I_j \subseteq [n]$ represents the sub-components whose production will be outsourced to manufacturer $j \in [m]$. See Fig. 5.4 for a pictorial representation in a simple toy example.

Correctness of an outsourcing compiler demands that the compiled circuit maintains the same functionality of the original circuit.

Definition 35 (Correctness)**.** *We say that an outsourcing compiler* Φ *is functionality preserving if for all circuits* Γ, *for all values of the initial memory* M_1, *and for any set of public inputs* $X_1, \ldots, X_q$, *the sequence of outputs* $Y_1, \ldots, Y_q$ *produced by running the original circuit* Γ *starting with state* M_1 *is identical to the sequence of outputs produced by running the transformed circuit* $\widehat{\Gamma}$ *starting with state* M_1 *(with all but negligible probability over the randomness of the compiler and the randomness of the original and compiled circuit).*

Security

We define security using the simulation paradigm. Our approach is similar in spirit to previous work on tamper-resilient circuit compilers (see, e.g., [IPSW06, FPV11]). In a nutshell, security is defined by comparing two experiments. In the first experiment, also called the real experiment, the circuit designer compiles the circuit and outsources

the production of some of the components in the compiled circuit to a set of m untrusted manufacturer. A subset of size t of the manufacturers are malicious, and controlled by a monolithic adversary A; of course the circuit designer does not know which manufacturers are malicious and which ones are honest. During production, A is allowed to completely change the outsourced circuit components under its control, whether by adding, removing or changing gates and/or wires. Later, the designer assembles the circuit by re-combining all the components (the outsourced ones and the ones built in-house). Finally A can access the assembled circuit in a *black-box* way, that is, it can observe inputs/outputs produced by running the assembled circuit (with some initial memory M_1).

In the second experiment, also called the ideal experiment, a simulator is given black-box access to the original circuit (initialized with initial memory M_1). The goal of the simulator is to produce an output distribution which is indistinguishable from the one in the real experiment. In its most general form, our definition allows the simulator to obtain a short leakage on the initial memory. This captures the feature that the adversary in the real experiment could learn at most a short amount of information on the private memory.

Real experiment. The distribution $\mathbf{Real}_{\mathsf{A},\Phi,\mathcal{C},\Gamma,M_1}(\kappa)$ is parameterized by the adversary $\mathsf{A} = (\mathsf{A}_0, \mathsf{A}_1)$, the set of corrupt manufacturers $\mathcal{C}$, the compiler Φ, and the original circuit Γ with initial memory M_1.

1. $(\widehat{\Gamma}, \alpha) \leftarrow \Phi(\Gamma)$: In the first step, the description of the original circuit Γ is given as input to the compiler Φ; the compiler outputs the description of the compiled circuit $\widehat{\Gamma}$ plus the auxiliary information $\alpha := ((\widehat{\Gamma}_1, \ldots, \widehat{\Gamma}_n), \mathcal{M}, (I_1, \ldots, I_m))$ which is used to specify how the compiled circuit is split into sub-components, how the different sub-components are connected (via the mapping function $\mathcal{M}$), and the subset of sub-components whose production is outsourced to each manufacturer (in the index sets I_j, for $j \in [m]$).

2. $(\{\widehat{\Gamma}'_i\}_{i\in I}, \tau) \leftarrow \mathsf{A}_0(1^\kappa, \{\langle\widehat{\Gamma}_i\rangle\}_{i\in I}, \langle\Gamma\rangle, \langle\widehat{\Gamma}\rangle)$: The adversary is given as input the description of the components from the index set $I = \cup_{j\in\mathcal{C}} I_j$, the description of the original circuit Γ, the description of the compiled circuit $\widehat{\Gamma}$, and returns the modified components along with some value τ that may contain some auxiliary state information.

3. $\widehat{\Gamma}' := (\widehat{V}', \widehat{E}')$: The compiled circuit $\widehat{\Gamma}'$ is rebuilt by replacing the components $(\widehat{\Gamma}_i)_{i\in I}$ with the modified components $(\widehat{\Gamma}'_i)_{i\in I}$, and by connecting the different components as specified by the mapping function $\mathcal{M}$.

4. $\mathsf{A}_1^{\widehat{\Gamma}'[M_1](\cdot)}(1^\kappa, \tau)$: Adversary A_1, with auxiliary information τ, is given oracle access to the rebuilt circuit $\widehat{\Gamma}'$ with compiled private memory M_1.

Simulation. The distribution $\mathbf{Ideal}_{\mathcal{S},\mathsf{A},\Phi,\mathcal{C},\Gamma,M_1,\ell}(\kappa)$ is parametrized by the simulator $\mathcal{S}$, the adversary $\mathsf{A} = (\mathsf{A}_0, \mathsf{A}_1)$, the compiler Φ, the set of corrupt manufacturers $\mathcal{C}$, the original circuit Γ with initial memory M_1, and some value $\ell \in \mathbb{N}$.

1. $f \leftarrow \mathcal{S}(1^\kappa, \langle\Gamma\rangle, \Phi, \mathsf{A}, \mathcal{C}, \ell)$: Given as input a description of the original circuit, of the compiler and of the adversary, the subset of corrupt manufacturers, and the parameter $\ell \in \mathbb{N}$, the simulator specifies an arbitrary polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^\ell$.

2. $\mathcal{S}^{\mathsf{A},\Gamma[M_1](\cdot)}(1^\kappa, L)$: The simulator takes as input leakage $L = f(M_1)$, and is given oracle access to adversary $\mathsf{A} = (\mathsf{A}_0, \mathsf{A}_1)$ and to the original circuit Γ with private memory M_1. We remark that the simulator is restricted to be fully black-box. In particular, $\mathcal{S}$ only accesses the modified sub-components returned by A_0 in a black-box way (i.e., without knowing their description).

Definition 36 (Security)**.** *We say that a (ρ, m)-outsourcing circuit compiler Φ is (ℓ, t)-secure if the following conditions are met.*

(i) Non-triviality: *$\rho < 1$, for sufficiently large values of $\kappa \in \mathbb{N}$.*

(ii) Simulatability: *For all $\mathcal{C} \subseteq [m]$ of size at most t and for all PPT adversaries A, for all circuits Γ, and for all initial values of the memory $M_1 \in \{0,1\}^*$, there exists a simulator $\mathcal{S}$ with running time poly$(|\mathsf{A}|, |\Gamma[M_1]|)$ such that*

$$\{\mathbf{Real}_{\mathsf{A},\Phi,\mathcal{C},\Gamma,M_1}(\kappa)\}_{\kappa\in\mathbb{N}} \approx_c \{\mathbf{Ideal}_{\mathcal{S},\mathsf{A},\Phi,\mathcal{C},\Gamma,M_1,\ell}(\kappa)\}_{\kappa\in\mathbb{N}}\,.$$

We observe that the above definition is only interesting for small values of ℓ (as, e.g., it becomes trivial in case $\ell = |M_1|$). Also notice that the non-triviality condition demands that the ratio between the size of the sub-components of the compiled circuit built in-house, and the size of the original circuit, should be less than one. This is necessary, as otherwise a manufacturer could simply produce the entire circuit by itself, without the help of any off-shore facility[13]. Clearly, the smaller ρ the better, as this means that a large fraction of the original circuit production can be outsourced.

Undetectability

We formally define what it means for an adversarial strategy to be undetectable by all black-box polynomial-time tests. Informally, this means that it is hard to distinguish the output of the original circuit from the output of the compiled circuit (after the outsourced sub-components have been maliciously modified). Importantly, the latter has to hold even if the testing strategy knows the initial content of the private memory and the description of all sub-components in the compiled circuit.

For simplicity, we give the definition in the case of a single manufacturer (i.e., $m = 1$); a generalization to the case $m \geq 2$ is immediate. The formalization borrows ideas from similar definitions in the setting of subversion (cf. Section 4.2) and algorithm-substitution attacks [BPR14, BJK15].

[13] We note however, that this definition can still make sense even in some cases where $\rho > 1$. This can be, for instance, when a circuit designer does not have access to the technology necessary to manufacture the entire circuit by itself.

Definition 37 (Undetectability)**.** *Let Φ be an outsourcing circuit compiler, and Γ be a circuit. We say that an adversary A is undetectable for Γ w.r.t. Φ if for all* PPT *algorithms* Test *there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that, for all initial values of the memory M_1, we have that $\mathbb{P}[\mathsf{Test}\ wins] \leq 1/2 + \nu(\kappa)$ in the following game:*

1. *The challenger picks $b \leftarrow\!\!\$\ \{0,1\}$, runs $(\widehat{\Gamma}, (\widehat{\Gamma}_1, \ldots, \widehat{\Gamma}_n), \mathcal{M}, I) \leftarrow \Phi(\Gamma)$, and returns $(M_1, \langle\Gamma\rangle, \langle\widehat{\Gamma}\rangle, (\langle\widehat{\Gamma}_1\rangle, \ldots, \langle\widehat{\Gamma}_n\rangle), \mathcal{M}, I)$ to* Test.
2. *Let $\widehat{\Gamma}'$ be the circuit implicitly defined by the sub-components $\{\widehat{\Gamma}_i\}_{i\in[n]\setminus I} \cup \{\widehat{\Gamma}'_i\}_{i\in I}$ together with the mapping function $\mathcal{M}$, where $\{\widehat{\Gamma}'_i\}_{i\in I} \leftarrow \mathsf{A}(1^\kappa, \{\langle\widehat{\Gamma}_i\rangle\}_{i\in I}, \langle\Gamma\rangle, \langle\widehat{\Gamma}\rangle)$.*
3. *Algorithm* Test*, can ask polynomially many queries of the type X_i. Upon input such query, the answer from the challenger depends on the value of the bit b:*
   - *In case $b = 0$, the output is Y_i where $(Y_i, M_{i+1}) \leftarrow\!\!\$\ \Gamma[M_i](X_i)$.*
   - *In case $b = 1$, the output is Y_i where $(Y_i, M_{i+1}) \leftarrow\!\!\$\ \widehat{\Gamma}'[M_i](X_i)$.*
4. *Algorithm* Test *outputs a bit b', and wins iff $b' = b$.*

5.3.2 Compilers based on VC

In this section we build secure outsourcing compilers that work for any circuit, in the presence of a *single* malicious manufacturer. The compilers are based on any verifiable computation (VC) scheme (satisfying certain properties) for the function computed by the underlying circuit.

We start by recalling the basic definitions for VC schemes. Next, we describe our first compiler, which requires a VC scheme satisfying input privacy. Still in this Section, we describe our second compiler, which can be instantiated with non-input-private VC schemes; our second compiler requires that once the first invalid output is produced, the compiled circuit overwrites its entire memory with the all-zero string and "self-destructs." As we show, this restriction is necessary. Finally, towards the end of this Section we discuss concrete instantiations of our compilers, based on state-of-the-art research on verifiable computing.

Prelude: Verifiable Computation

A verifiable computation scheme allows for a client to outsource the computation of a function to a (untrusted) server; the server produces a proof of correctness along with the output of the function. The client checks the correctness proof to decide whether the output provided by the server is accepted or not.

Definition 38 (Verifiable Computation)**.** *Let $\mathcal{F}$ be a function. A VC scheme $\mathcal{VC} = (\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ for function $\mathcal{F}$ consists of the algorithms described below.*

- *$(SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \kappa)$: The (randomized) key generation algorithm takes as input the function $\mathcal{F}$ and the security parameter κ, and outputs a public key PK and a secret key SK.*

- $(\Sigma_X, VK_X) \leftarrow \mathsf{ProbGen}_{SK}(X)$: *The (randomized) problem generation algorithm takes as input the value X and uses the secret key SK to compute an encoding Σ_X of X and a* secret *verification key VK_X.*

- $\Sigma_Y \leftarrow \mathsf{Compute}_{PK}(\Sigma_X)$: *The (deterministic) compute algorithm takes as input the encoded value Σ_X and uses the public key PK to compute an encoding of $Y = \mathcal{F}(X)$.*

- $Y \leftarrow \mathsf{Verify}_{SK}(VK_X, \Sigma_Y)$: *The (deterministic) verify algorithm takes as input the verification key VK_X and the value Σ_Y; it uses the secret key SK and VK_X to compute a value $Y \in \{0,1\}^* \cup \{\perp\}$, where symbol $\perp$ denotes that the algorithm rejects the value Σ_Y.*

A typical VC scheme needs to satisfy some properties that we formalize below.

Correctness. A VC scheme is correct if the ProbGen algorithm produces problem instances that allow for a honest server to successfully compute values Σ_Y such that $Y = \mathcal{F}(X)$.

Definition 39 (Correctness for VC schemes)**.** *Let $\mathcal{VC}$ be a VC scheme for some function $\mathcal{F}$. We say that $\mathcal{VC}$ is correct if for all values X the following holds:*

$$\mathbb{P}\left[Y = \mathcal{F}(X) : \begin{array}{r}(SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \kappa); (\Sigma_X, VK_X) \leftarrow \mathsf{ProbGen}_{PK}(X) \\ \Sigma_Y \leftarrow \mathsf{Compute}_{PK}(\Sigma_X); Y \leftarrow \mathsf{Verify}_{SK}(VK_X, \Sigma_Y)\end{array}\right] = 1.$$

Soundness. A VC scheme is sound if no malicious server can “trick" a client into accepting an incorrect output, i.e, some value Y such that $Y \neq \mathcal{F}(X)$. We require this to hold even in the presence of so-called verification queries [FGP14].

Definition 40 (Soundness for VC schemes)**.** *Let $\mathcal{VC}$ be a VC scheme for some function $\mathcal{F}$. We say that $\mathcal{VC}$ is sound if for all PPT adversaries A there exists some negligible function $\nu : \mathbb{N} \to [0,1]$ such that $\mathbb{P}[\mathsf{A}$ wins$] \leq \nu(\kappa)$ in the following game.*

1. *The challenger runs $(SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \kappa)$ to obtain the secret key SK and the public key PK, and sends PK to A.*

2. *Adversary A can make the following two types of queries to the challenger, that can be carried out polynomially many times in any order and in an adaptive way.*

   (i) Adversary A can specify an input X_i; the challenger computes $(VK_i, \Sigma_i) \leftarrow \mathsf{ProbGen}_{SK}(X_i)$ and sends Σ_i to A.

   (ii) Adversary A can specify verification queries $(i, \hat{\Sigma})$; the challenger computes $Y = \mathsf{Verify}_{SK}(VK_i, \hat{\Sigma})$ and returns 1 if $Y \neq \perp$, otherwise returns 0.

3. *Eventually, adversary A will output a pair (i^*, Σ^*); we say that A wins iff $Y^* \neq \mathcal{F}(X_{i^*})$ and $Y^* \neq \perp$, such that $Y^* = \mathsf{Verify}_{SK}(VK_{i^*}, \Sigma^*)$.*

Input privacy. A VC scheme is input-private if no server can learn the input value X that the function is being computed on.

Definition 41 (Input-privacy for VC schemes)**.** *Let $\mathcal{VC}$ be a VC scheme for some function $\mathcal{F}$. We say that $\mathcal{VC}$ is input private if for all PPT adversaries A there exists some negligible function $\nu : \mathbb{N} \to [0,1]$ such that $\mathbb{P}[\mathsf{A}\ wins] \leq \nu(\kappa)$ in the following game.*

1. *The challenger runs $(SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \kappa)$ to obtain the secret key SK and the public key PK, and sends PK to A.*

2. *Adversary A can make the following two types of queries to the challenger, that can be carried out polynomially many times in any order and in an adaptive way.*

   (i) Adversary A can specify an input X_i; the challenger computes $(VK_i, \Sigma_i) \leftarrow \mathsf{ProbGen}_{SK}(X_i)$ and sends Σ_i to A.

   (ii) Adversary A can specify verification queries $(i, \hat{\Sigma})$; the challenger computes $Y = \mathsf{Verify}_{SK}(VK_i, \hat{\Sigma})$ and returns 1 if $Y \neq \perp$, otherwise returns 0.

3. *Adversary A chooses two values X_0 and X_1 and sends them to the challenger.*

4. *The challenger samples a random bit $b \leftarrow_{\$} \{0,1\}$ and computes $(VK^*, \Sigma^*) \leftarrow \mathsf{ProbGen}_{SK}(X_b)$ forwarding Σ^* to A.*

5. *Adversary A can still specify to the challenger the queries described above, including special verification queries for the verification key VK^*.*

6. *Finally, A outputs a bit b'; we say that A wins if and only if $b = b'$.*

Outsourceability. A VC scheme is outsourceable if the time to encode the input plus the time to run a verification is smaller than the time to compute the function itself.

Definition 42 (Outsourceability for VC schemes)**.** *A VC scheme can be outsourced if it allows efficient generation and efficient verification. This means that for any X and any Σ_Y the time (or circuit size) required for $\mathsf{ProbGen}_{SK}(X)$ plus the time (or circuit size) required for $\mathsf{Verify}_{SK}(VK, \Sigma_Y)$ is $o(T)$, where T is the time (or circuit size) required to compute $\mathcal{F}(X)$.*

VC without input-privacy. The above definitions can be adapted to cast VC schemes without input-privacy, i.e schemes where the server is allowed to learn the input value X used by the client. For such VC schemes, algorithm $\mathsf{ProbGen}$ returns the value X in the clear along with the secret verification key VK_X; the correctness, soundness and outsourceability definitions can easily be adapted to this setting.

First Compiler

In this section we construct an outsourcing circuit compiler by using a VC scheme that satisfies the properties of correctness, soundness, input-privacy and outsourceability. Let Γ be a circuit. The idea is to invoke a VC scheme for the function $\mathcal{F}$ corresponding to the functionality computed by Γ. The compiled circuit will consist of four main components $\widehat{\Gamma}_{\mathsf{ProbGen}}$, $\widehat{\Gamma}_{\mathsf{Compute}}$, $\widehat{\Gamma}_{\mathsf{Verify}}$, and $\widehat{\Gamma}_{\$}$. The first three components are the circuit representations of the algorithms ProbGen, Compute and Verify corresponding to the underlying VC scheme; such components hard-wire keys (SK, PK) generated using algorithm KeyGen. The fourth component samples the random coins R_i to be used during each invocation of the circuit.

The production of component $\widehat{\Gamma}_{\mathsf{Compute}}$ will then be outsourced to a single untrusted facility, whereas all other components are built in-house (as their implementation needs to be trusted). Notice that the implementation of algorithm KeyGen can be thought of as a pre-processing stage that runs only once (and could be carried out in software).

An important observation is that the size of circuit $\widehat{\Gamma}_{\mathsf{Verify}}$ and $\widehat{\Gamma}_{\mathsf{ProbGen}}$ is independent, and much smaller, than the size of circuit $\widehat{\Gamma}_{\mathsf{Compute}}$. As discussed in the introduction, the size of $\widehat{\Gamma}_{\$}$ can also be considered to be constant (consisting only of a few gates). We describe our first compiler below in more details.

The compiler $\Phi^1_{\mathcal{VC}}$. Let Γ be a circuit, and $\mathcal{VC} = (\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ be a VC scheme for the function $\mathcal{F}$ implemented by Γ. Our first compiler is depicted in Fig. 5.5, and can be described as follows.

1. First run $(SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \kappa)$ once, obtaining the pair of keys (SK, PK).
2. Let $\widehat{\Gamma}_{\mathsf{Memory}}$ be a circuit component consisting only of memory gates, as needed by the original circuit Γ, storing the initial value of the private memory M_1.
3. Let $\widehat{\Gamma}_{\$}$ be a circuit outputting random coins $\widehat{R}_i$ (as needed in each invocation of the compiled circuit).
4. Define a component for each function ProbGen, Compute and Verify of the $\mathcal{VC}$ scheme as explained below.

    - $\widehat{\Gamma}_{\mathsf{ProbGen}}$: This component embeds the secret key SK, and it takes three inputs; the input X_i, the (current) private memory M_i, and random coins $\widehat{R}_i := R_i || R'_i$. It implements function $\mathsf{ProbGen}_{SK}(X_i || M_i || R_i; R'_i)$, that produces two outputs: an encoding Σ_{X_i,M_i,R_i}, and a verification key VK_{X_i,M_i,R_i}.
    - $\widehat{\Gamma}_{\mathsf{Compute}}$: This component embeds the public key PK, and it takes as input the encoding Σ_{X_i,M_i,R_i}. It implements function $\mathsf{Compute}_{PK}(\Sigma_{X_i,M_i,R_i})$, that produces the encoding $\Sigma_{Y_i,M_{i+1}}$ of $(Y_i, M_{i+1}) = \mathcal{F}(X_i, M_i; R_i)$ as output.
    - $\widehat{\Gamma}_{\mathsf{Verify}}$: This component embeds the secret key SK, and it takes two inputs; the encoding $\Sigma_{Y_i,M_{i+1}}$ and the verification key VK_{X_i,M_i,R_i}. It implements function $\mathsf{Verify}_{SK}(VK_{X_i,M_i,R_i}, \Sigma_{Y_i,M_{i+1}})$, to produce the output

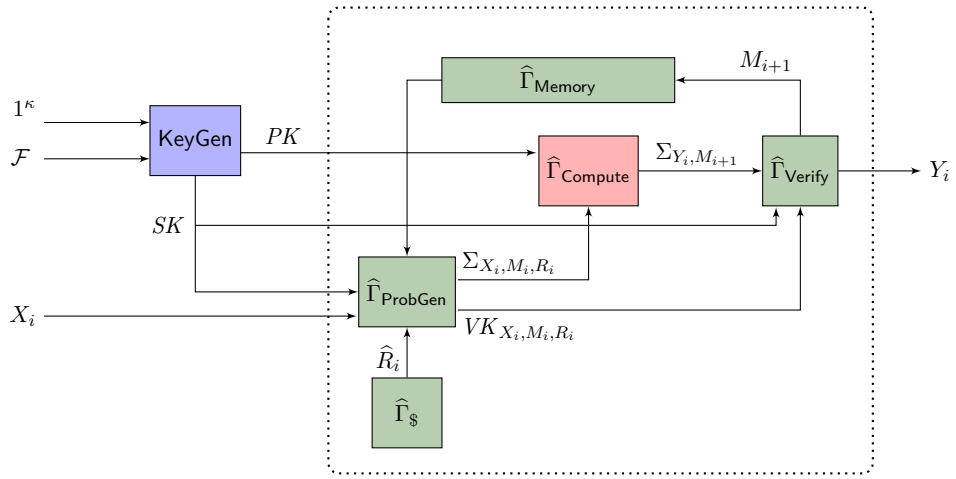

Figure 5.5. The description of compiler $\Phi^1_{\mathcal{VC}}$. The green parts (i.e., $\widehat{\Gamma}_{\mathsf{ProbGen}}$, $\widehat{\Gamma}_{\mathsf{Verify}}$, and $\widehat{\Gamma}_{\$}$) need to be built in-house, while the production of the red part (i.e., $\widehat{\Gamma}_{\mathsf{Compute}}$) can be outsourced; the blue part (i.e., KeyGen) is built only once (not necessarily in hardware). The dotted line depicts the circuit boundaries.

$Y_i \in \{0,1\}^* \cup \{\perp\}$, and eventually update the circuit private memory to M_{i+1}.

5. The output of $\Phi^1_{\mathcal{VC}}$ is defined as follows. The first output is a (description of the) compiled circuit $\widehat{\Gamma}$ as depicted in Fig. 5.5.

   The auxiliary information α consists of the components $\widehat{\Gamma}_{\mathsf{ProbGen}}, \widehat{\Gamma}_{\mathsf{Compute}}, \widehat{\Gamma}_{\mathsf{Verify}}$, $\widehat{\Gamma}_{\mathsf{Memory}}$, and $\widehat{\Gamma}_{\$}$, the mapping function $\mathcal{M}$ that describes the physical connections between such components (i.e., the arrows in Fig. 5.5), and the index set $I = \{2\}$ specifying the component $\widehat{\Gamma}_{\mathsf{Compute}}$ as a candidate for outsourcing.

The theorem below states that the compiler from Fig. 5.5 satisfies our strongest security notion (i.e., Definition 36 with $\ell = 0$), provided that the underlying VC scheme is correct, sound, input-private, and outsourceable.

Theorem 16. *Let Γ be an arbitrary circuit and let $\mathcal{VC}$ be a verifiable computation scheme for the function $\mathcal{F}$ computed by Γ, satisfying the properties of correctness, soundness, input-privacy and outsourceability. Then the compiler $\Phi^1_{\mathcal{VC}}$ is a correct, $(0,1)$-secure $(o(1),1)$-outsourcing circuit compiler.*

Remark 1 (On outsourcing memory gates)**.** *In the compiler depicted in Figure 5.5, $\widehat{\Gamma}_{\$}$ is being built in-house. In order to outsource private memory to a potentially malicious manufacturer we modify the above compiler as follows: instead of storing in $\widehat{\Gamma}_{\$}$ the value M_i in plaintext, we store $c \leftarrow \mathsf{AE}_{SK'}(M_i)$, where c is the encryption of M_i using a symmetric, semantically secure authenticated encryption scheme, with secret key SK'. Moreover, $\widehat{\Gamma}_{\mathsf{ProbGen}}$ is modified such that when receiving the private memory value c, it first decrypts it using SK' and then executes the original circuit $\widehat{\Gamma}_{\mathsf{ProbGen}}$ on the resulting plaintext. We also substitute $\widehat{\Gamma}_{\mathsf{Verify}}$ so that it outputs the encryption of M_{i+1}, under SK'. This modification enables the simulator to execute the circuit using the all-zeros bit-string as the initial memory value, and security*

follows by the semantic security of the encryption scheme. Finally, whenever the decryption of c gives $\perp$ the circuit output is $\perp$.

Proof idea. We give an intuition for the security proof. Correctness of the compiler and the fact that $\rho = o(1)$ follow immediately, respectively, from the correctness and the outsourceability of the underlying VC scheme. As for security, we need to build a simulator $\mathcal{S}$ that is able to "fake" the real experiment for all adversaries A, for all circuits Γ, and for all initial memory values M_1. The simulator runs compiler $\Phi^1_{\mathcal{VC}}$ upon input Γ, forwards the circuit component $\widehat{\Gamma}_{\mathsf{Compute}}$ to A obtaining a modified component $\widehat{\Gamma}'_{\mathsf{Compute}}$, and re-assembles the compiled circuit $\widehat{\Gamma}'$ plugging together all the required components. Thus, upon input a query X_i from A, the simulator simply runs $\widehat{\Gamma}$ upon input X_i and using some fixed memory (e.g., the all-zero string); if the output is invalid, S answers the query with $\perp$, and otherwise it answers the query by using black-box access to the original circuit.

Intuitively, by soundness of the underlying VC scheme, whenever the output of $\widehat{\Gamma}[M_i](\cdot)$ is not $\perp$, such value must be equal to the output of the function $\mathcal{F}(\cdot, M_i)$. On the other hand, the fact that the output is valid or not must be independent of the actual memory used for the computation, as otherwise one could break the input-privacy property of the VC scheme. With this in mind, on can show the indistinguishability between the real and the simulated experiment using a hybrid argument.

Proof. We start by showing that the outsourcing ratio parameter ρ of the compiler $\Phi^1_{\mathcal{VC}}$ is always smaller than 1, for sufficiently large values of the security parameter κ, thus meeting the non-triviality condition.

Claim 16.1. $\rho = o(1)$.

Proof. The non-triviality requirement from Definition 36 states that

$$\rho = \frac{|\widehat{\Gamma}_{\mathsf{ProbGen}}| + |\widehat{\Gamma}_{\mathsf{Verify}}| + |\widehat{\Gamma}_{\$}| + |\widehat{\Gamma}_{\mathsf{Memory}}|}{|\Gamma|} < 1.$$

The claim is that $\rho = o(1)$, i.e. $\lim_{\kappa\to\infty} \frac{\rho}{1} = 0$. By the outsourceability property of the $\mathcal{VC}$ scheme we know that $|\widehat{\Gamma}_{\mathsf{ProbGen}}|+|\widehat{\Gamma}_{\mathsf{Verify}}| = o(|\Gamma|)$, i.e. $\lim_{\kappa\to\infty} \frac{|\widehat{\Gamma}_{\mathsf{ProbGen}}|+|\widehat{\Gamma}_{\mathsf{Verify}}|}{|\Gamma|} = 0$. By substituting the expression for ρ, and by using the fact that the size of $\widehat{\Gamma}_{\$}$ can assumed to be constant and that the original and the compiled circuit contain the same number of memory gates, we obtain:

$$\lim_{\kappa\to\infty} \frac{(|\widehat{\Gamma}_{\mathsf{ProbGen}}| + |\widehat{\Gamma}_{\mathsf{Verify}}| + |\widehat{\Gamma}_{\$}| + |\widehat{\Gamma}_{\mathsf{Memory}}|)/|\Gamma|}{1} = 0.$$

This shows that function ρ converges to 0. As the size of each component is monotonously increasing with the security parameter, for sufficiently large κ, the outsourcing ratio ρ will always be smaller than 1, as desired. □

Claim 16.2. *The compiler $\Phi^1_{\mathcal{VC}}$ satisfies correctness.*

Proof. The correctness of the compiler $\Phi^1_{\mathcal{VC}}$ follows immediately from the correctness property of the underlying VC scheme. □

We proceed to prove security of $\Phi^1_{\mathcal{VC}}$. We need to build a simulator $\mathcal{S}$ that is able to "fake" experiment **Real** for all adversaries A, for all circuits Γ, and for all initial memory values M_1. A description of the simulator follows.

- Run the compiler $\Phi^1_{\mathcal{VC}}(\Gamma)$ supplying (a description of) the original circuit Γ; the output is (a description of) the compiled circuit $\widehat{\Gamma}$, and the auxiliary information $\alpha := ((\widehat{\Gamma}_{\mathsf{ProbGen}}, \widehat{\Gamma}_{\mathsf{Compute}}, \widehat{\Gamma}_{\mathsf{Verify}}, \widehat{\Gamma}_{\mathsf{Memory}}, \widehat{\Gamma}_{\$}), \mathcal{M}, \{2\})$.
- The description of component $\widehat{\Gamma}_{\mathsf{Compute}}$ is sent to adversary A together with the descriptions of Γ and $\widehat{\Gamma}$.
- Adversary A produces the component $\widehat{\Gamma}'_{\mathsf{Compute}}$ (that may be malicious) and sends it to the simulator; the circuit $\widehat{\Gamma}'$ is assembled using the components $(\widehat{\Gamma}_{\mathsf{ProbGen}}, \widehat{\Gamma}'_{\mathsf{Compute}}, \widehat{\Gamma}_{\mathsf{Verify}}, \widehat{\Gamma}_{\mathsf{Memory}}, \widehat{\Gamma}_{\$})$, via the mapping function $\mathcal{M}$.
- Upon input X_i from A, run $(Y_i, \widetilde{M}_{i+1}) \leftarrow \widehat{\Gamma}'[0^\mu](X_i)$, where $\mu := |M_1|$; if $Y_i = \perp$ then forward $\perp$ to A, otherwise query X_i to oracle $\Gamma[M_1](\cdot)$ and forward the output to A.

Let us write $\mathbf{R}$ and $\mathbf{S}$ for the distribution of the random variables in experiment **Real** and **Ideal** of Definition 36. Recall that these variables are parametrized by adversary A, simulator S, initial memory M_1, compiler $\Phi^1_{\mathcal{VC}}$, and circuit Γ, but we omit explicitly writing all these parameters to simplify the exposition.[14] We consider a new experiment $\mathbf{R}'$ that is exactly the same as $\mathbf{R}$, except that for all queries X_i such that $(Y'_i, M_{i+1}) \leftarrow \widehat{\Gamma}'[M_i](X_i)$ with $Y'_i \neq \perp$, the experiment computes $(Y_i, M_{i+1}) \leftarrow \mathcal{F}(X_i, M_i)$ and outputs Y_i.

The claim below shows that experiment $\mathbf{R}$ and $\mathbf{R}'$ are computationally indistinguishable.

Claim 16.3. *For all PPT adversaries* A*, and for all PPT distinguishers* D*, there exists a negligible function* $\nu : \mathbb{N} \to [0,1]$ *such that* $\Delta^{\mathsf{D}}(\mathbf{R}; \mathbf{R}') \leq \nu(\kappa)$*.*

Proof. Let $q \in poly(\kappa)$ be the number of input queries asked by A to its oracle, and let us define an event E that becomes true whenever there is some index $i^* \in [q]$ such that in experiment $\mathbf{R}'$ we have $Y_{i^*} \neq Y'_{i^*}$. Then, by Lemma 1, we have that $\Delta^{\mathsf{D}}(\mathbf{R}; \mathbf{R}') \leq \Delta(\mathbf{R}; \mathbf{R}' | \neg E) + \mathbb{P}[E]$. Note that the distributions $\mathbf{R}$ and $\mathbf{R}'$, conditioned on event E *not* happening are exactly the same. We proceed to show that the probability of event E is negligible.

Let us assume that there exists some circuit Γ, some initial memory M_1 and a PPT adversary A provoking event E with non-negligible probability. We build a PPT adversary A' that uses A in order to break the soundness property of the underlying VC scheme (cf. Definition 40).

Adversary $\mathsf{A}'(M_1, \mathcal{F}, \Gamma, \Phi)$:

1. Receive PK from the challenger, where $(SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \kappa)$.

[14] We also omit to mention the set of corrupt manufacturers $\mathcal{C}$, as in this case there is a single malicious manufacturer.

2. Run $(\widehat{\Gamma}, M_1, \alpha) \leftarrow \Phi(\Gamma, M_1)$, and send (the description of) circuit components $(\widehat{\Gamma}_{\mathsf{Compute}}, \Gamma, \widehat{\Gamma})$ to A. Adversary A produces the modified component $\widehat{\Gamma}'_{\mathsf{Compute}}$ and sends it to A'.
3. For all $i \in [q]$, upon input query X_i from A, do the following.
   - Sample random coins R_i, create the string $Z_i := X_i||M_i||R_i$ and send it to the challenger as an encoding query. In response the challenger computes $(VK_{Z_i}, \Sigma_{Z_i}) \leftarrow \mathsf{ProbGen}_{SK}(Z_i)$ and sends back the encoding Σ_{Z_i} to A'.
   - Run $\Sigma_{Y'_i, M_{i+1}} = \widehat{\Gamma}'_{\mathsf{Compute}}(PK, \Sigma_{Z_i})$ and send a verification query $(i, \Sigma_{Y'_i, M_{i+1}})$ to the challenger; the answer from the challenger is a verification bit d. If $d = 0$, then reply with $\perp$, otherwise compute $(Y_i, M_{i+1}) = \mathcal{F}(M_i, X_i; R_i)$ and reply with Y_i.
4. Finally, pick a random $i^* \in [q]$ and output the pair $(i^*, \Sigma_{Y'_{i^*}})$.

For the analysis, we note that the above simulation is perfect. In particular the outputs seen by A retain exactly the same distribution as in experiment $\mathbf{R}'$. Now A provokes event E with non-negligible probability, so there exists some index $i^* \in [q]$ such that Y_{i^*} is different from Y'_{i^*} where $(Y'_{i^*}, M_{i^*+1}) \leftarrow \widehat{\Gamma}'[M_{i^*}](X_{i^*})$. Since A' guesses the right index i^* with probability $1/q$, we obtain that A' wins the soundness game with non-negligible probability $\frac{1}{q} \cdot \mathbb{P}[E]$. This contradicts the soundness property of the underlying VC scheme, and thus concludes the proof of the claim. □

Claim 16.4. *For all PPT adversaries* A*, and for all PPT distinguishers* D*, there exists a negligible function* $\nu' : \mathbb{N} \to [0,1]$ *such that* $\Delta^{\mathsf{D}}(\mathbf{R}'; \mathbf{S}) \leq \nu'(\kappa)$.

Proof. Let $q \in poly(\kappa)$ be the number of input queries asked by A to its oracle. For an index $i \in [q]$ consider the hybrid experiment $\mathbf{H}_i$ that answers the first i queries as in $\mathbf{R}'$ and all the subsequent queries as in $\mathbf{S}$. We note that experiments $\mathbf{H}_{i-1}$ and $\mathbf{H}_i$ only differ in how the output is computed in position i, and that $\mathbf{H}_0 \equiv \mathbf{S}$ and $\mathbf{H}_q \equiv \mathbf{R}'$.

We now show that for all circuits Γ, all initial memories M_1, and all PPT adversaries A, each pair of adjacent hybrids $\mathbf{H}_{i-1}$ and $\mathbf{H}_i$ are computationally indistinguishable. Fix some $i \in [0, q]$, and assume that there exists some circuit Γ, some initial memory M_1, a PPT adversary A, and a PPT distinguisher D that distinguishes between the pair of hybrids $\mathbf{H}_{i-1}$ and $\mathbf{H}_i$. We build an adversary A' that uses (A, D) in order to break the input-privacy property of the underlying VC scheme (cf. Definition 41).

Adversary $\mathsf{A}'(M_1, \mathcal{F}, \Gamma, \Phi, i)$:

1. Receive PK from the challenger, where $(SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \kappa)$.
2. Run $(\widehat{\Gamma}, \alpha) \leftarrow \Phi(\Gamma)$, and send (the description of) circuit components $(\widehat{\Gamma}_{\mathsf{Compute}}, \Gamma, \widehat{\Gamma})$ to A. Adversary A produces the modified component $\widehat{\Gamma}'_{\mathsf{Compute}}$ and sends it to A'.

3. Upon input query X_j from D, such that $j \neq i$, answer as follows:
   - Sample random coins R_j. If $j < i$ create the string $Z_j := X_j||M_j||R_j$, else create the string $Z_j := X_j||0^\mu||R_j$.
   - Send Z_j to the challenger as an encoding query. In response the challenger computes $(VK_{Z_j}, \Sigma_{Z_j}) \leftarrow \mathsf{ProbGen}_{SK}(Z_j)$ and sends Σ_{Z_j} to A'.
   - Run $\Sigma_{Y_j,M_{j+1}} = \widehat{\Gamma}'_{\mathsf{Compute}}(PK, \Sigma_{Z_j})$, and send a verification query $(j, \Sigma_{Y_j,M_{j+1}})$ to the challenger.
   - The challenger replies with a decision bit d; if $d = 0$ then return $\perp$, otherwise compute $(Y_j, M_{j+1}) = \mathcal{F}(M_j, X_j; R_j)$ and return Y_j.
4. Upon input query X_i from A, answer as follows:
   - Sample random coins R_i, and create two strings $Z_0^* := X_i||M_i||R_i$ and $Z_1^* := X_i||0^\mu||R_i$.
   - Send Z_0^* and Z_1^* to the challenger as challenge inputs for the input-privacy game; the challenger replies with Σ_{Z^*} (which corresponds to the encoding of either Z_0^* or Z_1^*).
   - Run $\Sigma_{Y^*} = \widehat{\Gamma}'_{\mathsf{Compute}}(PK, \Sigma_{Z^*})$ and send a verification query (i, Σ_{Y^*}) to the challenger; the challenger replies with a decision bit d. If $d = 0$ then return $\perp$, otherwise compute $(Y_i, M_{i+1}) = \mathcal{F}(M_i, X_i; R_i)$ and return Y_i.
5. Finally, output whatever D outputs.

For the analysis, we note that the above simulation is perfect. In particular, depending on Z^* being either an encoding of Z_0^* or Z_1^* the view of (A, D) is identical to the view in either experiment $\mathbf{H}_{i-1}$ or $\mathbf{H}_i$. Hence, A' retains the same advantage as (A, D) which contradicts input-privacy of the underlying VC scheme. We conclude that there exist negligible functions $\nu', \nu'' : \mathbb{N} \to [0, 1]$ such that

$$\Delta^{\mathsf{D}}(\mathbf{R}'; \mathbf{S}) \leq \sum_{i=1}^{q} \Delta^{\mathsf{D}}(\mathbf{H}_{i-1}; \mathbf{H}_i) \leq q \cdot \nu'(\kappa) \leq \nu''(\kappa),$$

as desired. □

The statement now follows by Claim 16.3 and Claim 16.4, and by the triangle inequality, as

$$\Delta^{\mathsf{D}}(\mathbf{R}; \mathbf{S}) \leq \Delta^{\mathsf{D}}(\mathbf{R}; \mathbf{R}') + \Delta^{\mathsf{D}}(\mathbf{R}'; \mathbf{S}) \leq \nu(\kappa) + \nu'(\kappa).$$

This finishes the proof. □

Second Compiler

In this section we construct an outsourcing circuit compiler by using any VC scheme that satisfies the properties of correctness, soundness and outsourceability. The construction follows the same ideas of compiler $\Phi^1_{\mathcal{VC}}$ (cf. Section 5.3.2), with two

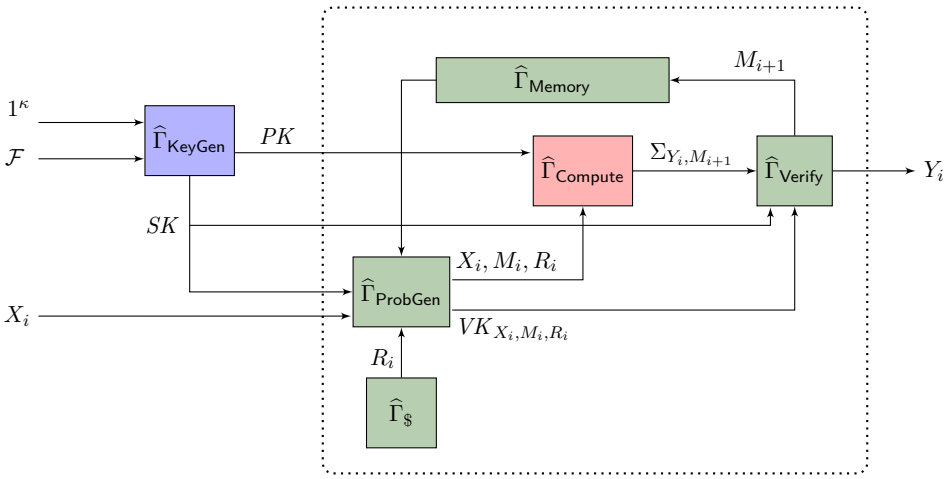

Figure 5.6. The description of compiler $\Phi^2_{\mathcal{VC}}$. Notice that component $\widehat{\Gamma}_{\mathsf{ProbGen}}$ does not need to hide its input, and that component $\widehat{\Gamma}_{\mathsf{Verify}}$ implements the self-destruct feature. The dotted line depicts the circuit boundaries.

main differences. First, as we rely on a VC scheme without input-privacy, the component $\widehat{\Gamma}_{\mathsf{ProbGen}}$ now outputs the values X_i, M_i, R_i in the clear. Second, the component $\widehat{\Gamma}_{\mathsf{Verify}}$ needs to implement a special "self-destruct" feature: The first time the component returns the special symbol $\perp$, the private memory is overwritten with the all-zero string.

As we argue later in this section, the self-destruct feature is *necessary*, in that, without such a feature, generic attacks against our compiler are possible, possibly exposing the entire private memory in an undetectable manner.

The compiler $\Phi^2_{\mathcal{VC}}$. Let Γ be a circuit, and $\mathcal{VC} = (\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ be a VC scheme for the function $\mathcal{F}$ implemented by Γ. The description of the compiler $\Phi^2_{\mathcal{VC}}(\Gamma)$ can be found in Fig. 5.6. The theorem below establishes that such a compiler is secure, provided that the original circuit to be produced is resilient to a logarithmic (in the security parameter) amount of leakage on its private memory.

Theorem 17. *Let Γ be an arbitrary circuit and let $\mathcal{VC}$ be a verifiable computation scheme for the function $\mathcal{F}$ computed by Γ, satisfying the properties of correctness, soundness and outsourceability. Then the compiler $\Phi^2_{\mathcal{VC}}$ is a correct $(\log(q)+1, 1)$-secure $(o(1), 1)$-outsourcing circuit compiler, where q is the number of oracle queries asked by adversary A in Definition 36.*

Proof idea. We give an intuition for the security proof. As for our first compiler, correctness of the compiler and the fact that $\rho = o(1)$ follow readily from the correctness and the outsourceability properties of the underlying VC scheme. As for security, we need to build a simulator $\mathcal{S}$ that is able to "fake" the real experiment for all adversaries A, for all circuits Γ, and for all initial memory values M_1. The simulator $\mathcal{S}$ is allowed to define an arbitrary polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^{\log(q)+1}$ that leaks a logarithmic amount of information regarding the initial private memory M_1. Intuitively, the function f hard-wires the randomness

for running the entire real experiment with adversary A; note that this randomness includes the randomness for the compiler, and the random coin tosses of A. The output of the function consists of: (i) a predicate abort which tells whether there was some round where the output of circuit $\widehat{\Gamma}'$ differs from the output of the original circuit Γ; (ii) the index q^* corresponding to the round where self-destruct was triggered.

In case $\mathsf{abort} = 1$ the simulator S simply gives up and aborts. Otherwise, it emulates the real experiment with the same randomness hard-wired in the function f, answering all the queries from A up until round $q^* - 1$ using black-box access to the original circuit, whereas all further queries are answered with $\perp$. On the one hand, this is a perfect simulation as long as $\mathsf{abort} = 0$. On the other hand, the soundness property of the underlying VC scheme ensures that $\mathsf{abort} = 0$ with all but a negligible probability. The proof follows.

Proof. The proof of the non-triviality and the correctness condition are similar to the proofs of Claim 16.1 and Claim 16.2 respectively, and are therefore omitted.

We proceed to prove the security of $\Phi^2_{\mathcal{VC}}$. We need to build a simulator $\mathcal{S}$ that is able to "fake" experiment **Real** for all adversaries A, for all circuits Γ, and for all initial memory values M_1. The simulator $\mathcal{S}$ is allowed to define an arbitrary polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^{\log(q)+1}$ that leaks a logarithmic amount of information regarding the initial private memory M_1. The function f is defined as follows.

- Sample all random coins for experiment **Real**, including the random coins ρ_{A} of adversary A, and the random coins ρ_Φ of the compiler Φ.
- Function f takes as input the memory M_1 and the sampled random coins $\rho = (\rho_{\mathsf{A}}, \rho_\Phi)$; note that f can completely simulate experiment **Real** using the memory M_1 and the sampled randomness ρ.
- For all $i \in [q]$ function f runs simultaneously the circuits $\widehat{\Gamma}'[M_i](X_i)$ and $\Gamma[M_i](X_i)$ using the initial memory M_1 and adversary A's randomness to sample the inputs X_i's.
- Define the event E which becomes true in case there exists some index $i^* \in [q]$ such that $\widehat{\Gamma}'[M_{i^*}](X_{i^*}) \neq \Gamma[M_{i^*}](X_{i^*})$.
- The outputs of function f are
  - $\mathsf{abort} \in \{0,1\}$: a predicate that is 1 if event E happens;
  - $q^* \in [q]$: the index corresponding to the round in which a self-destruct is triggered (if any).

The claim below shows that the probability of event E happening is negligible.

Claim 17.1. *For all PPT adversaries* A*, the probability that* f *returns* $\mathsf{abort} = 1$ *is negligible.*

Proof. Let us assume that there exists some circuit Γ, some memory M_1, and a PPT adversary A that provokes an $\mathsf{abort} = 1$ with non-negligible probability. We build a PPT adversary A' that uses A to break the soundness property of the underlying VC scheme (cf. Definition 40). The description of A' follows.

Adversary $\mathsf{A}'(M_1, \mathcal{F}, \Gamma, \Phi)$:

1. Receive PK from the challenger, where $(SK, PK) \leftarrow \mathsf{KeyGen}(\mathcal{F}, \kappa)$.
2. Run $(\widehat{\Gamma}, \alpha) \leftarrow \Phi(\Gamma)$ and send (the description of) circuit components $\widehat{\Gamma}_{\mathsf{Compute}}$, Γ and $\widehat{\Gamma}$ to A. Adversary A produces the modified component $\widehat{\Gamma}'_{\mathsf{Compute}}$ and sends it to A'.
3. For all $i \in [q]$, upon input query X_i answer as follows.
   - Sample random coins R_i, define the string $Z_i := X_i||M_i||R_i$, and run $\Sigma_{Y_i,M_{i+1}} = \widehat{\Gamma}'_{\mathsf{Compute}}(PK, Z_i)$.
   - Send a verification query $(i, \Sigma_{Y_i,M_{i+1}})$ to the challenger, receiving back a verification bit d. If $d = 0$ then return $\perp$, and set $M_{i+1} := 0^\mu$; otherwise, compute $(Y_i, M_{i+1}) = \mathcal{F}(M_i, X_i; R_i)$ and return Y_i.
4. Finally, pick a random $i^* \in [q]$ and output the pair $(i^*, \Sigma_{Y_{i^*},M_{i^*+1}})$.

For the analysis, note that the above simulation is perfect. In particular, the first time a verification query is rejected, the reduction returns $\perp$ and overwrites the private memory with the all-zero string. By definition of the event E, we know that there exists an index $i^* \in [q]$ such that the value Y_{i^*} corresponding to the encoding $\Sigma_{Y_{i^*},M_{i^*+1}}$ is different from the value Y_{i^*} computed via $\mathcal{F}(M_{i^*}, X_{i^*})$.

Since A' guesses the right index i^* with probability $1/q$, we conclude that A' breaks the soundness property of the underlying VC scheme with non-negligible probability $\frac{1}{q} \cdot \mathbb{P}[E]$. This concludes the claim proof. □

We proceed to describe how the simulator $\mathcal{S}$ uses the obtained leakage in order to fake the distribution in the real experiment (when run with the same randomness ρ initially sampled by the simulator to define the leakage function f).

- After getting the auxiliary input (abort, q^*), check if $\mathsf{abort} = 1$ and in this case stop with output "simulation failed." Otherwise, proceed to the next step.
- Upon input query X_i, such that $i < q^*$, simply forward the query X_i to oracle $\Gamma[M_1](\cdot)$ and output the answer Y_i obtained from the oracle.
- Upon input query X_i, such that $i = q^*$, output $\perp$ and initialize the memory $\widetilde{M}_i := 0^\mu$.
- Upon input query X_i, such that $i > q^*$, run $(Y_i, \widetilde{M}_{i+1}) = \widehat{\Gamma}'[\widetilde{M}_i](X_i)$ and output Y_i.

The claim below shows that, whenever $\mathsf{abort} = 0$, the above simulation is perfect.

Claim 17.2. *Whenever* $\mathsf{abort} = 0$*, for all circuits* Γ*, all initial memories* M_1*, and for all PPT adversaries* A*, the output produced by the above simulator* $\mathcal{S}$ *is identically distributed to the output of the experiment* $\mathbf{Real}_{\mathsf{A},\Phi,\Gamma,M_1}(\kappa)$ *(using the same randomness as sampled by* $\mathcal{S}$*).*

Proof. Notice that the simulator implements the self-destruct feature (overwriting the private memory with the all-zero string) at round q^*; this is exactly what happens in the real experiment (conditioning on the randomness ρ used to define the leakage function being the same as the one used in the experiment). Moreover, all queries before round q^* are answered by running the original circuit Γ. Again, this is a perfect simulation (as $\mathsf{abort} = 0$). The claim follows. □

Let us write $\mathcal{R}$ for the randomness space of experiment $\mathbf{Real}_{\mathsf{A},\Phi,\Gamma,M_1}(\kappa)$.[15] For $\rho \in \mathcal{R}$ let us write $\mathbf{Real}_{\mathsf{A},\Phi,\Gamma,M_1}(\kappa)[\rho]$ to denote the outcome of the real experiment when using the randomness ρ; similarly, let us write $\mathbf{Ideal}_{\mathcal{S},\mathsf{A},\Phi,\Gamma,M_1,\ell}(\kappa)[\rho]$ for the outcome of the ideal experiment when using the randomness ρ. Whenever $\mathsf{abort} = 0$, by the above claim, we have that for any ρ:

$$\mathbf{Real}_{\mathsf{A},\Phi,\Gamma,M_1}(\kappa)[\rho] = \mathbf{Ideal}_{\mathcal{S},\mathsf{A},\Phi,\Gamma,M_1,\ell}(\kappa)[\rho].$$

On the other hand, by Claim 17.1, for a random ρ we have that $\mathsf{abort} = 1$ happens only with a negligible probability. It follows that for all PPT distinguishers D there exists a negligible function $\nu : \mathbb{N} \to [0,1]$ such that

$$|\mathbb{P}\left[\mathsf{D}(\mathbf{Real}_{\mathsf{A},\Phi,\Gamma,M_1}(\kappa)) = 1\right] - \mathbb{P}\left[\mathsf{D}(\mathbf{Ideal}_{\mathcal{S},\mathsf{A},\Phi,\Gamma,M_1,\ell}(\kappa)) = 1\right]| \leq \nu(\kappa),$$

□

The case of memory outsourcing is identical to that of the first compiler and we refer the reader to Remark 1.

Necessity of self-destruct. We show that the self-destruct feature is *necessary* for the security of compiler $\Phi^2_{\mathcal{VC}}$ by presenting an *undetectable* attack (as per Definition 37) against the circuit $\widehat{\Gamma}'$ produced by compiler $\Phi^2_{\mathcal{VC}}$. The attack, which is described in details in Fig. 5.7, works for a large class of circuits, and leaks the entire initial private memory M_1 embedded in the compiled circuit.

Theorem 18. *Let* Γ *be any circuit with input size* $n = \omega(\log \kappa)$ *and let* $\Phi^2_{\mathcal{VC}}$ *be the compiler from Fig. 5.6 without the self-destruct capability. Then, the attack* A^* *described in Fig. 5.7 is undetectable for* Γ *w.r.t.* $\Phi^2_{\mathcal{VC}}$*, and leaks the entire initial private memory* M_1*.*

Proof. The second part of the statement follows directly by observing that knowledge of the trapdoor information τ allows to learn the value M_1 with overwhelming probability.

We proceed to show undetectability. Let $\mathbf{G}$ be the undetectability game described in Definition 37, where the adversary A is chosen to be adversary A^* from Fig. 5.7.

[15] In what follows we omit to parametrize the experiments by the set $\mathcal{C}$ of corrupt manufacturers, as we are considering the case of a single malicious manufacturer.

Consider the game $\mathbf{G}_0$, an identical copy of game $\mathbf{G}$ when $b = 0$, and consider the game $\mathbf{G}_1$ an identical copy of game $\mathbf{G}$ when $b = 1$. Abusing notation, let us write $\mathbf{G}_0$ and $\mathbf{G}_1$ for the distribution of the random variables corresponding to algorithm Test's view in games $\mathbf{G}_0$ and game $\mathbf{G}_1$ respectively. For an index $i \in [q]$ consider the hybrid game $\mathbf{H}_i$ that answers the first i queries as in $\mathbf{G}_0$ and all the subsequent queries as in $\mathbf{G}_1$. We note that game $\mathbf{H}_{i-1}$ and $\mathbf{H}_i$ only differ in position i, and that $\mathbf{H}_0 \equiv \mathbf{G}_1$ and $\mathbf{H}_q \equiv \mathbf{G}_0$.

We claim that, for all $i \in [q]$, it holds $\mathbf{H}_{i-1} \approx_c \mathbf{H}_i$. Fix some index i, and define the event E that the i-th query X_i happens to be equal to the secret value τ embedded in the modified component $\widehat{\Gamma}_{\mathsf{Compute}}$ (as described in Fig. 5.7). By Lemma 1, $\Delta(\mathbf{G}_0; \mathbf{G}_1) \leq \Delta(\mathbf{G}_0; \mathbf{G}_1 | \neg E) + \mathbb{P}[E]$. Clearly, conditioned on event E *not* happening, the distributions of game $\mathbf{G}_0$ and $\mathbf{G}_1$ are identical; this is because in such a case the modified component $\widehat{\Gamma}'_{\mathsf{Compute}}$ behaves exactly like the original component $\widehat{\Gamma}_{\mathsf{Compute}}$.

On the other hand, if $|\tau| = \omega(\log \kappa)$, the probability of event E is negligible. We conclude that for all PPT distinguishers D there exists a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that

$$\Delta^{\mathsf{D}}(\mathbf{G}_0; \mathbf{G}_1) \leq \sum_{i=1}^{q} \Delta^{\mathsf{D}}(\mathbf{H}_{i-1}; \mathbf{H}_i) \leq \nu(\kappa).$$

□

Concrete Instantiations

The area of verifiable computing has a long history in the cryptographic literature [BFLS91, Mic00, GKR08, GGP10]. We refer the reader to the excellent survey

Let Γ be any circuit with input size of n bits, and consider the compiled circuit $\widehat{\Gamma}$ produced by running $\Phi^2_{\mathcal{VC}}(\Gamma)$. Define the following adversarial strategy, aimed at building a modified circuit component $\widehat{\Gamma}'_{\mathsf{Compute}}$ that leaks the entire initial private memory M_1 when given some secret trapdoor information.

$\underline{\mathsf{A}^*(X_i, M_i, R_i)}$:

1. Choose a random value τ, such that $|\tau| = n$, and store it in a memory location.
2. Upon the first run, duplicate and store the content of memory M_1 into another memory location, and initialize a counter $j := 0$.
3. Upon input a tuple (X_i, M_i, R_i) such that $X_i \neq \tau$, output $\Sigma_{Y_i} = \mathsf{Compute}_{PK}(X_i, M_i, R_i)$. Otherwise, behave as follows:
   - If $M_1[j] = 0$, let Σ_{Y_i} be the all-zero string.
   - Else, compute $\Sigma_{Y_i} = \mathsf{Compute}(X_i, M_i, R_i)$.
   - Update counter $j \leftarrow (j + 1) \bmod |M_1|$ and output Σ_{Y_i}.

Figure 5.7. Undetectable attack against compiler $\Phi^2_{\mathcal{VC}}$ (without the self-destruct capability).

by Walfish and Blumberg [WB15] for a thorough introduction.

By now, several schemes and models for the problem of outsourcing computation are known (see, among others, [AIK10,BHR12,BCCT12,BCG+13,JNO14,ACG+14]). Below, we focus only on VC schemes suitable for the compilers described in this section.

First compiler. For the compiler $\Phi^1_{\mathcal{VC}}$, we need a VC scheme satisfying both soundness and input-privacy (in the presence of verification queries). The only known schemes meeting these requirements are the ones recently constructed by Fiore, Gennaro, and Pastro [FGP14] (relying on fully homomorphic encryption [Gen09]).

Second compiler. For the compiler $\Phi^2_{\mathcal{VC}}$, we need a VC scheme satisfying soundness (in the presence of verification queries), but no input-privacy is required. Therefore, we can instantiate this compiler using more efficient schemes based on SNARKs [Mic00, GLR11, BCCT12, DFH12, BCI+13, BCCT13, BCG+13, GGPR13, PHGR13, BCTV14, CFH+15, Gro16].

Extensions. While the compilers described in this section rely on non-interactive VC schemes, it is easy to generalize our constructions to also work with *interactive* VC. The difference is that the communication pattern between the components $\widehat{\Gamma}_{\mathsf{Compute}}$ and $\widehat{\Gamma}_{\mathsf{Verify}}$ would consist of multiple sequential messages, before the values Y_i and M_{i+1} are produced. VC schemes of this type were designed, e.g., in [GKR08,CMT12, GKR15]. See also [WHG+16] for an excellent discussion on how to implement those schemes in hardware.

5.3.3 Compiler based on MPC

In this section we present a compiler which is based on *multi-party computation* (MPC) techniques, aiming to improve the efficiency of the resulting circuit at the expense of achieving security in the weaker model where there are $m \geq 2$ manufacturers, a t-fraction of which is malicious (for some threshold $t \leq m-1$). Before presenting our compiler, we first revisit the core ideas of MPC, and then we give a generic definition for MPC protocols in the *client-server* model, along the lines of [Bea97].

MPC in the Client-Server Model

In MPC we consider p parties, where each party P_i, for $1 \leq i \leq p$, possesses an input X_i and they all wish to jointly compute the tuple $(Y_1, \ldots, Y_p) = \mathcal{F}(X_1, \ldots, X_p)$, where P_i receives Y_i. In the client-server model, the parties are divided into two categories: the parties that provide inputs and wish to receive the output of the computation (clients), and those performing the computation (servers). A *t-private* MPC protocol ensures that any adversary who controls up to t servers cannot leak any information related to the private inputs of the clients, besides the information that can be inferred by inspecting the output of the computation, and regardless of the number of corrupted clients. In our compiler the circuit corresponding to the code executed by the servers will be outsourced to a number of possibly malicious

manufacturers, that may apply arbitrary modifications against the circuit components. Thus, we require MPC protocols that are secure against active (malicious) attackers. Additional properties that might be useful are the following:

1. Correctness: The protocol computes the correct output.
2. Output delivery: It guarantees that the honest parties will receive the output of the computation.
3. Fairness: If at least one party learns the output, then, all the parties learn the output.
4. Identifiable abort: Whenever the protocol terminates due to an abort message, it is guaranteed that at least one malicious party will be caught (this might be useful for deactivating adversarial circuit components).

The general idea behind the compiler is the following. Let Γ be a circuit implementing some functionality $\mathcal{F}$, and let $\Pi_{\mathcal{F}}$ be a t-private MPC protocol realizing the function $\mathcal{F}$. Then, assuming the number of malicious manufacturers is at most $t < m$, the circuit $\widehat{\Gamma}$ will implement the code of $\Pi_{\mathcal{F}}$, and each $\widehat{\Gamma}_i$ will implement the code of the i-th server. Below we define the protocol framework that we are going to use for the rest of this section. The idea is to describe any MPC protocol using its next message function, denoted as Next.

Definition 43 (r-round protocols)**.** *Let C, S be sets of probabilistic interactive Turing machines, with cardinalities p, m, respectively. An r-round protocol Π for p clients and m servers is a tuple $(C, S, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Next})$, where $\mathsf{Next} = (\mathsf{Next}_1, \ldots, \mathsf{Next}_m)$, described as follows.*

- Setup: *Each client computes $(X_i^1, \ldots, X_i^m) \leftarrow \mathsf{Enc}(X_i)$, and sends X_i^j to the server indexed by j. Let $\mathbf{in}^j := (X_1^j, \ldots, X_p^j)$, and $\tau_j := 0$ (we assume that the network is fully connected, still the properties of the communication channel depend on the instantiation).*
- Computation: *For $i \in [r]$:*
  - *If $i \neq r$, for $j \in [m]$ execute $(o_1^j, \ldots, o_m^j, \tau_j') \leftarrow \mathsf{Next}_j(\mathbf{in}^j, \tau_j)$, send o_k^j, $k \neq j$, to the server with index k. Set $\mathbf{in}^j = (o_j^1, \ldots, o_j^m)$, and $\tau_j = \tau_j'$.*
  - *If $i = r$, for $j \in [m]$ execute $o^j \leftarrow \mathsf{Next}_j(\mathbf{in}^j, \tau_j)$, and send o_j to Dec.*
- Output: *Execute $(Y_1, \ldots, Y_p) \leftarrow \mathsf{Dec}(o_1, \ldots, o_m)$, and send Y_j to the client with index j.*

For any function $\mathcal{F}$, the protocol computing $\mathcal{F}$ will be denoted by $\Pi_{\mathcal{F}}$.

Informally, in the first step of the protocol execution, the clients encode their inputs, as it is prescribed by Enc, and then the main computation begins. The code executed by the servers at each round is defined by the function Next (the next message function). Hence, in the i-th round, server S_j computes Next_j upon the outputs and the state information τ produced by the other servers in round

$i-1$. One can also consider deterministic next message functions, assuming the randomness is given as input in each round. Below, we formally define *correctness* and *privacy* for MPC protocols.

Definition 44 (Correctness)**.** *Let $\mathcal{F}$ be a p-party functionality. We say that Π realizes $\mathcal{F}$ with perfect (resp., statistical) correctness if for any input $(X_1, \ldots, X_p)$, the probability that the output delivered to the i-th client during the protocol execution is different than Y_i, is 0 (resp., negligible in λ), where $(Y_1, \ldots, Y_p) = \mathcal{F}(X_1, \ldots, X_p)$ for $i \in [p]$.*

Definition 45 ((t,m)-privacy)**.** *Let λ be the security parameter, p be the number of parties (clients) and m be the number of servers, and let A be an adversary that may corrupt any set of parties $I_c \subseteq [p]$, and servers $I_s \subset [m]$, where $|I_s| \leq t$. We say that the protocol Π realizes $\mathcal{F}$ with (t,m)-privacy if there exists a PPT algorithm S such that for all sufficiently large $\lambda \in \mathbb{N}$,*

$$\mathsf{View}_{I_s,I_c}(\lambda, X_1, \ldots, X_p) \approx_c \mathsf{S}(1^\kappa, I_c, I_s, (X_i, Y_i)_{i \in I_c})$$

where $\mathsf{View}_{I_s,I_c}(\lambda, X_1, \ldots, X_p)$ denotes the joint view of the servers and clients in I_s and I_c, respectively, within an execution of the protocol upon inputs $X_1, \ldots, X_p$, and $(Y_1, \ldots, Y_p) = \mathcal{F}(X_1, \ldots, X_p)$.

The main idea behind the above definition is that the view of the attacker during the protocol execution can be computed based on its own input and output only.

The Compiler

Let Γ be a circuit implementing the function $\mathcal{F}(M_1, \cdot)$, where for any X and $i \in \mathbb{N}$, we have $(Y, M_{i+1}) = \mathcal{F}(M_i, X)$. Let $\Pi_{\mathcal{F}} = (C, S, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Next})$ be an r-round protocol realizing the function $\mathcal{F}$, over a set of m servers with a single client. The compiler produces $(\widehat{\Gamma}, \mathsf{aux}) \leftarrow \Phi_{\Pi_{\mathcal{F}}}(\Gamma)$, where

- $\widehat{\Gamma}$ is the circuit that implements $\Pi_{\mathcal{F}}$ (depicted in Figure 5.8 for the case $m = 2$ and $p = 1$), having as a sub-circuit $\widehat{\Gamma}_{\mathsf{Memory}}$, which is a circuit consisting only of memory gates, as needed by the original circuit Γ. During initialization, $\widehat{\Gamma}_{\mathsf{Memory}}$ stores the initial private memory value, M_1.
- $\alpha = ((\widehat{\Gamma}_1, \ldots, \widehat{\Gamma}_{m+2}), \mathcal{M}, (I_1, \ldots, I_m))$, where
  - $\widehat{\Gamma}_{m+1} = \widehat{\Gamma}_{\mathsf{Enc}}$ and $\widehat{\Gamma}_{m+2} = \widehat{\Gamma}_{\mathsf{Dec}}$, i.e., the circuits $\widehat{\Gamma}_{m+1}$ and $\widehat{\Gamma}_{m+2}$ implement the encoder, Enc, and the decoer Dec, of $\Pi_{\mathcal{F}}$, respectively.
  - For $i \in [m]$, $\widehat{\Gamma}_i$ is the circuit that implements the code of the i-th server, for the entire execution of $\Pi_{\mathcal{F}}$ (r-rounds). Those circuits can be implemented in a straightforward way using the next message function Next_i (cf. the sub-components $\widehat{\Gamma}_1$ and $\widehat{\Gamma}_2$ in Figure 5.8).
  - The mapping function $\mathcal{M}$ describes the physical connections between the circuits described above, and I_j, for $j \in [m]$, specifies the components that will be outsourced to the manufacturer with index j. In our case $I_j = \{j\}$.

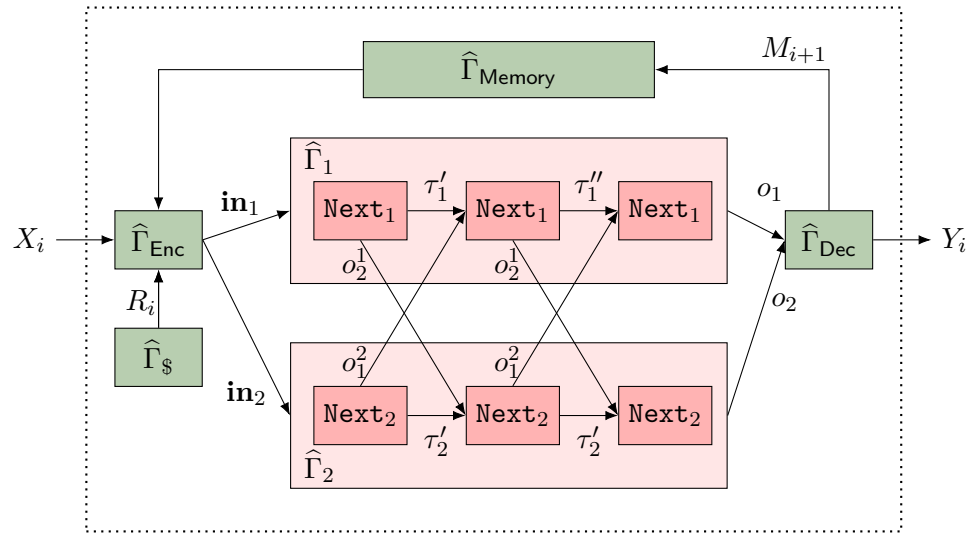

Figure 5.8. The MPC compiler for the case of $m = 2$ outsourcing facilities (from which *one* can be malicious). The components $\widehat{\Gamma}_1$ and $\widehat{\Gamma}_2$ can be outsourced, while the connectivity between them and the remaining components are built in-house. The dotted line depicts the circuit boundaries.

– In case the original circuit is randomized, in addition to the components described above, Φ also outputs a circuit $\widehat{\Gamma}_{\$}$ producing random coins R_i (as needed in each invocation of the circuit).

Our construction must be non-trivial (cf. Definition 36), thus the underlying protocol Π must satisfy the following outsourceability property.

Definition 46 (Outsourceability of procotols)**.** *A protocol* $\Pi = (C, S, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Next})$ *that realizes the function* $\mathcal{F}$ *can be outsourced if it satisfies the following condition: The circuit computing the encoding and decoding procedures* $(\mathsf{Enc}, \mathsf{Dec})$ *must be smaller than the circuit computing the function* $\mathcal{F}$*.*

We prove the following result.

Theorem 19. *Let* $\mathcal{F}$ *be any function, and let* $\Pi_{\mathcal{F}}$ *be a* (t, m)*-private MPC protocol for* $\mathcal{F}$*, satisfying the correctness and outsourceability properties. Then, the compiler* $\Phi_{\Pi_{\mathcal{F}}}$ *is a correct,* $(0, t)$*-secure,* $(o(1), m)$*-outsourcing circuit compiler.*

Proof. The correctness property of $\Phi_{\Pi_{\mathcal{F}}}$ follows directly by the correctness property of $\Pi_{\mathcal{F}}$. By the outsourceability property of the $\Pi_{\mathcal{F}}$, we have that $|\mathsf{Enc}|+|\mathsf{Dec}| = o(|\Gamma|)$, which implies that $\lim_{\kappa\to\infty} \frac{|\mathsf{Enc}|+|\mathsf{Dec}|}{|\Gamma|} = 0$. Having this in mind we can prove that the ratio ρ converges to 0, as κ goes to infinity using the expression for ρ, and assuming $\widehat{\Gamma}_{\$}$ is of constant size:

$$\lim_{\kappa\to\infty} \frac{(|\mathsf{Enc}| + |\mathsf{Dec}| + |\widehat{\Gamma}_{\$}|)/|\Gamma|}{1} = 0.$$

Thus, for sufficiently large κ the outsourcing ration ρ is smaller than 1.

Let $\mathcal{F}$ be any functionality and let Γ be the circuit implementing $\mathcal{F}$. Assuming that $\Pi_{\mathcal{F}}$ is a (t,m)-private MPC protocol for $\mathcal{F}$, we will prove that $\Phi_{\Pi_{\mathcal{F}}}$ is a $(0,t)$-secure, circuit compiler. Concretely (cf. Definition 36), we need to prove that for all $\mathcal{C} \subseteq [m]$ of size at most t, all PPT adversaries A, all circuits Γ, and for all initial values of the memory $M_1 \in \{0,1\}^*$, there exists a simulator $\mathcal{S}$ with running time $poly(|\mathsf{A}|, |\Gamma[M_1]|)$ such that

$$\{\mathbf{Real}_{\mathsf{A},\Phi,\mathcal{C},\Gamma,M_1}(\kappa)\}_{\kappa\in\mathbb{N}} \approx_c \{\mathbf{Ideal}_{\mathcal{S},\mathsf{A},\Phi,\mathcal{C},\Gamma,M_1,\ell(\kappa)}(\kappa)\}_{\kappa\in\mathbb{N}}, \tag{5.9}$$

for all sufficiently large values of κ. Let A be an attacker $\Phi_{\Pi_{\mathcal{F}}}$. The idea behind the proof is to relate the interaction between A and the circuits produced by $\Phi_{\Pi_{\mathcal{F}}}$, with the interaction between an attacker A' corrupting up to t, with a protocol $\Pi_{\mathcal{F}}$. Then, we will use the simulator $\mathcal{S}'$ that is given by the (t,m)-privacy of $\Pi_{\mathcal{F}}$ to construct a simulator $\mathcal{S}$, satisfying Eq. (5.9). In what follows, and for the sake of simplicity, we prove the needed assuming A is a single round attacker, and then we discuss how the proof easily extends to the setting in which we have multiple executions.

By the compiler definition, the protocol $\Pi_{\mathcal{F}}$ that $\Phi_{\Pi_{\mathcal{F}}}$ is based on, consists of two clients, C_1, C_2, where C_1 is the corrupted client that provides the public input to the circuit, X, and C_2 supplies the circuit with private input, M_i, and m servers. Let Γ be the circuit implementing $\mathcal{F}$. Given the adversary A for $\Phi_{\Pi_{\mathcal{F}}}$ we define the adversary $\mathsf{A}' = (\mathsf{A}'_0, \mathsf{A}'_1)$ against $\Pi_{\mathcal{F}}$ as follows:

- **(server corruption)** A'_0: execute $(\widehat{\Gamma}, \alpha) \leftarrow \Phi(\Gamma)$, where $\alpha := ((\widehat{\Gamma}_1, \ldots, \widehat{\Gamma}_n), \mathcal{M}, (I_1, \ldots, I_m))$, and sample $(\{\widehat{\Gamma}'_i\}_{i\in I}, \tau) \leftarrow \mathsf{A}_0(1^\kappa, \{\langle\widehat{\Gamma}_i\rangle\}_{i\in I}, \langle\Gamma\rangle, \langle\widehat{\Gamma}\rangle)$. Then corrupt the server S_i, for $i \in I$, so that S_i will execute the possibly modified circuit $\widehat{\Gamma}'_i$.
- **(protocol execution)** A'_1: participate in the protocol $\Pi_{\mathcal{F}}$ choosing the input for client C_1 (the corrupted client), according to the input value chosen by A_1. Concretely, execute the following steps: sample $X \leftarrow \mathsf{A}_1(1^\kappa, \tau)$, define the input of client C_1 to equal to X, receive the output of $\Pi_{\mathcal{F}}$ for client C_1, Y, for inputs (X, M), and forward Y to A_1.

We define the random variable View_{I_s,I_c}, $I_s = \mathcal{C}$, $I_c = \{1\}$, to be the view of A while indirectly interacting with $\Pi_{\mathcal{F}}$ through A'_1. Clearly, by the definition of A', the view of A while being executed by A', matches its view while executing the real world experiment of Definition 36, thus we have

$$\mathsf{View}_{I_s,I_c}(\lambda, X, M) = \mathbf{Real}_{\mathsf{A},\Phi,\mathcal{C},\Gamma,M_1}(\kappa). \tag{5.10}$$

Assuming $\Pi_{\mathcal{F}}$ is (t,m)-private against A', there exists exists a simulator $\mathcal{S}''$ that simulates the view of A' during the protocol execution. Let S' be code of $\mathcal{S}''$ that only outputs the view of A. Then we have that for all sufficiently large $\lambda \in \mathbb{N}$,

$$\mathsf{View}_{I_s,I_c}(\lambda, X, M) \approx_c \mathsf{S}'(1^\lambda, I_c, I_s, (X,Y)_{i\in I_c}). \tag{5.11}$$

Now we define the simulator S for A against $\Phi_{\Pi_{\mathcal{F}}}$. $\mathcal{S}$ on input $(1^\kappa, \langle\Gamma\rangle, \Phi, \mathsf{A}, \mathcal{C}, 0)$ executes the following steps:

- executes A_1 with oracle access to $\Gamma[M_1](\cdot)$, and constructs the pair (X,Y), i.e., it constructs the valid output of $\mathcal{F}$ on input X, chosen by A_1.

- executes $o \leftarrow \mathsf{S}'(\lambda, I_c, I_s, (X,Y)_{i\in I_c})$, where $I_s = \mathcal{C}$ and $I_c = \{1\}$, and outputs o.

Clearly, from Eq. (5.11) we have that S produces outputs which is computationally indistinghuishable from $\mathsf{View}_{I_s,I_c}(\lambda, X, M)$, and then using Eq. (5.10) we receive,

$$\mathbf{Real}_{\mathsf{A},\Phi,\mathcal{C},\Gamma,M_1}(\kappa) \approx_c \mathbf{Ideal}_{\mathcal{S},\mathsf{A},\Phi,\mathcal{C},\Gamma,M_1,\ell}(\kappa),$$

and this concludes the proof for attackers executing the protocol only once.

For multi-round attackers against the circuit compiler, we need to have multiple, sequential executions, of the same protocol, as a single execution computes a single circuit output. Moreover, the attacker is non-adaptive, and corrupts the servers only before the first protocol execution. By the composition theorem of [Can00], we have that any secure MPC protocol is also secure against sequential composition, even for adaptive adversaries. Using a standard hybrid argument, this gives rise to a simulator, $\mathcal{S}'$, that simulates the view of the attacker for all protocol executions, and the proof idea is identical to one given above: we relate the attacker against the compiler to an attacker against the protocol, and we use $\mathcal{S}'$ to construct a simulator $\mathcal{S}$ for the circuit compiler. □

The case of memory outsourcing is identical to that of the first compiler and we refer the reader to Remark 1.

Concrete Instantiations

Many MPC protocols satisfy the outsourceability property, as the values that feed the main computation, i.e., the output of the encoder, are independent of the function that is being evaluated, and mostly depends on the number of parties, as in the case of [GMW87] (where the same holds for decoding). An explicit (t, m)-private protocol is given in [DI05], for $t < m/2$, in which there is a pre-processing phase that can be implemented by the encoder, with running time independent of the function that is being evaluated. The construction uses secure point-to-point and broadcast channels, that can be implemented directly between the components, and besides privacy it also guarantees output delivery.

We can also easily adapt the SPDZ protocol [DPSZ12] to the client-server setting. The SPDZ protocol requires a pre-processing phase that is performed by the parties, and that will feed the encoder circuit who will perform the actual encoding (which is only a linear operation). The complete protocol requires a linear number of public-key operations in the circuit size $|\Gamma|$, with the encoder requiring only a linear number of operations in m. The efficiency of the pre-processing stage can be further improved [BDTZ16]. This construction does not guarantee output deliver, but it is secure against adversaries that corrupt up to $m - 1$ sub-components.

Finally, the construction of [IOZ14] can also be adapted to the client-server scenario and additionally provides security with identifiable aborts.

CHAPTER 6

Conclusions

In this thesis we covered different uses of backdoors in cryptography. We gave contributions on the legitimate usage of backdoors (the good) by defining a new security goal for chameleon hash functions. We also contributed to the better understanding of the implications and consequences on the malicious uses of backdoors (the bad). Finally, we contributed with immunization techniques against the malicious use of backdoors (the not so ugly). We briefly recall our contributions and discuss implications and future work next.

Chameleon-Hash transformation. We presented the first generic transformation from any public coin collision-resistant chameleon hash function to a secret coin enhanced collision-resistant chameleon hash function. The enhanced collision property can be fundamental for some applications, as we showed in Section 3.3.4 with our blockchain application. A natural direction for improvement of the transformation is to have it from even weaker assumptions. Another open problem is the possibility of having the transformation from a public coin CH to a public coin enhanced collision-resistant CH.

Subversion-Resilient signatures. We gave security definitions for signatures resilient against subversion attacks and we showed positive results for unique signatures and re-randomized signatures (by adopting a reverse firewall).

Our study has strong implications in practice and might influence the way digital signature schemes are selected or adopted in standards and protocols. A subverted signature scheme is arguably even more deceitful and dangerous in practice than subverted encryption. Indeed, it is well-known that authenticated encryption must involve digital certificates that are signed by Certification Authorities (CAs). If a CA is using a subverted signature scheme, it is reasonable to expect the signing key will eventually be exposed. With knowledge of the signing key, it is possible to impersonate any user and carry out elementary man-in-the-middle attacks. This

renders the use of any type of encryption utterly pointless and underlines the important role played by signatures in the context of secure communications.

Unfortunately, signature schemes currently employed to sign digital certificates, or used in protocols such as OTR, TLS/SSL, SSH, etc., are all susceptible to a subversion attack and their use should possibly be discontinued. The positive news however is that there already exist signature schemes that are subversion-resilient and they are efficient and well-established. This is in contrast with encryption where *good* schemes are not deployable in all contexts since they require retention of state information (see [BPR14]).

Open problems in this vein includes the analysis of virtually every cryptographic primitive under the subversion model.

Secure circuit fabrication. We put forward a simulation-based security definition for assessing security of ICs whose fabrication has (partially) been outsourced to an untrusted off-shore manufacturer. Our definition implies a strong guarantee, essentially saying that no matter how the manufacturer modified the outsourced IC sub-components, using the re-assembled IC in the wild cannot leak sensitive information about the private memory.

There are several interesting open questions in this direction. First, it might be interesting to explore variations of our model, for example by considering the case where there are several (non-colluding) manufacturers involved in the fabrication process. In such a case, it might be possible to obtain significant efficiency improvements, e.g., by relying on special VC schemes already suitable for similar settings [ACG$^+$14].

Second, one could try to instantiate our compilers with specialized VC schemes that are tailored for specific functionalities. Although there are already some schemes with this feature—e.g., [FGP14] constructs VC schemes tailored for (multivariate) polynomials and linear combinations—to the best of our knowledge, there is no concrete VC scheme for verifying the computation of a specific cryptographic functionality (such as AES).

Third, it would be interesting to explore different approaches in order to build compilers meeting our security definition without relying on verifiable computing or multiparty computation.

Bibliography

[ACF14] Michel Abdalla, Dario Catalano, and Dario Fiore. Verifiable random functions: Relations to identity-based key encapsulation and new constructions. *J. Cryptology*, 27(3):544–593, 2014.

[ACG+14] Prabhanjan Ananth, Nishanth Chandran, Vipul Goyal, Bhavana Kanukurthi, and Rafail Ostrovsky. Achieving privacy in verifiable computation with multiple servers - without FHE and without preprocessing. In *PKC*, pages 149–166, 2014.

[ACM+14] Per Austrin, Kai-Min Chung, Mohammad Mahmoody, Rafael Pass, and Karn Seth. On the impossibility of cryptography with tamperable randomness. In *CRYPTO*, pages 462–479, 2014.

[ADL14] Divesh Aggarwal, Yevgeniy Dodis, and Shachar Lovett. Non-malleable codes from additive combinatorics. In *STOC*, pages 774–783, 2014.

[AdM04] Giuseppe Ateniese and Breno de Medeiros. On the key exposure problem in chameleon hashes. In *SCN*, pages 165–179, 2004.

[ADW09] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[AFPW11] Martin R. Albrecht, Pooya Farshim, Kenneth G. Paterson, and Gaven J. Watson. On cipher-dependent related-key attacks in the ideal-cipher model. In *FSE*, pages 128–145, 2011.

[AGM+15] Shashank Agrawal, Divya Gupta, Hemanta K. Maji, Omkant Pandey, and Manoj Prabhakaran. A rate-optimizing compiler for non-malleable codes against bit-wise tampering and permutations. In *TCC*, pages 375–397, 2015.

[AHI11] Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. In *Innovations in Computer Science*, pages 45–60, 2011.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP*, pages 152–163, 2010.

[AL11] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly-secure multiparty computation. *ECCC*, 18:36, 2011.

[AMV15] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. In *CCS*, pages 364–375, 2015.

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *EUROCRYPT*, pages 56–73, 2004.

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *J. Cryptology*, 21(2):149–177, 2008.

[BBG13] James Ball, Julian Borger, and Glenn Greenwald. Revealed: how US and UK spy agencies defeat internet privacy and security. *Guardian Weekly*, September 2013.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, pages 41–55, 2004.

[BC10] Mihir Bellare and David Cash. Pseudorandom functions and permutations provably secure against related-key attacks. In *CRYPTO*, pages 666–684, 2010.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ICS*, pages 326–349, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *ACM STOC*, pages 111–120, 2013.

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: verifying program executions succinctly and in zero knowledge. In *CRYPTO*, pages 90–108, 2013.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, pages 315–333, 2013.

[BCM11] Mihir Bellare, David Cash, and Rachel Miller. Cryptography secure against related-key attacks and tampering. In *ASIACRYPT*, pages 486–503, 2011.

[BCR+13] Emmanuel Bresson, Dario Catalano, Mario Di Raimondo, Dario Fiore, and Rosario Gennaro. Off-line/on-line signatures revisited: a general unifying paradigm, efficient threshold variants and experimental results. *Int. J. Inf. Sec.*, 12(6):439–465, 2013.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security Symposium*, pages 781–796, 2014.

[BDI+99] Mike Burmester, Yvo Desmedt, Toshiya Itoh, Kouichi Sakurai, and Hiroki Shizuya. Divertible and subliminal-free zero-knowledge proofs for languages. *J. Cryptology*, 12(3):197–223, 1999.

[BDTZ16] Carsten Baum, Ivan Damgård, Tomas Toft, and Rasmus Winther Zakarias. Better preprocessing for secure multiparty computation. In *ACNS*, pages 327–345, 2016.

[Bea97] Donald Beaver. Commodity-based cryptography (extended abstract). In *ACM STOC*, pages 446–455, 1997.

[Bei11] Amos Beimel. Secret-sharing schemes: A survey. In *IWCC*, pages 11–46, 2011.

[Ber03] Daniel J. Bernstein. Proving tight security for standard rabin-williams signatures, 2003.

[Ber08] Daniel J. Bernstein. Proving tight security for Rabin-Williams signatures. In *EUROCRYPT*, pages 70–87, 2008.

[BFGM01] Mihir Bellare, Marc Fischlin, Shafi Goldwasser, and Silvio Micali. Identification protocols secure against reset attacks. In *EUROCRYPT*, pages 495–511, 2001.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *ACM STOC*, pages 21–31, 1991.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[BHN11] Mark Beaumont, Bradley Hopkins, and Tristan Newby. Hardware trojans — prevention, detection, countermeasures (a literature review). Technical report, Australian Government Department of Defence, 07 2011.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *ASIACRYPT*, pages 134–153, 2012.

[BJK15] Mihir Bellare, Joseph Jaeger, and Daniel Kane. Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks. In *CCS*, pages 1431–1440, 2015.

[BK03] Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In *EUROCRYPT*, pages 491–506, 2003.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptology*, 17(4):297–319, 2004.

[BPR14] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In *CRYPTO*, pages 1–19, 2014.

[BPRW15] Allison Bishop, Valerio Pastro, Rajmohan Rajaraman, and Daniel Wichs. Essentially optimal robust secret sharing with maximal corruptions. *IACR Cryptology ePrint Archive*, 2015:1032, 2015.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS*, pages 62–73, 1993.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures—How to sign with RSA and Rabin. In *EUROCRYPT*, pages 399–416, 1996.

[BR14] Mihir Bellare and Todor Ristov. A characterization of chameleon hash functions and new, efficient designs. *J. Cryptology*, 27(4):799–823, 2014.

[BR15] Shivam Bhasin and Francesco Regazzoni. A survey on hardware trojan detection techniques. In *IEEE ISCAS*, pages 2021–2024, 2015.

[BRPB14] Georg T. Becker, Francesco Regazzoni, Christof Paar, and Wayne P. Burleson. Stealthy dopant-level hardware trojans: extended version. *J. Cryptographic Engineering*, 4(1):19–31, 2014.

[BSW06] Dan Boneh, Emily Shen, and Brent Waters. Strongly unforgeable signatures based on computational diffie-hellman. In *PKC*, pages 229–240, 2006.

[BSW13] Elette Boyle, Gil Segev, and Daniel Wichs. Fully leakage-resilient signatures. *J. Cryptology*, 26(3):513–558, 2013.

[But15] Vitalik Buterin. On public and private blockchains. `https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/`, 2015.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 2000.

[CFH+15] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 253–270, 2015.

[CJ07] Benoît Chevallier-Mames and Marc Joye. A practical and tightly secure signature scheme without hash function. In *CT-RSA*, pages 339–356, 2007.

[CL02] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *SCN*, pages 268–289, 2002.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*, pages 56–72, 2004.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT*, pages 402–414, 1999.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Innovations in Theoretical Computer Science*, pages 90–112, 2012.

[CNB09] Rajat Subhra Chakraborty, Seetharam Narasimhan, and Swarup Bhunia. Hardware trojan: Threats and emerging solutions. In *IEEE HLDVT*, pages 166–171, 2009.

[Coi] Bitcoin venture capital. `http://www.coindesk.com/bitcoin-venture-capital/`.

[Cor00] Jean-Sébastien Coron. On the exact security of full domain hash. In *CRYPTO*, pages 229–235, 2000.

[Cor02] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In *EUROCRYPT*, pages 272–287, 2002.

[CP92] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, pages 89–105, 1992.

[CPP06] Benoît Chevallier-Mames, Pascal Paillier, and David Pointcheval. Encoding-free ElGamal encryption without random oracles. In *PKC*, pages 91–104, 2006.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, pages 13–25, 1998.

[CS00] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. *ACM Trans. Inf. Syst. Secur.*, 3(3):161–185, 2000.

[CTZD10] Xiaofeng Chen, Haibo Tian, Fangguo Zhang, and Yong Ding. Comments and improvements on key-exposure free chameleon hashing based on factoring. In *Inscrypt*, pages 415–426, 2010.

[CZK04] Xiaofeng Chen, Fangguo Zhang, and Kwangjo Kim. Chameleon hashing without key exposure. In *ISC*, pages 87–98, 2004.

[CZS+10] Xiaofeng Chen, Fangguo Zhang, Willy Susilo, Haibo Tian, Jin Li, and Kwangjo Kim. Identity-based chameleon hash scheme without key exposure. In *ACISP*, pages 200–215, 2010.

[CZS+14] Xiaofeng Chen, Fangguo Zhang, Willy Susilo, Haibo Tian, Jin Li, and Kwangjo Kim. Identity-based chameleon hashing and signatures without key exposure. *Inf. Sci.*, 265:198–210, 2014.

[CZT+11] Xiaofeng Chen, Fangguo Zhang, Haibo Tian, Baodian Wei, and Kwangjo Kim. Discrete logarithm based chameleon hashing and signatures without key exposure. *Computers & Electrical Engineering*, 37(4):614–623, 2011.

[Dam87] Ivan Damgård. Collision free hash functions and public key signature schemes. In *EUROCRYPT*, pages 203–216, 1987.

[dC16] Michael del Castillo. The dao attacked: Code issue leads to $60 million ether theft. `http://www.coindesk.com/dao-attacked-code-issue-leads-60-million-ether-theft/`, June 2016.

[DeR] Chris DeRose. Why blockchain immutability is a perpetual motion claim. `http://www.coindesk.com/immutability-extraordinary-goals-blockchain-industry/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed:+CoinDesk+%28CoinDesk+-+The+Voice+of+Digital+Currency%29`.

[Des88a] Yvo Desmedt. Abuses in cryptography and how to fight them. In *CRYPTO*, pages 375–389, 1988.

[Des88b] Yvo Desmedt. Subliminal-free authentication and signature (extended abstract). In *EUROCRYPT*, pages 23–33, 1988.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *TCC*, pages 54–74, 2012.

[DFK+06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *TCC*, pages 285–304, 2006.

[DFMV13] Ivan Damgård, Sebastian Faust, Pratyay Mukherjee, and Daniele Venturi. Bounded tamper resilience: How to go beyond the algebraic barrier. In *ASIACRYPT*, pages 140–160, 2013.

[DFMV15] Ivan Damgård, Sebastian Faust, Pratyay Mukherjee, and Daniele Venturi. The chaining lemma and its application. In *ICITS*, pages 181–196, 2015.

[DFP15] Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. A more cautious approach to security against mass surveillance. In *FSE*, pages 579–598, 2015.

[DFS16] Stefan Dziembowski, Sebastian Faust, and François-Xavier Standaert. Private circuits iii: Hardware trojan-resilience via testing amplification. In *ACM CCS*, 2016.

[DGG+15] Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. A formal treatment of backdoored pseudorandom generators. In *EUROCRYPT*, pages 101–126, 2015.

[DHLW10] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Efficient public-key cryptography in the presence of key leakage. In *ASIACRYPT*, pages 613–631, 2010.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *CRYPTO*, pages 378–394, 2005.

[DK12] Dana Dachman-Soled and Yael Tauman Kalai. Securing circuits against constant-rate tampering. In *CRYPTO*, pages 533–551, 2012.

[DK14] Dana Dachman-Soled and Yael Tauman Kalai. Securing circuits and protocols against $1/\text{poly}(k)$ tampering rate. In *TCC*, pages 540–565, 2014.

[DLSZ15] Dana Dachman-Soled, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Locally decodable and updatable non-malleable codes and their applications. In *TCC*, pages 427–450, 2015.

[DMS16] Yevgeniy Dodis, Ilya Mironov, and Noah Stephens-Davidowitz. Message transmission with reverse firewalls - secure communication on corrupted machines. In *CRYPTO*, pages 341–372, 2016.

[Dod03] Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. In *PKC*, pages 1–17, 2003.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012.

[DPW10] Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In *Innovations in Computer Science*, pages 434–452, 2010.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *PKC*, pages 416–431, 2005.

[EGM96] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *J. Cryptology*, 9(1):35–67, 1996.

[ES98] Shawna Meyer Eikenberry and Jonathan Sorenson. Efficient algorithms for computing the jacobi symbol. *J. Symb. Comput.*, 26(4):509–523, 1998.

[Eth] Ethereum project. `https://www.ethereum.org/`.

[FF02] Marc Fischlin and Roger Fischlin. The representation problem based on factoring. In *CT-RSA*, pages 96–113, 2002.

[FGK+13] David Mandell Freeman, Oded Goldreich, Eike Kiltz, Alon Rosen, and Gil Segev. More constructions of lossy and correlation-secure trapdoor functions. *J. Cryptology*, 26(1):39–74, 2013.

[FGP14] Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In *ACM CCS*, pages 844–855, 2014.

[FHN+12] Sebastian Faust, Carmit Hazay, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Angela Zottarel. Signature schemes secure against hard-to-invert leakage. In *ASIACRYPT*, pages 98–115, 2012.

[Fis03] Marc Fischlin. The Cramer-Shoup strong-RSA signature scheme revisited. In *PKC*, pages 116–129, 2003.

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO*, pages 152–168, 2005.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In *INDOCRYPT*, pages 60–79, 2012.

[FMNV14] Sebastian Faust, Pratyay Mukherjee, Jesper Buus Nielsen, and Daniele Venturi. Continuous non-malleable codes. In *TCC*, pages 465–488, 2014.

[FMNV15] Sebastian Faust, Pratyay Mukherjee, Jesper Buus Nielsen, and Daniele Venturi. A tamper and leakage resilient von Neumann architecture. In *PKC*, pages 579–603, 2015.

[FMVW14] Sebastian Faust, Pratyay Mukherjee, Daniele Venturi, and Daniel Wichs. Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In *EUROCRYPT*, pages 111–128, 2014.

[FNV15] Antonio Faonio, Jesper Buus Nielsen, and Daniele Venturi. Mind your coins: Fully leakage-resilient signatures with graceful degradation. In *ICALP*, pages 456–468, 2015.

[FPV11] Sebastian Faust, Krzysztof Pietrzak, and Daniele Venturi. Tamper-proof circuits: How to trade leakage for tamper-resilience. In *ICALP*, pages 391–402, 2011.

[Fry00] Niklas Frykholm. Countermeasures against buffer overflow attacks. Technical report, RSA Data Security, Inc., November 2000.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31(4):469–472, 1985.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *EUROCRYPT*, pages 626–645, 2013.

[GHR99] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In *EUROCRYPT*, pages 123–139, 1999.

[GIP+14] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In *STOC*, pages 495–504, 2014.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, pages 281–310, 2015.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *ACM STOC*, pages 113–122, 2008.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27, 2015.

[GL10] David Goldenberg and Moses Liskov. On related-secret pseudorandomness. In *TCC*, pages 255–272, 2010.

[GLM+04] Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In *TCC*, pages 258–277, 2004.

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier cs-proofs. *IACR Cryptology ePrint Archive*, 2011:456, 2011.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *ACM STOC*, pages 218–229, 1987.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[GOR11] Vipul Goyal, Adam O'Neill, and Vanishree Rao. Correlated-input secure hash functions. In *TCC*, pages 182–200, 2011.

[GQ88] Louis C. Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In *CRYPTO*, pages 216–231, 1988.

[Gre14] Glenn Greenwald. No place to hide: Edward Snowden, the NSA, and the U.S. surveillance state. *Metropolitan Books*, May 2014.

[Gro06] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *ASIACRYPT*, pages 444–459, 2006.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, pages 305–326, 2016.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, pages 415–432, 2008.

[HC13] Steve Hargreaves and Stacy Cowley. How porn links and ben bernanke snuck into bitcoin's code. `http://money.cnn.com/2013/05/02/technology/security/bitcoin-porn/index.html`, 2013.

[Her11] Mathias Herrmann. Improved cryptanalysis of the multi-prime φ - hiding assumption. In *AFRICACRYPT*, pages 92–99, 2011.

[HJK12] Dennis Hofheinz, Tibor Jager, and Edward Knapp. Waters signatures with optimal security reduction. In *PKC*, pages 66–83, 2012.

[HK07] Dennis Hofheinz and Eike Kiltz. Secure hybrid encryption from weakened key encapsulation. In *CRYPTO*, pages 553–571, 2007.

[HK12] Dennis Hofheinz and Eike Kiltz. Programmable hash functions and their applications. *J. Cryptology*, 25(3):484–527, 2012.

[HKM12] Mike Hamburg, Paul Kocher, and Mark Marson. Analysis of Intel's Ivy Bridge digital random number generator. Technical report, Cryptography Research, Inc., 03 2012.

[Hop13] Curt Hopkins. If you own Bitcoin, you also own links to child porn. `http://www.dailydot.com/business/bitcoin-child-porn-transaction-code/`, 2013.

[HW09a] Susan Hohenberger and Brent Waters. Realizing hash-and-sign signatures under standard assumptions. In *EUROCRYPT*, pages 333–350, 2009.

[HW09b] Susan Hohenberger and Brent Waters. Short and stateless signatures from the RSA assumption. In *CRYPTO*, pages 654–670, 2009.

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Structure in Complexity Theory Conference*, pages 134–147, 1995.

[Inf] Blockchain Info. Bitcoin hashrate distribution. `https://blockchain.info/pools`.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *CRYPTO*, pages 369–386, 2014.

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits II: keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.

[IR88] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *CRYPTO*, pages 8–26, 1988.

[Jag15] Tibor Jager. Verifiable random functions from weaker assumptions. In *TCC*, pages 121–143, 2015.

[JK99] Benjamin Jun and Paul Kocher. The Intel random number generator. Technical report, Cryptography Research, Inc., 04 1999.

[JNO14] Thomas P. Jakobsen, Jesper Buus Nielsen, and Claudio Orlandi. A framework for outsourcing of secure computation. In *CCSW*, pages 81–92, 2014.

[JW15] Zahra Jafargholi and Daniel Wichs. Tamper detection and continuous non-malleable codes. In *TCC*, pages 451–480, 2015.

[KK12] Saqib A. Kakvi and Eike Kiltz. Optimal security proofs for full domain hash, revisited. In *EUROCRYPT*, pages 537–553, 2012.

[KKS11] Yael Tauman Kalai, Bhavana Kanukurthi, and Amit Sahai. Cryptography with tamperable and leaky memory. In *CRYPTO*, pages 373–390, 2011.

[KR00] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *NDSS*, 2000.

[KT13] Aggelos Kiayias and Yiannis Tselekounis. Tamper resilient circuits: The adversary at the gates. In *ASIACRYPT*, pages 161–180, 2013.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[KW03] Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In *ACM CCS*, pages 155–164, 2003.

[Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. Technical report, SRI International Computer Science Laboratory, 10 1979.

[LJM11] Eric Love, Yier Jin, and Yiorgos Makris. Enhancing security via provably trustworthy hardware intellectual property. In *IEEE HOST*, pages 12–17, 2011.

[LJMP90] Arjen K. Lenstra, Hendrik W. Lenstra Jr., Mark S. Manasse, and John M. Pollard. The number field sieve. In *ACM STOC*, pages 564–572, 1990.

[LKG+09] Lang Lin, Markus Kasper, Tim Güneysu, Christof Paar, and Wayne Burleson. Trojan side-channels: Lightweight hardware trojans through side-channel engineering. In *CHES*, pages 382–395, 2009.

[LL12] Feng-Hao Liu and Anna Lysyanskaya. Tamper and leakage resilience in the split-state model. In *CRYPTO*, pages 517–532, 2012.

[Luc04] Stefan Lucks. Ciphers secure against related-key attacks. In *FSE*, pages 359–370, 2004.

[Lys02] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In *CRYPTO*, pages 597–612, 2002.

[Mak15] Marie A. Mak. Trusted Defense Microelectronics: Future Access and Capabilities Are Uncertain. Technical report, United States Government Accountability Office, 10 2015.

[Man11] Alfonso Cevallos Manzano. Reducing the share size in robust secret sharing. Master's thesis, Mathematisch Instituut Universiteit Leiden, the Netherlands, 2011.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130, 1999.

[MS15] Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic reverse firewalls. In *EUROCRYPT*, pages 657–686, 2015.

[MWPB09] David R. McIntyre, Francis G. Wolff, Christos A. Papachristou, and Swarup Bhunia. Dynamic evaluation of hardware trust. In *IEEE HOST*, pages 108–111, 2009.

[NPS01] David Naccache, David Pointcheval, and Jacques Stern. Twin signatures: an alternative to the hash-and-sign paradigm. In *ACM CCS*, pages 20–27, 2001.

[NR94] Kaisa Nyberg and Rainer A. Rueppel. Message recovery for signature schemes based on the discrete logarithm problem. In *EUROCRYPT*, pages 182–193, 1994.

[NS12] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. *SIAM J. Comput.*, 41(4):772–814, 2012.

[NT01] Moni Naor and Vanessa Teague. Anti-presistence: history independent data structures. In *ACM STOC*, pages 492–501, 2001.

[NVZ14] Jesper Buus Nielsen, Daniele Venturi, and Angela Zottarel. Leakage-resilient signatures with graceful degradation. In *PKC*, pages 362–379, 2014.

[Oka92] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *CRYPTO*, pages 31–53, 1992.

[One96] Aleph One. Smashing the stack for fun and profit. *Phrack Magazine*, 7(49):File 14, 1996.

[OS90] H. Ong and Claus-Peter Schnorr. Fast signature generation with a Fiat Shamir-like scheme. In *EUROCRYPT*, pages 432–440, 1990.

[Ott06] Martin Otto. *Fault Attacks and Countermeasures*. PhD thesis, University of Paderborn, Germany, 2006.

[PB04] Jonathan D. Pincus and Brandon Baker. Beyond stack smashing: Recent advances in exploiting buffer overruns. *IEEE Security & Privacy*, 2(4):20–27, 2004.

[PB16] Kevin Petrasic and Matthew Bornfreund. Beyond bitcoin: The blockchain revolution in financial services. `http://www.whitecase.com/publications/insight/beyond-bitcoin-blockchain-revolution-financial-services`, 2016.

[Pea15] Jordan Pearson. The bitcoin blockchain could be used to spread malware, interpol says. `http://motherboard.vice.com/read/the-bitcoin-blockchain-could-be-used-to-spread-malware-interpol-says`, 2015.

[Per13] Nicole Perlroth. Government announces steps to restore confidence on encryption standards. `http://bits.blogs.nytimes.com/2013/09/10/government-announces-steps-to-restore-confidence-on-encryption-standards/?_r=0`, September 2013.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252, 2013.

[Pie09] Krzysztof Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482, 2009.

[PLS13] Nicole Perlroth, Jeff Larson, and Scott Shane. N.S.A. able to foil basic safeguards of privacy on web. *The New York Times*, September 2013.

[Pot10] Miodrag Potkonjak. Synthesis of trustable ics using untrusted CAD tools. In *DAC*, pages 633–634, 2010.

[PSas16] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. Cryptology ePrint Archive, Report 2016/454, 2016.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC*, pages 73–85, 1989.

[RB07] Phillip Rogaway and Mihir Bellare. Robust computational secret sharing and a unified account of classical secret-sharing goals. In *CCS*, pages 172–184, 2007.

[Rey11] Leo Reyzin. Lecture notes: Extractors and the leftover hash lemma, March 2011.

[SB15] Jean-Pierre Seifert and Christoph Bayer. Trojan-resilient circuits. In Al-Sakib Khan Pathan, editor, *Securing Cyber-Physical Systems*, chapter 14, pages 349–370. CRC Press, Boca Raton, London, New York, 2015.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[Sch12] Sven Schäge. Strong security from probabilistic signature schemes. In *PKC*, pages 84–101, 2012.

[SCO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, pages 566–598, 2001.

[Sco02] Mike Scott. Authenticated ID-based key exchange and remote log-in with simple token and PIN number. *IACR Cryptology ePrint Archive*, 2002:164, 2002.

[Seu14] Yannick Seurin. On the lossiness of the rabin trapdoor function. In *PKC*, pages 380–398, 2014.

[SF08] Christian Schridde and Bernd Freisleben. On the validity of the phi-hiding assumption in cryptographic protocols. In *ASIACRYPT*, pages 344–354, 2008.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[Sha07a] Hovav Shacham. A Cramer-Shoup encryption scheme from the linear assumption and from progressively weaker linear variants. *IACR Cryptology ePrint Archive*, 2007:74, 2007.

[Sha07b] Brian Sharkey. Trust in Integrated Circuits Program. Technical report, DARPA, 03 2007.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.

[Sho06] Victor Shoup. *A computational introduction to number theory and algebra*. Cambridge University Press, 2006.

[Sim83] Gustavus J. Simmons. The prisoners' problem and the subliminal channel. In *CRYPTO*, pages 51–67, 1983.

[Sim84] Gustavus J. Simmons. The subliminal channel and digital signature. In *EUROCRYPT*, pages 364–378, 1984.

[Sip92] Michael Sipser. The history and status of the P versus NP question. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 603–618, 1992.

[SRA81] Adi Shamir, Ronald L. Rivest, and Leonard M. Adleman. *Mental Poker*. Springer US, 1981.

[SS93] Jeffrey Shallit and Jonathan Sorenson. A binary algorithm for the jacobi symbol. *SIGSAM Bull.*, 27(1):4–11, January 1993.

[ST01] Adi Shamir and Yael Tauman. Improved online/offline signature schemes. In *CRYPTO*, pages 355–367, 2001.

[Ten] Jeni Tennison. What is the impact of blockchains on privacy? `https://theodi.org/blog/impact-of-blockchains-on-privacy`.

[U.S13] U.S Government Accountability Office. Financial regulatory reform: Financial crisis losses and potential impacts of the dodd-frank act. Technical report, 2013.

[VV83] Umesh V. Vazirani and Vijay V. Vazirani. Trapdoor pseudo-random number generators, with applications to protocol design. In *FOCS*, pages 23–30, 1983.

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.

[WB86] Lloyd R. Welch and Elwyn R. Berlekamp. Error correction of algebraic block codes, 1986. US Patent 4,633,470.

[WB15] Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Commun. ACM*, 58(2):74–84, 2015.

[Wee12] Hoeteck Wee. Public key encryption against related key attacks. In *PKC*, pages 262–279, 2012.

[WHG+16] Riad S. Wahby, Max Howald, Siddharth J. Garg, Abhi Shelat, and Michael Walfish. Verifiable asics. In *IEEE SP*, pages 759–778, 2016.

[WS11] Adam Waksman and Simha Sethumadhavan. Silencing hardware backdoors. In *IEEE Symposium on Security and Privacy*, pages 49–63, 2011.

[YY96] Adam L. Young and Moti Yung. The dark side of "black-box" cryptography, or: Should we trust Capstone? In *CRYPTO*, pages 89–103, 1996.

[YY97] Adam L. Young and Moti Yung. Kleptography: Using cryptography against cryptography. In *EUROCRYPT*, pages 62–74, 1997.