Zero-Knowledge Proofs in Theory and Practice

David Bernhard

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

April 17, 2014

Abstract

Zero-knowledge proof schemes are one of the main building blocks of modern cryptography. Using the Helios voting protocol as a practical example, we show mistakes in the previous understanding of these proof schemes and the resulting security problems. We proceed to define a hierarchy of security notions that solidifies our understanding of proof schemes: weak proof schemes, strong proof schemes and multi-proofs. We argue that the problems in Helios result from its use of weak proofs and show how these proofs can be made strong. We provide the first proof of ballot privacy for full Helios ballots with strong proofs.

In Helios, a proof scheme commonly known as Fiat-Shamir-Schnorr is used to strengthen encryption, a construction also known as Signed ElGamal or more generally, Encrypt+PoK. We show that in the Encrypt+PoK construction, our hierarchy of proof scheme notions corresponds naturally to a well-known hierarchy of security notions for public-key encryption: weak proofs yield chosen-plaintext secure encryption, strong proofs yield non-malleable encryption and multi-proofs yield chosen-ciphertext secure encryption.

Next, we ask whether Signed ElGamal is chosen-ciphertext secure, a question closely related but not identical to whether Fiat-Shamir-Schnorr proofs are multi-proofs. We answer both these questions negatively: under a reasonable assumption, the failure of which would cast doubt on the security of Schnorr-like proofs, we prove that Signed ElGamal cannot be shown to be chosen-ciphertext secure by a reduction to the security of plain ElGamal. This answers an open question, to our knowledge first asked by Shoup and Gennaro in a paper published in 1998.

Dedication and Acknowledgements

I dedicate this thesis to my grandfather, Roland Ernest Cove, from whom I have inherited my interest in mathematics. He passed away before I came to study here in Bristol but I am sure he would have been proud to have a grandson with a PhD.

My work over the last years would not have been possible without a tremendous amount of help from family, friends and academic collaborators. I would like to thank my supervisors, Nigel Smart and Bogdan Warinschi, all members of the Bristol cryptography group and my collaborators on published papers from other institutions: Véronique Cortier, Marc Fischlin, Stephan Neumann, Olivier Pereira, Ben Smyth and Melanie Volkamer.

I would also like to thank all my friends from the University of Bristol Explorers Club for the good times we have had together and the walks that have given me strength and energy, without which I could not have done my work of the past three years, or written this thesis.

Author's Declaration

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

Signed: ..... Date: .....

Contents

1 Introduction

A folklore saying in computer science, often attributed to Dijkstra, goes that "Computer science is not about computers, any more than surgery is about knives or astronomy is about telescopes". This, then, is not a thesis about encryption. If we had to settle on a single term to capture the essence of modern cryptography, the closest we can think of is "trust". On the one hand, cryptographic tools such as encryption or digital signatures aim to create trust — trust that a message cannot be read or tampered with by others; trust that a person really is who they say they are; trust that I am sending my credit card number to the shop I am trying to do business with and not to a fraudster.

Trust is a social concept, not a physical one. Nonetheless, physical items play their part in establishing and maintaining trust between humans, for example opaque paper envelopes in sealed-bid auctions, signatures on dotted lines or candle-wax seals. What these tools and techniques strive for in the physical realm, cryptography aims to replicate in the world of information. Information does not behave like an everyday object — it does not exist at a point in space and time and can in principle be freely duplicated but is notoriously hard to destroy — yet the existence and utility of information are not only beyond doubt but central to the society we live in.

The properties of information, as opposed to those of physical objects, allow cryptography to propose solutions that at a first glance look like contradictions. In Chapter 6 on voting, we analyse a system that is designed to not let anyone know anyone else's vote, yet at the same time everyone can check that all votes are counted correctly. One of the key tools in this process, which forms the main topic of this thesis, is known as a zero-knowledge proof scheme. Such a scheme is a recipe for two mutually distrustful parties to cooperate and achieve a common aim, without either of the two being able to cheat the other. The closest physical analogue of a zero-knowledge protocol is perhaps the "cut and choose" technique for two children to divide up a cake:

Definition 1.1 ("Cut and choose" cake protocol).

1. The first child cuts the cake into two pieces.
2. The second child selects a piece, leaving the remaining one for the first child.

The "security notion" achieved by this protocol is that neither child will have reason to complain that their piece of cake is smaller than the other piece. The principle of "cut and choose" is in fact used directly in sophisticated cryptographic protocols.

If cryptography is supposed to create trust, do we first have to trust cryptography? Since we started writing this thesis, revelations in the media have confirmed not only the obvious — that our spies engage in spying — but that there have been attempts to subvert cryptographic protocols and standards to leave open a "back door" for our intelligence agencies. Yet many people face a far bigger hurdle to trusting cryptography than possible compromise by our intelligence agencies: modern cryptography is based on "pure" mathematics such as number theory, inaccessible to a majority of the human population. While cryptography comes into play every time we withdraw money from an ATM, make a call on a mobile phone or visit all but the most basic of websites, the trust barrier to adopting cryptographic techniques in nationwide elections is high and rightly so.

One could be tempted to say that at least within the cryptographic community, one should publish every construction in full detail, develop mathematical security models and assumptions and verify each other's work in a spirit of constructive criticism. Yet we then face a different problem, as Halevi famously stated:

> ...as a community, we generate more proofs than we carefully verify (and as a consequence some of our published proofs are incorrect). [H05]

In the same year, Snow from the NSA pointed out a less academic view of the matter:

> We do not "beta-test" on the customer; if my product fails, someone might die. [S05]

There has been much debate on the role of security proofs in cryptography and their relation (or lack thereof) to practice. For us, a big concern is that cryptographic security proofs often consist of much drudgery in exchange for very little insight. As a consequence, not only are

our proofs not verified carefully but occasionally also written with a certain amount of handwaving and imprecision. In our previously published work, both on Direct Anonymous Attestation (DAA) and on voting, we have analysed schemes that were thought or even "proven" to be secure. In both cases we found otherwise. The problems in both cases were related to the use and modelling of zero-knowledge proof schemes. By mathematically analysing components of these schemes at a greater level of detail than had been done previously, we uncovered issues that had been omitted in earlier work.

We suggest that cryptographers must not only consider the process by which the community verifies security arguments but also actively seek to make these arguments as simple as possible to write and verify. We see one of the future challenges for cryptography in designing models and methods by which we can reduce the painstaking and time-consuming processes of proof creation and verification or better still, automate these processes.

The discipline known as formal methods (which Snow [S05] also suggested in passing) offers some hope of automating mundane proof-verification tasks; more recently tools such as EasyCrypt [EC] have appeared that aim to automate verification of a particular style of security analysis widely used in cryptography and in this thesis in particular, so-called game-based security. However, the tools available for automated analysis at the moment have a steep learning curve and require much knowledge and experience to use. We are not yet at the point where we could apply such tools to our own work although we consider this a valuable direction to pursue in the future. For now, we choose in this thesis to adopt an intermediate step, giving all security analysis in the code-based game-playing framework of Bellare and Rogaway [BR06] which aims for precision and the possibility of automation.

We are not yet satisfied with any approach for reasoning about zero-knowledge proofs including the one in this thesis, which is at most a passable approximation. One of our problems is that we rely on techniques that work well for reasoning about two-party systems yet the security models for proof schemes include further, hypothetical parties such as "simulators" and "extractors". In any case, we believe that there is much interesting future work in the area of zero-knowledge.

1.1 Publications

The following is an overview of our publications and their relation to this thesis. We analysed direct anonymous attestation (DAA), contributing to a new security model and fixing an earlier, broken scheme together with Fuchsbauer, Ghadafi, Smart and Warinschi [BF+11] in a work

published in the International Journal of Information Security. Later, we assisted Ghadafi and Fuchsbauer [BFG13] in constructing the first DAA scheme that does not require random oracles (to be introduced in Section 2.6.1) in work published at the International Conference on Applied Cryptography and Network Security (ACNS).

This thesis is not concerned with DAA. However, some ideas on security models in general and the flaws that we and others discovered a previously proposed and supposedly "proven secure" DAA scheme helped influence us to choose the code-based game-playing framework of Bellare and Rogaway as a basis for the security arguments in this thesis, hoping to avoid such problems. We introduce this framework formally in Section 2.5.2.

Our work on zero-knowledge proofs started with our analysis of the Helios voting system together with Cortier, Pereira, Smyth and Warinschi [BC+11] that we published at Esorics. We develop the ideas of this paper further in Chapter 6 of this thesis. We aimed to prove that Helios satisfies a property that we call ballot privacy (our Definition 6.5) if "done properly", since Cortier and Smyth had found some potential attacks on Helios. In a later work available on eprint [BPW12a], we weakened the conditions under which we could show ballot privacy. In this thesis, we show ballot privacy under even weaker preconditions, yielding the first proof to consider full Helios ballots rather than individual ciphertexts (Theorem 6.12). As part of our analysis of Helios, we needed to establish that the encryption used in Helios satisfies a security property known as non-malleability (addressed in Section 3.5). Based on the existing literature, we assumed this was obvious — but it turned out to be incorrect once we analysed a certain construction known as Encrypt+PoK (Section 5.2 in this thesis) in detail.

The problem with Helios was that it used a technique that we called the weak Fiat-Shamir transformation (discussed in Section 5.1.1) which turns out to be problematic. We wrote a paper on this transformation and its pitfalls with Pereira and Warinschi [BPW12b] which we published at Asiacrypt. In this thesis, we build on the results of the Asiacrypt paper in Section 5.1.

As a spin-off from our main line of work on zero-knowledge proofs, we compared our notion of ballot privacy to other ideas in the literature. With Cortier, Pereira and Warinschi [BCPW12] we defined a notion of privacy based on conditional entropy and showed that our original notion implies the entropy-based one; we published this work at ACM-CCS. With Smyth [BS13], we compared privacy to ballot independence, formalising the notions and showing their equivalence for "Helios-like" schemes. This work, like our original paper on Helios, was also published at Esorics. Independently, with Neumann and Volkamer [BNV13] we studied the practical applicability of so-called malleable proofs and published a paper at Vote-ID. We do not discuss the works cited in this paragraph any further in this thesis.

Shifting our focus from Helios to the zero-knowledge proofs themselves, we began to explore the different strengths in which these proofs come. When one uses such proofs in conjunction with encryption schemes then there is an interesting correspondence between security levels for encryption and proofs. We had shown in our Asiacrypt paper that the well-known "signed ElGamal" scheme is non-malleable (a "folklore" result, for which we could find no previous proof however). Yet it was an open problem, to our knowledge first asked by Shoup and Gennaro in 1998, whether this scheme also meets the stronger notion of chosen-ciphertext security (our Definition 3.2). With Fischlin and Warinschi, we defined a notion of multi-proofs (Definition 5.28) with which we can provably achieve chosen-ciphertext security (Theorem 5.40). Using a technique known as meta-reductions, we further showed that proofs based on the Fiat-Shamir transformation (Section 4.3.3) such as used in Helios are not multi-proofs (Theorem 5.32). By adapting both the techniques in this proof and the assumptions of the theorem, we finally showed that under plausible conditions, the encryption scheme under consideration is not chosen-ciphertext secure (Theorem 5.50), giving a negative answer to a long-standing open problem. The work referenced in this paragraph is currently in submission to Eurocrypt and as such, unpublished. We reproduce a large part of this work in Chapter 5 of this thesis, albeit with a slightly different presentation founded on code-based game-playing.

We give a complete list of our publications to date and their key topics in the following table.

reference	**published**	**topics**
[BF+11]	IJIS	DAA security models
[BC+11]	Esorics	ballot privacy definition, proof using CCA
[BPW12a]	eprint	ballot privacy from non-malleable encryption
[BPW12b]	Asiacrypt	weak proofs and attacks on Helios
[BCPW12]	ACM CCS	entropy-based notion of ballot privacy
[BFG13]	ACNS	DAA without random oracles
[BS13]	Esorics	relationship between ballot privacy and independence
[BNV13]	Vote-ID	practicality of mixnets based on malleable proofs

IJIS is the International Journal of Information Security; ACM CCS is the Association for Computing Machinery (conference on) Computer and Communications Security and ACNS is the (International Conference on) Applied Cryptography and Network Security.

1.2 Outline of this thesis

In Chapter 2 we introduce the mathematical tools required to reason about the security of cryptographic schemes. We begin by introducing concepts from algebra and complexity theory that appear throughout cryptography before we introduce game-based security starting from Section 2.4. We define the code-based game-playing framework in Section 2.5.2 and give a formal definition of a game in this framework (Definition 2.8) which we use in the rest of this thesis. In Chapter 3 we discuss public-key encryption and its security levels, the cornerstone being Definition 3.2. Much of our work on proof schemes is motivated by the desire to understand how they can interact with public-key encryption and how this affects these security levels. In Chapter 4 we introduce proof schemes and the main security notions such as zero-knowledge (Definition 4.4) and proof of knowledge (Definition 4.7).

Our main contributions are in Chapter 5. We first distinguish weak and strong proofs and highlight the weaknesses of weak proofs. Next, we introduce multi-proofs (informally: "even stronger proofs") in Definition 5.28 and prove a separation between strong and multi-proofs in Theorem 5.32. Along the way we apply these concepts to the Encrypt+PoK construction built upon ElGamal encryption and see that there is a correspondence between weak proofs and chosen-plaintext attacks, strong proofs and non-malleable encryption (Theorem 5.21) and multi-proofs and chosen-ciphertext attacks (Theorem 5.40). We further prove that strong proofs alone are insufficient to achieve chosen-ciphertext security in Theorem 5.50.

Finally, in Chapter 6 we define ballot privacy for voting and apply the theory from the previous chapters to show that Helios, with strong proofs, meets our notion. We summarise the main results of this thesis in the following table.

item	summary
Section 5.1.1	Problems with weak Fiat-Shamir.
Definition 5.5	Strong proofs.
Theorem 5.14	Strong Fiat-Shamir is strong and simulation sound.
Definition 5.28	Multi-proofs.
Theorem 5.32	Fiat-Shamir-Schnorr is not a multi-proof.
Theorem 5.40	Encrypt+multi-PoK yields CCA security.
Theorem 5.50	No reduction can show CCA security of Signed ElGamal.
Definition 6.5	Ballot privacy for voting systems.
Theorem 6.12	Helios (with strong proofs) satisfies ballot privacy.

2 Preliminaries

In this chapter we recall the mathematical foundations of cryptography. As undergraduates, we found ourselves perplexed the first time we encountered some of the mathematical tools and modes of reasoning used in a course on cryptography. Only later did it become clear that these were standard mathematical ideas, used widely across the whole field of computer science, albeit presented to us in new and different ways. In this chapter we therefore make a deliberate choice to present many of our tools as techniques from other areas of mathematics and computer science that cryptographers have borrowed and built upon. We will give hints in each section on cryptographic applications of these tools and then switch to discussing notions that are truly cryptographic in origin from Section 2.4 onwards.

2.1 Terminology and notation

We use the following terms frequently in this thesis which we introduce here in an informal manner. An algorithm is a non-deterministic always-halting Turing machine. Our algorithms will also be efficient in a complexity-theoretic sense, which we define in Section 2.3.1. The terms cryptographic primitive, scheme or protocol refer to the principal objects of study in this thesis and we often use them interchangeably depending on the context. A (security) game is one method of reasoning about the security of a scheme, discussed in more detail in Section 2.4.1. A reduction is an algorithm turning an attack against a scheme into an attack against a more basic primitive, with the aim of deriving security of the scheme from that of the primitive. Reductions are a technique of complexity theory and treated in more detail in Sections 2.3.5 and 2.4.1.

Notation. We summarise the notation in this thesis in Figure 2.1. We use $\twoheadrightarrow$ to denote randomised algorithms: by $\texttt{Encrypt} : PK \times M \twoheadrightarrow C$ we mean that `Encrypt` is a randomised algorithm taking inputs in PK and M and producing an output in C. This is equivalent to saying that there is some space R of "random coins" such that `Encrypt` is a function $PK \times M \times R \to C$.

notation	example	meaning
`monospace`	`Encrypt`, `challenge`	algorithm/procedure/oracle names
$\leftarrow$	$y \leftarrow f(x)$	assignment
$\rightarrow$	$f : X \rightarrow Y$	function definition
$\twoheadleftarrow$	$b \twoheadleftarrow \{0,1\}$	uniform random sampling
$\twoheadrightarrow$	$\texttt{Encrypt} : PK \times M \twoheadrightarrow C$	randomised algorithm
$\bot$		a special symbol denoting "failure"
$\cup$	$M \cup \{\bot\}$	disjoint union (coproduct)
$\mathbb{N}$		natural numbers, including 0
$[\,]$		empty list
::	$L \leftarrow L :: l$	append to list

Figure 2.1: Overview of notation in this thesis.

Cryptographic schemes. When we define a cryptographic scheme, we use the following layout and conventions: First, we give the sets that the scheme uses (for example, messages and keys etc.). Secondly, we give the algorithms of the scheme as maps on the sets on which they operate, for example $\texttt{DeriveKey} : SK \rightarrow PK$. Thirdly, we give any constraints (pre- or post-conditions or invariants) that must hold for all instances of the scheme; these are usually summarised in a property called correctness.

Security properties of a scheme are the subject of later, separate definitions. This separation allows us to define the "type" of a scheme first and then consider different levels or configurations of security properties. In this thesis the property called correctness applies exclusively to the requried behaviour of a scheme in the absence of adversaries; the terms "encryption scheme" and "correct encryption scheme" are thus synonyms whereas a "semantically secure encryption scheme", for example, is a specialisation thereof.

2.2 Group theory

We briefly review some of the basic notions of group theory. A group consists of a set $\mathbb{G}$ together with an operation $\star : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}$ that is associative, has a neutral element and has inverses. Often one does not distinguish between $\mathbb{G}$ as a set and as a group and uses the same symbol for both. If $\star$ is also commutative one calls the group commutative or Abelian. An important

example of a group is $\mathbb{Z}$ with the operation $\star$ defined as $+$, the integers with addition: one can build many other useful groups starting with $\mathbb{Z}$.

A ring consists of an Abelian group with an extra operation $\circ$ that is associative and distributes over $\star$ and has a neutral element. As a consequence of the distributive law, the neutral element of the group operation $\star$ annihilates under $\circ$, that is if e is the neutral element then for any $g \in \mathbb{G}$, $e \circ g = e = g \circ e$. An important example of a ring is again $\mathbb{Z}$ with $\star := +$ and $\circ := \cdot$, the usual operations of integer addition and multiplication. One often calls the neutral element of addition the zero of a ring and the operations, ring addition and ring multiplication. A ring is called commutative if its second operation (ring multiplication) is commutative; the first one (ring addition) is commutative by definition. If a commutative ring has inverses for any element except the zero (which would be impossible except if the ring contain exactly one element), it is called a field.

Given a basic structure, one can build further structures "over" the basic one. Given a field $\mathbb{F}$ with operations $+, \cdot$ and a zero 0, a vector space over $\mathbb{F}$ is a group $\mathbb{V}$ with an operation $\oplus : \mathbb{V} \times \mathbb{V} \to \mathbb{V}$ and an extra operation $\odot : \mathbb{F} \times \mathbb{V} \to \mathbb{V}$ that distributes over $\oplus$ and associates with $\cdot$, i.e. for any $f, g \in \mathbb{F}$ and any $V \in \mathbb{V}$ we have $(f \cdot g) \odot V = f \odot (g \odot V)$, further for the neutral element 1 of $\mathbb{F}$ under $\cdot$ we have $1 \odot V = V$ for any $V \in \mathbb{V}$.

The construction of a vector space never mentions the inverse of $\cdot$ in the field and we can do the same construction starting from a ring instead of a field; the resulting set and operations are then called a module over the ring.

A homomorphism is a map from one object of a category to another that preserves structure. For our purposes, the categories of interest are those of groups, rings, fields, vector spaces and modules — we do not define categories formally here. For example, a ring homomorphism from $(R, +, \cdot)$ to $(S, \oplus, \odot)$ is a map $h : R \to S$ such that for all $a, b \in R$ we have $h(a + b) = h(a) \oplus h(b), h(a \cdot b) = h(a) \odot h(b)$ and applying h to the neutral element of R (under $+$ or $\cdot$) yields the corresponding neutral element of S.

Group exponentiation. For any group $\mathbb{G}$ with operation $\star$, neutral element e and inverse operation $G \mapsto \widehat{G}$, there is an operation called group exponentiation

$$\mathbb{Z} \times \mathbb{G} \to \mathbb{G}, \quad (z, G) \mapsto G^z := \begin{cases} e & z = 0 \\ \underbrace{G \star G \star \ldots \star G}_{z \text{ times}} & z > 0 \\ (-z) \star \widehat{G} & z < 0 \end{cases}$$

Fixing a group element G, exponentiation is a group homomorphism $\mathbb{Z} \to \mathbb{G}$ as $G^{z+w} = G^z \star G^w,\ G^0 = e,\ G^{-z} = \widehat{G^z}$ as one can easily check. If $\mathbb{G}$ is Abelian then for any fixed $z \in \mathbb{Z}$, the operation $G \mapsto G^z$ is also homomorphic as $G^z \star H^z = (G \star H)^z,\ e^z = e,\ \widehat{G}^z = \widehat{G^z}$.

For $G \in \mathbb{G}$ we can define the span $\langle G \rangle := \{G^z \mid z \in \mathbb{Z}\}$ which is itself a group, the "subgroup of $\mathbb{G}$ generated by G". If there is a $G \in \mathbb{G}$ such that $\langle G \rangle = \mathbb{G}$ we say that G is a generator of $\mathbb{G}$ and that $\mathbb{G}$ is cyclic. Cyclic groups need not be infinite: the group $\mathbb{Z}_n$ for $n \in \mathbb{N}$ consists of the set $\{0, 1, \ldots, n-1\}$ with addition "modulo n" and is a finite cyclic group with generator 1. (In fact all finite cyclic groups are isomorphic to such a $\mathbb{Z}_n$ and the number n is called the order of the group.)

If $\mathbb{G}$ is a finite cyclic group and G a generator thereof then for any $H \in \mathbb{G}$ there is a unique $z \in \mathbb{Z}_{|\mathbb{G}|}$ such that $H = G^z$. In fact, in this case group exponentiation to base G is an isomorphism between $\mathbb{G}$ and $\mathbb{Z}_{|\mathbb{G}|}$ (with operation $+$). In the expression $H = G^z$, we call z the discrete logarithm of H to base G.

This operation is of supreme importance in cryptography. Looking ahead, if we have a cyclic group $\mathbb{G}$ in which the group operation and group inversion are efficiently computable then group exponentiation to a given generator is also efficiently computable (via the so-called "square and multiply" method and a number of optimisations thereof). However, taking the discrete logarithm is one of the operations that is believed to be "computationally hard" in suitable groups; giving us a construction upon which one can base much of modern cryptography.

2.3 Complexity theory

Much of cryptography builds upon notions of complexity theory, adding the extra dimension of "security". For example, consider a safe with an n-digit code. The owner who knows the code to open the safe has to enter n digits. However, an adversary with no knowledge of the code to our safe has a search space of 10^n possible codes which is exponentially larger than n. Similarly, in a cryptographic scheme with an n-bit key the workload for legitimate participants will typically be a low-degree polynomial in n whereas the effort to break a secure scheme should be exponential in n; one of our aims as cryptographers is to argue that the workload to break our schemes is indeed infeasibly large. We give a short introduction to some complexity-theoretic preliminaries in order to formalise these notions.

Languages. In theoretical computer science, an alphabet Σ is any finite set, for all practical purposes we can restrict ourselves to considering $\{0, 1\}$. Elements of an alphabet are called

characters or symbols. A word over an alphabet is a sequence of characters, including the empty word that contains no characters. The length of a word w is denoted by $|w|$. The set of all words of length n over an alphabet Σ is denoted Σ^n and the set of all finite words over Σ is denoted Σ^*, formally defined as $\bigcup_{n\in\mathbb{N}} \Sigma^n$. This set is therefore countably-infinite. A subset L of Σ^*, i.e. a set of words, is a language over Σ. We can for all practical purposes assume that any set appearing in this thesis is some subset of $\{0,1\}^*$.

Turing machines. A deterministic Turing machine decides a language L if for any word in L it outputs "accept" and for any word of Σ^* not in L it outputs "reject", in both cases taking at most a finite number of steps which may depend on the word in question. A non-deterministic Turing machine decides a language L if it always terminates in a finite number of steps on any input and random choices, and for any word $x \in L$ there is some choice of random values that makes it output "accept" when run on these inputs. For other random choices, the decider may mistakenly output "reject" but for any word $y \notin L$, no matter what random choices it makes, it always eventually outputs "reject". A string of random values r that cause a decider to output "accept" on input a word $x \in L$ is called a witness or "proof of membership" for x.

Landau O-notation. A central quantity in complexity theory is the running time of an algorithm in relation to the length of its inputs. Formally, we can express such a running time by a function $T : \mathbb{N} \to \mathbb{N}$ where $T(n)$ is the maximal number of steps an algorithm takes on any input of length n (assuming it always halt). We define the set $F(\mathbb{N})$ as the set of all functions $\mathbb{N} \to \mathbb{N}$; several important definitions concern particular subsets of $F(\mathbb{N})$.

Complexity-theoretic notions such as efficiency of an algorithm can be expressed concisely as subsets of $F(\mathbb{N})$ using Landau O-notation, perhaps better known to computer scientists as "big-O" notation. This notation describes the set of functions that grow asymptotically at most as fast as a given function f, i.e. such functions g where the limit $g(n)/f(n)$ as $n \to \infty$ is bounded (in this formulation, assuming $f(n)$ does not take the value 0 for large n). We define the notion in an equivalent manner that does not involve limits.

Definition 2.2 (Landau O). For a function $f \in F(\mathbb{N})$,

$$O(f) := \{g \in F(\mathbb{N}) \mid \exists c, n_0 \in \mathbb{N}\ \forall n \geq n_0\ c \cdot f(n) \geq g(n)\}$$

O-notation can be used in a slightly informal manner where the meaning is clear: we say that "an algorithm takes $O(n)$ steps" to mean that the function measuring the number of steps the algorithm takes is an element of $O(n)$.

2.3.1 Efficiency

The first important notion of complexity theory for cryptographers is that of an efficient algorithm, which attempts to model computations that are feasible to carry out in practice. Efficiency as defined in complexity theory and cryptography is an asymptotic notion, a characterisation of the running time of an algorithm on inputs of size n as n tends to infinity.

Taking an algorithm to mean a Turing machine that halts in finite time on any input, we would like to define a subset $\mathbf{E} \subseteq F(\mathbb{N})$ and call an algorithm efficient if its asymptotic running time is an element of $\mathbf{E}$. We consider only deterministic algorithms for the following motivation. If we start from the following four postulates,

1. An efficient algorithm shall be able to execute one step, for any input length.
2. An efficient algorithm shall be allowed to read its entire input.
3. Given two efficient algorithms, the algorithm that runs these two in sequence shall still be efficient.
4. An efficient algorithm shall be able to call other efficient algorithms as subroutines.

we get the conditions (1) $(n \mapsto 1) \in \mathbf{E}$, (2) $(n \mapsto n) \in \mathbf{E}$, (3) $f, g \in \mathbf{E} \Rightarrow (n \mapsto f(n) + g(n)) \in \mathbf{E}$, (4) $f, g \in \mathbf{E} \Rightarrow (n \mapsto f(n) \cdot g(n)) \in \mathbf{E}$. This gives us, as the smallest possible class $\mathbf{E}$, the set of all functions bounded by polynomials in n, which is the usual definition of the running time of efficient algorithms:

$$\mathbf{E} \;=\; \bigcup_{k \in \mathbb{N}} O(n \mapsto n^k) \;=\; \{f \mid \exists c, k \in \mathbb{N}\; \forall n \in \mathbb{N}\; f(n) \leq c \cdot n^k\}$$

The closure properties (1)–(4) of the set $\mathbf{E}$ mean we get the same notion if we make slight changes to the notion of "step" or execution model, or use an alphabet with more than two characters. This observation also motivates the common choice to ignore such details when talking about efficient algorithms.

Definition 2.3 (efficient). A deterministic algorithm is efficient if there is a constant $k \in \mathbb{N}$ such that on any input x, letting $n = |x|$ be the length of the input, the algorithm takes at most $O(n^k)$ steps.

To extend this notion to randomised algorithms, one usually gives such an algorithm access to an unbounded source of random bits but asks that the number of steps taken be polynomially bounded in the length of the input alone, counting the drawing of a random bit as a single step.

2.3.2 P and NP

Switching our viewpoint from algorithms to languages, the classes $\mathbf{P}$ and $\mathbf{NP}$ are the classes of languages decidable by efficient deterministic and non-deterministic algorithms respectively: informally, $\mathbf{P}$ is the class of (decision) problems one can solve efficiently whereas $\mathbf{NP}$ is the class of problems which can be solved efficiently given suitable "help" or whose solutions can be checked efficiently.

Many interesting problems in cryptography fall into the $\mathbf{NP}$ category and one hopes that they do not fall into $\mathbf{P}$, putting aside for a moment the distinction between decision problems (where the output can be seen as one bit) and more general computation problems. For example, given a ciphertext and a suitable key, computing the contained message should be an efficient operation whereas without the key this should be infeasible.

Definition 2.4 (Complexity class P). The complexity class $\mathbf{P}$ is the class of all languages $L \subseteq \{0,1\}^*$ such that there exists a deterministic Turing machine M that decides L and a constant $k \in \mathbb{N}$ such that for any input word w, letting $n = |w|$ be the length of w, M halts in at most $O(n^k)$ steps.

Among several ways to define $\mathbf{NP}$, we choose one that aligns closely with the definition of $\mathbf{P}$.

Definition 2.5 (Complexity class NP). The complexity class $\mathbf{NP}$ is the class of all languages $L \subseteq \{0,1\}^*$ such that there exists a *non*-deterministic Turing machine M that decides L and a constant $k \in \mathbb{N}$ such that for any input word w, letting $n = |w|$ be the length of w, M halts in at most $O(n^k)$ steps.

Side note — BPP, P ?= NP. The common notion of the "class of efficiently solvable (decision) problems" is called $\mathbf{BPP}$ and contains all decision problems that can be solved by an algorithm with access to an unbounded source of random bits, running in a number of steps polynomially bounded by the length of its input and which may give an incorrect answer with probability no more than one third. Drawing a random bit counts as one step but random bits to not contribute to the input length; the constant of one third is arbitrary. It is clear that $\mathbf{P} \subseteq \mathbf{BPP}$ and further conjectured that the two might be equal. We do not mention $\mathbf{BPP}$ any further in this thesis and omit a formal definition. It is equally obvious that $\mathbf{P} \subseteq \mathbf{NP}$; whether the two are equal ranks among the greatest unsolved problems in mathematics and a positive answer might have devastating results for cryptography.

2.3.3 NP relations

A relation R on a pair of sets X, W is a subset of $X \times W$ or, equivalently, a function $X \times W \to \{0,1\}$. We use the notation $(x,w) \in R$ and $R(x,w) = 1$ synonymously. A relation R induces a set

$$L(R) := \{x \in X \mid \exists w \in W \; R(x,w) = 1\}$$

When X is the set of words over some alphabet (for example, $\{0,1\}$), $L(R)$ is a language over this alphabet, hence the name. An $\mathbf{NP}$ relation is a relation where the decision problem associated to its induced language is in $\mathbf{NP}$:

1. There exists a polynomial p such that for every $x \in L(R)$ there exists some w such that $|w| \leq p(|x|)$. Call the restriction of R to such pairs (x,w) of the above form $R^{\wp}$. (It is obvious that $L(R) = L(R^{\wp})$.)

2. There exists an algorithm that on input a pair (x,w), takes time polynomial in the length of x alone and for any $(x,w) \in R^{\wp}$ outputs 1; for any (x,w) with $x \notin L(R)$ it outputs 0.

In this case we often call an $x \in L(R^{\wp})$ an instance and a corresponding w a witness. The reason that the length of witnesses cannot be arbitrary is that for a statement such as "x is a satisfiable circuit" with on the order of $|x|$ inputs, a "witness" w consisting of $2^{|x|}$ zeroes would buy an efficient algorithm enough time to check all possible inputs. A satisfying assignment to the circuit however is a valid witness.

Cryptographers are especially interested in **NP** relations where it is hard to compute a witness given an instance. Curiously, cryptographically "useful" relations are all but disjoint from known **NP**-complete ones (to be defined in Section 2.3.5).

2.3.4 Negligibility

A negligible function is one that vanishes (converges to zero) faster than the inverse of every polynomial. In cryptography, negligible functions serve, as their name suggests, to model risks one chooses to neglect when declaring a system to be secure. For example, in any system using a n-bit key, there is a 2^{-n} chance of guessing the key in one attempt, which is negligible (both in the sense of our definition, and in practice for suitably large n); an adversary making an efficient (polynomial in n) number of guesses will still have negligible guessing probablity as the limit of $p(n)2^{-n}$ converges to zero for any polynomial p and for $n \to \infty$.

Definition 2.6 (Negligible). The set of negligible functions is the subset of $F(\mathbb{N})$ defined by

$$\mathbf{Negl} := \{f : \mathbb{N} \to \mathbb{N} \mid \forall k \in \mathbb{N}\, \exists n_0 \in \mathbb{N}\, \forall n \geq n_0\, f(n) < n^{-k}\}$$

For probabilities, the "dual" notion to negligible is overwhelming: a function f with range $[0, 1]$ is overwhleming if $(n \mapsto 1 - f(n))$ is negligible. In cryptography we often demand that desirable properties hold with overwhelming probability, for example that encrypting a message and then decrypting the resulting ciphertext with the correct keys yield the original message.

2.3.5 Reductions

A reduction is a technique to show "hardness" of some problem given the hardness of a different problem. For example, once we have established that the Halting Problem is not decidable, we can show other problems to be undecidable by a reduction to the Halting Problem. The key step

in a reduction can seem counter-intuitive as it is performed "backwards": to show that problem X is hard given that problem Y is hard (for reasonable notions of "hard" and "problem"), give an algorithm that takes an instance of Y and turns it into an instance of X.

Perhaps the most famous examples of reductions come from the concept of NP-completeness. A problem X in $\mathbf{NP}$ is NP-complete if $X \in \mathbf{P}$ would imply $\mathbf{NP} \subseteq \mathbf{P}$, i.e. $\mathbf{P} = \mathbf{NP}$. Cook's famous theorem [C71] that 3-SAT is NP-complete is an abstract reduction from 3-SAT to *any* NP problem. Following on from this result, many problems in $\mathbf{NP}$ have been shown to be NP-complete by concrete reductions to 3-SAT (directly or indirectly, via other NP-complete problems); a paper of Karp [K72] gives 21 famous NP-complete problems and reductions.

In cryptography, reductions are one of the main techniques for reasoning about the security of cryptographic schemes. Most of our security arguments in this thesis will be based on reductions.

2.4 Security models

We are now ready to address topics specific to cryptography and present several different approaches used by cryptographers to reason about security.

2.4.1 Security games and reductions

We give an overview of the game-playing paradigm using terminology borrowed from software engineering. A cryptographic primitive or scheme can be seen as an interface, containing a number of procedure names and signatures and constraints such as pre- and post-conditions or invariants. For example, an encryption scheme contains key generation, encryption and decryption algorithms and constraints such as that encrypting a message then decrypting it with a matching key returns the original message. The ElGamal public-key encryption scheme [E85], for example, is an implementation of the encryption scheme interface.

Game-based security aims to capture security properties as further procedure signatures and constraints on the interfaces of schemes or primitives. We formally define for each interface what it means for an implementation of this interface to be "broken". For example, an implementation of encryption might be declared broken if there exists a procedure `break` that takes a ciphertext and returns the plaintext, without access to any keys. Such an extension of an interface with a notion of "brokenness" is called a cryptographic game.

Game-based security arguments take the form of reductions between interfaces that propagate the notion of "breaking", just like reductions in complexity theory propagate the notion of

"solving". The following example and Figure 2.7, while very informal, illustrate this point. The reduction from 3-COLOURING to 3-SAT is by Karp [K72] and has nothing to do with cryptography. This reduction contains an algorithm that takes a 3-SAT instance (a circuit) and creates a graph that is 3-colourable if and only if the circuit is satisfiable. Therefore, if there is an algorithm that solves 3-COLOURING, there is also an algorithm to solve 3-SAT. Conversely, if 3-SAT is hard to solve then so is 3-COLOURING. The reduction from semantic security of ElGamal encryption to the decisional Diffie-Hellman problem is a standard security argument in cryptography. Here, the reduction's algorithm takes a so-called Diffie-Hellman instance and creates an ElGamal ciphertext. Therefore, if there is an algorithm to break ElGamal then there is also an algorithm to break Diffie-Hellman instances. Conversely, if Diffie-Hellman instances are secure then so is ElGamal.

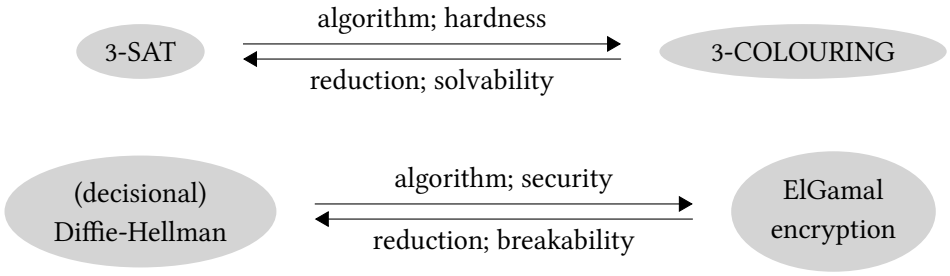


Figure 2.7: Well-known reductions in complexity theory and cryptography.

2.4.2 Simulation-based security

Simulation-based security is an alternative to the game-based approach. Here, one starts with a description of an ideal protocol or functionality, to be run by a trustworthy party. An ideal voting protocol for example might involve all voters announcing their vote to the trustworthy party in private, after which the party announces the election result to everyone in turn [G04]. To prove a real protocol secure, one shows that it emulates the ideal protocol in the following manner: for every attacker against the real protocol, there is a simulator that can perform an equivalent attack against the ideal protocol. The fact that the ideal protocol excludes certain attacks by design implies that the corresponding attacks cannot occur in the real protocol either.

Among the formalisations of the simulation-based approach, the Universal Composability (UC) framework by Canetti [C01] is perhaps the most famous. As the name suggests, a key feature of this framework is a composition theorem that says, informally, that if (1) a protocol

P (for example, a voting protocol) emulates an ideal functionality F given access to an ideal functionality G for some primitive (for example, encryption), and (2) Q is a real protocol that emulates functionality G, then (3) P given access to Q also emulates F.

2.4.3 Dolev-Yao models

Protocols for key exchange, authentication or similar "higher-level" constructions, where primitives such as encryption or digital signatures are assumed to be available, should be able to handle many different sessions between numerous parties that may run concurrently. This places formidable difficulties in front of a would-be security prover who must deal with any possible combination of interactions in numerous sessions. (Some versions of the UC model can handle multiple sessions but even "basic" UC security is hard to show: the protocols one encounters in practice are typically not proven to be UC secure.)

An approach pioneered by Dolev and Yao [DY83] views protocols at an abstract level as an algebra of data types and terms, allowing for an easier mathematical treatment while abstracting away any details of the underlying schemes. This approach has lead to a whole field of security analysis known as "formal methods", with concepts such as the applied pi-calculus [AF01, RS11] and tools such as Coq [Coq], to name but one. The ideal in this approach is to have proofs checked and where possible even constructed in a fully automatic manner.

Formal methods are a great tool for finding certain classes of attacks. Cortier and Smyth for example [CS11, CS13] applied formal methods to Helios and found issues with ballot privacy leading to numerous further papers including several of our own. Formal methods are comparatively easy to automate and their security notions are in our opinion often the easiest to understand of any of the types of security modelling presented in this thesis.

If a formal analysis does not find any attacks on a protocol however, concluding that the protocol is secure is another matter. Formal methods capture protocols at a high level of abstraction compared to cryptographic games and it is not always clear if the components in a protocol securely implement this abstraction: there is a whole field of research known as computational soundness [AR00] that aims to establish under what conditions an abstract formalisation applies to given primitives. For example, the issues with Helios that we discovered recently [BPW12b] only manifest themselves at a lower level of abstraction than the level chosen by Cortier and Smyth [CS11, CS13] in their analysis.

2.5 Game-based security

Among the models presented above, we find game-based security the most suitable for our purposes. Game-based security allows for the most modular security arguments and allows us to spell out important low-level details more easily than the other models, first treating each aspect of a system in isolation before combining them into a high-level view.

In the long term, we hope that game-based proofs will be able to be formally specified and verified with the same level of automation as Dolev-Yao style proofs while retaining their expressiveness to handle the level of detail necessary for cryptography. The EasyCrypt framework [EC] by Barthe et al. is a tool that attempts to offer formal verification of game-based proofs, however it requires a background in automated theorem-proving as well as cryptography and we have not yet had the time to learn how to use it. With this future application in mind we choose to express our games and security arguments in a pseudocode language based on the code-based game playing framework of Bellare and Rogaway.

2.5.1 Security parameters

In cryptography, it is common to use a security parameter that is provided as a (usually implicit) extra input to all algorithms. Security parameters are often denoted by the Greek letter λ and provided in unary notation, for which one writes 1^λ. This allows us to talk about, e.g. key generators as efficient algorithms, which take no actual inputs but have the sole task of drawing random bits and "formatting" them as cryptographic keys. While the concepts of key-length and security parameter are related, they are distinct: a key-length is a parameter of a practical implementation of a scheme which one believes to have some security properties for some fixed value whereas a security parameter is a concept used in the theoretical analysis of asymptotic security properties.

In this thesis, unless stated otherwise we make the convention that a security parameter is implicit in all algorithms and spaces. Thus if we say a decryption algorithm is an efficiently computable map $d : C \times K \to M$ (where C is a set of ciphertexts, K of keys and M of messages) we mean that for every $\lambda \in \mathbb{N}$ there are spaces $C_\lambda, K_\lambda, M_\lambda$ and a map $d_\lambda : C_\lambda \times K_\lambda \to M_\lambda$ which is computable by an algorithm running in polynomial time in its inputs (c, k) and the implicit parameter 1^λ.

2.5.2 The code-based game-playing framework

Code-based game-playing is a framework by Bellare and Rogaway for specifying games and reductions between them formally and precisely. Their paper [BR06] first appeared officially at Eurocrypt 2006 although the paper had already been known and commented on well before that — the first "eprint" version is dated 2004.

Code-based games are written in a particular programming language. Variables can be of type integer, string, set of strings, array indexed by strings and boolean; all variables can take the special value $\perp$ (undefined). Uninitialised variables hold the following values: strings and sets are empty, integers zero, booleans false and arrays undefined at every index. In the original presentation, boolean variables once set to true could not be reset, we do not make this convention here. The language supports computation and comparison, assignment, random assignment from a finite set, conditionals or "ifs", iterations ("for") over finite sets and a return operation. This language guarantees that all algorithms written in it will terminate in finite time (it is deliberately not Turing-complete). We give the definition of a game based on their framework.

Definition 2.8 (game, adversary). A game is a collection of procedures or "oracles" which may include ones named `initialise` and `finalise`. Procedures of a game may share state between each other and between calls (all variables are global unless specified otherwise).

An adversary is an algorithm that can send and receive values over some external interface. To run a game with an adversary, one first runs the procedure `initialise` and gives its output to the adversary.

The adversary may call any of the oracles except `initialise` and `finalise`. Unless specified otherwise, the adversary may call all other oracles any number of times and in any order. The adversary may terminate by returning a value. We pass this value to the `finalise` procedure, which may itself return a value. These two return values are called the adversary output and game output respectively.

2.5.3 Success probability and distinguishing advantage

Security games commonly come in one of two forms, modelling different types of security properties: bad event-type games and indistinguishability games.

1. A "bad event" game asks the adversary to trigger a certain event.
2. An indistinguishability game asks the adversary to guess which of two scenarios she is interacting with.

"Bad event" games capture trace properties, security requirements that can be refuted by looking at the trace of a single execution of an adversary interacting with a scheme. For example, we may declare an encryption scheme broken if the adversary manages to output the secret key. The adversary may trigger the bad event in such a game by supplying a value meeting particular conditions or the bad event may be a condition on the game's internal state, such as two secret values colliding.

In a "bad event" game, the finalisation procedure outputs 1 if and only if the bad event has happened. Alternatively one can take the finalisation procedure's output in an execution as the definition of the bad event. The probability that a particular adversary triggers this event is known as the adversary's success probability against the game and the associated security property is the postulate that no (efficient) adversary has a non-negligible success probability.

Indistinguishability games model properties which cannot be captured by a single trace of an execution. For example, in an encryption scheme we may ask that an adversary be unable to guess whether a ciphertext represents the bit 1 or 0. In this case, the adversary always has a one half chance of success by guessing at random and the security property asks that no adversary can do better than this. We cannot tell from a single execution where the adversary happened to guess correctly, whether the adversary has broken the scheme or just been lucky.

There are two equivalent presentations of indistinguishability games. In the first, there is a single game that begins by choosing a bit $b \twoheadleftarrow \{0,1\}$ which the game may use later on in its procedures. The `finalise` procedure takes a bit g (for "guess") as input from the adversary and returns 1 if $b = g$ (the guess was correct), otherwise 0. In the second form, one gives a pair of games with the same procedure names and the adversary outputs a guess g as to which of the two games she is interacting with. The `finalise` algorithm, which we leave implicit, takes a bit g as input and outputs $\neg g$ in one game and g in the other.

The adversary's success probability in an indistinguishability game is still the probability that the game outputs 1 but it should ideally be close to one half rather than zero. We can convert this quantity into a function that should be negligible in the usual sense by measuring the difference between the probabilities of the adversary returning 1 in the cases $b = 0$ and $b = 1$ or equivalently, between the two games. This new quantity is called the adversary's distinguishing advantage against the game.

Definition 2.9 (success probability, distinguishing advantage). In a "bad event" type game, the adversary's success probability σ is the probability that the game returns 1.

In an indistinguishability game, let G be a random variable denoting the adversary's guess and let B be a random variable for the game's random bit. Then, the success probability of an adversary is the probability $\sigma := \mathbf{Pr}[G = B]$ and her distinguishing advantage is $\alpha := \mathbf{Pr}[G = 1 \mid B = 1] - \mathbf{Pr}[G = 1 \mid B = 0]$.

For indistinguishability games, a conditional probability argument on B, using that its distribution is uniform, gives the equation $\sigma = 1/2 + \alpha/2$ or equivalently $\alpha = 2\sigma - 1$, so the two notions completely determine each other. Depending on the notion in question, success probability or distinguishing advantage may be easier to compute directly.

2.5.4 "Old style" game-based notation

In the code-based game-playing framework, we define a game by a set of procedures; the adversary's success probability or distinguishing advantage can be left implicit. For readers more familiar with an older way of presenting games and advantages as experiments and random variables, we give two examples to help translation to the code-based style.

For our first example, we define one-wayness of a function $f : X \to Y$ in both "old" and "new" styles.

Definition 2.10 (One-way function). A function $f : X \to Y$ is one-way if for any efficient algorithm $\mathcal{A}$, the following probability $\mathbf{Succ}^{\mathrm{OW}}_{\mathcal{A},f}$ is negligible where F is an oracle that on input an $x \in X$ returns $f(x)$.

$$\mathbf{Succ}^{\mathrm{OW}}_{\mathcal{A},f} := \mathbf{Pr}\left[x^* \twoheadleftarrow X; y^* \leftarrow f(x^*); x' \leftarrow \mathcal{A}^{F(\cdot)}(y^*) : x^* = x'\right]$$

Code-based definition. A one-way function is defined by the following game.

```
1  oracle initialise
2    x* ↞ X
3    y* ← f(x*)
4    return y*
5
6  oracle F(x)
7    return f(x)
8  oracle finalise(x′)
9    if y* = f(x′) then
10     return 1
11   else
12     return 0
13   end if
```

We can translate this simple game from code-based to "experiment" style by defining the success probability as

$$\mathbf{Succ}_{\mathcal{A}} := \mathbf{Pr}\left[in \leftarrow \texttt{initialise}(); out \leftarrow \mathcal{A}^{\texttt{oracles}}(in) : \texttt{finalise}(out) = 1\right]$$

The problem with this notation is that it does not scale to more complex games involving more than two parties. When we define multi-proofs, there will be an extractor, a simulator and multiple provers all interacting in a "concurrent" manner. Indeed, the notation $\mathcal{A}^{\texttt{oracle}()}()$ suggests that the adversary drives the execution and calls the oracles as subroutines. This viewpoint is unsatisfactory in a "rewinding" setting, where the game may have many copies of the adversary running at the same time. Our presentation of code-based games deliberately shifts the focus from the success probability to the game itself — the one entity that exists only once in the entire system, handles the initial set-up and produces the final output. The adversary $\mathcal{A}$ never appears in the code-based definition at all: it is sufficient to define the functionality of the game on its own, independently of other algorithms such as the adversary that it might interact with.

As a second example, we define a pseudorandom function $f : X \to Y$ using indistinguishability games.

Definition 2.11. A function $f : X \to Y$ is pseudorandom if for any efficient adversary $\mathcal{A}$ the following distunguishing advantage $\mathbf{Adv}^{\text{PRF}}_{\mathcal{A},f}$ is negligible. Here $F : X \to Y$ is a random function, i.e. for any $x \in X$ the value $F(x)$ is uniform in Y and independent of the values of F at any other $x' \neq x$.

$$\mathbf{Adv}^{\text{PRF}}_{\mathcal{A},f} := \mathbf{Pr}\left[\mathcal{A}^{f(\cdot)}() = 1\right] - \mathbf{Pr}\left[\mathcal{A}^{F(\cdot)}() = 1\right]$$

Code-based definition. A pseudorandom function is defined by the following games.

Game "L"

```
1 oracle F(x)
2   return f(x)
```

Game "R"

```
3 oracle F(x)
4   if V[x] = ⊥ then
5     V[x] ← Y
6   end if
7   return V[x]
```

We chose to spell out the random function in the code-based presentation. Again, in this simple case there is a straightforward translation into "experiment" notation. This particular game does not require an initialisation algorithm. The finalisation algorithm for an indistinguishability-based game presented as a pair of games can always be left implicit as it simply returns the adversary's guessed bit.

2.5.5 The fundamental principle of indistinguishability games

We give a high-level overview of how we work with indistinguishability games. Examples in this thesis include showing IND-CPA security of ElGamal encryption from the DDH assumption or ballot privacy of "minivoting" (leading up to Helios) from non-malleability of the deployed encryption scheme.

Start with a primitive P that has a security notion defined using a pair of games P_L, P_R that both offer the same interface I^P and that should be indistinguishable for the primitive to be secure. From this, construct a scheme S with security defined as indistinguishability of a pair of games S_L, S_R whose interface we call I^S. A security reduction from S to P is an efficient algorithm R which makes use of an algorithm offering interface I^P and itself offers interface I^S, such that $R \circ P_L = S_L$ and $R \circ P_R = S_R$. Then for any adversary A using interface I^S with distinguishing advantage α between S_L and S_R, $A \circ R$ is an adversary using interface I^P with advantage α between P_L and P_R.

The above discussion serves to give a useful intuition of the process involved in proving reductions between industinguishability games. We caution however that the notation used is very informal, despite having the appearance of mathematical formulae: we never formally defined what we mean by composition of algorithms, for example. While this may be possible to

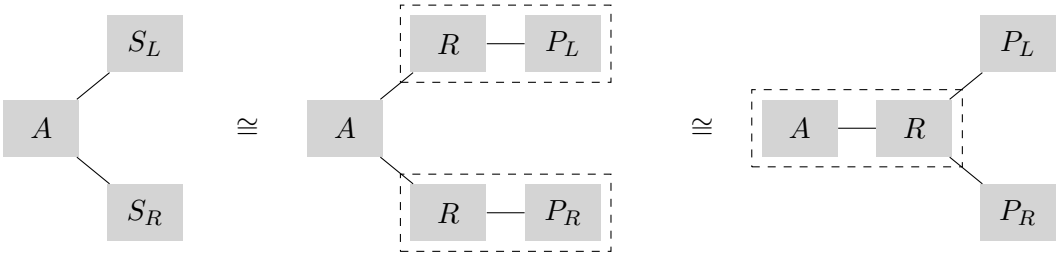


Figure 2.12: Reduction R from games S_L/S_R to P_L/P_R.

formalise easily for the simple situation presented above, in "practice" reductions can involve multiple primitives, external algorithms and models such as random oracles, black-box access to multiple instances of the same algorithm and losses in the distinguishing advantage and further complications. We say this because an obvious question to ask is whether it be possible to prove reductions entirely at an abstract level and dispense with some of the detailed analysis that one currently needs to repeat in each security argument. Our answer is negative: we are not aware of any framework that allows security arguments to be conducted entirely at an abstract level and that "scales" beyond trivial cases to schemes of the complexity of those that we wish to analyse. For the time being we find it useful to have these abstractions or "sketches" in mind while designing security arguments but a sketch should not be confused with an actual security argument.

2.6 Random oracles and common reference strings

Apart from specific computational assumptions, two general ones often called "models" are central to cryptography: the random oracle (RO) model and the common reference string (CRS) model. Unlike say the DDH assumption, these models are not statements that we assume particular computations to be infeasible. Instead, these are statements about the environment in which a scheme executes. These environments capture the properties necessary to deploy, for example, non-interactive zero-knowledge proofs, where one of the central arguments is that while no prover can make a convincing proof unless she have some witness to the truth of a statement (or, in a weaker form, that the statement simply be true), there exists a hypothetical simulator that can do exactly that in a thought-experiment: make proofs irrespective of the truth of the statement in question. For this to work, the simulator must have some power not accorded to any real prover and the random oracle and common reference string models describe two different envi-

ronments which allow us to reason about a simulator with extra powers over the environment. It is also possible to have an environment with both random oracles and common reference strings.

2.6.1 Random oracles

A random oracle is an algorithm that takes inputs from some domain, returns outputs from some range and acts as follows: for any input never queried before, it picks a random element of its range, stores the input/output mapping and returns the new element. For inputs queried previously, the random oracle answers consistently. A random oracle can thus be seen as an efficient instantiation via lazy sampling of a random function. A protocol is in the random oracle (RO) model if all algorithms have implicit access to one or more random oracles.

As the domain of a random oracle one can usually choose $\{0,1\}^*$ so that one can use "anything" as input to the random oracle. We assume the range $\mathcal{R}$ of the random oracle is finite, efficiently samplable and exponentially large in the security parameter, for example $\{0,1\}^\lambda$ for security parameter λ. This makes the probability of guessing an oracle output in advance negligible.

Random oracles were introduced by Bellare and Rogaway [BR93] to formalise a concept of replacing "securely chosen" random bits by applications of cryptographic hash functions. The original paper contains two applications, both of which we address in this thesis: first, they construct and strengthen public-key encryption; secondly, they analyse the derivation of non-interactive proofs from interactive ones using the technique commonly attributed to Fiat and Shamir [FS86]. These proofs, in both interactive and non-interactive instantiations, are efficient enough that they have seen use in practice. Indeed, part of this thesis concerns their application in the Helios voting system. The idea known as the Fiat-Shamir transformation is to replace an interactive verifier, whose job is to pick a value chosen uniformly at random after the prover has sent her some values (and thus committed to them), with the value of a hash function taken on the prover's previous values. The random oracle model provides a basis upon which one can build a formal analysis of this transformation.

2.6.2 Common reference strings

A common reference string generator is an efficient algorithm `GenCRS` that returns a value from some distribution, which we call a common reference string. The role of a common reference string is twofold: in the real world, it is simply available to all algorithms as an extra input. This property is no different from a setup parameter, such as the description of a group and its

operations over which a cryptographic protocol is to be performed. What distinguishes common reference strings is that in security arguments, one can work with CRSs generated by a different algorithm that come with a "trapdoor", such that they look like a proper CRS to anyone without the trapdoor but accord extra abilities to anyone with it.

For example, the Pedersen commitment scheme [P91] picks a CRS consisting of two generators G, H in a group such that the discrete logarithm of one to the other is unknown to anyone. To commit to a value x, one chooses a random r and computes the commitment $G^x H^r$; to reveal the committed value one reveals both x and r. The idea here is that if you can find an $x' \neq x$ and an r' such that $G^x H^r = G^{x'} H^{r'}$ then you can take also the discrete logarithm of H to base G, which is assumed to be a hard problem. In security proofs, one can pick $H = G^t$ and give the trapdoor value t to some fictional party, allowing her to change committed values: if she has committed to $C = G^x H^r$ but would prefer to reveal $x' = x + \delta$, she can compute the required value as $r' = r - \delta/t$ (assuming a group of prime order, hence the exponents are in a finite field).

2.6.3 Formal definitions

A random oracle and CRS generator/oracle are given in Figure 2.13. The range $\mathcal{R}$ of the random oracle must be exponentially large in the security parameter and the sampling operation (in line 3) must be efficient. The trapdoor generator(s) are not part of the definition of the CRS model itself but of particular schemes that one can instantiate in this model.

```
1  oracle RO(x)
2    if H[x] = ⊥ then
3        H[x] ↞ R
4    end if
5    return H[x]

6  oracle CRS(x)
7    if crs = ⊥ then
8        crs ← GenCRS()
9    end if
10   return crs
```

Figure 2.13: Random oracle and common reference string generator.

A protocol is in the random oracle (RO) model if all algorithms have implicit access to (at least) one random oracle and in the common reference string (CRS) model if all algorithms have implicit access to (at least) one common reference string. In the security notions, the oracles and generators can be treated specially. For example, we will call a proof system (RO/CRS)–zero-knowledge if there is a simulator such that the simulator, when put in charge of oracle/string queries, is indistinguishable from some other party together with a real oracle/generator.

2.7 Computational assumptions

Computational assumptions are, broadly speaking, the security properties found on the lowest levels of a stack of cryptographic schemes. Common examples concern the hardness of certain operations in groups, often related to the taking of discrete logarithms.

On problems and assumptions. Computational assumptions are asymptotic in nature. It is meaningless to claim that taking discrete logarithms is asymptotically hard in a particular finite group $\mathbb{G}$ — there is always a constant-time brute-force algorithm where the constant is the group order. Instead, we should say that we assume taking discrete logarithms is asymptotically hard in a particular family of groups $(\mathbb{G}_\lambda)_{\lambda\in\mathbb{N}}$. However, we view shifting our point of view from individual groups to families of groups as unhelpful — we can formulate most interesting relations between computational assumptions and cryptographic constructions "locally". For example, in any group $\mathbb{G}$ an algorithm that takes discrete logarithms yields one that also solves the computational and decisional Diffie-Hellman problems in essentially the same running time and an attack on the IND-CPA property of ElGamal encryption over a group $\mathbb{G}$ yields an attack on the DDH property in the same group $\mathbb{G}$ with a loss of $1/2$ in distinguishing advantage (Lemma 3.7).

Motivated by this observation, we choose to write "problem" for a property relating to a single group (formulated as a security game) and "assumption" for the corresponding security property induced on families of groups. Statements on problems imply asymptotic statements on assumptions: the example above implies that in any family of groups $(G_\lambda)_{\lambda\in\mathbb{N}}$, if the DDH assumption holds then ElGamal encryption defined over this family is IND-CPA secure. In the rest of this section we will give problems and leave the assumptions implicit. For example, when we give the DLOG problem in Defintion 2.14, it is understood that the DLOG assumption holds in a family of groups $\mathbb{G}_\lambda$ indexed by a security parameter $\lambda \in \mathbb{N}$ if there is no efficient algorithm that on input 1^λ (the security parameter in unary notation) and some description of the group $\mathbb{G}_\lambda$, solves the DLOG problem (i.e. wins the game in Definition 2.14) with non-negligible probability in $\mathbb{G}_\lambda$ as λ tends to infinity.

Generating and describing groups. In the last paragraph we mentioned algorithms that take as input "some description of a group" which is part of a family. Let us make this statement slightly more precise. A family of groups $(\mathbb{G}_\lambda)_{\lambda\in\mathbb{N}}$ can be formally described by a mapping that takes as input a value $\lambda \in \mathbb{N}$ and outputs a (description of a) group $\mathbb{G}_\lambda$. It is clear that there can be many different such mappings so we should not read $\mathbb{G}_\lambda$ as "*the* group for security level λ", indeed the expression $\mathbb{G}_\lambda$ is not defined except in the context of a particular mapping. Statements about

families of groups such as "taking discrete logarithms is hard" can also be interpreted as referring to particular mappings into the collection of all groups. (Since the collection of all groups is a proper class not a set, we should not call such a mapping a function either. This point seems of little practical relevance however.)

Such mappings are sometimes named explicitly, for example Seurin and Treger [ST13] define a group generator `GpGen` as an algorithm that on input 1^λ outputs a group of prime order roughly 2^λ along with its order and a particular generator. Seurin and Treger then define assumptions such as CDH or DDH relative to group generators (note that they use "problem" where we would say "assumption"):

> "We say that the CDH problem is hard relatively to `GpGen` if the following advantage ... [CDH game] ... is negligible for any PPT adversary A." [ST13]

As described above, in this thesis we instead take the "local" view and prefer to define a problem over and individual group; the above form of definition can unambiguously be recovered from the local one. At the end of this section we give a couple of more concrete examples of group generators.

Discrete logarithms. In any finite, cyclic group $\mathbb{G}$ with generator G, any group element H has a unique value $x \in \mathbb{Z}_{|\mathbb{G}|}$ such that $H = G^x$, where $(x, G) \mapsto G^x$ is group exponentiation.

Definition 2.14 (DLOG problem). Let $\mathbb{G}$ be a cyclic group with generator G. The discrete logarithm problem is to compute x given a challenge $H = G^x$, where $x \twoheadleftarrow \mathbb{Z}_{|\mathbb{G}|}$ is uniformly random. The associated game is in Figure 2.15.

Diffie-Hellman. Given a cyclic group $\mathbb{G}$ with generator G and operation $\cdot$, we can define a further operation $\otimes : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ by $H \otimes K := G^{h \cdot k}$ where $H = G^h$ and $K = G^k$ are the discrete logarithms of the operands and the product in the exponent is taken in the integers. This operation has many interesting properties, above all it is assumed not to be efficiently computable in common cryptographic groups yet is clearly easy to compute if one knows at least one of the discrete logarithms involved. This operation forms the basis of both the Diffie-Hellman key exchange protocol [DH76] and the ElGamal encryption scheme [E85]. We state the assumption that this product is hard to compute:

```
1  oracle initialise
2    x ↞ Z_|G|
3    H ← G^x
4    return H

5  oracle finalise(y)
6    if x = y then
7        return 1
8    else
9        return 0
10   end if
```

Figure 2.15: The DLOG game.

Assumption 2.16 (CDH). Let $\mathbb{G}$ be a finite cyclic group with generator G. The computational Diffie-Hellman or CDH problem is for uniformly random $h, k \twoheadleftarrow \mathbb{Z}_{|\mathbb{G}|}$ and $H \leftarrow G^h, K \leftarrow G^k$, to compute $H \otimes K = G^{h \cdot k}$ on input H, K. The game for CDH is in Figure 2.18.

The Diffie-Hellman product is assumed to be not only hard to compute without either of the discrete logarithms but even hard to verify:

Assumption 2.17 (DDH). In a group $\mathbb{G}$ with generator G, the decisional Diffe-Hellman or DDH problem is for uniformly random $h, k, z \twoheadleftarrow \mathbb{Z}_{|\mathbb{G}|}$ and $H \leftarrow G^h, K \leftarrow G^k, Z \leftarrow G^z$ to distinguish $(H, K, H \otimes K)$ from (H, K, Z). The game for DDH is in Figure 2.18.

A triple of the form (H, K, L) with $L = H \otimes K$ is called a Diffie-Hellman triple and one can phrase the DDH problem as distinguishing whether a given triple has this property or not.

Relations. It is trivial to see that an algorithm breaking the DLOG assumption can be used to break the CDH assumption too and solving CDH in turn solves DDH. In the opposite direction, no efficient reductions are known and one assumes that the heirarchy DLOG $-$ CDH $-$ DDH is strict. Indeed, there are groups in which DLOG is assumed to be hard yet DDH is easy, namely

groups equipped with so-called bilinear pairings that have found many applications in cryptography. The assumption that CDH is hard even in the presence of a DDH oracle is sometimes referred to as the gap Diffie-Hellman assumption.

CDH.

```
1  oracle initialise
2    h, k, ← ℤ_|𝔾|
3    H ← G^h, K ← G^k
4    return H, K
5
6  oracle finalise(L)
7    if L = H^k  then // = H ⊗ K
8        return 1
9    else
10       return 0
11   end if
```

DDH.

```
1  oracle initialise
2    b ← {0, 1}
3    h, k, z ← ℤ_|𝔾|
4    H ← G^h, K ← G^k, Z ← G^z
5    L ← H^k  // = H ⊗ K
6    if b = 0 then
7        return H, K, L
8    else
9        return H, K, Z
10   end if
11
12 oracle finalise(g)
13   if b = g then
14       return 1
15   else
16       return 0
17   end if
```

Figure 2.18: Games for the CDH and DDH problems.

One-more discrete logarithm. Based on the discrete logarithm problem, we consider the following game. You have access to many discrete logarithm challengers, must solve the challenge of any one and can "open" any others to get their value of x. Formally, you get even more: you may make as many "instances" as you like and ask discrete logarithm queries on as many arbitrary elements as you like. You win if you solve all instances and have made fewer discrete logarithm queries than instances.

Definition 2.19 (OMDL assumption). The one-more discrete logarithm problem or OMDL is to win the game in Figure 2.20. The adversary may call the `instance` and `dlog` oracles many times, in any order.

```
1  oracle initialise
2    i ← 0, j ← 0
3    D ← [ ]
4    return
5
6  oracle instance
7    x ↞ ℤ_|𝔾|
8    D[i] ← x
9    i ← i + 1
10   return G^x
11
12 oracle dlog(Y)
13     j ← j + 1
14     return dlog_G(Y)
15 oracle finalise(x⃗)
16   if i = 0 or j ≥ i or
17       |x⃗| < i then
18       return ⊥
19   end if
20   for k = 0 to i − 1 do
21       if x_k ≠ D[k] then
22           return 0
23       end if
24   end for
25   return 1
```

Figure 2.20: One-more discrete logarithm.

Here, i records the number of instances obtained and j the number of instances opened so winning conditions include $i > 0$ and $i > j$. We allow ourselves to index the vector $\vec{x}$ starting from 0 instead of 1 to match the array D. In Line 14, the `dlog` oracle takes a discrete logarithm. This operation is not efficiently computable if the DLOG assumption holds in $\mathbb{G}$ so we have an inefficient game. Since games are only thought-experiments and often used in a "black box" manner anyway, this does not matter: the OMDL notion still makes sense. There is a weaker version of this notion that is sufficient for some of our purposes however which we will also mention here and which does yield an efficient game: instead of a `dlog` oracle, the adversary gets an `open` oracle taking a vector $\vec{c}$ of length i as input (for the current value of i) and after incrementing j by 1 returns $\sum_{k=0}^{i-1} c_k \cdot D[k]$. The adversary wins the game if she provides all

discrete logarithms of the instances and the linear span of the discrete logarithms she opened is of smaller dimension than the problem space spanned by the instances.

Some specific families of groups. In practice, two common constructions yield cryptographically useful families of groups. The first is to pick, on input a $\lambda \in \mathbb{N}$, a strong prime q of order approximately λ bits, i.e. $2^\lambda < q < 2^{\lambda+1}$. A strong prime is a prime q such that $p = (q-1)/2$ is also prime. Taking $\mathbb{Z}_q^\times$ as our base group, we pick any element g that has order p in this group, yielding a cyclic group $\mathbb{G}$ of order p generated by g as a subgroup of the base group. Group addition is defined as $a +_\mathbb{G} b := (a + b) \pmod q$ and group exponentiation becomes modular exponentiation modulo q.

The second construction creates cyclic groups $\mathbb{G}_\lambda$ as order-p subgroups of the group of points on elliptic curves, where p is related to the desired security parameter. An element of such a group is either a point (x, y) whose components satisfy a particular equation such as $y^2 = x^3 + ax + b \pmod p$ for constants a, b, or a special "point at infinity". Toegether with a particular formula for point addition, the set of such points on a curve forms a group with the point at infinity as its neutral element. Details of this construction and methods for choosing suitable curves are outside the scope of this thesis.

3 Public-key encryption

Public-key encryption was first proposed, at least in public, by Diffie and Hellman in their 1976 paper "New Directions in Cryptography" [DH76] and first implemented by Rivest, Shamir and Adleman (RSA) in 1978 [RSA78]. Among the later proposals for public-key encryption schemes, the 1985 one by ElGamal [E85] stands out: many later schemes are extensions of the original ElGamal one. ElGamal encryption uses the same mathematical principles as Diffie and Hellman's original construction for key exchange. The ElGamal scheme and extensions thereof using proof schemes (Chapter 4) form the subject matter of a large part of this thesis.

The founding paper for the mathematical analysis of the security of public-key encryption was written by Goldwasser and Micali in 1984 [GM84]. In this paper, they not only proposed a model for encryption-scheme security but proposed the very concept of modelling security formally in the first place:

> "This paper proposes an encryption scheme with the following property: *whatever is efficiently computable about the cleartext given the cyphertext, is also efficiently computable without the cyphertext.*" [GM84]

Formally, this is the notion of semantic security (w.r.t. functions), which as a side-effect creates a need for encryption to be non-deterministic. These ideas (among others) gained Goldwasser and Micali the 2012 Turing Award, the Association of Computing Machinery crediting them with "[turning] cryptography from an art into a science" [ACM].

3.1 Formal definition

We follow the spirit of Bellare et al.'s overview paper [BDPR98]. A public-key encryption scheme contains a key generation algorithm that produces a public key and a secret key. With the public key, one can use the encryption algorithm to encrypt messages, producing ciphertexts. The decryption algorithm, on input a ciphertext and a matching secret key, will output the message. We let the decryption algorithm output a special symbol $\perp$ to denote an invalid ciphertext.

Definition 3.1 (Public-Key Encryption). A public-key encryption scheme is given by the following spaces and algorithms.

Spaces.	
M	message space
PK	public key space
SK	secret key space
C	ciphertext space

Algorithms.	
`KeyGen`	$\twoheadrightarrow PK \times SK$
`Encrypt`	$PK \times M \twoheadrightarrow C$
`Decrypt`	$SK \times C \to M \cup \{\perp\}$

A public-key encryption scheme must satisfy the following correctness property: for any $m \in M$, letting

$$(pk, sk) \leftarrow \texttt{KeyGen()}, \quad c \leftarrow \texttt{Encrypt}(pk, m), \quad m' \leftarrow \texttt{Decrypt}(sk, c)$$

we have $m = m'$.

We are using our convention that there is a security parameter available to all algorithms and that the algorithms are efficient, meaning polynomial-time in the length of their inputs and the security parameter.

Dependent spaces. A new feature of our style of definitions is that we explicitly name all spaces and give signatures of all algorithms based on these spaces. We believe that this allows us greater precision in describing certain constructions such as the way the Encrypt+PoK transformation turns one encryption scheme into another (Definition 5.16). Further, the more "strongly typed" approach that this style of definition allows may have uses in computer-aided verification of security arguments — we have EasyCrypt [EC] in particular in mind.

This comes at the cost of some generality however; our style does not apply directly to schemes with dependent spaces. In the famous RSA encryption scheme, the key generation algorithm does not run on a given group but rather chooses parameters that make up a group itself; each keypair thus describes a separate group whose order must be kept secret. RSA encrypts messages that can be mapped into this group: not only does the message space vary with each keypair but a sender will typically not even know the exact message and ciphertext spaces since this would allow her to decrypt as well and defeat the purpose of a public-key scheme.

In this thesis, we restrict ourselves to schemes based on discrete logarithms and groups whose order depends only on the security parameter and can therefore be made public. Since none of the constructions in this thesis involve dependent spaces, we believe that the benefits of clearly defining all spaces up front outweigh the loss of generality that this choice incurs.

3.2 Defining security

A security notion for encryption should both formalise intuitive security requirements and be comfortable to work with formally (i.e. prove schemes secure under this notion). Goldwasser and Micali [GM84] gave two notions, which we present in an informal manner since we only use them as motivation.

Semantic security After picking a function f on the message space, it should be no easier to guess $f(m)$ given an encryption of a random message m than guessing $f(m)$ for an unknown, random message m directly.

Indistinguishability It should be infeasible to find a pair of messages such that, given an encryption of either message chosen at random, you have a better-than-random chance of guessing which message was encrypted.

Indistinguishability was called polynomial security in the original paper and semantic security can be named more precisely by adding "with respect to functions", since a separate notion with respect to relations exists too. Semantic security is the property that perhaps best captures the intuitive demand for security: that it be not only infeasible to extract messages from ciphertexts without the key, but also the first bit of the encrypted message, the parity of all bits in the message etc. — indeed, any function of the message.

Goldwasser and Micali proved that polynomial security (indistinguishability) implies semantic security. Since indistinguishability is also the easier of the two notions to work with, it has become the standard notion in cryptography.

3.3 The IND security game

We present an abstract security game that captures a class of indistinguishability-based notions. Initially, the game creates a keypair and gives the adversary the public key. We use the presentation in which the game picks a random bit and the adversary's aim is to guess this bit: a scheme

will be called secure with respect to a version of this game if no efficient adversary can guess the bit better than at random with non-negligible probability. Once in the game, the adversary may pick two messages m_0 and m_1 of equal length. The game will encrypt m_b and give the adversary the resulting ciphertext c^*; this is the only use of the bit b.

Versions of this game differ in if and when the adversary may ask the game to decrypt ciphertexts. In the weakest instantiation, IND-CPA, the game will never decrypt anything. In the strongest, CCA2, the adversary may at any time ask for a decryption of anything except c^*. In IND-CCA1, a notion mainly of historical interest, the adversary may only decrypt before receiving c^*. Finally, in IND-1-CCA, not to be confused with IND-CCA1, the adversary may only ask the game to decrypt once, after seeing c^*, but may ask for any number of ciphertexts (except c^*) to be decrypted. This variation makes the adversary's decryptions non-adaptive in the sense that she cannot wait to see the decryption of her first ciphertext before choosing the second one.

Definition 3.2 (IND security). The IND security notions are given by the game in Figure 3.3 where the `decrypt` oracle can be called as described in the following table.

notion	`decrypt` available
IND-CPA	never
IND-CCA1	only before calling `challenge`
IND-CCA2	always
IND-1-CCA	once, after `challenge`, with a list of ciphertexts

In the IND-1-CCA game, the `decrypt` oracle processes each ciphertext in the list individually and returns the list of results.

In other words, a family of encryption schemes $(E_\lambda)_\lambda$ indexed by a security parameter λ is said to have an IND security property if the advantage of any efficient adversary against the relevant game instantiated for E_λ is negligible as a function of λ. Equivalently, one may say that a particular construction of an encryption scheme out of a group such as ElGamal (Section 3.4) has an IND security property in a particular λ-indexed family of groups.

A remark on the message lengths. The condition that the two challenge messages be of the same length is to prevent a trivial guess based on the length of the returned ciphertext. To encrypt

```
1  oracle initialise
2    (pk, sk) ← KeyGen()
3    b ↞ {0, 1}
4    return pk

5  oracle finalise g
6    return b = g

7  oracle challenge(m_0, m_1)
8    // may only be called once
9    // precondition: |m_0| = |m_1|
10   c* ← Encrypt(pk, m_b)
11   return c*

12 oracle decrypt(c)
13   // when this can be called depends on the
14   // exact notion in question.
15   if c = c* then // false if c* not yet defined
16       return ⊥
17   else
18       return Decrypt(sk, c)
19   end if
```

Figure 3.3: Indistinguishablity-based security notions for encryption.

messages of arbitrary lengths, it is usual to split messages into "blocks" and then encrypt a block at a time; in which case it may be appropriate to weaken the condition to the two messages being the same number of blocks in length. For the public-key schemes we consider, the message space will usually be a group and all valid messages will have the same length of "1 group element". In this case the length restriction becomes redundant.

Historical note. IND-CCA1 was defined by Naor and Yung [NY90] who originally called it security against chosen-ciphertext attacks before Rackoff and Simon [RS91] defined IND-CCA2, also known as security against adaptive chosen ciphertext attacks. The former notion can therefore also be called security against non-adaptive chosen ciphertext attacks or "lunchtime attacks". When one speaks of chosen-ciphertext security nowadays, sometimes just called CCA, the notion meant is that of IND-CCA2. (We will see why the "IND" part can be dropped too in a minute.) IND-1-CCA is the notion of Bellare and Sahai [BS99].

Real or random. For some proofs it is easier to present the IND game in a "real or random" version, where the challenge oracle takes one message m as input and returns either a ciphertext for m or a ciphertext for a randomly sampled $m' \twoheadleftarrow M$, depending on the bit b. We present, in an abstract manner, an efficient reduction transforming an IND adversary with advantage α into a "real or random" adversary with advantage $\alpha/2$.

Lemma 3.4. If there is an adversary with advantage α against an IND game as presented above then there is an adversary of the same class (the reduction between the two is efficient) against the "real or random" version of the corresponding IND game with advantage $\alpha/2$.

Proof. Recall that the advantage of an adversary against an indistinguishaiblity game is $\alpha := \mathbf{Pr}[G = 1 \mid B = 1] - \mathbf{Pr}[G = 1 \mid B = 0]$ where G is a random variable for the adversary's output and B for the game's random bit. Let p_0 be the probability that the adversary outputs 1 when interacting with the game where $B = 0$ and p_1 be the same probability when $B = 1$, i.e. $\alpha = (p_1 - p_0)$. Assume w.l.o.g. that the "real or random game" picks a bit b' and returns "real" ciphertexts when $b' = 1$ and random ones when $b' = 0$. Our reduction picks a random bit $c \twoheadleftarrow \{0, 1\}$. To handle a challenge query, it receives two messages m_0, m_1 and passes m_c on to the game, returning the received ciphertext. When the adversary makes her guess g, the reduction outputs 1 if $g = c$, otherwise 0. All other queries (the initial messages and decryption queries if the game in question allows them) our reduction forwards between the adversary and the game. The advantage of our reduction is $\mathbf{Pr}[G' = 1 \mid B' = 1] - \mathbf{Pr}[G' = 1 \mid B' = 0]$ where G' is the reduction's output and B' the "real or random" game's bit.

- In the case $B' = 1$, the game returns real ciphertexts so the reduction's output G' is exactly the original adversary's success probability, since the reduction's output is 1 if the adversary guessed c correctly. We thus have $\mathbf{Pr}[G' = 1 \mid B' = 1] = 1/2 + \alpha/2$.

- In the case $B' = 0$, the game returns random ciphertexts and ignores the input to the challenge oracle. The probability of the adversary guessing c is thus $1/2$ exactly.

Taken together, these two points give the reduction an advantage of $\alpha' = \alpha/2$, as claimed. *q.e.d.*

3.4 ElGamal

We present the ElGamal encryption scheme [E85] that achieves IND-CPA security under the DDH assumption. Extending ElGamal to achieve higher security notions is a topic in the next chapter.

Definition 3.5 (ElGamal). Let $\mathbb{G}$ be a finite cyclic group of prime order (written multiplicatively) and G a generator. The ElGamal encryption scheme is defined as follows: $M = \mathbb{G}, PK = \mathbb{G}, SK = \mathbb{Z}_{|\mathbb{G}|}, C = \mathbb{G} \times \mathbb{G}$ and the algorithms are in Figure 3.6.

```
algorithm KeyGen
    sk ↞ Z_|G|
    pk ← G^sk
    return pk, sk

algorithm Encrypt(pk, m)
    r ↞ Z_|G|
    A ← G^r
    B ← pk^r · m
    return A, B

algorithm Decrypt(sk, C)
    parse (A, B) ← C
    m ← B/A^sk
    return m
```

Figure 3.6: ElGamal encryption.

Correctness of ElGamal is easy to verify. As to security,

Lemma 3.7. ElGamal encryption is IND-CPA secure if DDH holds in the underlying group. More precisely, an adversary with advantage α against IND-CPA of ElGamal yields one against DDH with advantage $\alpha/2$.

Proof. We give a reduction from the "real or random" version of IND-CPA security of ElGamal to DDH in Figure 3.8. The reduction receives a challenge triple (A, B, C) from the DDH game where C is either G^{ab} (for the values of a, b such that $A = G^a, B = G^b$) or C is random in G. In the former case, the reduction coupled with the DDH challenger operates identically to the "real" part of the IND-CPA game since a is the "secret key", b the randomness used to encrypt and $C = (G^a)^b$. In the latter case, since $C = G^c$ is random in $\mathbb{G}$, we can write $C \cdot m$ as G^{ab+r} where $r = c - ab + \mu$ for μ the discrete logarithm of m to basis G. Since c was uniform, so is r. Therefore we have a ciphertext for a random message, just like in the "random" game. We conclude that this reduction preserves the adversary's advantage. Since the step from IND-CPA to the "real or random" version halved the advantage, the claim in the lemma follows. *q.e.d.*

```
1  oracle initialise(A, B, C)
2    return A
3
4  oracle finalise(g)
5    return g
6  oracle challenge(m)
7    c* ← (B, C · m)
8    return c*
```

Figure 3.8: Reduction from IND-CPA of ElGamal encryption to DDH.

3.5 Non-malleability

Consider a voting scheme where voters encrypt their votes with the voting authority's public key and publish the ciphertexts. The authority will decrypt all ciphertexts at the end of the election and compute the tally. IND-CPA security guarantees that Eve, a dishonest voter, cannot decrypt Alice's ciphertext and thus learn how she voted. But this is not enough: we also want to prevent Eve from taking Alice's ciphertext and modifying it to a ciphertext for exactly the opposite vote, then recasting this as her own vote. In this case, Eve might not be able to decrypt Alice's ciphertext or even her own one, but she can still influence the election unfairly as long as the authority can decrypt both ciphertexts to tally.

This property was first considered by Dolev, Dwork and Naor in 1991 [DDN91]. (Their original example concerned auctions; the application to voting is based on work by Cortier and Smyth [CS11, CS13].) Informally, what we want can be described as semantic security with respect to *relations*:

Semantic security w.r.t. relations After seeing a challenge ciphertext for random message, it should be hard to produce a new ciphertext and a relation on messages such that the relation is more likely to hold between the new encrypted message and the challenge message than between the new message and a random message unrelated to the challenge.

Bellare and Sahai [BS99] proved several definitions of non-malleability to be equivalent. The "classical" one [BDPR98] is NM-CPA, which we present in Figure 3.9 as a pair of games that should be indistinguishable to efficient adversaries. In the first game, the adversary may call `challenge` once with a description of a distribution D on messages, upon which the game draws a message m^* from this distribution and returns an encryption c^* of it (this generalises the choice of exactly two messages in IND notions). In the second game, the `challenge` oracle draws two messages from the distribution and returns an encryption of the first. The adversary may provide a vector of ciphertexts and a relation to the `finalise` oracle. None of these ciphertexts may be identical to the challenge ciphertext, or the game just returns 0. The game then decrypts the adversary's ciphertexts (returning 0 if any of them is invalid) and evaluates the relation. The first game evaluates the relation on the decrypted messages and the challenge message; the second game evaluates the relation on the decrypted messages and the second, random message (that is unrelated to the challenge message).

There are two further notions NM-CCA1 and NM-CCA2 obtained by giving the adversary access to a decryption oracle as in the corresponding IND games. In the former case, the adversary can only decrypt before she has called the challenge oracle; in the latter the only restriction is that she cannot ask to decrypt c^* itself.

Since IND-type games can be easier to work with than the NM game, we state the following lemma by Bellare and Sahai [BS99] that says non-malleability is equivalent to security against a single decryption query with multiple ciphertexts. In other words, the adversary may ask as many decryption queries as she likes as long as they are non-adaptive.

Lemma 3.10. NM-CPA security is equivalent to IND-1-CCA security.

3.5.1 Controlled-malleable encryption

A possible criticism of non-malleability notions is that they are too strong: given any non-malleable scheme, add an extra bit to the end of all ciphertexts during encryption and have

Game 0.

```
1  oracle initialise
2    (pk, sk) ← KeyGen()
3    return pk
4
5  oracle challenge(D)
6    m* ↞ D
7
8    c* ← Encrypt(pk, m*)
9    return c*
10
11 oracle finalise(c⃗, R)
12   if c* ∈ c⃗ then
13       return 0
14   end if
15   m⃗ ← map((c ↦ Decrypt(sk, c)), c⃗)
16   if ⊥ ∈ m⃗ then
17       return 0
18   end if
19   if R(m*, m⃗) then
20       return 1
21   else
22       return 0
23   end if
```

Game 1.

```
oracle initialise
  (pk, sk) ← KeyGen()
  return pk

oracle challenge(D)
  m* ↞ D
  m′ ↞ D
  c* ← Encrypt(pk, m*)
  return c*

oracle finalise(c⃗, R)
  if c* ∈ c⃗ then
      return 0
  end if
  m⃗ ← map((c ↦ Decrypt(sk, c)), c⃗)
  if ⊥ ∈ m⃗ then
      return 0
  end if
  if R(m′, m⃗) then
      return 1
  else
      return 0
  end if
```

Figure 3.9: Non-malleability game (NM-CPA).

the new decryption algorithm strip the last bit of a ciphertext before decrypting as before. This new scheme is clearly malleable but otherwise "as good as" before (in fact, Bellare et al. apply this construction to a formerly non-malleable scheme to prove the separation between IND-CPA and NM-CPA [BDPR98]). However, such easily detectable malleability should not be a problem in practice. We give a notion for encryption that we call controlled-malleable encryption that captures the idea that detectable malleability is harmless. We first presented this notion in a paper with Smyth [BS13].

We start with the IND-1-CCA game where the adversary may not ask for a decryption of the challenge ciphertext and generalise this equality check to an equivalence relation. In applications where one in principle desires non-malleability but ciphertexts are public (voting with bulletin boards is a particularly good example), one can sometimes replace non-malleability with controlled malleability.

Definition 3.11 (controlled-malleable encryption). Let R be an efficiently computable equivalence relation on a set C. A public-key encryption scheme is controlled-malleable with respect to R if its ciphertext space is C and the scheme is secure in the variant of the IND-1-CCA game in Figure 3.12.

```
1 oracle decrypt(c)
2   if R(c, c*) then
3       return ⊥
4   else
5       return Decrypt(sk, c)
6   end if
```

Figure 3.12: Controlled-malleable encryption.

Note that the way we applied our change to the IND game, we could equally well define a notion of "controlled CCA". The term "controlled-malleable" we borrowed from Chase et al. [CKLM12] who use it for a type of proof scheme. Other notions of restricted malleability appear in the cryptographic literature with similar properties as ours such as notions of replayable security (RCCA) [CKN03, PR07], constrained security (CCCA) [HK07] or benign malleability [S01].

Controlled versus benign malleability. The last-cited definition by Shoup is closest in spirit to our notion although Shoup uses an equivalence relation defined such that equivalent ciphertexts must decrypt to the same plaintext. His notion does not capture the possibility of an equivalence relation that modifies the underlying plaintexts in an "obvious" manner, which we require to reason about Helios ballots in Chapter 6. For example, take a non-malleable encryption scheme E on a message space M and construct an encryption scheme for message space $M \times M$ by simply encrypting both messages separately with scheme E. Now, a ciphertext of the form (C, C') can easily be modified to (C', C): if the former encrypted a message pair (m, m') then the new ciphertext will decrypt to (m', m). Applied to Helios ballots, this is one of the issues pointed out by Cortier and Smyth [CS11, CS13]. This is why our equivalence relations, unlike Shoup's, must be able to handle detectable ciphertext-modifications that also modify the contained plaintexts.

3.6 Relations among security notions

Bellare et al. give the main security notions for encryption in their overview paper [BDPR98]. The obvious relations hold: A CCA2 notion implies the corresponding CCA1 notion, which in turn implies the corresponding CPA notion. Further, a NM notion implies its IND counterpart. IND-CCA2 is equivalent to NM-CCA2: if ciphertexts were malleable, the adversary could ask to decrypt a modified version of the challenge ciphertext in the IND-CCA2 game. For this reason, we can just speak of CCA2 as a security notion. In fact, since this is the strongest notion among the ones present here and sometimes called the "correct" notion for public key encryption, CCA2 is sometimes just called "chosen-ciphertext security" or CCA for short. (A word of warning: IND-CCA1 predates IND-CCA2 and the papers introducing the former called it "chosen-ciphertext security", although this term nowadays is reserved for the latter.)

Lemma 3.13. The relations between the security notions for encryption are exactly as in Figure 3.14: arrows indicate reductions and if one notion N cannot be reached from another notion M by arrows, there is an irreduction i.e. a scheme that satisfies the notion M but not N.

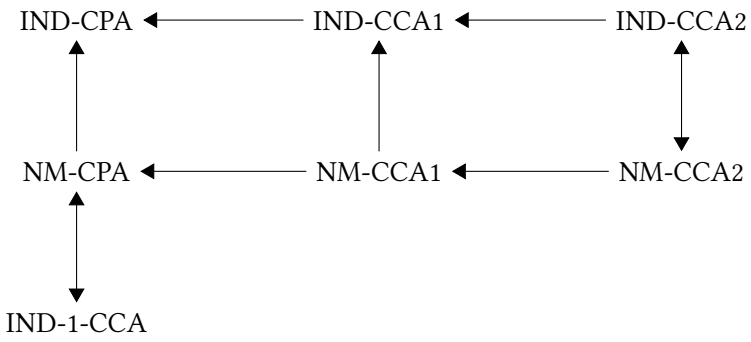


Figure 3.14: Relations between security notions for encryption.

3.7 Homomorphic encryption

Homomorphic encryption schemes allow operations on ciphertexts without access to the secret key, resulting in a ciphertext for a corresponding operation on the underlying plaintexts.

Definition 3.15 (homomorphic encryption). A public-key encryption scheme is homomorphic if the message space M is a group under some operation $\star$, the space of random choices R_{Enc} for the encryption algorithm is a group under some $\circ$ and there is an efficient algorithm $\mathsf{Add} : PK \times C \times C \to C$ on the space of ciphertexts such that for any pk, m, m', r, r',

$$\mathsf{Add}(pk, \mathsf{Encrypt}(pk, m; r), \mathsf{Encrypt}(pk, m'; r')) = \mathsf{Encrypt}(pk, m \star m'; r \circ r')$$

Homomorphic encryption schemes admit an efficient algorithm $\mathsf{Rand} : PK \times C \twoheadrightarrow C$ such that $\mathsf{Rand}(pk, c)$ is a uniformly random ciphertext in the space of all ciphertexts for the originally encrypted message:

$$\mathsf{Rand}(pk, c) \quad := \quad r \twoheadleftarrow R_{Enc};\ \mathsf{return}\ \mathsf{Add}(pk, c, \mathsf{Encrypt}(pk, e_M, r))$$

where e_M is the neutral element of M as a group under $\star$. This makes homomorphic encryption schemes rerandomisable. (Rerandomisable encryption can also be defined in a weaker manner

where the Rand algorithm does not need to produce a uniform distribution; this weaker notion does not concern us further in this thesis.)

The ElGamal encryption scheme presented in Section 3.4 is homomorphic: the operation on messages is the group operation in the underlying group and the Add algorithm is just the group operation applied component-wise.

3.7.1 Homomorphic and non-malleable encryption?

Homomorphic encryption has practical applications: one principle for constructing voting schemes is to have every voter encrypt her vote, homomorphically add all encrypted votes and then have the authorities decrypt the resulting ciphertext without anyone ever learning an individual's vote. However, homomorphism and non-malleability seem mutually exclusive and we have already argued that the latter is desirable too in an election scheme.

There are encryption schemes in which ciphertexts consist of two parts, the first being homomorphic and the second non-malleable. Taken together, the two parts form a non-malleable ciphertext as the second part of a ciphertext is non-malleable and it is hard to create new second part if one has tampered with the first. However, one can combine the first parts homomorphically to give a new ciphertext.

To construct a voting scheme out of such an encryption scheme, one demands that voters submit both parts of their ciphertexts. When voting has closed, the authorities check that ciphertexts are well-formed, discard the second parts and homomorphically tally only the first parts.

This principle of two-part ciphertexts was described by Wikström [W08] under the name submission security. Wikström's definition is for the case where both parts together are CCA secure, but the generalisation to non-malleability is easy. We will present our own variation on submission-security in Section 3.7.3 that differs from the original in some technical points. For example, we omit the original requirement of an extra secret key component to verify ciphertexts. But first, we will give an example of a CCA submission-secure encryption scheme.

3.7.2 Cramer-Shoup encryption

The scheme of Cramer and Shoup [CS08] is famous for being the first practically efficient proposal for a provably CCA2 secure encryption scheme that avoids the use of random oracles in its security proof. Instead it uses a collision-resistant hash function $\mathbf{H} : \mathbb{G}^3 \to \mathbb{Z}_{|\mathbb{G}|}$, i.e. it is assumed to be infeasible to find two distinct inputs x, y with $\mathbf{H}(x) = \mathbf{H}(y)$. Further, Cramer-Shoup is an extension of ElGamal similar to the ones we will consider in Chapter 4 on zero-knowledge proofs

(in fact the extension part can be seen as a zero-knowledge proof if one slightly generalises the notion). Cramer-Shoup encryption, as shown by Wikström [W08], is even a submission-secure extension of ElGamal.

Definition 3.16 (Cramer-Shoup encryption). Let $\mathbb{G}$ be a cyclic group and G a generator thereof. Let $\mathbf{H}$ be a collision-resistant hash function $\mathbb{G}^3 \to \mathbb{Z}_{|\mathbb{G}|}$. The Cramer-Shoup encryption scheme has spaces $SK = \mathbb{Z}_{|\mathbb{G}|}^5$, $PK = \mathbb{G}^4$, $M = \mathbb{G}$ and $C = \mathbb{G}^4$. The algorithms are as described in Figure 3.17.

```
1  algorithm KeyGen
2    x, x̄, y, ȳ, z ↞ Z_|G|
3    w ↞ Z_|G|; Ḡ ← G^w
4    pk ← (Ḡ, G^x Ḡ^x̄, G^y Ḡ^ȳ, G^z)
5    sk ← (x, x̄, y, ȳ, z)
6    return (pk, sk)

7  algorithm Encrypt(pk, m)
8    r ↞ Z_|G|
9    parse (Ḡ, U, V, Z) ← pk
10   (A, B, C) ← (G^r, Ḡ^r, Z^r · m)
11   c ← (A, B, C, (U^r V^r)^H(A,B,C))
12   return c

13 algorithm Decrypt(sk, c)
14   parse (A, B, C, D) ← c
15   parse (x, x̄, y, ȳ, z) ← sk
16   // check the proof
17   if A^x B^x̄ (A^y B^ȳ)^H(A,B,C) ≠ D then
18       return ⊥
19   end if        // proof ok:
20   return C/A^z  // plain ElGamal decryption
```

Figure 3.17: Cramer–Shoup encryption.

We refer the reader to the original paper [CS08] for the CCA2 security proof and to Wikström [W08] for a proof of submission-security. In the Cramer-Shoup scheme, $(z, Z = G^z)$ is an ElGamal keypair. The first and third components A, C in a Cramer-Shoup ciphertext are an ElGamal encryption (and thus homomorphic). B, D are an augmentation of an ElGamal ciphertext and $\bar{G}, U, V$ can be seen as an augmentation of the public key. The part marked "check the proof" in the decryption algorithm requires the secret key to check a ciphertext.

3.7.3 Verifiable augmented encryption

Although submission security is close to the notion of "non-malleable and homomorphic encryption" that we want, the original requirement for a secret key to check ciphertext validity is too strong for us. We propose a notion of verifiable augmented encryption that comes with a public checking algorithm and applies to notions including NM-CPA (modelled as IND-1-CCA), IND-CCA1 and IND-CCA2. We first introduced a version of this notion at Esorics 2011 [BC+11], where we called it "voting-friendly encryption".

Definition 3.18 (augmentation). Let $(\texttt{KeyGen}, \texttt{Encrypt}, \texttt{Decrypt})$ be a public-key encryption scheme. An augmentation of this scheme is an encryption scheme defined as follows:

- The new public key space is $PK^+ = PK \times PK_{Aug}$ and there is an algorithm $\texttt{AugmentKey} : PK \twoheadrightarrow PK^+$ that turns original public keys into augmented ones.
- The new ciphertext space is $C^+ = C \times C_{Aug}$.
- There is a deterministic verification algorithm $\texttt{Verify} : PK^+ \times C^+ \to \{0, 1\}$.

The new scheme is then

$\texttt{KeyGen}^+$ Run KeyGen to get the original keys and AugmentKey to augment the public key.

$\texttt{Encrypt}^+$ This algorithm depends on the scheme in question. See below.

$\texttt{Decrypt}^+$ Run Verify on the (augmented) public key and ciphertext. If this returns 0, return $\perp$. Otherwise, extract the embedded original ciphertext and run the original Decrypt on this and the secret key.

By $C^+ = C \times C_{Aug}$ we mean the product in a "categorical" sense: there are deterministic algorithms that project an augmented ciphertext onto an original ciphertext and an augmentation part, such that one can recover the original and augmented ciphertext from these two parts. We further demand that these projection and recombination algorithms are efficient.

For the extensions of ElGamal that we will consider in Chapter 4, the augmented encryption algorithm takes a particular form, although we choose not to restrict to this form in our defini-

tion above. These augmented encryption algorithms run in two stages, first drawing randomness $r_1 \twoheadleftarrow R_{Enc}$ and producing a basic ciphertext $c \leftarrow \texttt{Encrypt}(pk, m; r_1)$ then producing an augmentation $c_{Aug} \leftarrow \texttt{Augment}(pk, pk_{Aug}, m, r_1)$ which may involve drawing further random coins. Here, `Augment` is an algorithm defined as part of the scheme, often involving some kind of non-interactive zero-knowledge proof. The augmented ciphertext is then the pair $c^+ = (c, c_{Aug})$.

For examples of verifiable augmented encryption we must defer the reader to the next chapter where we will present the Encrypt+PoK construction.

3.8 Threshold encryption

Threshold encryption [DF89, SG98, FP01] allows a group of people to generate a key together such that everyone ends up with their own secret key share and there is one common public key. Anyone can now encrypt messages to the group with the public key; to decrypt messages a certain threshold number of group members must act together. At no point in such a decryption operation does any one person, or group of persons smaller than the threshold, need to know the secret key (and thus be able to decrypt other messages). Instead, each group member takes the ciphertext and produces a decryption share. Given enough shares, anyone can run a public recombination algorithm to obtain the decrypted message.

On key generation. The common definition of threshold encryption [SG98] defines a key generation algorithm that outputs a secret key for each group member and a common public key. In the security model, the key generation algorithm "is run" (by the security game, which is equivalent to a trusted party) and each group member gets their key. Unfortunately this is not what we need in practice, as in a real implementation some party would have to be in charge of running this algorithm, and thus would know everyone's secret keys. Instead, what we want is a key generation protocol (as opposed to an algorithm) that allows the group to generate their keys together, as described in the introduction above.

For the ElGamal-based schemes that we will use, we can simplify the protocol and give two algorithms: a key share generation algorithm, that each group member uses to generate a pair of shares, one secret and one public; and a key combination algorithm that takes all public key shares and generates the public key. We note that for other schemes such as BBS/DLIN [BBS04] encryption, devising a threshold key generation protocol from an algorithm is not easy and this fact is often overlooked in the literature, as we argued recently [BNV13].

Formal definition. In our notion of a threshold encryption scheme, a number of decryptors each generate a private key share and a public key share. The decryptors combine these public key shares to create a regular public key for encryption. To decrypt a ciphertext, each decryptor computes a decryption share; these can be combined to recover the message.

Definition 3.19 (threshold encryption). A threshold encryption scheme for parameters (k, n) (decryption threshold and total number of decryptors, $k \leq n$) is described by the following spaces and algorithms.

Spaces and algorithms. M is the message space, PK the public key space and C the ciphertext space. PK_S is the space of public key shares, D_S the space of decryption shares and SK_S the space of secret key shares.

algorithm	signature
`KeyShareGen`	$\twoheadrightarrow PK_S \times SK_S$
`KeyCombine`	$(PK_S)^n \to PK \cup \{\bot\}$
`Encrypt`	$PK \times M \twoheadrightarrow C$
`DecryptShare`	$SK_S \times C \twoheadrightarrow D_S \cup \{\bot\}$
`Combine`	$C \times (D_S)^k \to M \cup \{\bot\}$

Correctness. For any $m \in M$, we demand that after producing n pairs $(pk_i, sk_i) \leftarrow$ `KeyShareGen`, running $pk \leftarrow$ `KeyCombine`$(pk_1, \ldots, pk_n)$ we have $pk \neq \bot$; further running $c \leftarrow$ `Encrypt`(pk, m) and then for $i = 1 \ldots n$, $d_i \leftarrow$ `DecryptShare`(sk_i, c) we have $d_i \neq \bot$ everywhere and for any subset S of k decryption shares we get that `Combine`$(c, (d_s)_{s \in S})$ yields m.

Security notions. All IND notions can be converted to the threshold setting. In the key generation phase, we allow an adversary against a k-out-of-n threshold scheme to contribute up to k public key shares and have the game generate the remaining $n - k$ key shares correctly, keeping their secret keys secret and giving the adversary the public key shares. However, if the `KeyCombine` algorithm fails, the game halts before the adversary can make any challenges. (This

allows for schemes in which public keys are accompanied by proofs of knowledge of their associated secret keys: if the adversary fail to provide such proofs for her key shares, the game halts.)

The adversary may challenge the game as usual and, if the notion in question allows for decryption queries, the adversary gets decryption shares from the $n - k$ decryptors managed by the game. The adversary is expected to handle her own partial decryptions for her k "dishonest" decryptors. The aim of the adversary remains to guess the secret bit used in challenge queries.

We omit giving a formal definition of the threshold security games as they are outside the scope of this thesis.

Threshold ElGamal. As an example we give the $(n-1)$-out-of-n threshold version of ElGamal, such as currently used in the Helios voting protocol [A08]. To generate a key share, each decryptor just generates an ElGamal key pair and provides a proof of knowledge of her secret key along with the public key (we will describe these in Chapter 4). To combine n public key shares, one first verifies all proofs. If they verify, the public key is the group operation $PK = \prod_{i=1}^{n} PK_i$ on the key shares, a normal ElGamal public key (with the associated secret key $sk = \sum_{i=1}^{n} sk_i$). Encryption is just ElGamal encryption under this public key. To partially decrypt a ciphertext (C, D) with a secret key share sk_i, compute $D_i \leftarrow C^{sk_i}$ (and make a zero-knowledge proof of correct decryption, if necessary). To combine decryption shares, check proofs if necessary then compute $M = D / \prod_{i=1}^{n} D_i$.

Deploying threshold encryption. As a rule of thumb, results concerning encryption schemes also apply to threshold encryption as long as the number of dishonest key-holders is below the threshold and *all keys are generated correctly*. Ensuring correct key generation for threshold cryptosystems is beyond the scope of this thesis; for ElGamal it is possible but with a few subtleties as shown in a sequence of works starting with one by Gennaro et al. [GJKR99] whereas in recent work we have shown it to be a more complex matter than previously thought for another popular encryption scheme [BNV13].

Once key generation is sorted, since keys are typically information-theoretically securely shared, a proof of a protocol involving a threshold scheme proceeds in two parts, first a reduction to the single-key version of the protocol then a proof in the single-key setting. For the rest of this thesis we will ignore threshold issues since they are orthogonal to the analyses of the protocols themselves that we will perform.

4 An introduction to zero-knowledge proofs

This chapter is about proof schemes, protocols with which Peggy "the Prover" can try and prove something to Veronica "the Verifier" who can either accept or reject Peggy's claim. Suppose that Peggy wishes to prove to Veronica that she knows the code to a safe. Peggy could simply tell Veronica the code. This is a perfectly good proof scheme but Veronica ends up knowing not just that Peggy knows the code, but the code itself too. Peggy could also just state that she knows the code, "trust me". This protects Peggy's secret knowledge, but might not convince Veronica. However, Peggy could also let Veronica observe the closed safe, ask her to look away then enter the code and open the safe, allowing Veronica to deduce that Peggy knows the code without Veronica gaining the ability to open the safe herself. This is what cryptographic proof schemes aim to achieve, security guarantees for both Peggy's secret and Veronica's trust.

Peggy's knowledge is protected by properties such as zero-knowledge, which informally says that Veronica gains no extra knowledge from Peggy by following a proof scheme with her, beyond whatever Peggy wanted to prove in the first place. The security properties for Veronica can include a protocol being sound, i.e. Peggy cannot convince Veronica of a false claim. The scheme can also be a "proof of knowledge" which informally means that Peggy cannot convince Veronica unless she actually know what she claims to know; we formalise this in Definition 4.4.

Formalising zero-knowledge and proofs of knowledge is a difficult task, principally due to the fact that these notions require a collection of assumptions — commonly known as a model — to yield a formal notion and different models exist leading to different variations of these notions.

Historical overview. The concept of zero-knowledge proofs was introduced by Goldreich, Micali and Rackoff [GMR85, GMR89] for interactive proofs. Their paper was first published in 1985. Feige, Fiat and Shamir first defined proofs of knowledge in 1988 [FFS88] and improved this definition (according to Bellare and Goldreich [BG92]) in 1990 [FS90] in a paper that also introduced the notions of witness hiding and witness independent proofs. Also in 1988, Blum, Feldman and Micali [BFM88] considered non-interactive zero-knowledge proofs for the first time. Goldreich, Micali and Wigderson [GMW91] proved in 1991 that zero-knowledge proofs exist for all lan-

guages in **NP**. There are many notions and variations of zero-knowledge, proof of knowledge and related concepts and several papers attempting to simplify, correct or unify existing notions. The most important of these papers is the 1992 work of Bellare and Goldreich [BG92], presenting a viewpoint centered upon the verifier and defining the idea of a knowledge error, allowing for a quantitative analysis of protocols.

Sigma protocols and random oracles. The cited result [GMW91] that all **NP** languages admit zero-knowledge proofs, like many of the above definitions, says little about how one could construct such proofs in a practical manner. A branch of the tree of zero-knowledge notions and schemes that does yield practical protocols begins with the 1986 work of Fiat and Shamir [FS86]. This work on the one hand contains interactive protocols where the prover sends the verifier a value thus committing to it, the verifier produces a random challenge and the prover "opens" a combination of the original value and the challenge — an early version of the ideas underlying so-called sigma protocols. These protocols are in fact proofs of knowledge, although this notion is never mentioned in the paper — it had not yet been invented at the time the paper was published.

On the other hand, the observation with which this paper is most often credited (despite Bellare and Neven [BN06] attributing it to Blum) is that one can replace a verifier whose task is to create random challenges after seeing certain commitments from the prover, by a "pseudo random function f" on said commitments. This idea yields a non-interactive protocol and is commonly known as the Fiat-Shamir transformation. A formal analysis of this idea was given by Bellare and Rogaway in 1993 [BR93], modelling the function f as a "random oracle".

Schnorr, in 1991 [S91], gave the first true sigma protocol, a three-move protocol with a verifier picking random challenges, for proving knowledge of discrete logarithms. Schnorr's protocol is not only practical but is in fact also used in practice; indeed he originally motivated the protocol as an identification scheme efficient enough to run on smart cards.

The principles of Schnorr's protocol apply to many other types of problems, yielding similarly efficient protocols. Chaum and Pedersen [CP92] proposed a sigma protocol for proving possession of a Diffie-Hellman triple and knowledge of the underlying exponents in 1992. Camenisch developed a large number of sigma protocols and a general theory for them in his 1998 PhD thesis [C98], including a concise notation to describe the statements proved with such protocols, developed together with Stadler in 1997 [CS97b, CS97c]. Sigma protocols are sometimes also called generalised Schnorr proofs. Camenisch, Kiayias and Yung give a long list of papers using these in their 2009 work [CKY09], also pointing out some pitfalls with their use.

Our presentation of the theory of zero-knowledge proofs is divided into two chapters, the current and the following one. In this chapter we introduce standard constructions and security notions, with a particular emphasis on sigma protocols. In the following chapter we will present our own work that builds upon the theory in this chapter.

The reader will notice that we present some notions using code-based games whereas we use a mainly textual description for others. This is a deliberate choice. The difference between the two presentations is not merely one of style: adopting our code-based notation implies fixing a particular view of the execution model and choosing a fixed representation of inter-procedure communication. We use code-based notation for our own definitions in the following chapter and for some properties that we need in code-based form to develop and prove our own notions. Where our notation would imply a change to the spirit of the original definition, we choose not to adopt a code-based presentation. Thus we reproduce Bellare and Rogaway's well-known definition of a knowledge verifier without any changes and define a proof of knowledge in a text-based manner whereas we present the definition of zero-knowledge and our own definition of a strong proof in the next chapter using a code-based game.

Recap: NP relations. In much of the rest of this chapter we will discuss proof systems over **NP** relations (Section 2.3.3). These are relations $R \subseteq X \times W$ that can be computed in time polynomial in the length of the first input $x \in X$ alone. For such a relation R, the language $L(R)$ is defined as the set of $x \in X$ for which there exists a $w \in W$ making $R(x, w)$ hold. For a relation R to be in **NP**, we further demand that for every $x \in L(R)$ there is also a $w \in W$ such that the length of w is itself polynomial in the length of x; the collection of all such pairs (x, w) we denote $R^{\wp}$.

4.1 Proof schemes

The basic object of study in this chapter is the proof scheme, a protocol between a prover and a verifier in which the prover can make a claim that the verifier can accept or reject. A proof scheme itself is not necessarily a proof, just like an encryption scheme is not necessarily secure: we take the approach of first defining the class of schemes that we consider and then introducing security notions in later definitions.

Definition 4.1 (Proof scheme). An interactive proof scheme for an **NP** relation R on sets X, W is a pair of interactive, efficient algorithms $(\texttt{Prove}, \texttt{Verify})$. `Prove`

takes a pair $(x, w) \in X \times W$ as input; `Verify` takes an $x \in X$ as input and outputs a value in $\{0, 1\}$ which is taken as the output of the scheme. If `Verify` outputs 1 we say that it accepts the interaction with `Prove`, otherwise we say that it rejects.

As usual when dealing with **NP** witnesses, the running time of the prover is measured in terms of the first input X only.

A proof scheme must satisfy the following correctness property: for any instance/witness pair $(x, w) \in R^{\wp}$, if we let `Prove`(x, w) interact with `Verify`(x) the outcome is that `Verify` accepts, with overwhelming probability over all random choices by both parties.

A non-interactive proof scheme is a special case of an interactive one in which the prover sends the verifier a single message; the verifier then accepts or rejects without further interaction. Since this thesis is mainly concerned with applications of non-interactive proofs and some notions are easier to formulate for the non-interactive case, or indeed have only been defined in the non-interactive case, we give non-interactive proof schemes in a separate definition. We will later revert to discussing a class of interactive proof schemes known as sigma protocols, with which we will build non-interactive ones in the random oracle model.

Definition 4.2 (Non-interactive proof scheme). A non-interactive proof scheme for an **NP** relation $R \subseteq X \times W$ is given by a space Π and a pair of efficient algorithms $(\texttt{Prove}, \texttt{Verify})$ where $\texttt{Prove} : X \times W \twoheadrightarrow \Pi$ and $\texttt{Verify} : X \times \Pi \to \{0, 1\}$. The running time of `Prove` is measured with respect to its input in X alone.

An element $\pi \in \Pi$ such that for some $x \in X$, `Verify`$(x, \pi) = 1$ is called a proof for x.

A non-interactive proof scheme must satisfy the following correctness property: for any $(x, w) \in R^{\wp}$, letting $\pi \leftarrow$ `Prove`(x, w) then `Verify`$(x, \pi) = 1$ with overwhelming probability over the random choices of `Prove` (`Verify` is deterministic).

We indicate the notions related to proof schemes that we treat in this thesis in Figure 4.3. The dimensions interactive/non-interactive and plain/ROM/CRS/both are to some extent independent of each other and of the security properties but not every combination makes sense. Arrows are

implications (so a simulation sound proof scheme is also sound and zero-knowledge). The notions of (ss-)mPoK are our own and we will present them in the next chapter.

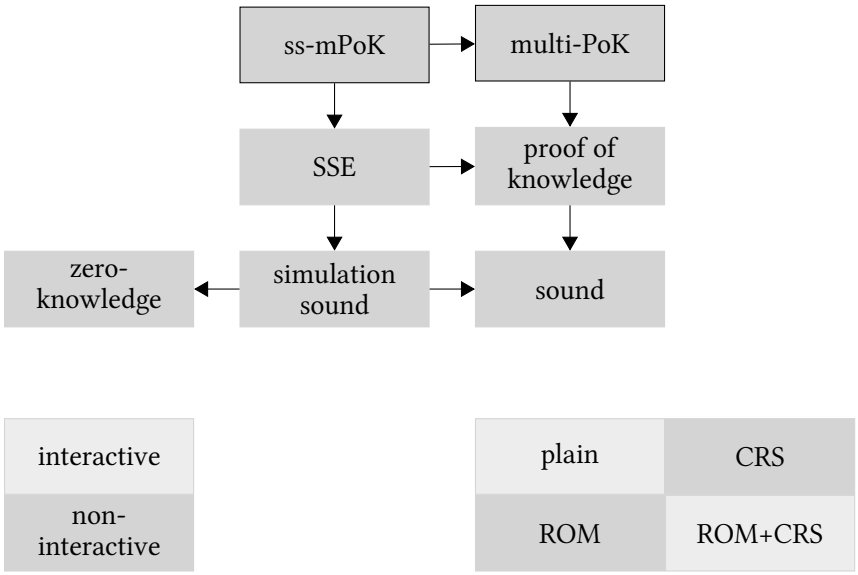


Figure 4.3: Notions related to proof schemes.

4.2 Security properties

We consider non-interactive proofs of knowledge except where stated otherwise from here on.

4.2.1 Zero-knowledge

We wish to show that a verifier gains no knowledge from seeing a proof of a statement, beyond the statement itself and the truth of the statement. To achieve this we postulate the existence of a simulator that can produce proofs that look like real ones given only an instance and being assured that the statement is true on this instance.

We model this in a game in which a distinguisher plays against a `prove` oracle that takes a instance/witness pair and does one of two things. After checking correctness of the witness, either it produces a regular proof or it passes only the instance on to the simulator.

Since the simulator is producing a proof without a witness, which is exactly what a regular prover should not be able to do, if the system is to offer the verifier any security guarantees, we

need to give the simulator some extra capabilities. At this point the notion of zero-knowledge diverges into random oracle model zero-knowledge and common reference string zero-knowledge. In this thesis, we choose to present a unified definition highlighting the fact that these notions are just expressions of the same idea in different models. (Our definition also shows that it is possible to construct a proof system that is zero-knowledge and uses neither the CRS nor the RO model. However, such a proof system cannot be sound or a proof of knowledge since anyone could just simulate the simulator, so such a system would not be of any use.)

Definition 4.4 (zero-knowledge). A proof system $(\texttt{Prove}, \texttt{Verify})$ for a relation R is zero-knowledge if there exists an efficient simulator $\texttt{S}$ such that the following two games are indistinguishable. If the distinguisher may call the oracle multiple times, the proof system is also called multi-theorem zero-knowledge, otherwise single-theorem zero-knowledge. In the second game,

- In the CRS model, the simulator is responsible for the CRS oracle.
- In the RO model, the simulator is responsible for the random oracle.

First game.

```
1 oracle prove(x, w)
2   if not R(x, w) then
3       return ⊥
4   end if
5   π ← Prove(x, w)
6   return π
```

Second game.

```
oracle prove(x, w)
  if not R(x, w) then
      return ⊥
  end if
  π ← S(x)
  return π
```

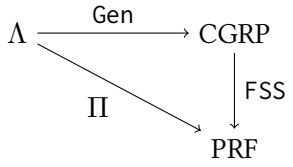
The idea behind deploying zero-knowledge in a protocol is that if there is any party gaining illicit knowledge after seeing a proof, we perform a thought experiment in which we replace the proof in question by a simulated one which by definition contains no knowledge (other than of the instance). If the party's behaviour now changes, then we have a distinguisher between real and simulated proofs, breaking the zero-knowledge property.

We remark that the definition of zero-knowledge does not give an exact class of inputs on which the simulator is expected to work. In the definition, the simulator only runs on correct

instances, however if there is some superset $L^* \supsetneq L$ of the set L of correct instances such that L and L^* are indistinguishable, we can consider a game in which the simulator is given an $x' \in L^* \setminus L$ as input. Since the simulator is itself efficient and therefore cannot distinguish such an $x' \in L^* \setminus L$ from a correct one, we can conclude that the simulator still produces a valid proof. Further variations on this theme appear in this thesis as "generic simulation arguments" in Section 5.2.3.

Local and global definitions of ZK. As we explained in Section 2.7, we prefer to state our definitions locally (i.e. with respect to a single group) and leave implicit that the actual definitions of security properties refer to families of objects indexed by security parameters. We give some more details for the case of zero-knowledge.

Informally, our starting point is the class of cyclic groups which we might denote CGRP. A family of groups indexed by a security parameter is a mapping Gen from the set Λ of the natural numbers in unary notation into CGRP. The same principle applies to other classes of cryptographic objects such as encryption or proof schemes. A particular proof scheme over a group such as Fiat-Shamir-Schnorr (which we introduce formally in Definition 5.12) can be seen as a mapping FSS from CGRP to the class of proof schemes PRF. Our definition of zero-knowledge (Definition 4.4) gives a game for a single group. Composing the mappings FSS and Gen yields a security parameter-indexed family of proof schemes $\Pi = (P_\lambda)_{\lambda \in \Lambda} \subset \mathrm{PRF}$, on which it is syntactically correct to state the claim that the zero-knowledge game is asymptotically hard. Expressed as a commutative diagram,



The objects on which we can state asymptotic security properties are arrows or paths out of Λ. We could either claim that the Fiat-Shamir-Schnorr construction FSS is zero-knowledge w.r.t. a particular family of groups described by Gen or we could regard the group generation as part of a concrete scheme and claim that the family of proof schemes Π is zero-knowledge.

The reader may have guessed that we would find it an interesting future project to investigate reductionist security from a category-theoretic viewpoint in which Gen and FSS are morphisms between categories such as Λ, CGRP or PRF.

Applying the above intuition, we give as an example the formal statement that the FSS construction is zero-knowledge, which we prove in Lemma 4.21 in a particular model. Note that the

proof of the lemma is entirely "local", that is it describes how to construct a simulator given any fixed group $\mathbb{G} \in \mathrm{GRP}$.

"Let Gen be a mapping that on input a security parameter λ in unary notation outputs a (description of a) group $\mathbb{G}_\lambda$. Let $\Gamma_0(\mathbb{G}_\lambda)$ be the first game of Definition 4.4 where $(\mathtt{Prove}, \mathtt{Verify})$ is instantiated over $\mathbb{G}_\lambda$ and let $\Gamma_1(\mathbb{G}_\lambda)$ be the second game where the simulator S takes a description of $\mathbb{G}_\lambda$ and 1^λ as extra, implicit parameters.

Fiat-Shamir-Schnorr is zero-knowledge w.r.t. Gen if for any efficient algorithm A, the quantity

$$\begin{aligned}\mathbf{Adv}^{\mathrm{ZK}}_{A,\mathsf{Gen}}(\lambda) := \quad & \mathbf{Pr}\left[\mathbb{G}_\lambda \leftarrow \mathsf{Gen}(1^\lambda); A^{\Gamma_0(\mathbb{G}_\lambda)}(\mathbb{G}_\lambda) = 1\right] - \\ & \mathbf{Pr}\left[\mathbb{G}_\lambda \leftarrow \mathsf{Gen}(1^\lambda); A^{\Gamma_1(\mathbb{G}_\lambda)}(\mathbb{G}_\lambda) = 1\right]\end{aligned}$$

is negligible in λ. Here $A^x(i)$ means the output of A when it gets input i and may interact with algorithm x."

Similar examples could be given for all further definitions in this section. In each case, the asymptotic version is to be obtained by composing the local definition/game(s) with a mapping from Λ into the domain of the construction in question.

4.2.2 Soundness or proofs of statements

While the zero-knowledge property protects a prover from a malicious verifier learning her witness, the security properties for the verifier against a malicious prover come in different flavours. Informally, we summarise:

soundness If the verifier accepts, then (most likely) the statement is correct. Equivalently, it is hard to create proofs for false statements.

proof of knowledge If the verifier accepts, then the prover "knows" a witness to the instance (which implies that the statement is correct). The formalisation of the prover "knowing" a witness is that there is a thought experiment in which a witness can be extracted from the prover.

simulation soundness One cannot produce fresh proofs of false statements even if one has seen a proof made by the simulator (which may have involved a false statement). Fresh here means that proofs made by the simulator do not count; taking a simulated proof and modifying it does count, so simulation-soundness implies that proofs are in some sense non-malleable.

We give a definition of soundness following Bellare and Goldreich's notion [BG92] of a knowledge verifier.

Definition 4.5 (soundness). A proof scheme is δ-sound for a function $\delta : \mathbb{N} \to [0, 1]$ against a class $\mathcal{P}$ of provers if for any $x' \notin L(R)$ and any $P' \in \mathcal{P}$, after P' interacts with `Verify`(x') the result is that `Verify` rejects with probability at least $\delta(|x|)$, the probability being taken over the random choices of `Verify` alone.

If δ is a negligible function we simply call the proof scheme sound.

The classes $\mathcal{P}$ of provers that are of typical interest are the class of all efficient provers, in which case we can call the proof scheme computationally δ-sound, or the class of all (computationally unbounded) provers.

4.2.3 Proofs of knowledge

A more complex property that a proof scheme can satisfy is being a proof of knowledge for some relation. This property is often explained intuitively by saying that no prover can get the verifier to accept a proof unless the prover actually know a witness to the instance she is proving. This notion is then formalised using a knowledge extractor, an algorithm with some extra abilities that allow it to extract a witness from the prover. What we believe the proof of knowledge property is really about is this: in a security argument, one can treat a proof of knowledge as if the prover provide a witness to her instance along with the proof. Indeed, sometimes the proof of knowledge property is formulated as "witness extended emulation" [L03], saying that in a security argument one can replace a prover with an emulator that provides instances and proofs indistinguishable from those of a real prover and additionally supplies the witnesses to all instances.

The extractor's extra abilities broadly fall into two categories: trapdoors to a CRS and rewinding in the random oracle model. In the former case, the extractor gets an extra trapdoor value along with the CRS, allowing it to extract witnesses. The extractor usually does not need to interact further with the prover; we call such an extractor a straight-line extractor. In the latter case, proofs hold w.r.t. a random oracle and the extractor has black-box access to many copies of the prover. The extractor can run two identical copies of the prover and give them different answers to the same random oracle query, then extract a witness from the two resulting proofs. We call such an extractor a rewinding extractor. The situation is more complex however as there are at least two more types of extractors. We summarise the known types:

1. Straight-line extractors in the CRS model. The Naor-Yung construction [NY90] to strengthen an encryption scheme to a CCA secure one uses double encryption and a non-interactive proof that need not be a proof of knowledge. The second encryption and the non-interactive proof taken together can be viewed as a proof of knowledge of the plaintext (indeed, the main argument in the CCA proof is that one can extract the plaintext) with the second encryption key as the CRS.

2. Rewinding extractors in the random oracle model. The Fiat-Shamir-Schnorr protocol is perhaps the best-known example.

3. Straight-line extractors in the random oracle model. A proof scheme by Fischlin [F05] has an extractor that needs only to see the transcript of random oracle queries made by the prover to extract a witness. The Chaum-Pedersen signed ElGamal scheme [ST13] uses a random oracle proof with straight-line extractor too in its construction, although a key is required to check the proof.

4. The CRS+random oracle model. Some constructions can be interpreted as proof schemes in the random oracle model where the extractor also gets a trapdoor to a CRS. The Naor-Yung construction where the original "small" non-interactive proof is in the random oracle model meets this case. The TDH2 encryption scheme [SG98, BGP11], another CCA secure extension of ElGamal, can be seen as using a random oracle model proof and a trapdoor.

The definition of proofs of knowledge in the case of interactive proof systems was given as follows by Bellare and Goldreich [BG92]. Their definition gives a concrete security bound δ as a function of the witness length and so avoids a dependency on a security parameter.

Definition 4.6 (knowledge verifier). V is a knowledge verifier for relation R with knowledge error $\delta : \mathbb{N} \to [0, 1]$ if there exists a P such that for all $x \in L(R)$, after interacting with P then V accepts with probability one and there exists a $c > 0$ and an extractor K such that for every P' and every $x \in L(R)$, if after interacting with P' then V accepts with probability $p(|x|)$ greater than $\delta(|x|)$ then $K(x)$ with oracle access to $P'(x)$ outputs a witness in an expected number of steps

$$\frac{|x|^c}{p(|x|) - \delta(|x|)}$$

We present a definition of a proof of knowledge as a game between a prover P and a knowledge extractor K. The aim of the prover P is to produce a proof on which the extractor cannot extract a witness; the aim of the extractor K is to produce such a witness. This definition lends itself to generalisations, which we will address in Section 5.3.

Definition 4.7 (proof of knowledge). Let $(\texttt{Prove}, \texttt{Verify})$ with $\texttt{Prove} : X \times W \twoheadrightarrow \Pi$ and $\texttt{Verify} : X \times \Pi \to \{0,1\}$ be a non-interactive proof scheme. Call an algorithm P a prover for $x \in X$ if P outputs a value in Π and with overwhelming probability (in some security parameter), running $\pi \leftarrow P()$ satisfies $\texttt{Verify}(x, \pi) = 1$.

A proof scheme is a proof of knowledge for a relation $R \subseteq X \times W$ if there is an efficient extractor K such that for any $x \in X$ and any prover P for x, after running $\pi \leftarrow P()$ the extractor $K(x, \pi)$ outputs a witness w with overwhelming probability such that $R(x, w) = 1$.

Our definition allows for non-uniform provers, that is the extractor must work for provers with a hard-coded x' that can only produce proofs w.r.t. this particular x'. An euqivalent viewpoint is that the extractor must work for families $(P_x)_{x \in X}$ of provers where each P_x individually is an efficient algorithm (in some security parameter as implicit input), even if the algorithms may vary arbitrarily with x. This is a stronger definition than considering only efficient algorithms $P(x, 1^\lambda)$ that output a proof for any x. It is also why the prover $P()$ takes no explicit inputs. In our definition, we aimed to capture the key point independently of a particular model; the following are specialisations to the most common models.

- If the proof scheme is to be a PoK with a rewinding extractor, then K has black-box access to further copies of the prover P with the same randomness as the one used to define π.
- If the proof scheme is in the random oracle model, K gets a transcript of all random oracle queries and responses made by the initial execution of P. For a rewinding proof in the random oracle model, K may handle random oracle queries for all further copies of P.
- If the proof scheme is in the CRS model, K may initially provide a CRS that all copies of P get as an input. A prover for x in the CRS model is an algorithm that, given access to a correctly generated CRS, outputs a proof that verifies w.r.t. this CRS with overwhelming probability.

The definition of provers in the CRS model lets K submit a "manipulated" CRS as long as P cannot distinguish it from a real one: were this the case, P would not have to output a valid proof anymore given a "bad" CRS and K would have to find a witness by itself. If K can find witnesses without P's help, the proof scheme is trivially a proof of knowledge.

4.2.4 Simulation soundness

Simulation soundness is the property that one can simulate proofs and still expect soundness from all proofs except the simulated ones. Typically simulation soundness is not a problem (and sometimes not even considered) for random oracle based proofs yet becomes interesting for CRS-based proofs, for example those of Groth and Sahai [GS08] (standard techniques exist to transform such proofs into simulation-sound ones but this property is not "for free"). These proofs are zero-knowledge (in some instantiations) because it is possible to set up a "hiding" CRS with which one can simulate proofs and they are proofs of knowledge because one can set up a "binding" CRS allowing for witness extraction. However, one has to choose at the outset which kind of CRS one wants to create. Simulation soundness was first defined by Sahai [S99] for the very purpose that makes it interesting to us, namely to boost encryption schemes from chosen-plaintext to chosen-ciphertext security which we cover in detail in Section 5.2 below.

Definition 4.8 (simulation soundness). A proof scheme is simulation sound for a relation R if it is zero-knowledge and for any efficient prover P' who may make a single $\mathtt{S}(x)$ query to the zero-knowledge simulator directly (i.e. without going through the $\mathtt{prove}$ oracle), the probability of the prover returning a instance/proof pair (x, π) such that $\mathtt{Verify}(x, \pi) = 1$, π was not obtained from the simulator yet $x \notin L(R)$ is negligible. In a model requiring a CRS or random oracle, the simulator handles these calls.

In applications of simulation soundness, what we often want is a simulation sound proof of knowledge such that we can still extract witnesses after using the simulator. This is not automatically guaranteed so in our notion of simulation sound multi-proofs (Definition 5.30) we will demand this property explicitly. However, while we have a definition for this property in the random oracle model, we leave the correct handling of the CRS model in this case for future work.

4.3 Sigma protocols

Sigma protocols (the name comes from Cramer [C96]) are three-round interactive protocols with the following layout, illustrated in Figure 4.10:

1. The prover sends a value known as a commitment to the verifier.
2. The verifier draws a challenge uniformly at random and returns it to the prover.
3. The prover computes a response based on her initial knowledge, the verifier's challenge and her commitment and sends this to the verifier, who accepts or rejects the prover's claim.

Definition 4.9 (sigma protocol). A sigma protocol is described by the following sets and efficient algorithms.

Sets.	
X	instances
W	witnesses
Com	commitments
Ch	challenges
Res	responses
St	prover state

Algorithms.	
`Commit`	$X \times W \twoheadrightarrow Com \times St$
`Respond`	$X \times W \times Com \times Ch \times St \to Res$
`Verify`	$X \times Com \times Ch \times Res \to \{0,1\}$

The protocol is given by the following algorithms for the prover and verifier respectively.

The prover runs `Commit` on input her instance and witness, stores the resulting state and sends the commitment to the verifier, receiving a challenge in return. She then runs `Respond` on all this data to produce a response, which she sends to the verifier.

The verifier takes an instance as input, receives a commitment then picks a challenge uniformly at random from Ch and returns it to the prover. The verifier then waits for a response, after receiving this she runs `Verify` on the instance, commitment, challenge and response and accepts if and only if this returns 1.

A sigma protocol is correct for a relation R on $X \times W$ if for any (x, w) in R the execution of the sigma protocol with these inputs causes the verifier to accept.

A 4-tuple in $X \times Com \times Ch \times Res$ resulting from the execution of a sigma protocol is called a transcript. A transcript is accepting if `Verify` returns 1 on it.

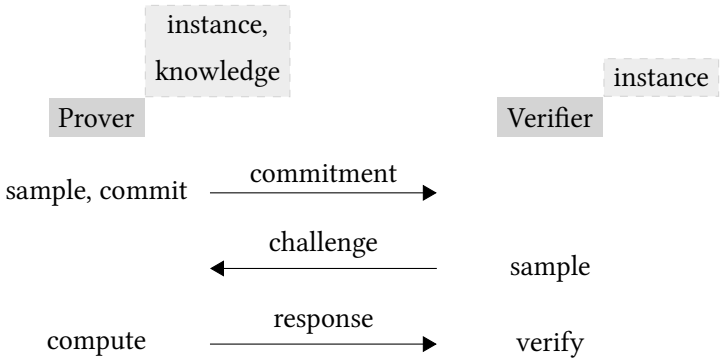


Figure 4.10: A sigma protocol.

A point that we will mention later is how the verifier gets her input $x \in X$. An alternative way of viewing the protocol is that the prover sends x along with the commitment in the first message to the verifier. All further properties of interactive proofs built from sigma protocols work equally well under both viewpoints, but the non-interactive versions obtained with the Fiat-Shamir transformation will differ.

4.3.1 The Schnorr protocol

We present Schnorr's protocol [S91] as an example. Schnorr's protocol proves knowledge of a discrete logarithm: the instance is an $X \in \mathbb{G}$ where $\mathbb{G}$ is a group with generator G and $R(X, x) := (X = G^x)$.

Definition 4.11 (Schnorr protocol). The Schnorr protocol is the sigma protocol defined in Figure 4.12.

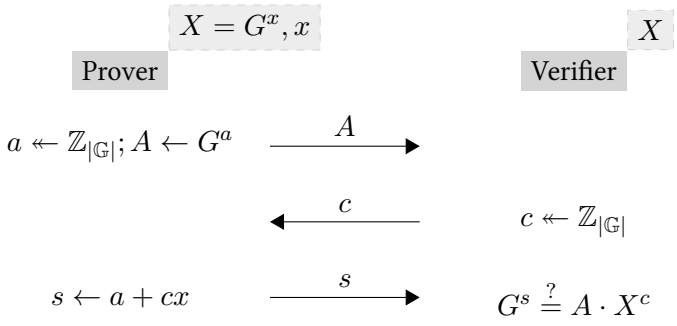


Figure 4.12: The Schnorr protocol.

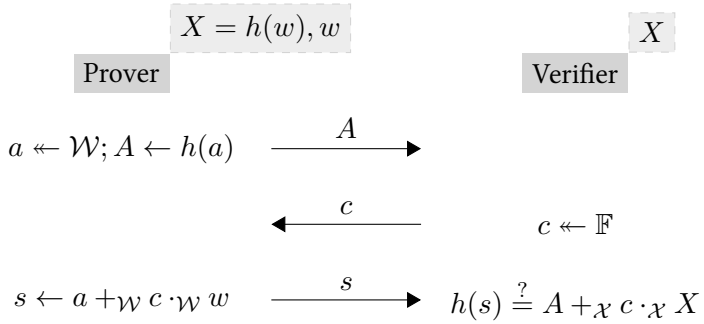
Schnorr's protocol has the following properties, which we will define soon and prove based on the template sigma protocol below: it is honest-verifier zero-knowledge and has special soundness, which gives normal soundness with a negligible soundness error. The non-interactive version obtained via the Fiat-Shamir transformation (Section 4.3.3 below) will be fully zero-knowledge and a simulation sound extractable strong proof in the random oracle model (as defined in the next chapter).

4.3.2 A template protocol

We can generalise Schnorr's protocol to get a template that describes many generalised Schnorr proofs, following our master thesis [B09] but building the construction over vector spaces instead of rings.

Let $\mathbb{F}$ be a finite field and $\mathcal{W}, \mathcal{X}$ be finite vector spaces over $\mathbb{F}$. (We are interested not so much in the spaces having finite dimension over $\mathbb{F}$ as in being able to draw elements from $\mathbb{F}$ and $\mathcal{W}$ uniformly at random.) Recall that a vector-space homomorphism $h : \mathcal{W} \to \mathcal{X}$, also known as a linear map, is a map on the underlying sets that preserves addition and field multiplication, i.e. for $w, w' \in \mathcal{W}$ and $f \in \mathbb{F}$ we have $h(w +_{\mathcal{W}} w') = h(w) +_{\mathcal{X}} h(w')$ and $h(f \cdot_{\mathcal{W}} w) = f \cdot_{\mathcal{X}} h(w)$.

Definition 4.13 (template protocol). The sigma protocol template for proving knowledge of a preimage of $h : \mathcal{W} \to \mathcal{X}$ is as follows:



This protocol is correct since if both parties follow it,

$$h(s) = h(a +_{\mathcal{W}} c \cdot_{\mathcal{W}} w) = h(a) +_{\mathcal{X}} c \cdot_{\mathcal{X}} h(w) = A +_{\mathcal{X}} c \cdot_{\mathcal{X}} X$$

As an application of this theory, consider the protocol by Chaum and Pedersen [CP92] for proving that a triple of elements is a Diffie-Hellman triple. We illustrate this protocol in Figure 4.14. Taking $\mathbb{F} = \mathbb{F}_p$ for a prime p, a Diffie-Hellman triple in a cyclic group $\mathbb{G}$ of order p with generator G is a triple (G^x, G^y, G^{xy}) for $x, y \in \mathbb{Z}_p$. Choosing $\mathcal{W} = (\mathbb{F}_p)$ and $\mathcal{X} = \mathbb{G}^2$, the map $\mathcal{W} \to \mathcal{X} : y \mapsto (G^y, X^y)$ for any fixed group elements G, X is linear. (Note that $(x, y) \mapsto G^{xy}$ would not be a linear map.)

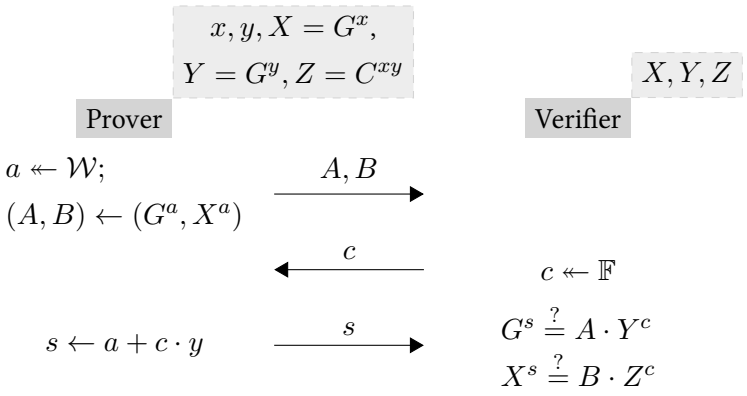


Figure 4.14: The Chaum–Pedersen protocol.

Security for the prover. A protocol following our template is honest-verifier zero-knowledge, that is there exists a simulation algorithm that on input X produces tuples (X, A, c, s) indistinguishable from transcripts of runs of the protocol with the honest verifier who chooses c uniformly and independently of X and A. In fact, the simulation algorithm is even better, it can take X and c as input and produce A, s as output. This kind of simulator is also known as a c-simulator and the property of possessing one is sometimes called special honest-verifier zero-knowledge. Protocols with a c-simulator are trivially honest-verifier zero-knowledge.

Definition 4.15 (honest-verifier zero-knowledge). A sigma protocol represented as $(\texttt{Prove}, \texttt{Verify})$ is honest-verifier zero-knowledge for a relation R if there exists a simulator `sim` such that for any $(X, w) \in R$, the simulator on input an instance X returns a transcript (X, A, c, s) that is indistinguishable from a transcript of a run of $\texttt{Prove}(X, w)$ with $\texttt{Verify}(X)$.

A sigma protocol as above has a c-simulator if there is an algorithm `c_sim` that takes a pair (X, c) as input and for any $(X, w) \in R$ and $c \in Chal$, the c-simulator on input (X, c) produces an accepting transcript (X, A, c, s) distributed indistinguishably from a transcript of a run of $\texttt{Prove}(X, w)$ with $\texttt{Verify}(X)$ conditioned on the event that `Verify` chooses challenge c.

The c-simulator and the induced honest-verifier simulator for our template protocol are given in Figure 4.16. The key idea is that the simulator is free to choose the order in which it creates the transcript, unlike the real prover who must commit to her commitment before she sees the challenge. Indeed, the simulator picks the commitment last, picking a random response and challenge (or in case of the c-simulator, taking a challenge as input) then solving the verification equation for the commitment.

Observe that for fixed X, the distribution of simulated triples is identical to that of protocol transcripts: c is uniform in its domain and (A, s) are uniform subject to the verification equation holding.

The prover thus has security guarantees against a verifier who chooses her challenge randomly: such a verifier cannot learn anything about the prover's knowledge, other than that the prover knows what she claims to know. What about a dishonest verifier? If our verifier chooses her challenge as a hash of the prover's commitment, the distribution of such transcripts is no longer

```
1 oracle c_sim(X, c)
2   s ↞ W
3   A ← h(s) +_X (−c) ·_X X
4   return (X, A, c, s)
5 oracle sim(X)
6   c ↞ F
7   return c_sim(X, c)
```

Figure 4.16: C-simulator and simulator for sigma protocols.

simulable. This does not imply that such a verifier learn anything about the prover's witness but is still not a satisfactory situation. Two approaches exist in the literature to solve this problem: one can restrict the challenge space, yielding full zero-knowledge at the cost of soundness which can be recovered by running the protocol many times. Alternatively, one can add an extra round to the start of the protocol in which the verifier commits to her challenge before seeing the prover's commitment.

We do not pursue these techniques further here because we are primarily interested in non-interactive proof schemes derived from sigma protocols. Using the very technique that caused problems in the interactive case, choosing the challenge as a hash of the prover's commitment (and the instance), we obtain non-interactive full zero-knowledge.

Security for the verifier. To assess the verifier's security guarantees, we introduce special soundness.

Definition 4.17 (special soundness). Let a sigma protocol for a relation R be given. A matching pair is a pair of accepting transcripts (X, A, c, s) and (X, A, c', s') with $c \neq c'$.

The sigma protocol has special soundness if there is an efficient extraction algorithm that on input any matching pair, outputs a witness w such that $R(X, w)$ holds (where X is the first component in the matching transcripts).

The preimage-proof protocol for a linear map has this property since $w' \leftarrow \frac{1}{c-c'} \cdot (s - s')$ has the property

$$h(w') = \frac{1}{c-c'} \cdot (h(s) - h(s')) = \frac{1}{c-c'} \cdot (A + c \cdot X - A - c' \cdot X) = \frac{c-c'}{c-c'} \cdot X = X$$

Special soundness implies soundness in the following sense.

Lemma 4.18. A sigma protocol for a relation R that has special soundness is also $1 - \frac{1}{|Ch|}$ sound.

Proof. Consider the verifier at the point in time when she has received an instance X and commitment A but not yet chosen a challenge, in a protocol execution with an arbitrary P'. To each challenge $c \in Ch$ we can assign a probability $p(c)$ that the prover P' will deliver an accepting response (the honest prover chooses responses deterministically, but P' may not). Write C for the random variable denoting the challenge that the verifier chooses and Acc for the random variable that is 1 if the verifier accepts, else 0. The probability of the verifier accepting is

$$\mathbf{Pr}[Acc = 1] = \sum_{c \in Ch} \mathbf{Pr}[Acc = 1 \mid C = c] \cdot \mathbf{Pr}[C = c] = \frac{1}{|Ch|} \sum_{c \in Ch} p(c)$$

Suppose that the instance X in question is not in $L(R)$. In this case, at most one of the terms in the sum can be non-zero: if any two distinct values c and c' have non-zero probability, this implies existence of values s and s' such that (X, A, c, s) and (X, A, c', s') are both accepting transcripts, but then by special soundness, if we were given these transcripts we could extract a witness, therefore a witness must exist and $X \in L(R)$, yielding a contradiction. Since the value of $p(c)$ is bounded by 1 for any c, we get $Pr[Acc = 1 \mid X \notin L(R)] \leq \frac{1}{|Ch|}$ as desired. *q.e.d.*

4.3.3 The Fiat-Shamir transformation

In the Fiat-Shamir transformation, one replaces the verifier's random choice of challenge in a sigma protocol by the value of a hash function on the elements obtained so far (instance and commitment), yielding a non-interactive proof system.

Definition 4.19 (Fiat-Shamir transformation). The Fiat-Shamir transformation of a sigma protocol by function $\mathbf{H}$ is the non-interactive proof system in Figure 4.20. We assume $\mathbf{H}$ is a function $X \times Com \to Ch$.

```
oracle prove(x, w)
  (com, st) ← Commit(x, w)
  ch ← H(x, com)
  res ← Respond(x, w, com, ch, st)
  return (com, ch, res)

oracle verify(x, com, ch, res)
  if H(x, com) ≠ ch then
      return 0
  end if
  if Verify(x, com, ch, res) ≠ 1
      then return 0
  end if
  return 1
```

Figure 4.20: The Fiat-Shamir transformation.

This transformation is usually done using a cryptographic hash function that is assumed to be "(pseudo)random". Bellare and Rogaway [BR93] proposed the random oracle model to reason about security of the Fiat-Shamir construction. This model idealises the hash function $\mathbf{H}$ as a randomly chosen function between its domain and range. The essence of the random oracle model lies in how the function $\mathbf{H}$ appears in the formulations of the zero-knowledge and proof of knowledge properties: the postulated algorithm (simulator or extractor) acts as an oracle for the function $\mathbf{H}$ and can thus choose its values suitably.

Lemma 4.21. Fiat-Shamir transformed sigma protocols following our template are zero-knowledge proof systems in the random oracle model if the image of the linear map h has exponential size (relative to the security parameter).

Proof. We give a simulator S in Figure 4.22 that can handle simulation queries and random oracle queries, then argue that it is indistinguishable from the Prove algorithm and a true random oracle. Let Ch be the range of the random oracle. We pick a challenge and use the c-simulator to create a simulated proof, then define our random oracle at the relevant input to return the chosen challenge. This technique is called "patching" or "programming" the oracle. In the event that the adversary has already called the oracle at this value, the simulator fails. We will show that the chance of this happening is negligible.

As long as the simulator does not fail, we already know that the simulated proofs returned by the c-simulator are identically distributed to proofs made by the Prove algorithm. Challenges

picked by the simulator are uniformly random in Ch, again just like a true random oracle would pick them. The simulator fails if H is already defined at a point that it picked (through the c-simulator) uniformly at random in the domain of the oracle. Since the simulator drew these points freshly from an exponentially large set, the chance of the adversary having called the oracle at these points is negligible. *q.e.d.*

```
1  oracle sim(x)
2    c ↞ Ch
3    (x, a, c, s) ← c_sim(x, c)
4    q ← (x, a)
5    if H[q] ≠ ⊥ and H[q] ≠ c then
6        fail
7    end if
8    H[q] ← c // program RO
9    return (a, c, s)
10 oracle RO(x)
11   if H[x] = ⊥ then
12       H[x] ↞ Ch
13   end if
14   return H[x]
```

Figure 4.22: Zero-knowledge simulator for sigma protocols.

4.3.4 Hash compression

In our definition above, the prover sends the verifier the tuple (com, ch, res). An easy optimisation is to send just the pair (com, res) and have the verifier recompute $ch = \mathbf{H}(x, com)$ since she needs to perform this computation anyway to verify the challenge. In Schnorr's protocol, a proof with this optimisation is in $\mathbb{G} \times \mathbb{Z}_{|\mathbb{G}|}$. Since group elements are typically larger when encoded than group exponents, a technique called hash compression (our term) allows to reduce a proof to $\mathbb{Z}_{|\mathbb{G}|} \times \mathbb{Z}_{|\mathbb{G}|}$. In essence, one transmits the pair (ch, res) instead. The verifier then recomputes the commitment by solving the verification equation using the prover's provided challenge and then verifies that the hash of the instance and computed commitment matches the challenge. We express these two steps as one in the formula

$$ch \stackrel{?}{=} \mathbf{H}(X, G^{res}/X^{ch})$$

We can also apply this technique directly to our abstract template to get the equation

$$ch \stackrel{?}{=} \mathbf{H}(h(res) +_{\mathcal{X}} (-ch) \cdot_{\mathcal{X}} X)$$

and thus have a proof in $\mathbb{F} \times \mathbb{F}$ as opposed to $\mathcal{X} \times \mathbb{F}$. We warn that hash compression is not secure in some applications of sigma protocols like Chaum-Pedersen signed ElGamal (see Section 5.2.5) due to the need for the challenge to remain hidden. Indeed, the original authors [ST13] first overlooked this point and gave the hash-compressed version of their scheme; the paper has since been revised.

4.3.5 Disjunctive proofs

Given two linear maps h and k and sigma protocols following our template for proving knowledge of preimages, it is easy to create a proof that one knows both a preimage w under h and a preimage u under k for some elements in the respective ranges: just run the two sigma protocols, either in sequence or "in parallel" by combining the respective commitments, challenges and responses into single messages. We could call this transformation that the (category-theoretic) product of two sigma protocols.

Where there are products, the question of coproducts arises: given two such template protocols for maps h, k can we construct a sigma protocol to prove that we know a preimage w of h **or** a preimage u of k for given range elements — without revealing which of the two preimages we know? Cramer [C96] gives a construction for such disjunctive proofs assuming that the two protocols share the same challenge space and that this space is a group. This will typically be the case.

The idea behind the disjunctive proof is to run the two sigma protocols "in parallel". The verifier provides a single challenge; the prover picks a challenge for each protocol such that the sum of the prover's two challenges equals the verifier's challenge. This allows the prover to simulate one proof and then subtract her simulated challenge from the verifier's one to obtain a challenge for the other proof. Since real and simulated proofs are indistinguishable, this construction hides which of the two proofs is the real one.

This construction retains special soundness and therefore soundness and the proof-of-knowledge property since any two disjunctive proofs on the same instance that differ in the verifier's challenge must also differ in the challenge for at least one of the contained, individual proofs. We omit the details and proofs for this disjunctive construction. We will prove in the following sections that (strong) Fiat-Shamir transformed proofs built upon our template satisfy a property that we will define soon known as simulation sound extractability which implies soundness, zero-knowledge and proof-of-knowledge; the same holds for disjunctions of such proofs using the construction sketched above.

5 Advanced topics on zero-knowledge proofs

In this chapter we present our own work on the theory of zero-knowledge proofs. We begin by distinguishing weak from strong proofs based on our work with Pereira and Warinschi [BPW12b] and develop the underlying theory further; in particular we investigate the issue of soundness in greater depth in Section 5.1.2. The presentation of this material using code-based games is also new to this thesis. Next, we investigate "even stronger proofs" that we call multi-proofs. This work is based on a paper with Fischlin and Warinschi that has not yet been published. In this thesis we adopt a slightly different proof strategy to the paper, performing some of the proof steps in a different but equally valid order.

Encrypt-and-prove schemes. The application of proofs of knowledge to strengthen encryption was mentioned by Dolev, Dwork and Naor in 1991 in their paper on non-malleability [DDN91]; Naor and Yung used zero-knowledge proofs in their double-encryption technique to achieve non-adaptive chosen ciphertext security (CCA1) in 1990 [NY90] and Rackoff and Simon in 1991 [RS91] used non-interactive zero-knowledge proofs of knowledge to define and achieve adaptive chosen-ciphertext security (CCA2). In 1999, Sahai [S99] showed that the Naor-Yung construction also yielded CCA2 security under some extra conditions on the proofs involved, including the important notion of simulation soundness that they introduced in this paper. At a casual glance, it would seem like non-interactive zero-knowledge proofs of knowledge are exactly what one needs to turn a basic encryption scheme into a CCA2 secure one; the Naor-Yung construction reinforces this intuition. However, Shoup and Gennaro pointed out in 1998 [SG98] that the "obvious" security proof for an equally obvious and efficient construction does not work. The construction in question adds a Fiat-Shamir transformed sigma-protocol proving knowledge of the plaintext and randomness used to encrypt to all ciphertexts, a technique known as "encrypt+PoK" in general and "TDH0" or "signed ElGamal" when applied to the ElGamal [E85] encryption scheme. Tsiounis and Yung [TY98] claimed a CCA2 proof of this construction in 1998 under a "knowledge assumption" that is hard to justify without resorting to the generic group model; Schnorr and Jakobsson [SJ00] gave a proof in 2000 explicitly using the generic group model. However,

these proofs actually prove a slightly stronger notion than CCA2 (which is technically known as "plaintext awareness") yet Seurin and Treger [ST13] recently proved that signed ElGamal is not "plaintext aware" without generic group assumptions. Their result does not disprove CCA2 security of signed ElGamal however.

Our contributions. In 2012, together with Pereira and Warinschi [BPW12b] we showed that the Fiat-Shamir transformation comes in two forms that are not usually distinguished by previous security notions, yet one of the two does not gain any security at all when used in encrypt-and-prove schemes since the resulting encryption scheme is malleable. In this thesis, we continue this line of research and give what we believe is strong evidence that the signed ElGamal construction, even in its stronger form, is not in fact CCA2 secure under common assumptions. Conversely, we give a new and stronger notion of proofs of knowledge that we call multi-proofs under which the "obvious" approach to CCA2 does work and prove this, although the proof will turn out to be far from obvious. Our results come in the form of a definition of multi-proofs and three main theorems:

- Fiat-Shamir-Schnorr, the proof scheme underlying signed ElGamal, is not a multi-proof unless a problem commonly thought to be hard (one-more discrete logarithm) is easy. (Theorem 5.32)
- A simulation sound multi-proof in an encrypt+PoK construction (with an IND-CPA basic scheme) does give CCA2 security. (Theorem 5.40)
- Signed ElGamal cannot be shown CCA2 secure by any reduction in the random oracle model unless ElGamal with interactive Schnorr proofs is insecure. (Theorem 5.50)

5.1 Strong proofs

The Fiat-Shamir transformation has been presented in different ways in the literature using different inputs to the hash function. Our presentation follows the original scheme by Fiat and Shamir in which the hash is taken over the instance and commitment of the sigma protocol. One can hash extra elements, for example nonces or identifiers to achieve domain separation when a protocol uses multiple proofs; this does not affect the security of the transformation. Further, one can include an arbitrary message in the hash input; the resulting scheme is then known as a "signature of knowledge", a digital signature with any NP statement as the public key and a

witness as the secret key. Applied to the Schnorr protocol, the resulting scheme is unsurprisingly known as a "Schnorr signature".

Another common variation on the Fiat-Shamir transformation however comes with potential security problems. Unfortunately, existing theory does not easily separate this variation, which we call the weak Fiat-Shamir transformation, from the original one. Indeed, the proof of knowledge property for the original one follows from special soundness yet the weak variation has this property too.

In this section, we reproduce our arguments from our recent paper with Pereira and Warinschi [BPW12b]: first we introduce the weak Fiat-Shamir transformation and show some problems with soundness, knowledge extraction and malleability. Secondly, we give a notion of strong proofs that avoids these problems and prove that the original Fiat-Shamir transformation, which we call strong Fiat-Shamir, is indeed a strong proof. We conclude this section with the popular Fiat-Shamir-Schnorr scheme of which we will study applications in the rest of this thesis.

5.1.1 Weak Fiat-Shamir

A variant of the Fiat-Shamir transformation appears in some papers such as Bellare and Rogaway's foundational paper for the random oracle model [BR93] where the function H is only applied to the commitment, not the instance. We call this variant the weak Fiat-Shamir transformation:

Definition 5.1 (Weak Fiat-Shamir). The weak Fiat-Shamir transformation is a variant of the Fiat-Shamir transformation (Definition 4.19) in which $\mathbf{H}$ is a function $Com \to Ch$ and line 3 of algorithm `prove` is replaced by $ch \leftarrow \mathbf{H}(com)$; lines 7–8 of `verify` similarly become `if` $\mathbf{H}(com) \neq ch$ `then return 0`.

Exponent inversion. Consider the weak Fiat-Shamir transformation of Schnorr's protocol in Figure 5.2. If the exponent group $\mathbb{Z}_{|\mathbb{G}|}$ is a field, we can invert elements and get the algorithm P to create "simulated" instance/proof pairs.

This prover, when fed with a random group element A produces an instance X and proof A, c, s that verify w.r.t. $\mathbf{H}$ but never needs the discrete logarithms of A or X. Informally, this prover does not need to "know" the discrete logarithms. What about special soundness? One

```
algorithm prove(X, w)
  // X = G^w
  a ↞ Z_{|G|}, A ← G^a
  c ← H(A)
  s ← a + cw  // in Z_{|G|}
  return (A, c, s)
```

```
algorithm verify(X, π)
  (A, c, s) ← π
  if c = H(A) ∧ G^s = A · X^c then
      return 1
  else
      return 0
  end if
```

```
algorithm P(A)
  c ← H(A)
  s ↞ Z_{|G|}
  X ← (G^s/A)^{1/c}
  return (X, A, c, s)
```

Figure 5.2: Weak Fiat-Shamir-Schnorr and a "prover".

argument goes: special soundness does not help here since giving P a different answer to the hash query produces not only a different c but a different X too. Much more damningly, one can construct a reduction from extraction to DLOG. Give P an A obtained from a discrete logarithm challenger: if any extractor succeeds in obtaining w s.t. $X = G^w$, compute $a = s - cw$ to solve the challenge. So if DLOG is hard in some group $\mathbb{G}$ then a weak Fiat-Shamir-Schnorr proof cannot imply "knowledge" of the discrete logarithm of the instance in question. This idea generalises to our proof template: extraction implies an inverter for the homomorphism h.

For comparison we give the algorithms to make Schnorr proofs for the honest prover, zero-knowledge simulator and weak Fiat-Shamir "cheater" side-by-side in Figure 5.3. Note the different order of execution and that only the honest prover holds a witness.

5.1.2 Soundness

When using Schnorr's protocol, the existence of discrete logarithms is not up for debate: Schnorr's protocol is exclusively concerned with knowledge of a discrete logarithm. Consider the weak Fiat-Shamir transformation of the Chaum-Pedersen protocol in Figure 5.4, proving that (X, Y, Z) is a Diffie-Hellman triple i.e. $\log_G(X) = \log_Y(Z)$. This protocol is not even sound! By the same argument as above, for any A, B there is a prover P that creates a "simulated" proof.

Prover	**Simulator**	**Cheater**
`input:` X, w	`input:` X	`input:` A
$a \twoheadleftarrow \mathbb{Z}_{\lvert\mathbb{G}\rvert}, A \leftarrow G^a$	$s \twoheadleftarrow \mathbb{Z}_{\lvert\mathbb{G}\rvert}$	$c \leftarrow \mathbf{H}(A)$
$c \leftarrow \mathbf{H}(A)$	$c \twoheadleftarrow \mathbb{Z}_{\lvert\mathbb{G}\rvert}$	$s \twoheadleftarrow \mathbb{Z}_{\lvert\mathbb{G}\rvert}$
$s \leftarrow a + c \cdot w$	$A \leftarrow G^s / X^c$	$X \leftarrow (G^s/A)^{1/c}$
	$H[A] \leftarrow c$ `// for RO`	
`return` (X, A, c, s)	**`return`** (X, A, c, s)	**`return`** (X, A, c, s)

Figure 5.3: The weak Fiat-Shamir transformation.

A quick calculation on the exponents shows that (X, Y, Z) need not be a Diffie-Hellman triple: writing a, b, x, y, z for the discrete logarithms of A, B, X, Y, Z to base G we get $cx = s - a$ and $cz = sy - b$ so $xy = z$ holds if and only if $ay = b$ but algorithm `P` works on any inputs A, B, Y.

The Helios voting scheme [A08] uses such proofs to demonstrate validity of a ciphertext in a ballot where Y is fixed in advance as the election public key. Here one can choose A and B independently and uniformly at random to make a ciphertext (X, Z) and proof (A, B, c, s) distributed indistinguishably from a valid ciphertext (unless the DDH problem is easy) yet the ciphertext is with overwhelming probability not valid and breaks the election's tallying process. This attack, described further in Section 6.9 was one of our motivating examples for studying the problems of the weak Fiat-Shamir transformation [BPW12b]: we have successfully carried it out in practice against a test installation of Helios.

We can generalise this attack to our sigma protocol template as well. We defined the homomorphism h as a map $\mathcal{W} \to \mathcal{X}$, inducing a set $\mathbf{Im}(h) \subseteq \mathcal{X}$ of valid instances. The cheater however starts with an arbitrary $A \in \mathcal{X}$. From the verification equation and the linearity of h it is clear that the resulting instance X is an element of the image of h if and only if A was already in this image. The cheater can therefore always generate proofs of instances in $\mathcal{X} \setminus \mathbf{Im}(h)$ whenever this set is non-empty and efficiently sampleable.

Whether or not this technique allows the cheater further control over the structure of X depends on the properties of the homomorphism h in question. For example, in Helios this technique creates a valid-looking ballot for a random value (modulo the order of the group used) but cannot create a ballot for a value of one's choice, which would be a more powerful attack since one could add or subtract an arbitrary value from a candidate's tally. If all the trustees

```
1  algorithm prove(X, Y, Z, w)
2    // X = G^w ∧ Z = Y^w
3    a ↞ Z_|G|, A ← G^a, B ← Y^a
4    c ← H(A, B)
5    s ← a + cw  // in Z_|G|
6    return (A, B, c, s)

7  algorithm verify(X, π)
8    (A, B, c, s) ← π
9    if c = H(A, B) ∧
10       G^s = A · X^c ∧ Y^s = B · Z^c then
11       return 1
12   else
13       return 0
14   end if

15 algorithm P(A, B, Y)
16   c ← H(A, B)
17   s ↞ Z_|G|
18   X ← (G^s/A)^(1/c)
19   Z ← (Y^s/B)^(1/c)
20   return (X, Y, Z, A, B, c, s)
```

Figure 5.4: Weak Chaum-Pedersen and "prover".

in a Helios election and at least one voter are dishonest however, an extension of this attack by Pereira [BPW12b] creates an encryption key pair with a trapdoor that allows exactly one, pre-determined modification to the election result. Worse still, this attack is undetectable as all values are provably indistinguishable from honestly generated ones as long as the result lies in an acceptable range (for example, the tally does not exceed the number of votes cast) and there is uncertainty about the votes cast by a number of other voters.

5.1.3 Malleability

While the definition of a proof of knowledge does not require proofs to be non-malleable, in some applications non-malleable proofs are desirable, particularly in the Encrypt+PoK construction that we will consider soon. In the original or strong Fiat-Shamir transformation, proofs are non-malleable in the sense that changing any components requires a new hash to be computed, forcing the entire proof to be recomputed from scratch. We will show that encryption schemes constructed with the help of such proofs meet the formal definition of non-malleability.

However, weak Fiat-Shamir proofs can be malleable. Given a "weak Schnorr" instance and proof (X, A, c, s), the verification equations are $c = \mathbf{H}(A)$ and $G^s = A \cdot X^c$. One can now easily modify such a proof by picking a random $r \twoheadleftarrow \mathbb{Z}_{|\mathbb{G}|}$ and setting $X' \leftarrow X \cdot G^{r/c}$ and $s' \leftarrow s + r$. In our proof template, one sets $X' \leftarrow X + \frac{1}{c} \cdot h(r)$. The new instance/proof pair (X', A, c, s') will still verify if the original one did since we have not modified the elements of the hash equation and we have simply multiplied both sides of the sigma protocol verification equation by G^r.

In an application using such proofs, one can detect a mauled proof if the original proof is also available since the two will share the same hash c. In Helios, one can detect all known attacks of the Cortier-Smyth type [CS11, CS13] by comparing hashes and raising an alarm if the same hash value is ever used twice. These are attacks that involve malicious voters submitting modifications of ballots already posted by other voters, threatening the privacy of the original voter.

5.1.4 Strong proofs

A proof of knowledge has an extractor that can produce witnesses from proofs, for any instance selected and given to a verifier *in advance*. In other words, if we have a verifier who only accepts proofs relating to one particular instance then the extractor works for any prover that convinces this verifier. A strong proof of knowledge on the other hand has an extractor that finds witnesses from any prover who can make her own instance/proof pair; the prover is free to chose her own instance. In a rewinding setting, running the prover multiple times and giving different answers to random oracle queries may potentially lead such a prover to output proofs for different instances, if one is not careful.

In defining a strong proof, we face a hurdle common to most definitions in the area of zero-knowledge: while it is easy to define a "strong proof in the rewinding RO model" or indeed any other suitable model, it is not so easy to give a generic definition that specialises across different models. In the literature, ROM and non-ROM based definitions have evolved in parallel and the issue of weak proofs has only ever appeared in the ROM (although the root cause of weakness is rewinding rather than the random oracle). While many non-ROM definitions and schemes are strong by default (in fact even straight-line and hence multi-proofs), ROM proofs are usually trivially simulation-sound whereas this property can take extra effort to achieve outside the ROM. We first give a general definition of strong proofs that sacrifices some precision in exchange for less dependence on a particular model, then we give the rewinding-based ROM instantiation in full code-based detail and prove that the strong Fiat-Shamir transformation meets this definition.

Definition 5.5 (strong proof, generic version). Let a non-interactive proof system $(\texttt{Prove}, \texttt{Verify})$ be given with sets X, W, Π. A strong prover is an algorithm P that produces outputs in $X \times \Pi$ s.t. if $(x, \pi) \leftarrow P()$ then with overwhelming probability, $\texttt{Verify}(x, \pi)$ returns 1.

The proof system is a strong proof of knowledge for a relation $R \subseteq X \times W$ if there is an efficient extractor K such that after running $(x, \pi) \leftarrow P()$, $K(x, \pi)$ can produce a witness w satisfying $R(x, w)$ with overwhelming probability.

This definition, like that of a proof of knowledge, has different interpretations in the various models:

- In the random oracle model, the initial run of the prover has access to a real random oracle and the condition that the proof must verify with overwhelming probability holds w.r.t. this oracle. The extractor K gets the transcript of all oracle queries made by the prover.
- A strong proof with rewinding extractor allows the extractor K black-box access to further copies of the prover with the same randomness as the initial invocation. In the random oracle model, the extractor answers random oracle queries for all further copies of the prover.
- In the CRS model, the extractor provides the initial CRS that all copies of the prover use.

To give a code-based version of this definition, we require a three-party game-based definition containing the game itself, the adversary and the extractor.

Definition 5.6 (strong proof in rewinding ROM). A non-interactive proof system $(\texttt{Prove}, \texttt{Verify})$ on sets X, W, Π is a strong proof of knowledge in the rewinding ROM for a relation $R \subseteq X \times W$ if there is an efficient extractor K such that the following holds.

For any strong prover P (who may call the RO), the probability of the game in Figure 5.7 returning 1 is overwhelming.

In Figure 5.7, the set R_P is the space of random coins for the prover and R_{RO} is the range of the random oracle. The oracles in this game are called by the extractor, not the prover, except for `P_main.ro` that handles random oracle queries for the main run of the prover (line 4) and stores them in the list H, which is returned to the extractor on line 5. The extractor may launch further copies of the prover with the `instance` oracle; the extractor is responsible for answering random oracle queries made by these instances.

```
1  oracle initialise
2    r ↞ R_P
3    H ← [ ]
4    (x*, π*) ← P(r) // main run
5    return (H, x*, π*)
6
7  oracle P_main.ro(x)
8    if H[x] = ⊥ then
9      H[x] ↞ R_RO
10   end if
11   return H[x]
12 oracle instance
13   (x, π) ← P(r)
14   return (x, π)
15
16 oracle finalise(w*)
17   if R(x*, w*) then
18     return 1
19   else
20     return 0
21   end if
```

Figure 5.7: Strong proof game.

As the name suggests, the strong Fiat-Shamir transformation yields strong proofs.

Theorem 5.8. The strong Fiat-Shamir transformation of specially sound sigma protocols yields strong proofs in the random oracle model with a rewinding extractor.

Proof. For any prover that outputs a valid instance/proof pair $(x, \pi = (com, ch, res))$ we know that ch must be the value of the random oracle on input (x, com) so the prover must have queried the oracle at this point except with a negligible probability that she just guessed the value (since in Section 2.6.1 we stated that a random oracle's range is exponentially large). The rewinding extractor launches a copy of the prover and gives it the same answers to all queries up to the point when the prover makes the oracle query on the pair (x, com) for which the

main copy made a proof. If no such query exists, the extractor aborts. From this query on, the extractor chooses fresh, random answers to all random oracle queries. This technique is known as "forking" the prover.

If the prover again makes a proof on the same oracle query then we conclude that the forked copy of the prover makes a proof on the same instance as the main copy, since the two runs had identical inputs and randomness up to the point when they fork. Further, with overwhelming probability the extractor chose different challenges for the main and forked copies since these choices were independent and uniform. We use these facts to invoke the special-soundness extractor, obtaining the required witness.

The forking lemma of Bellare and Neven [BN06] gives the probability that the prover will make another proof using this challenge (in relation to the probability of the prover making such a proof in the main run). While the forking lemma is a general statement on probabilistic algorithms, we give the lemma specifically adapted to special soundness extraction. The proof of the lemma remains the same and we refer the reader to the original paper for the proof.

Lemma 5.9 (forking lemma). Let A be an algorithm that makes up to q random oracle queries and returns values (x, π). Let I be a random variable that is $i > 0$ if (x, π) is a valid instance/proof pair and the proof π uses algorithm A's i-th random oracle query as its challenge. Otherwise, $I = 0$. Let acc be the probability that $I > 0$ in an execution of algorithm A.

Let frk be the probability that in an execution of the strong proof experiment, $I = i > 0$ in the main run of the prover (i.e. the prover makes a valid proof) and the forked copy run by our extractor receives a different answer to the i-th oracle query yet still makes a valid proof on this query (so special soundness extraction succeeds).

Then (where $\mathcal{R}$ is the range of the random oracle)

$$\mathsf{frk} \geq \mathsf{acc} \cdot (\mathsf{acc}/q - 1/|\mathcal{R}|)$$

For our purposes it is sufficient to note that acc will be overwhelming in the security parameter, q polynomial (since we consider efficient provers) and $1/|\mathcal{R}|$ negligible as $\mathcal{R}$ is exponentially large. So frk is still overwhelming, proving that our extractor works as required. *q.e.d.*

5.1.5 Forking and weak proofs

Our proof above invokes the forking lemma in the same way as it is commonly used in the literature. In our case this use happens to be correct but we have swept an important argument under the carpet (the reader is encouraged to try and discover which one, before continuing).

The forking lemma operates at a high level of abstraction: the original version does not even mention "proofs" at all but just references algorithms that return arbitrary values and an index i and makes a statement on the probability of two correlated runs of an algorithm returning the same non-zero index. In particular, the forking lemma does not distinguish between weak and strong proofs. (Our formulation includes the term "strong proof experiment" but the same lemma holds, with exactly the same proof, if "weak" is substituted.) Yet we know that weak proofs are not necessarily sound so there cannot be a proof of their soundness based on the forking lemma. If the above proof is repeated for the weak transformation, it must fail somewhere.

To understand this discrepancy, we must look at the use of the forking lemma in some detail. The post-conditions of the forking lemma guarantee a certain probability that two runs of the prover return values $(x, \pi) = (x, (a, c, s))$ and $(x', (a', c', s'))$ such that both are valid proofs on the same index.

To prove that the strong Fiat-Shamir transformation yields strong proofs, we wish to feed these values to the special-soundness extractor and obtain a witness to the instance x. The extractor has three preconditions on such inputs, apart from their validity as instance/proof pairs:

$$(1) \quad x = x' \qquad (2) \quad a = a' \qquad (3) \quad c \neq c'.$$

Condition 3 is already accounted for in our formulation of the forking lemma. The probability of both challenges colliding is $1/|\mathcal{R}|$ since they are independent and both uniform; this explains the subtraction of the corresponding term in the equation for frk.

Condition 2 can be argued as follows. In the first run, the prover got c as the result of a query $\mathbf{H}(x, a)$ on exactly the element a appearing in the prover's output, otherwise the random variable I would not have been set accordingly. In the second run, the prover obtained identical inputs and outputs up to the point where she asked $H(x', a')$ to get c', therefore she must have asked this query on the same value of a.

For the strong transformation, since x was also a hash input the same reasoning applies to show $x = x'$, discharging condition 1. This is the point where an attempt to prove the weak transformation sound breaks down: since it uses only $c = \mathbf{H}(a)$, there is no reason that the prover must return the same x again. The forked prover still has an identical view of the main one

up to the point when the two fork on the critical hash query which implies that all values chosen before this time are identical in both runs — but nothing prevents a weak prover from choosing or modifying an instance (x-value) after this point. Indeed, the weak Fiat-Shamir cheater described earlier will always come up with different values of x and x' when $c \neq c'$. So while the forking lemma still holds for the weak transformation, its postconditions are too weak to satisfy the preconditions for special-soundness extraction.

As a corollary to this observation, we answer the question "Which parts of an instance/proof pair do we need to hash?" — while hashing the entire instance (the strong transformation) always works, there are practical cases where one can still obtain security while hashing fewer values. For example, consider a verifier that only accepts proofs made on one particular instance (imagine non-interactive Schnorr proofs used in an authentication protocol where an instance is fixed during set-up as the public key of some party). In this case, we get condition 1 for free since any proof on a different instance will not convince the verifier; any prover that convinces the verifier with probability acc must therefore have a probability at least acc of using one particular instance and the corresponding probability frk of satisfying the extractor's preconditions. In other words, the weak Fiat-Shamir transformation is actually fine in this case.

We state the condition for hashing (parts of) the instance as follows. The sigma-protocol commitment needs to be hashed in any case.

> Two successive runs of the prover with the same randomness, which both convince the verifier, also produce the same instance.

Note that this condition is automatically satisfied for all parts of the instance that are hashed, by the argument above. For any non-hashed part it remains to show why it cannot vary between runs, for example that it is fixed in advance between the prover and verifier.

5.1.6 Simulation sound extractability

Simulation soundness guarantees soundness after using the simulator. However, soundness of a proof of knowledge is usually proved using an extractor and security properties of schemes using proofs of knowledge often rely on the extractor too. Simulation sound extractability (SSE) says that the extractor still works after using the simulator, with the obvious restriction that one cannot extract from a simulated proof. We base our presentation of SSE on our earlier work [BPW12b]; SSE was originally introduced by Groth [G06].

Definition 5.10 (simulation sound extractability). An SSE prover is like a strong prover but can additionally make `prove` queries to the zero-knowledge simulator, as specified in Figure 5.11. An SSE prover must with overwhelming probability both (1) return a pair (x, π) that verifies and (2) never return a pair (x, π) where it produced π using a `prove` query on statement x and any witness w.

If the extractor launches further copies of the prover, the extractor handles both `ro` and `prove` queries of these copies. The extractor gets the transcripts H and Π of all queries made by the main prover. A proof scheme is simulation sound extractable (SSE) in the random oracle model if there is an extractor K such that for any SSE prover P, the probability of the SSE game returning 1 is overwhelming.

```
1  oracle initialise
2    r ↞ R_P
3    H ← [ ]; Π ← [ ]
4    (x*, π*) ← P(r) // main run
5    return (H, Π, x*, π*)
6
7  oracle P_main.ro(x)
8    h ← S.RO(x)
9    H ← H :: (x, h)
10   return h
11 oracle P_main.prove(x, w)
12   if not R(x, w) then
13     return ⊥
14   end if
15   // use simulator
16   π ← S.sim(x)
17   Π ← Π :: (x, π)
18   return π
```

Figure 5.11: Changes to the strong proof game (Figure 5.7) for simulation sound extractability.

Our notion of SSE is a straightforward extension of the notion of a strong proof, obtained by adding `prove` queries to the notion and routing `ro` queries of the main prover through the simulator. We have renamed the simulator's main oracle to `S.sim(x)` compared with `S(x)` in Definition 4.4. In the CRS model, defining SSE is a more complex task which we leave for future work. Briefly, the problem is that the simulator may need a CRS of one kind with a simulation trapdoor whereas the extractor may need an extraction trapdoor and the implications of different options for "Who gets to create the CRS?" are not yet clear to us.

5.1.7 Fiat-Shamir-Schnorr

The Fiat-Shamir-Schnorr protocol is the simplest instance of a non-interactive proof derived from a sigma protocol. We will study it later on so we give a formal definition here.

Definition 5.12 (Fiat-Shamir-Schnorr). The Fiat-Shamir-Schnorr proof is the construction in Figure 5.13.

```
algorithm Prove(x, X)
  // precondition: X = G^x
  a ↞ Z_{|G|}, A ← G^a
  c ← H(X, A) // strong FS
  s ← a + cx
  return (A, c, s)
```

```
algorithm Verify(X, A, c, s)
  if c ≠ H(X, A) or
     G^s ≠ A · X^c then
     return 0
  else
     return 1
  end if
```

Figure 5.13: Fiat-Shamir-Schnorr.

Fiat-Shamir-Schnorr is simulation sound extractable, the strongest security notion that we have defined so far. In fact this holds for a whole class of similar protocols:

Theorem 5.14. Consider the strong Fiat-Shamir transformation of a sigma protocol following our template for a linear map h. Then this scheme is simulation sound extractable if h is injective and the range of the random oracle is exponentially large.

In particular, Fiat-Shamir-Schnorr is a simulation sound extractable proof scheme.

Proof. Our proof uses two ideas. First, the extractor and simulator for proofs following our template can work independently: the simulator only needs to program the random oracle on points which it has chosen itself with high entropy. Secondly, strong Fiat-Shamir proofs are non-malleable in the sense that for a given challenge c it is impossible to find two different proofs except if a collision occurs in the random oracle.

For proofs following our template, the strong-proof extractor runs an additional copy of the prover and replays replies to random oracle queries until it "forks" the prover on the query used to create the proof; the event that there is no such query has negligible probability since the oracle range is exponentially large.

If the prover makes a `prove` query, we run the zero-knowledge simulator from Figure 4.22 which patches the oracle at a point of its own choice. We need to argue that the extractor will not refuse to operate under these conditions. Suppose that we implemented a two-stage oracle for the prover: when the prover asks a `RO` query, we first see if the simulator has patched that oracle at the queried point and return the patched value if it has. Otherwise, we delegate the call to a "real" random oracle. The combination of the prover and the simulator can then be seen as an extended prover w.r.t. the "real" oracle and we can conclude that the extractor must still work against this prover. The extractor only ever interacts with the real oracle: simulated proofs and patched oracle values are internals of this extended prover as long as the prover never asks an extraction query on a proof with a patched oracle value.

To argue that the extractor must never deal with patched oracle values, suppose that the simulator creates a proof (X, A, c, s). The prover is banned by the SSE game from sending this exact proof to the extractor. If the prover sends a proof with a different challenge value c' which was not set by the simulator, the extractor must still work as we have argued above. For any valid proof (X', A', c, s') that shares a challenge with a simulated one, validity implies that c is the oracle value at (X', A') which means that $X = X'$ and $A = A'$ or else a collision has occurred in the random oracle. Since the oracle range is exponentially large and the simulator only patches the oracle to uniformly random values, the probability of such a collision is negligible. It remains to argue that validity implies $s = s'$. For a valid proof, $h(s') = A + c \cdot X = h(s)$ but we assumed that h is injective so $s = s'$ follows. In conclusion, the prover cannot modify a simulated proof without either breaking its validity or choosing a new hash value. *q.e.d.*

We caution that specially sound, zero-knowledge strong proofs are not automatically SSE by the above argument. The requirement for "unique responses" is necessary: we thank D. Unruh for pointing out that if you add an extra bit to the response which is discarded before checking the verification equation, the resulting proof scheme is no longer SSE.

5.2 Encrypt+PoK

Rackoff and Simon [RS91] introduced the notion of security under adaptive chosen ciphertext attack at Crypto '91. We call this notion CCA. It asks that an attacker cannot distinguish a "challenge" encryption of one of two messages of her choice, even when she is allowed to ask for decryption of any other ciphertext both before selecting her two messages and after obtaining the challenge ciphertext.

To prove that an encryption scheme meets this strong notion of security, the usual proof strategy — indeed, the only proof strategy that we know, although it is not always presented this way — is to argue that the decryption oracle is "useless" for the scheme in question because it only tells the attacker what she knows already, i.e. to create a valid ciphertext that she can submit to the decryption oracle, the attacker must already know the contained message.

The idea that Rackoff and Simon used to construct the first CCA-secure encryption scheme was first proposed by Blum, Feldman and Micali [BFM88]: define a ciphertext as a pair consisting of a "basic" ciphertext and a non-interactive proof that the creator of this ciphertext knows the contained message. The intuitive security argument is now that the attacker cannot submit a valid ciphertext to her decryption oracle without making a proof that she already know the encrypted message, in which case she may as well not bother.

The formal argument is a bit more complex. Assume that the basic encryption scheme from such a construction is IND-CPA secure and we wish to show CCA-security of the "extended" construction by means of a reduction. The reduction must perform two tasks: first, when getting a "basic" challenge ciphertext from the IND-CPA challenger, the reduction must augment it with a proof since the adversary expects an "extended" ciphertext. This is easy: if the proof scheme is zero-knowledge, the reduction uses the simulator.

Secondly, the reduction must handle decryption queries which do not exist in the IND-CPA game. It is not enough to tell the adversary that her query is redundant; the reduction has to answer it. The intuition behind proofs of knowledge, as we stated in Section 4.2.3, is that in a security argument we can treat them as if a witness (in our case, the encrypted message) were provided along with the proof. If we can set up the reduction correctly to make formal use of this intuition, the reduction can then answer decryption queries by extracting messages from proofs and returning these to the adversary. We summarise this idea in Figure 5.15.

This construction technique is known as Encrypt+PoK. Unfortunately, the intuition why it should achieve CCA security does not hold in general and most specifically, it does not hold for the most efficient known implementation of Encrypt+PoK, a scheme commonly known as signed

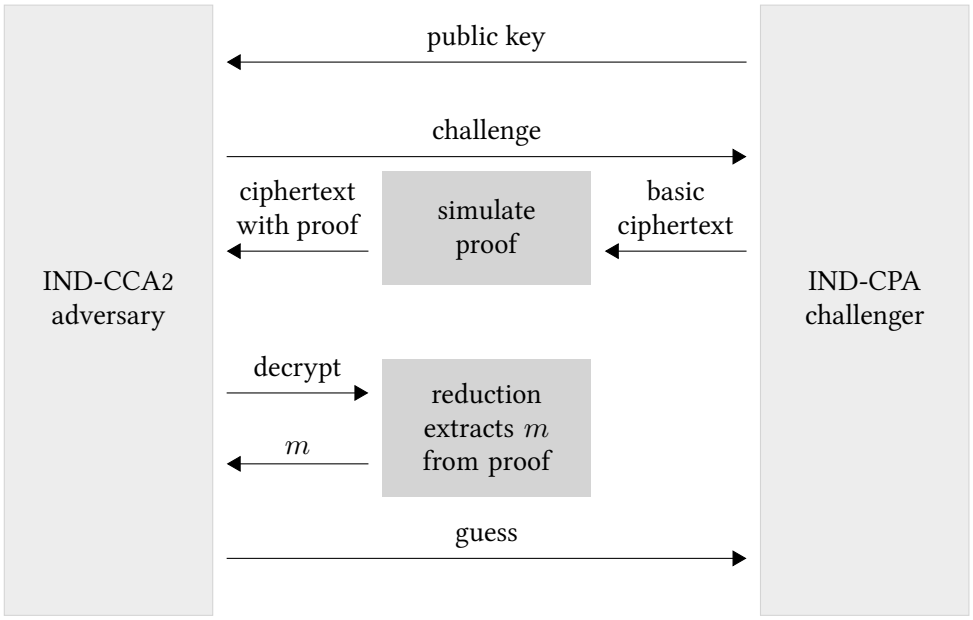


Figure 5.15: Outline of Encrypt+PoK security proof.

ElGamal. A chosen-ciphertext adversary may ask a sequence of decryption queries adaptively, i.e. choose a ciphertext, ask for a decryption then choose a second ciphertext after seeing the decryption of the first one. The obvious way to deal with such an adversary is just to use the proof scheme's extractor each time. If the proof has a straight-line extractor, all is well. Random oracle model proofs however often have rewinding extractors to exploit the special soundness property. The total running time of an extractor and all its copies of the prover is only bounded by a polynomial (in the security parameter) for a single extraction. If one uses the extractor repeatedly to deal with a CCA adversary, the running time can double for each decryption query, resulting in an extractor that takes $\Theta(2^n)$ time to extract from $O(n)$ decryption queries. This problem was first pointed out by Shoup and Gennaro [SG98] although they could neither prove that this exponential blow-up is unavoidable nor give an alternative proof technique to avoid it for signed ElGamal (their other schemes, TDH1 and TDH2, use more involved proofs that do avoid the problem).

Tsiounis and Yung [TY98] proved signed ElGamal CCA secure under a "knowledge assumption" which essentially represents the intuition that making a proof of knowledge implies that you "know" a witness. This assumption was not formally justified or reduced to a lower-level computational model and doing so would seem impossible without invoking the generic group

model. Schnorr and Jakobsson [SJ00] proved CCA security using both the random oracle and generic group models. Their generic group argument relies on being able to extract the witness from the prover's hash query: if you can ask for the hash of G^x in a generic group with no other generators given then you must "know" x. This argument leads to a straight-line extractor. Seurin and Treger however have proved [ST13] that in the random oracle model (without generic groups), if the CDH assumption is hard then signed ElGamal cannot have a straight-line extractor.

In the following sections we analyse the security of Encrypt+PoK and claim the following security levels which we consider one of the principal contributions of this thesis.

scheme	**security**
ElGamal + weak Fiat-Shamir	IND-CPA, not NM-CPA [BPW12b]
ElGamal + strong Fiat-Shamir	NM-CPA [BPW12b], not CCA (Section 5.7)
Encrypt + ss-multi-PoK	CCA (Section 5.6)
Encrypt + ss-straight-line PoK	CCA, examples: [SG98, CS08, ST13]

5.2.1 Formal definition

We give a formal definition of the Encrypt+PoK construction that we will use in future proofs concerning this technique. Informally, what we need is that after extracting a witness from a proof in a ciphertext we can efficiently compute "the" message for this ciphertext. Unfortunately the definition of "the" is a bit involved since we need to prevent an adversary from being able to produce ciphertexts that decrypt one way but the extractor would extract a different message. First we ask for the encryption and proof schemes in the construction to be "compatible" by asking for maps m, w that relate messages and witnesses.

Definition 5.16 (compatible). Let E be a public-key encryption scheme for sets M, PK, SK, C and P be a proof scheme for a relation R with sets X, W, Π. The two are compatible if the instance space X of the proof scheme is $PK \times C$, i.e. an instance is a pair consisting of a public key and a ciphertext, and the following hold:

- There is an efficient algorithm $m : X \times W \to M$ such that for any keypair (pk, sk) in the image of the key generation algorithm of E, any ciphertext $c \in C$ and any $w \in W$, if $R((pk, c), w)$ holds then

$$\texttt{Decrypt}(sk, c) = m((pk, c), w)$$

- There is an efficient algorithm $w : PK \times M \times R_{Enc} \to W$ such that if c is a ciphertext generated as `Encrypt`$(pk, m; r)$ then $R((pk, c), w(pk, m, r))$ holds.

Informally, this says that given a witness to a key/ciphertext pair you can extract the message and given all inputs to the encryption algorithm including the randomness you can find a witness necessary to make a proof. The reason for this abstract formulation (rather than asking for $M = W$) is that some schemes use proofs of knowledge of the message and randomness in a ciphertext, others use proofs that are formally just proofs of knowledge of the randomness but given the randomness one can compute the message from a ciphertext, yet other schemes allow you to extract only the message, yet another version (as found in Helios) not only allows you to extract a message but also prove that the message lies in some domain such as $\{0, 1\}$.

Definition 5.17 (Encrypt+PoK). Let E be an encryption scheme and P a compatible proof. The Encrypt+PoK construction on E and P is the encryption scheme in Figure 5.18.

Sets. Keys and messages are those of E, possibly with some restrictions on messages. Ciphertexts are pairs consisting of an E-ciphertext and a P-proof.

Algorithms. Key generation is the same as for E, encryption and decryption are given in Figure 5.18. In the figure, we refer to algorithms of E and P by prefixing them with the scheme name and a dot.

5.2.2 Signed ElGamal

The simplest Encrypt+PoK construction known to date combines ElGamal encryption with a Fiat-Shamir-Schnorr proof and is variously known as signed ElGamal [SJ00, ST13] or TDH0 [SG98].

```
algorithm Encrypt(pk, m)
  r ↞ R_Enc
  e ← E.Encrypt(pk, m; r)
  w ← w(pk, m, r)
  π ← P.prove(pk, e, w)
  return (e, π)
```

```
algorithm Decrypt(sk, c)
  parse (e, π) ← c
  if not P.Verify((pk, e), π) then
      return ⊥
  end if
  m ← E.Decrypt(sk, e)
  return m
```

Figure 5.18: Encrypt+PoK.

Definition 5.19 (signed ElGamal). The signed ElGamal encryption scheme over a group $\mathbb{G}$ with generator G is the construction in Figure 5.20.

```
algorithm KeyGen
  sk ↞ Z_|G|
  pk ← G^sk
  return (pk, sk)
```

Spaces:
$M = \mathbb{G}$
$C = \mathbb{G}^3 \times \mathbb{Z}^2_{|\mathbb{G}|}$

```
algorithm Encrypt(pk, m)
  r, u ↞ Z_|G|
  A ← G^r, U ← G^u
  B ← pk^r · M
  c ← H(pk, A, B, U)
  s ← u + cr
  return (A, B, U, c, s)
```

```
algorithm Decrypt(sk, C)
  parse (A, B, U, c, s) ← C
  if c ≠ H(G^sk, A, B, U) or
  G^s ≠ U · A^c then
      return ⊥
  end if
  M ← B/A^sk
  return M
```

Figure 5.20: Signed ElGamal.

In Figure 5.20 we have not included hash compression (Section 4.3.4) with which we could reduce the ciphertext size to $\mathbb{G}^2 \times \mathbb{Z}^2_{|\mathbb{G}|}$. Further we could omit c in the ciphertext as the decryptor

must recompute it anyway to check the ciphertext (line 14). Our analysis of signed ElGamal is unaffected by this choice. We will prove in Section 5.7 that Signed ElGamal cannot be shown CCA secure under reasonable assumptions. For now, we show what security it does achieve.

5.2.3 Non-malleability

Signed ElGamal, when implemented properly with the strong Fiat-Shamir transformation, gives NM-CPA security. We can generalise this result to say that all Encrypt+PoK constructions using IND-CPA secure encryption and an SSE proof are IND-1-CCA secure, which is easier to prove and equivalent to NM-CPA by Lemma 3.10.

Theorem 5.21. An Encrypt+PoK construction based on an IND-CPA secure encryption scheme and a compatible SSE proof is IND-1-CCA secure and hence non-malleable (NM-CPA).

As a corollary, signed ElGamal with the strong Fiat-Shamir transformation is non-malleable.

Proof. We prove the theorem by reducing IND-1-CCA security of the Encrypt+PoK construction to IND-CPA security of the basic encryption scheme. Our reduction obtains a public key from the IND-CPA challenger and passes it to the adversary.

Assume that some adversary A has advantage α against the IND-1-CCA challenger. We begin by modifying the IND-1-CCA challenger to use the simulator, that exists by the zero-knowledge property of the proof scheme, to simulate the proof on the challenge ciphertext.

Generic simulation argument 1. We claim that if the advantage of A against this modified game drops to some $\alpha' < \alpha$ then there is a distinguisher between real and simulated proofs with advantage $\widehat{\alpha}$ such that $\alpha' = \alpha - 2\widehat{\alpha}$. This is an easy calculation with probabilities: Let p_0 be the probability that A outputs 1 in the IND-1-CCA game when $b = 0$ and p_1 the corresponding probability when $b = 1$; by definition, $\alpha = p_1 - p_0$ (assuming w.l.o.g. $p_1 \geq p_0$ as we could flip the output bit otherwise to get a better advantage). Let p_0^S be the probability that A outputs 1 when interacting with the modified IND-1-CCA game with simulated proofs and bit $b = 0$ and p_1^S the same for $b = 1$; so $\alpha' = p_1^S - p_0^S$ (again w.l.o.g. this difference is positive).

Now consider a distinguisher D that is trying to break the zero-knowledge property of the proof scheme. The distinguisher D runs A against a modified IND-1-CCA game picking a bit

b at random and making a `prove` query for the proof on the challenge ciphertext. D forwards the adversary's output bit to its challenger. Let DP be a random variable for the output of our distinguisher when interacting with the prover and BP a random variable for the distinguisher's bit in this experiment; let DS and BS be the same when D is interacting with the simulator S. We compute the advantage of D as $\widehat{\alpha} := \mathbf{Pr}[DP = 1] - \mathbf{Pr}[DS = 1]$ and condition on the bit b to get

$$\begin{aligned}\widehat{\alpha} =& \mathbf{Pr}[DP = 1 \mid BP = 1] \cdot \mathbf{Pr}[BP = 1] + \mathbf{Pr}[DP = 1 \mid BP = 0] \cdot \mathbf{Pr}[BP = 0] \\ &- \mathbf{Pr}[DS = 1 \mid BS = 1] \cdot \mathbf{Pr}[BS = 1] - \mathbf{Pr}[DS = 1 \mid BS = 0] \cdot \mathbf{Pr}[BS = 0]\end{aligned}$$

Since the game chooses the bit b uniformly at random, the non-conditional terms are all $1/2$. If D is interacting with the real prover then the combination of D and the prover is identical to the real IND-1-CCA game towards its simulated copy of A whereas if D is interacting with the simulator then its copy of A sees exactly the modified IND-1-CCA game where the challenge proof is always simulated. So replacing the conditional probabilities with terms defined above this term is in fact

$$\widehat{\alpha} = \frac{1}{2}\left(p_1 + p_0 - p_1^S - p_0^S\right) = \frac{1}{2}\left(\alpha + \alpha'\right)$$

This gives our claimed formula, $\alpha' = \alpha - 2\widehat{\alpha}$ and shows that if the advantage of our adversary drops noticeably when changing real for simulated proofs, we get a distinguisher breaking the zero-knowledge property.

Generic simulation argument 2. We give an argument that we will repeat in later proofs and that shows a generic approach for dealing with simulators when you do not have a witness. We have a basic encryption scheme E and a proof scheme Π combined via the Encrypt+PoK transformation into an encryption scheme E'. When the IND-1-CCA adversary against E' makes a challenge query, the corresponding game creates an E-ciphertext e and uses the simulator to make a proof π then returns the E'-ciphertext consisting of the pair (e, π) to the adversary. We would like to delegate the E-ciphertext to an IND-CPA challenger for E but need the accompanying witness to pass to the `prove` oracle. Let us write out the challenge oracle, inlining the code of the `prove` oracle, in Figure 5.22.

By the definition of compatibility between E and P, algorithm $w(\ldots)$ will always return a valid witness so the check in lines 7–9 is redundant. We can therefore eliminate both checking and creating the witness w and call the simulator directly without changing the returned proofs. We could perform a similar argument even if we were to give the simulator a false statement for which no witness exists, assuming the false statement is indistinguishable from a real one.

```
1  oracle challenge(m_0, m_1)                    oracle challenge(m_0, m_1)
2    b ↞ {0,1}                                     b ↞ {0,1}
3    r ↞ R_Enc                                     r ↞ R_Enc
4    e ← E.Encrypt(pk, m_b; r)                     e ← E.Encrypt(pk, m_b; r)
5    w ← w(pk, m, r)
6    // S.prove((pk, e), w):
7    if not R((pk, e), w) then
8        return ⊥
9    end if
10   π ← S((pk, e))                                π ← S((pk, e))
11   return (e, π)                                 return (e, π)
```

Figure 5.22: Generic simulation argument 2: removing the witness check.

Reduction to IND-CPA. We reduce to IND-CPA by creating a reduction out of our modified game that obtains a public key from its challenger and forwards this to the adversary. To handle challenge queries, the reduction forwards the messages to its challenger and obtains an E-ciphertext, then simulates a proof as described above and returns the ciphertext/proof pair to the adversary. Writing out the code of this reduction and the IND-CPA challenger we see that the resulting algorithm is identical to the modified IND-1-CCA challenger from the last step. The adversary's advantage is therefore unaffected by the modifications so far.

We still need to handle the one decryption query in the IND-1-CCA game. When the adversary makes her call $\mathsf{decrypt}(\vec{c})$ with ciphertexts $c_i = (e_i, \pi_i)$ we would like to rewind the adversary once on each proof, fork on the corresponding random oracle query and extract using special soundness. Since the adversary is efficient, she makes at most a polynomial number q of ciphertexts from which we need to extract so we should need to launch at most q copies of the adversary. (The big difference to a CCA reduction is that we never need to reply to decryption queries from rewinding copies.)

Unfortunately, the bounds that we obtain by the usual forking lemma [BN06] do not guarantee an overwhelming probability of extraction in all proofs. Instead, we need to allow the reduction several attempts at extracting from each proof and use the generalised forking lemma of Bagherzandi et al. [BCJ08]. We again state the lemma restricted to the application we require and refer the reader to the original paper for the proof.

Lemma 5.23 (generalised forking lemma). Suppose that A is an adversary making up to q random oracle queries who outputs a vector of n valid instance/proof pairs, with probability at least ε. Suppose that the extractor makes up to $q \cdot (8n/\varepsilon) \ln(8n/\varepsilon)$ attempts to extract from each proof. Then the extractor obtains witnesses to all proofs with probability at least $\varepsilon/8$ and runs in time bounded by $q \cdot (8n^2/\varepsilon) \ln(8n/\varepsilon)$ times the running time of the adversary plus a constant overhead.

The generalised forking lemma was designed to extract from signature-forgers who output forgeries with some probability ε. In our case, we can assume that the adversary always asks her decryption query with valid proofs: if she submit an invalid proof, the reduction can just return the "decryption" $\perp$ directly and handle this case like a missing decryption query. If the adversary does not make any decryption query at all or queries fewer ciphertexts than expected, the reduction can just add ciphertexts of its own and discard the decryptions it gets back. We can therefore take $\varepsilon = 1$ and get an extractor that succeeds with probability $1/8$ and runs in time $8t^3 \ln(8t)$ where t is a bound on the number of queries the adversary can make, counting a decryption query with m ciphertexts as m queries.

If the extractor fails, our reduction aborts the adversary and outputs a random guess to the IND-CPA challenger. If the extractor succeeds, our reduction answers the main copy of the adversary's decryption query, since compatibility of E and P lets it convert witnesses back to messages. When the adversary makes a guess on the bit b, the reduction returns this guess to its challenger.

If our adversary had advantage α' against the last game-hop (with the modified IND-1-CCA challenger) then the reduction succeeds against the IND-CPA challenger with advantage $\alpha'/8$.

Conclusion. An adversary against IND-1-CCA of the Encrypt+PoK construction with advantage α can be efficiently converted into either a zero-knowledge distinguisher against the proof scheme with some advantage $\widehat{\alpha}$ or an IND-CPA adversary against the basic encryption scheme with advantage $1/8(\alpha - 2\widehat{\alpha})$. If α was non-negligible then so is at least one of these two quantities. *q.e.d.*

5.2.4 TDH2

As a brief excursion we mention another ElGamal+PoK type construction that *is* CCA secure in the random oracle model, the TDH2 scheme of Shoup and Gennaro [SG98] which they proposed as a substitute for signed ElGamal after arguing why no proof of CCA security for signed ElGamal could be found. In fact the version we present in Figure 5.24 is an adaptation by Bulens, Giry and Pereira [BGP11] that removes an extra hash invocation in the original. We assume a group $\mathbb{G}$ with two generators G and $\bar{G}$ such that the discrete logarithm of one to the other is unknown (one could implement this by having the key generator/decryptor pick a random $y \twoheadleftarrow \mathbb{Z}_{|\mathbb{G}|}$ and set $\bar{G} \leftarrow G^y$ as it does not matter if the decryptor knows y).

```
1  algorithm KeyGen
2    sk ↞ Z_|G|, pk ← G^sk
3    return (pk, sk)
4
5  algorithm Encrypt(pk, M)
6    r, s ↞ Z_|G|
7    A ← G^r, B ← M · pk^r
8    Ā ← Ḡ^r, S ← G^s, S̄ ← Ḡ^s
9    c ← H(A, B, Ā, S, S̄)
10   t ← s + cr
11   return (A, B, Ā, S, S̄, c, t)
12 algorithm Decrypt(sk, C)
13   parse (A, B, Ā, S, S̄, c, t) ← C
14   if c ≠ H(A, B, Ā, S, S̄) then
15       return ⊥
16   end if
17   if G^t ≠ S · A^c
18      or Ḡ^t ≠ S̄ · Ā^c then
19       return ⊥
20   end if
21   return B/A^sk
```

Figure 5.24: TDH2 (with [BGP11] optimisation).

We could apply hash compression (Section 4.3.4) to eliminate two group elements from ciphertexts giving a ciphertext size of $\mathbb{G}^3 \times \mathbb{Z}^2_{|\mathbb{G}|}$. For this price we get CCA security under the DDH assumption in the random oracle model and a verifiable augmentation of ElGamal preserving its homomorphic features.

We immediately recognise this scheme as an ElGamal+Chaum-Pedersen PoK construction: (A, B) is the ElGamal ciphertext and $S, \bar{S}$ are commitments used in the proof of knowledge of an r such that $A = G^r$ and $\bar{A} = \bar{G}^r$. The security reduction has the extractor know the discrete logarithm y of $\bar{G}$ to base pk, that is it computes $y \twoheadleftarrow \mathbb{Z}_{|\mathbb{G}|}, \bar{G} \leftarrow pk^y$ which allows for straight-line extraction as $M = B/\bar{G}^{1/y}$.

5.2.5 Chaum-Pedersen signed ElGamal

Seurin and Treger [ST13] give a variation on signed ElGamal that achieves CCA security. They call this scheme Chaum-Pedersen signed ElGamal. It is interesting for several reasons. First, it is the most compact CCA extension of ElGamal known to date that preserves the homomorphic property of a contained plain ElGamal ciphertext (ECIES/DHIES [S01] have an even smaller overhead but use a block cipher to encrypt the message, losing the homomorphic property). Secondly, like Cramer-Shoup, Chaum-Pedersen signed ElGamal requires the secret key to check validity of ciphertexts; it involves a modified sigma protocol but requires the challenge to be kept secret.

The idea behind the protocol is the following. An ElGamal ciphertext for message M with randomness r is a pair $(A, B) = (G^r, pk^r \cdot M)$. Instead of performing a Schnorr proof of knowledge of r, one writes $B = A' \cdot M$ and uses a Chaum-Pedersen proof to show knowledge of an r such that both $A = G^r$ and $A' = pk^r$. The difference to TDH2 is that we are now using the public key as our second base rather than a random group generator. Of course, we cannot give out A' without revealing the plaintext M, so the proof cannot be publicly checkable. Instead, the ciphertext contains just the base-G components of the instance and commitment of the proof. The decryptor recomputes the base-pk half A' of the proven statement using the secret key and can similarly recompute S', the base-pk half of the commitment. The decryptor can therefore check the proof.

An important anomaly here is that the challenge must remain hidden to outsiders. Given c, an adversary could compute a candidate $A'_\mu \leftarrow B/\mu$ for any test message μ and then compute a tentative value of S'_μ by solving the verification equation using c. She can then try and recompute c as $c_\mu = \mathbf{H}(B, A, A'_m u, S, S'_m u)$: she gets $c = c_\mu$ if and only if μ was the encrypted message, breaking IND-CPA security. As Seurin and Treger explain [ST13], they originally proposed the hash-compressed (see Section 4.3.4) version of their scheme but this is actually insecure. On a theoretical level, the CCA proof of this scheme proceeds by showing "plaintext awareness". It is well-known that plaintext awareness *together with* IND-CPA yields CCA [BDPR98]. The requirement to check IND-CPA separately is not vacuous: the trivial encryption scheme where ciphertexts are simply the messages in clear is also plaintext aware. Hash-compressed Chaum-Pedersen signed ElGamal is plaintext aware too, but not IND-CPA and therefore insecure.

We present the scheme in Figure 5.25 and give the security theorem here; we refer the reader to the original paper [ST13] for the proof.

```
algorithm KeyGen
  sk ↞ Z_|G|, pk ← G^sk                    // ElGamal keypair
  return (pk, sk)

algorithm Encrypt(pk, M)
  r, s ↞ Z_|G|
  A ← G^r, A' ← pk^r                       // instance
  B ← M · A'                               // ElGamal encryption
  S ← G^s, S' ← pk^s                       // commitment
  c ← H(B, A, A', S, S')                   // challenge
  t ← s + cr                               // response
  return (A, B, S, t)                      // ElGamal ct and
                                           // ''half'' the proof
algorithm Decrypt(sk, C)
  parse (A, B, S, t) ← C
  S' ← S^sk, A' ← A^sk                     // recompute missing half
  c ← H(B, A, A', S, S')                   // and challenge
  if G^t ≠ S · A^c or pk^t ≠ S'/A'^c then  // verify proof
      return ⊥
  else
      return B/A'                          // ElGamal decryption
  end if
```

Figure 5.25: Chaum-Pedersen signed ElGamal.

Theorem 5.26 (Chaum-Pedersen signed ElGamal security). In the random oracle model and assuming DDH is hard in the underlying group, Chaum-Pedersen signed ElGamal is CCA secure (in fact, also ROM plaintext aware).

Moreover, it is the minimal plaintext aware scheme in the sense that removing any set of the elements from ciphertexts and/or the hash yields a scheme which is provably not plaintext aware, as long as CDH is hard in the underlying group.

The stipulation that CDH be hard in the second half of the theorem is there to avoid the following problem: suppose CDH were easy. Then plain ElGamal would be plaintext aware and so, trivially, would any extension thereof that keeps the original ciphertext intact. No such scheme could be IND-CPA secure however.

The proof of plaintext awareness is in the random oracle model but does not require any rewinding and associated "oracle programming" to extract from the proof: from the transcript of all hash queries, the extractor simply reads off the A' component of a ciphertext and recovers the message as $M \leftarrow B/A'$ directly.

Minimality of plaintext awareness immediately gives an interesting corollary (actually a theorem in the original paper):

Theorem 5.27. Signed ElGamal (Definition 5.19) is not plaintext aware in the random oracle model.

This result was the first proof of a limitation of the security of the signed ElGamal. It excludes the existence of a straight-line extractor for the Fiat-Shamir-Schnorr proof scheme, however as Seurin and Treger point out [ST13, Page 16] it does not say anything about CCA security of signed ElGamal or the possibility of a rewinding extractor for Fiat-Shamir-Schnorr that does not suffer an exponential blow-up. To explore this problem we can use the notion of a multi-proof which captures exactly the idea of "rewinding, but not too much". We will prove that Fiat-Shamir-Schnorr is not a multi-proof and use the techniques developed to argue for a negative answer to the above question.

5.3 Multi-proofs

A multi-proof is a stronger version of a proof of knowledge for which the intuition behind the Encrypt+PoK construction does hold (assuming simulation soundness). This intuition is thus not incorrect, but neither the usual notion of a proof of knowledge (even in our strong version) nor the Fiat-Shamir-Schnorr scheme are powerful enough to satisfy it. Multi-proofs sit between Fiat-Shamir proofs and straight-line proofs in a hierarchy of strength: while straight-line proofs are automatically "multi" if suitably defined, multi-proofs can still admit a rewinding extractor as long as one does not "rewind too much". Achieving chosen-ciphertext security in the presence of a rewinding proof is a very involved matter. The security arguments and reductions for CCA security using straight-line proofs such as those of Fischlin [F05] or Seurin and Treger [ST13] can easily fit on a single page, yet even with tweaked margins, fonts and spacing an early draft of our proof struggled to fit into 9 pages (even though we left out some details).

Our definition of a multi-proof begins analogously to that of a (strong) proof of knowledge, involving an extractor playing against a prover. Our prover can now ask more than one `extract` query (as opposed to returning a single instance/proof pair) and the extractor must return the witnesses to the prover, allowing later extraction queries to depend on the witnesses returned from earlier ones. We also abandon the restriction that strong provers produce a correct proof with overwhelming probability, in favour of the following convention: the security game checks all instances, proofs and witnesses passed between the prover and the extractor. If any one of the prover's proofs is not valid, the extractor wins immediately; if any of the extractor's witnesses is not valid w.r.t. the proof it is supposed to be extracting from, the extractor loses immediately. We do however demand that provers be efficient in the sense that they make at most polynomially many queries and take at most polynomial time between queries, since a prover could otherwise just run forever and prevent the extractor from winning.

Although we define multi-proofs to be non-interactive, a multi-prover is an interactive algorithm since it expects a witness in return for each proof, before making the next one. This forces us to abandon the idea of an initial run of the prover with which the extractor cannot interfere in favour of a main run that the extractor might interact with concurrently with other, "rewinding" runs. In the main run of the prover, if the proof scheme is in the random oracle model then a random oracle external to the extractor handles the prover's oracle queries. The extractor may query the transcript of this random oracle but not interfere with its output — the oracle of the main run is not programmable by the extractor. The extractor has full control over random oracles in "rewinding" invocations of the prover.

Notation for the multi-proof game. Further, we have to change the notation of our game slightly as it is now a "two-sided" oracle, accepting queries from both the extractor and the copies of the prover. We prefix oracles with `P_` if (copies of) the prover can call them and `K_` if the extractor can call them. For the prover, since the game may be dealing with many invocations we further distinguish between oracles for the main and rewinding invocations, thus `oracle P_main.halt` handles the `halt` command for the main prover and `oracle P_rew` handles all calls for the rewinding invocations of the provers (since the game does not need to keep track of proofs here, one oracle can handle all possible queries from these provers). After running `initialise`, the extractor is the first "player" to run. Concurrency in our multi-proof game is "token based", i.e. only one algorithm is running at any time and a call to another algorithm suspends the caller until some other algorithm calls it back. In addition to the `return` statement which returns a value to the caller, we introduce a `send` statement that hands control to a party specified, i.e. in oracle `P_main.ro` the command "`return` h" returns a value h to the main prover whereas in `P_main.extract` the command "`send` x, π `to K`" does what its name suggests, handing control to the extractor. A send command terminates the currently executing oracle just like a return command.

We assume there is some implicit addressing scheme by which the extractor K can address many "rewinding" copies of the prover: either many copies of the `K_rew` oracle, or one copy with an extra implicit parameter for addressing; the details are not important for our purposes. We can further assume that whenever K calls a copy of the prover that has not run yet, the game launches a new copy and discards the initial message from K. We deliberately choose not to settle on any fixed representation of these details in our definition.

Every copy of the prover runs with the same random string.

Definition 5.28 (multi-proof). Let $(\texttt{Prove}, \texttt{Verify})$ be a non-interactive proof scheme with sets X, W, Π.

The proof scheme is a δ–multi-proof of knowledge for a relation $R \subseteq X \times W$ if there is an efficient extractor K such that for any efficient prover P the game in Figure 5.29 halts with `''win''` with probability at least $1 - \delta$. If this holds for a negligible δ, we simply call the proof scheme a multi-proof for R.

We leave the definition of multi-proofs in the CRS or CRS+ROM models for future work. In the multi-proof game, one may be tempted to put the extractor in charge of the CRS. However, in the simulation sound version of the multi-proof game later in this section, there is a simulator and an extractor and both may require trapdoors for the CRS leading to problems that we have not yet solved.

```
1  oracle initialise
2    r ↞ R_P
3    s⃗ ← ⊥
4    H ← [ ]
5    return
6
7  oracle K_list
8    return H
9
10 oracle K_main(w⃗)
11   if s⃗ ≠ ⊥ and ¬R_φ(s⃗, w⃗) then
12       halt with ''lose''
13   end if
14   send w⃗ to P
15
16 oracle K_rew(m)
17   send m to P
18 oracle P_rew(m)
19   send m to K
20
21 oracle P_main.halt
22   halt with ''win''
23
24 oracle P_main.ro(x)
25   h ← RO(x) // use a RO
26   H ← H :: (x, h)
27   return h
28
29 oracle P_main.extract(x⃗, π⃗)
30   if not Verify(x⃗, π⃗) then
31       halt with ''win''
32   end if
33   s⃗ ← x⃗
34   send x⃗, π⃗ to K
```

Figure 5.29: Multi-PoK game.

Overview of the game. Initially, procedure `initialise` sets up some variables: r is the random string that all copies of the prover P will use and $\vec{s}$ will hold the last vector of instances presented by the main invocation of P to K for extraction, so that the game can check that K returns valid witnesses. The game then runs the extractor K. During the game, whichever party is active can communicate with the game through its oracles. The main invocation of the prover P has an oracle `P_main` that handles three types of queries: `halt`, `ro` and `extract`. All other invocations of the prover have an oracle `P_rew` that forwards all queries to K.

K has one oracle to communicate with each copy of the prover: `K_main` for the main prover and a `K_rew` for each other prover. Additionally, K has a `K_list` oracle to look at the list of random oracle queries made by the main prover. When the extractor K calls `K_main` to communicate with the main prover, except for the first message to start up the prover it will be returning candidate witnesses to the prover's last extraction query. The game checks that the witnesses are correct; K immediately loses (the entire game halts with value "lose") if K provided an incorrect witness. Otherwise, the game passes the witnesses to the prover.

The main prover's oracles act as follows. If the main invocation of P halts, K wins: the game halts with value "win". As usual, a random oracle handles random oracle queries made by the main invocation of the prover; the game records all queries in a list H so that K can track these queries with its list oracle. When the main prover makes an extraction query, the game first checks the proofs: if these fail to verify, K wins immediately. The game then records the instances in the global variable $\vec{s}$ for later verification of K's witnesses and passes the extraction query on to K. All queries by the other instances of the prover are passed to K directly; the game does not concern itself with checking their witnesses or proofs. If K aborts the execution, the outcome of the game is "lose"; the only way K can win is to get the main prover to issue a `halt` query (assuming w.l.o.g. that the prover never ask an extraction query on an invalid proof — after all, she could check all proofs herself).

Simulation-sound multi-proofs. To construct a CCA secure Encrypt+PoK scheme, we must not only extract from proofs to answer decryption queries but also simulate proofs to handle the challenge query, both in the same security game. This requires the "multi" analogue of simulation soundness which we define by giving the prover the additional option to make simulation queries denoted `prove`. For the main invocation of the prover, we use the simulator to simulate proofs. Since the simulator must be in charge of the random oracle, if present, we forward the main invocation's random oracle queries to the simulator but track them in a list H as before. We denote calls to the simulator for requesting simulated proofs and random oracle queries by `S.sim` and `S.ro` respectively. The game stores all simulated proofs in a list Π, initially empty, to prevent them from appearing in an extraction query.

For proofs made by rewinding invocations of the prover, we check the witness then send only the instance to the extractor. This prevents the extractor from using a rewinding invocation to obtain the witnesses used in simulation queries of the main invocation. The extractor may choose to run an instance of the simulator itself for simulation queries by rewinding invocations of the prover and remains in control of random oracle queries for these invocations.

Definition 5.30 (ss-mPoK). A proof scheme is a simulation sound multi-proof of knowledge (ss-mPoK) for a relation R if it is zero-knowledge w.r.t. a simulator S and there is an efficient extractor K such that for any efficient prover P, the extension to the multi-proof game in Figure 5.31 halts with ‘‘win’’ with overwhelming probability (or a δ–ss-mPoK if the extractor wins with probability at least $1 - \delta$).

```
35 oracle P_main.prove(x, w)
36   if not R(x, w) then
37       return ⊥
38   end if
39   // use simulator
40   π ← S.sim(x)
41   Π ← Π :: (x, π)
42   return π
43
44 oracle P_rew.prove(x, w)
45   if not R(x, w) then
46       return ⊥
47   end if
48   send x to K
```

```
49 oracle P_main.extract(x⃗, π⃗)
50   if ∃i : (x_i, π_i) ∈ Π then
51       halt with ‘‘win’’
52   end if
53   if not Verify(x⃗, π⃗) then
54       halt with ‘‘win’’
55   end if
56   s⃗ ← x⃗
57   send x⃗, π⃗ to K
58
59 oracle P_main.ro(x)
60   h ← S.RO(x)
61   H ← H :: (x, h)
62   return h
```

Figure 5.31: Simulation-sound extension to the multi-PoK game. (The prove queries are new and the extract and ro ones replace their counterparts in the multi-proof game.)

5.4 Examples of multi-proofs

We give some examples of multi-proofs. It is an open question, that we plan to address in future work, whether there is a “sensible” multi-proof in which rewinding is inherent — after all, part of the appeal of multi-proofs for us is that they do not exclude rewinding yet still yield CCA secure Encrypt+PoK constructions (when they are also simulation sound). Our two examples here are therefore straight-line proofs.

The Naor-Yung transformation. Naor and Yung [NY90] proposed the following CCA secure scheme: take an IND-CPA secure encryption scheme and generate two keypairs. To produce a ciphertext, encrypt each message twice — once under each key — and then add a non-interactive, simulation sound proof that both ciphertexts contain the same message. To decrypt, check the proof then decrypt either ciphertext.

The security reduction from CCA security of the Naor-Yung transformation to IND-CPA of the basic encryption scheme obtains one public key from the challenger and picks the second keypair itself. To answer the challenge query, the reduction obtains a challenge ciphertext from its challenger, makes a second ciphertext on a random message and then simulates a proof that the two ciphertexts contain the same message. (There are some subtleties involved here which we omit in this thesis. One can show that if the adversary notices this trick, we can break either the basic encryption scheme directly or tell real from simulated proofs.) To answer decryption queries, the reduction uses its own secret key on the second ciphertext. As long as the proof is simulation sound, this works fine. The proof can be a Fiat-Shamir style proof: since it only has to be simulation sound and not a proof of knowledge, the reduction never needs to extract from a proof, avoiding the problems in the following sections.

If we view the combination of the second ciphertext and the simulation sound proof as a proof scheme, the Naor-Yung transformation becomes an Encrypt+PoK construction on the first ciphertext. This proof scheme is in the CRS model with the second public key as a CRS and the second secret key as a trapdoor. If the simulation sound proof is in the random oracle model, our proof scheme lives in the ROM+CRS model. ROM or not, this proof scheme is a simulation sound multi-proof: the extractor uses the CRS trapdoor (secret key) just like the CCA–to–IND-CPA reduction and never launches any "rewinding" copies of the prover. The simulator too acts like the reduction above, creating random ciphertexts then calling the simulator of the original simulation sound proof scheme. We can thus propose an alternative view of the Naor-Yung transformation: instead of taking IND-CPA secure encryption schemes to CCA secure ones with the help of a simulation sound proof, it takes simulation sound proofs to ss-multi-proofs with the help of an IND-CPA secure encryption scheme.

The Fischlin transformation. Fischlin [F05] gives a variation on the Fiat-Shamir transformation in his 2005 CRYPTO paper. A Fischlin proof is a sequence of strong Fiat-Shamir proofs such that each proof verifies individually and an additional "overall" condition holds: for each proof, compute the hash (random oracle value) of (a) the sequence of all sigma protocol commitments, and (b) the challenge and response to this individual proof. Then sum these hash values. A

Fischlin proof is valid if a certain number of low-order bits in this sum are all zero. The exact conditions and the security analysis of this scheme are outside the scope of this thesis.

Fischlin has shown that the only way one can create such a proof is to create the entire sequence of commitments, then try various challenges and their responses for each proof in the sequence until one finds a challenge/response pair that makes the low-order bits of the required hash small enough. Since the hashes to be summed include the commitments to all proofs in the sequence, the prover cannot start over with a different commitment but is forced to try different challenges to the same proof.

Fischlin's proofs are simulation sound multi-proofs in the ROM. They inherit zero-knowledge and simulation soundness from the Fiat-Shamir transformation; the multi-extractor need only observe the random oracle queries made by a prover to compute the "overall" hashes and is guaranteed to find in the hash inputs, a pair of proofs from which she can extract using special soundness (with overwhelming probability, for a choice of constants given by Fischlin).

Fischlin has proved that his transformation yields simulation sound multi-proofs. Since the work in question is not yet published, we do not address this point any further in this thesis.

TDH2, Cramer-Shoup and Chaum-Pedersen signed ElGamal? It is tempting to view other CCA extensions of ElGamal as Encrypt+multi-PoK transformations just like we did for the Naor-Yung transformation above. Unfortunately, this often leads to a "proof scheme" for a statement that is formally outside the class **NP** since the statement involves the secret key. To construct CCA secure encryption this is not a problem since only the decryptor needs to check the proofs. It is however a problem to view the proof as a standalone scheme, since the multi-proof game or a proposed extractor can no longer efficiently check proofs by themselves. We plan to address this issue in future work and investigate whether all known CCA transformations of ElGamal that preserve the homomorphic property on the basic ciphertext can be viewed as Encrypt+multi-PoK constructions. (Typically, these multi-proofs will also be straight-line.)

5.5 Fiat-Shamir-Schnorr is not a multi-proof

We come to the first of our three promised theorems, that the Fiat-Shamir-Schnorr non-interactive proof scheme (Def. 5.12) is not a multi-proof (Def. 5.28) if the one-more discrete logarithm assumption (Def. 2.19) holds in the group in question. The proof of this theorem constructs a reduction which can solve the one-more discrete logarithm problem, given any extractor that wins the multi-proof game.

Theorem 5.32. Suppose that pseudorandom functions exist. Then Fiat-Shamir-Schnorr is not a multi-proof under the one-more discrete logarithm assumption (over any family of groups) in the random oracle model.

Namely, for any extractor that wins the multi-proof game against a particular efficient prover with running time $\Theta(n)$ there is a reduction such that either the reduction solves the one-more discrete logarithm assumption in the underlying group or the extractor must invoke $\Omega(2^n)$ copies of the prover.

We could remove the stipulation that pseudorandom functions exist at the cost of using an inefficient prover. Since the extractor and reduction only make black-box use of the prover, we do not see this as a problem (in the proof we will change the PRF into a random function anyway).
Proof. The following is an overview of the proof.

1. We begin with a prover following an idea of Shoup and Gennaro [SG98], making a chain of proofs in which each depends on the last and then asking extraction queries in reverse order. The effect of this technique is that a straightforward special soundness approach to extraction takes exponential time.

2. This prover uses a (pseudo)random function to make its chain of proofs. We externalise this function and memoize it in one place for all copies of the prover. We also encapsulate the witnesses to proofs together with the (pseudo)random function.

3. Next, we switch the prover to one who obtains one-more discrete logarithm challenges and uses these to make proofs. Since the extractor may use special soundness to recover the witness to any proof it wants, we must use our challenger's discrete logarithm oracle whenever the extractor "opens" a proof.

4. We define an event E that the extractor answers an extraction query for which we do not know the witness.

   - If event E ever occurs, we solve the one-more discrete logarithm problem.
   - Then, we argue that as long as event E does not occur, any extractor successful against our prover must require exponential time.

Step 1: the prover. Assume that F is a pseudorandom function $Keys \times \{0,1\}^* \to \mathbb{Z}_{|\mathbb{G}|}$, i.e. it takes a key (written as a subscript) and an encoding of any number of elements as input and returns a group exponent. We begin with the prover on the left side of Figure 5.33. Next to our prover we give the code of a "fictional prover" that displays the intuition behind how we will use our prover to reduce to the one-more discrete logarithm problem, although we cannot use this prover directly as we have to take care of the reduction's rewinding capabilities. Note that after drawing the initial PRF key, our prover is deterministic (the variant where we use a true random function is completely deterministic). We define a few terms that we will use later in the proof:

- The history of a copy of the prover in the multi-PoK game is the sequence of values $c_0, c_1, \ldots$ that she has computed so far.
- A copy C is a prefix of another copy D if the history of C is a prefix of the history of D. Two equal histories are both prefixes of each other (so every invocation is always a prefix of itself).
- C is a strict prefix of D if C is a prefix of D but not vice versa, i.e. D is "ahead" of C.
- Two copies C, D have forked if there is some i such that both C, D have computed a value of c_i but obtained different values for c_i.

Histories and prefixes thus define a partial order on all copies of the prover at any point in time.

Step 2: rewriting the prover. In the multi-proof game, the extractor interacts with many copies of the prover which all use the same pseudorandom function on the same key (since all copies run with the same random string). We first rewrite the prover so as to externalise the pseudorandom function and place it in an oracle named `instance`. Secondly, we memoize the pseudorandom function using one memoizer for all copies of the prover. This gives us a three-layered system with the copies of the prover at layer 1, the memoizer at layer 2 and the pseudorandom function at layer 3 which we will later switch to a random function and then to the one-more discrete logarithm challenger. Since all copies of the prover call the PRF with the same key k, we can move the key down to layer 3. (As the prover wants her instances in pairs, we call the memoizer's oracle `pair` and have it sample two instances at a time.)

Next, we observe that the random elements x, a from the PRF are immediately converted into an instance/commitment pair $X = G^x, A = G^a$. We pull this operation down to layer 3 too and have the `instance` oracle here return the group elements X, A directly. However, to create the response s the prover needs the exponents x, a so we let the instance oracle store these exponents

real prover

```
1  algorithm Prover
2    k ↞ Keys
3    c_0 ← 0
4    for i = 1 to n do
5        x_i ← F_k(1, c_0, ..., c_{i-1})
6        a_i ← F_k(2, c_0, ..., c_{i-1})
7        X_i ← G^{x_i}
8        A_i ← G^{a_i}
9        c_i ← ro(X_i, A_i)
10   end for
11   for i = n to 1 step −1 do
12       s_i ← a_i + c_i · x_i
13       w ← extract(X_i, A_i, c_i, s_i)
14       if G^w ≠ X_i then
15           halt
16       end if
17   end for
18   halt()
```

fictional prover for OMDL

```
1  algorithm Prover
2
3    c_0 ← 0
4    for i = 1 to n do
5
6
7        X_i ← instance()
8        A_i ← instance()
9        c_i ← ro(X_i, A_i)
10   end for
11   for i = n to 1 step −1 do
12       s_i ← dlog(A_i · X_i^{c_i})
13       w ← extract(X_i, A_i, c_i, s_i)
14       if G^w ≠ X_i then
15           halt
16       end if
17   end for
18   halt()
```

Figure 5.33: Real and fictional provers for OMDL.

and introduce a new oracle `response` to create the necessary responses s, keeping the exponents encapsulated at layer 3. The prover's corresponding oracle is called `prove`. We sketch the layout of the three-layered system we have constructed in Figure 5.34.

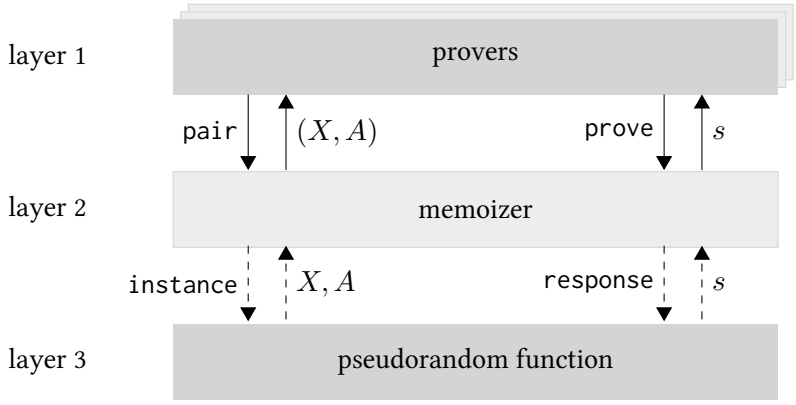


Figure 5.34: Decomposition of the provers into a three-layered system.

We also adapt the memoizer at layer 2 to memoize proof responses: if another copy of the prover has already made the exact same proof, we just return the stored response. If the provers have already made two proofs on the same instance/commitment pair (X, A) with different challenges then instead of calling down to layer 3 the memoizer uses special soundness to compute further proof responses by itself.

Layer 1 — prover

```
1  algorithm Prover
2    c_0 ← 0
3    for i = 1 to n do
4        (X_i, A_i) ← pair(c⃗)
5        c_i ← ro(X_i, A_i)
6    end for
7    for i = n to 1 step −1 do
8        s_i ← prove(X_i, A_i, c_i)
9        w ← extract(X_i, A_i, c_i, s_i)
10       if G^w ≠ X_i then
11           halt
12       end if
13   end for
14   halt()
```

Figure 5.35: Code of the prover at layer 1.

On layer 1 (Figure 5.35) we have the copies of our prover that now delegate element and proof creation to the memoizer. By $\vec{c}$ in the prover code we mean the history $c_0, c_1, \ldots$ of all c-values computed so far.

Layer 2 — memoizer

```
15 oracle pair(c⃗)
16   if V[c⃗] = ⊥ then
17       X ← instance(1, c⃗)
18       A ← instance(2, c⃗)
19       V[c⃗] ← (X, A)
20       L[(X, A)] ← (2, ⊥, ⊥)
21   end if
22   return V[c⃗]
23
24 oracle prove(X, A, c)
25   if L[(X, A)] = ⊥ then
26       return ⊥
27   end if
28   (ϕ, u, v) ← L[(X, A)]
29   if ϕ = 2 then
30       s ← response(X, A, c)
31       L[(X, A)] ← (1, c, s)
32       return s
33   else if ϕ = 1 then
34       if u = c then
35           // repeated query
36           return v
37       end if
38       s ← response(X, A, c)
39       // special soundness:
40       // we can open this one
41       x ← (c − u)^{−1} · (s − v)
42       a ← v − u · x
43       L[(X, A)] ← (0, x, a)
44       return s
45   else
46       // Already opened.
47       s ← v + c · u
48       return s
49   end if
```

Figure 5.36: The memoizer on layer 2.

On layer 2 (Figure 5.36) the memoizer maintains two global lists. V maps histories to challenges: the next pair of challenges that a copy with history $\vec{c}$ will obtain is $V[\vec{c}]$. L stores all discrete logarithms that the memoizer has learnt so far and for each challenge, an associated potential ϕ. An entry (ϕ, u, v) for a pair (X, A) encodes the following information:

- if $\phi = 2$ then the provers have not yet made any proofs on this pair and u, v are set to $\perp$.
- If $\phi = 1$ then the provers have made exactly one proof with instance X and commitment A, using challenge u and response v.

- If $\phi = 0$ then the provers have made at least two proofs on this instance/commitment pair with different challenges; the extractor therefore knows the witness by special soundness. In this case u holds the discrete logarithm of X and v that of A, allowing the memoizer to make further proofs "for free".

Layer 3 — (P)RF

```
50 oracle instance(n, c⃗)
51   if k = ⊥ then
52       k ↞ Keys
53   end if
54   s ← F_k(n, c⃗)
55   T ← G^s
56   D[T] ← s // store for later
57   return T
58 oracle response(X, A, c)
59   // recover our dlogs
60   x ← D[X], a ← D[A]
61   if x = ⊥ or a = ⊥ then
62       halt // cannot happen
63   end if
64   return a + c · x
```

Figure 5.37: The PRF on layer 3.

On layer 3 (Figure 5.37), when using the PRF we store the discrete logarithms in a list D to be able to answer layer 2's queries. Since layer 1 (the prover) only asks `prove` queries on challenges it got from the PRF (through the memoizer) and the sampler in turn only asks `response` queries on elements thus obtained from the prover, we conclude that the `response` query is only ever asked on group elements that the `instance` oracle made in the first place, for which D will contain the discrete logarithms. The `response` oracle therefore will never have to halt on Line 62.

All these changes only affect how our system operates internally but not its behaviour towards the game or extractor. In particular these changes do not affect the extractor's success probability.

Step 3: reduction to OMDL. We have set everything up so we can switch the single algorithm at layer 3. Assume that an efficient extractor K exists that can win the multi-proof game against our prover. First, we replace the PRF by a true random function. By the assumption that the function was pseudorandom, the combination of the extractor, game and layers 1 and 2 — everything except the (P)RF — cannot notice any difference since this whole construction is efficient.

At this point we note that we could have started with a random function instead of a PRF in the first place and carried out the exact same operations in step 2. Our original prover would have been inefficient but accessible only in a "black box" manner to the game and extractor. The

random function that we introduced above can be simulated efficiently by lazily sampling new values when needed; the reason that lazy sampling would not have worked with the original prover is that the extractor has access to many copies of the same prover with the same random string, which all need access to the same random function.

Next we switch our memoized random function to an OMDL challenger. The OMDL challenger is stateful so we need the memoizer in place for this step since different copies of the prover should return the same proofs if given the same inputs and random oracle values. `pair` queries made by copies of the prover while not a prefix of any other copy however yield independent and uniformly random values which is exactly what the OMDL challenger produces. Call these queries "fresh".

$\texttt{instance}(n, \vec{c})$ queries become `instance` queries to the OMDL challenger (with no parameters) since if the memoizer makes an instance query to the OMDL challenger as opposed to returning a memoized value then the query must be "fresh". $\texttt{proof}(X, A, c)$ queries become $\texttt{dlog}(A \cdot X^c)$ queries. We have set up the memoizer so that it makes discrete logarithm queries only when absolutely necessary. We claim that this game-hop from a random function to an OMDL challenger is *perfectly* indistinguishable. Indeed, by studying the code of the OMDL game in Definition 2.19 we see that it creates elements identically to our previous `instance` oracle with a random function, sampling an element x uniformly from $\mathbb{Z}_{|\mathbb{G}|}$ and returning G^x. The `dlog` oracle too, the way we use it in our reduction, does the same job as our former `response` oracle since there is only one possible witness to complete a Schnorr proof. Note that we only use discrete logarithm queries on "linear combinations" of the OMDL challenger's instances, so we could also reduce to the efficiently simulable version of the OMDL problem in Section 2.7.

Step 4: exponential lower bound. At any time in the game, each pair of instances (X, A) has a potential $\phi \geq 0$. We can define a global potential Φ as the sum of the potentials of all instances created so far as stored in the list L. Whenever we draw a pair of instances, we add a new entry to L with potential 2. Every call that results in a discrete logarithm query decreases the potential of the associated instances by 1. No other instance or discrete logarithm queries occur than the ones mentioned above so the global potential Φ represents the number of instance queries made minus the number of discrete logarithm queries made to the OMDL challenger. Since we never make more than two `dlog` queries on any instance, neither the global nor a local potential can ever become negative.

We want to assume w.l.o.g. that no two elements returned from separate instance queries by the OMDL challenger ever collide. Since they are uniformly random independent group elements,

the probability of a collision after polynomially many instances have been drawn is negligible.

We also briefly mention why we can ignore the case that the extractor returns incorrect witnesses. The extractor cannot return an incorrect witness to the main copy without immediately losing the multi-proof game. The extractor is free to return what she wants to rewinding copies but we have constructed our adversary to verify all witnesses: a rewinding copy getting an incorrect witness will immediately halt and be of no further use to the extractor.

Lemma 5.38. Let C be the event that two instances returned from the OMDL challenger collide. Let E be the event that the extractor returns a correct witness to some proof (X, A, c, s) by any copy of the prover while the potential of (X, A) is 1.

If events C or E occur, the reduction wins against the OMDL game. The collection of "everything except the OMDL challenger" is therefore an efficient algorithm that wins the OMDL game with at least the same probability as events C or E occurring in this experiment.

(Note that the extractor will never return a witness to a proof at potential 2 since the potential has to drop to 1 in order to make the proof in the first place.)

Proof. If event E occurs and C has not occurred then the reduction has obtained the discrete logarithm x of the instance X of the pair in question and can compute the other discrete logarithm for free as in Line 42. For all other pairs (X', A') the reduction can now obtain the discrete logarithms as follows: for a pair at potential 2 it asks $\texttt{dlog}(X')$ and $\texttt{dlog}(A')$ and drops the associated potential to 0. For a pair at potential 1 it asks only $\texttt{dlog}(X')$, dropping the local potential to 0, and computes the other one for free as explained above using the list L. For a pair at potential 0 the discrete logarithms are already stored in L. The reduction now has all discrete logarithms but the global potential is still at $\Phi = 1$ so the reduction wins the one-more discrete logarithm game that it is interacting with.

If C occurs then the reduction can open all pairs except the one that caused the collision, then ask the discrete logarithm of the other, non-colliding element in the pair that triggered event C. The reduction now has the discrete logarithm for the colliding element for free while keeping the global potential at 1 so it can again win the OMDL game. *q.e.d.*

Lemma 5.39. If the extractor wins the multi-proof game and events E and C have not occurred then the extractor must have launched at least 2^n copies of the prover.

Proof. Suppose that the extractor wins without events E or C having occurred. We construct a complete binary tree with nodes of the form (i, k) where i is an identifier for some copy of the prover (one could take 0 for the main copy and $i > 0$ for the ith "rewinding" copy that the extractor started) and k is an integer. Our tree will have the following invariants.

1. If a node (i, k) is present in the tree then the copy of the prover referenced by i has completed her history $c_0, c_1, \ldots c_n$ of challenges and progressed to the point where she has received the correct witness in response to her k-th extraction query.

2. The two children of any node (i, k) in the tree reference distinct copies of the prover that differ in the component c_{n-k+1} of their respective histories.

3. If $D = (j, l)$ is a descendant of $P = (i, k)$ then $l < k$ and the copy j of the prover shares all challenges up to and including c_{n-k} with copy i.

As the root of our tree, we choose $(0, n)$ where 0 refers to the main copy of our prover. By assumption, the extractor has won so main copy has halted and therefore received all n correct witnesses. The extractor cannot have halted the main copy of the prover by providing an incorrect witness since the game already checks these witnesses and the extractor would have lost in this case. In conclusion, the root meets our invariants.

To each node (i, k) with $k \geq 1$ we recursively add two children. The first child is $(i, k - 1)$. Nodes inherit invariants 1 and 3 from their parent; invariant 2 we will deal with when we add the second child.

For the second child of (i, k), since copy i has received the correct response to its k-th extraction query by invariant 1 and event E has not occurred, the potential of the associated instance/-commitment pair (X, A) must be at 0 so there must be some invocation i' that made a proof (a) on the same pair (X, A) and (b) with a different challenge, allowing the memoizer to obtain the witness in question by special soundness. We add $(i', k - 1)$ as the second child of (i, k); if several i' satisfy the above conditions we pick any one.

By (a) and since there were no collisions from the OMDL challenger, i' must share the same prefix $c_0, \ldots c_{n-k}$ with i in order to use the same instance/commitment pair. This proves invariant 3 since as we descend the tree, the counter k decreases forcing shared prefixes between a node and its parent to become longer. It also proves invariant 1 since to make a proof on its $(n-k+1)$-st challenge which due to the reverse order of proofs is its k-th proof, i' must have received the correct response to its $(k-1)$-st extraction query and all previous ones. By (b), i' must have made its k-th proof on a different challenge to i, i.e. i and i' differ in c_{n-k+1} proving invariant 2.

Our complete tree has 2^n leaves of the form $(i, 0)$ all of which represent invocations that have progressed to the point they made their first proof, in particular they have computed their entire history of challenges $c_0, \ldots, c_n$. We claim that the identifiers on these leaves are pairwise disjoint, proving that the extractor must have launched 2^n copies of the prover. Suppose for the sake of contradiction that two leaves L, M share the same identifier. Then there is a unique path in the tree connecting these two nodes and a unique highest node N on this path.

W.l.o.g. L is in the subtree rooted at the first child C of $N = (i, k)$ and M in the subtree rooted at the second child D of N. Therefore, by invariant 3, L as a descendant of N's first child $C = (i, k-1)$ must share all elements of its history up to c_{n-k+1} with N but M as a descendant of N's second child D shares history up to c_{n-k+1} with D. Since by invariant 2, C and D differ in c_{n-k+1}, so do L and M which contradicts the assumption that they represent the same copy. Therefore no two leaves can refer to the same copy of the prover and there must have been 2^n distinct copies of the prover that progressed until their first proof. *q.e.d.*

Conclusion. If we run any extractor against our reduction, one of three outcomes must occur: either the reduction solves the OMDL problem against which it is working, the extractor launches at least exponentially many copies of the prover or the extractor loses the multi-proof game. We conclude that in any group, Fiat-Shamir-Schnorr cannot have an efficient extractor in the sense of Definition 5.28 if the one-more discrete logarithm problem is hard, completing the proof of Theorem 5.32. *q.e.d.*

5.6 Multi-proofs yield chosen-ciphertext security

In this section we will prove that simulation sound multi-proofs in the sense of Definition 5.30 yield CCA secure encryption schemes in the Encrypt+PoK transformation. Our proof is to our knowledge the first proof of CCA security using a rewinding extractor. All previous such proofs,

whether for a particular scheme such as Cramer-Shoup or TDH2 or for the generic Encrypt+PoK construction in the generic group model by Schnorr and Jakobsson [SJ00] relied on a straight-line extractor. This makes such proofs comparatively easy: we will see that the main source of complexity in our proof is that we must handle multiple copies of the adversary in a reduction to a single IND-CPA challenger. Conversely, Shoup and Gennaro [SG98] argued and we will show that the rewinding extractor for the Fiat-Shamir-Schnorr construction is insufficient to obtain CCA security. These results place multi-proofs between the special soundness based rewinding proofs and straight-line proofs. They show that rewinding is not necessarily an obstacle to obtaining CCA security.

Theorem 5.40. Consider an IND-CPA secure encryption scheme E and a compatible simulation sound multi-proof P in the sense of Definition 5.30. Then the Encrypt+PoK construction on E and P according to Definition 5.17 is CCA secure.

Proof. The proof does the same thing as all other CCA proofs for Encrypt+PoK based schemes that we are aware of, namely use the extractor to answer decryption queries in a reduction to IND-CPA security of the basic encryption scheme E. Most of the proof concerns itself with achieving a consistent simulation of multiple copies of the adversary towards the extractor.

Step 1: simulating proofs. We begin by considering a CCA adversary interacting with the corresponding game. Looking at the challenge oracle written out for an Encrypt+PoK construction (Figure 5.41, left-hand side), we make two changes: first, we replace the proof on the challenge ciphertext by a simulated proof using the `prove` oracle provided by the simulator from the ss-mPoK property of P (Definition 5.30). In the random oracle model, this involves delegating the random oracle to the simulator. Since we are assuming an ss-mPoK and hence zero-knowledge w.r.t. S, the adversary's advantage against this modified game changes at most negligibly.

(To be precise, if α is the original CCA adversary's advantage and α' the CCA advantage after switching to simulated proofs then we can construct a reduction against the zero-knowledge game with advantage α_S such that $\alpha' \geq \alpha - 2\alpha_S$. This is the same argument as in the proof of Theorem 5.21.)

Next, we observe that the `prove` oracle takes as input an instance/witness pair, checks the witness w.r.t. relation R and then discards it. Since the challenge oracle creates the witness using algorithm w from Definition 5.16 (Line 4 in the figure) this witness is correct by definition. We

therefore eliminate the witness creation and checking code and call the `sim` oracle of S directly. This step is internal to the game and does not change any outputs to the adversary so it preserves the adversary's advantage. The original and modified challenge oracle are given in Figure 5.41.

```
1  oracle challenge(m_0, m_1)
2    r ↞ R_Enc
3    e ← E.Encrypt(pk, m_b; r)
4    w ← w(pk, m_b, r)
5    π ← P.prove((pk, e), w)
6    c* ← (e, π)
7    return c*
```

```
1  oracle challenge(m_0, m_1)
2    r ↞ R_Enc
3    e ← E.Encrypt(pk, m_b; r)
4    ~~w ← w(pk, m_b, r)~~
5    ~~if not R(x, w) then~~
6      ~~return ⊥~~
7    ~~end if~~
8    π ← S.sim((pk, e))
9    c* ← (e, π)
10   return c*
```

Figure 5.41: Modifying the challenge oracle to remove the witness check.

Step 2: using the extractor. We modify the decryption oracle from the CCA game to output an extraction query, wait for a witness and then compute the decrypted message from the witness (by Definition 5.16). Of course the decryption oracle must first check that the ciphertext is well-formed, i.e. that the proof verifies. In this way, the pair consisting of the adversary and modified game becomes an ss-mPoK prover. We call the modified CCA game the reduction from now on, since it connects the adversary and the extractor and translates queries between the two.

As part of this modification, the new prover must ask her random oracle and simulation queries on an external interface (which connects to the ss-mPoK game) too. Technically, such a prover does not have access to a simulator directly but only to a `prove` oracle. However, we can discard the witness-check for the same reason that we could do so in step 1 of this proof above. In particular, even in "rewinding" copies of the prover, the extractor never gets to see the witnesses. The list Π of simulated proofs is also irrelevant as the only entry it could contain is the pair $((pk, e), \pi)$ representing the challenge ciphertext but the CCA adversary is already banned from asking for a decryption of the challenge ciphertext. If the adversary manages to make a new "small" (IND-CPA) ciphertext $e' \neq e$ with respect to which the challenge proof π verifies then according to the ss-mPoK definition, the extractor must still work.

In the last line of the decryption oracle, we return the message extracted from the witness as defined in the definition of compatibility between E and P. Compatibility guarantees that we thus obtain "the" message encrypted in a ciphertext so our new decryption oracle returns the same value as the original one, for any inputs. Therefore the advantage of the main copy of the adversary is not affected by these changes as long as the extractor answers all its queries; the ss-mPoK definition guarantees an extractor that wins its game with overwhelming probability against our prover which implies that it answers all extraction queries of the main copy of the adversary.

```
11 oracle decrypt(c)
12   if c = c* then // false if c* not yet defined
13       return ⊥
14   end if
15   parse (e, π) ← c
16   if not Verify((pk, e), π) then
17       // decryption of bad proof is always ⊥.
18       return ⊥
19   end if
20   w ← Extract((pk, e), π)
21   return m((pk, c), w) // return the message
```

Figure 5.42: The modified decryption oracle.

(If we allowed the extractor to fail with probability δ then for an adversary with advantage α' against the game from step 1, using the extractor would give an adversary with advantage $(1 - \delta)\alpha'$ against the bit b in the current game).

Step 3: reduction to IND-CPA. We would like to replace the challenge ciphertext component e from the original encryption scheme E in the main run of the prover with an IND-CPA challenge. However, the extractor is only guaranteed to work when given access to multiple copies of the same prover but we have only one challenge query available. Allowing the IND-CPA challenger to answer multiple challenge queries would incur a loss in the adversary's advantage proportional to the number of queries, which we can avoid.

We obtain a public key from the challenger and give it to all copies of the reduction rather than have the reduction create its own keypair. Because the IND-CPA challenger generates public keys

as uniformly random group elements just like the the reduction used to do too, this change does not affect the distribution of public keys. Nor does the fact that we now only generate one public key for all copies of the reduction change anything: all copies used to generate their key from the same random string and so obtained the same key.

When the main copy of the adversary makes its challenge query, we have our reduction forward the query to the IND-CPA challenger and simulate the proof on the returned E-ciphertext e as before. The caveat here is that the extractor may want to have another copy of the prover run identically to the main one, giving her the same answers to all queries. We therefore make the following changes to how the bit b is created. First, we defer selecting the bit b until it is needed in the challenge oracle. Next, immediately before picking b, we have the reduction ask a random oracle query $\texttt{ro}(m_0, m_1)$ on the two challenge messages. The main copy of the reduction asks this query immediately before calling the IND-CPA challenger.

The adversary is unaffected by these changes as they do not change any elements that the reduction returns to the adversary (we ignore the result of the extra random oracle query). The extractor by definition works for all efficient provers so it will still work for the new one that asks this extra query. Further, the extractor has no control over any values returned to the main copy of the prover: it can only read but not write to the main prover's random oracle and we have already established that the extractor has no freedom of choice in the witnesses that it returns to the main prover.

We give a diagram of the system constructed so far in Figure 5.43. This diagram can be viewed from three different perspectives. First, the bold lines represent the ss-mPoK game: there is an extractor K interacting through the game $G_{\texttt{ss-mPoK}}$ with multiple copies of a prover, one of which is the "main" copy (represented by a bold border). These provers each consist of a copy of the CCA adversary A and a reduction R that performs the query translations which we have described so far, e.g. turning `decrypt` queries from A into `extract` queries sent to the ss-mPoK game and K.

Secondly, the copy of the adversary A within the main prover (with the bold border) is the one that we are claiming cannot tell whether it is contained in this system, or interacting with a CCA challenger directly. From this perspective, the whole construction except for this one adversary is a CCA challenger.

Thirdly, above all these provers is a single instance of the IND-CPA challenger $G_{\texttt{IND-CPA}}$, for the basic encryption scheme E. The whole construction minus this challenger is an IND-CPA adversary.

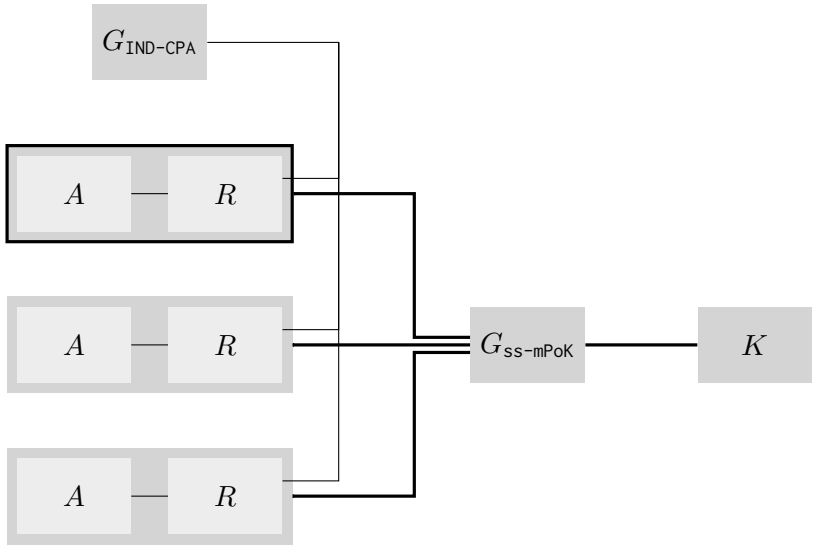


Figure 5.43: Diagram of the interaction with the ss-mPoK game and extractor.

"**Coin splitting**". We conduct a thought-experiment similar to a step in the proof of Theorem 5.32. Imagine that we used a random function to pick the bit b and the randomness r used in the challenge oracle for encryption. As inputs to this random function, we use some randomly drawn bits (from the random source shared between all copies of the reduction) and the history of all queries and responses made by the prover so far. The result of this thought-experiment is that while two copies of the prover that obtain identical answers to their queries (whether from the extractor or the simulator and its random oracle) continue to ask identical queries, the challenge oracles in any two "forked" copies of the reduction act as if they had independent sources of randomness. The CCA adversary would not notice as the bit b and the randomness r in the challenge ciphertext are still uniform thanks to the extra random bits in the input to the random function. The extractor is still interacting with multiple copies of the same prover (adversary plus reduction) so it must still work, and still has no control over any values returned to the main copy of the adversary.

We choose our bits b in rewinding copies of the reduction as follows. When the time comes to choose a bit b immediately before performing the challenge encryption, if the current copy's history is a strict prefix of any other copy then we just replay the same challenge ciphertext as the copy of which we are a prefix. Otherwise, we pick a fresh bit b and some randomness for encryption uniformly at random and make a new E-challenge ciphertext e on m_b (of the two

messages supplied in the query). Finally, we ask for a proof on e — as we are a rewinding copy, the extractor will have to handle this — and return the resulting challenge ciphertext to our copy of the adversary.

The main copy of the prover creates her challenge ciphertext using the IND-CPA challenger and the simulator. The only remaining case is what to do if the main copy of the prover must handle a challenge query while she is a prefix of another. This is why we added the extra random oracle query to all copies of the reduction just before making the challenge ciphertext: since the extractor does not control the main copy of the adversary's random oracle, the only way the main copy could be a prefix at this time is if the extra query happens to produce the same answer as the extractor previously gave to a rewinding copy. This only happens with negligible probability so we ignore this case.

We summarise the rules for dealing with challenge queries:

	main copy	**rewinding copy**
fresh query	use IND-CPA challenger	pick fresh bits and encrypt
prefix of another copy	cannot happen	replay ciphertext

Conclusion. If our adversary had a non-negligible advantage against the original CCA game then the main copy of our adversary still has a non-negligible advantage against the game that reduces to the IND-CPA challenger. In fact, the adversary's advantage changes only negligibly in our game-hops. We can simply forward the main copy of the adversary's guess to the challenger to complete our proof that simulation sound multi-proofs boost IND-CPA to CCA security in the Encrypt+PoK construction. *q.e.d.*

5.7 On the CCA security of Signed ElGamal

As explained in the introduction, intuitively the signed ElGamal scheme (Definition 5.19) should be CCA secure but Shoup and Gennaro [SG98] pointed out a flaw in the "obvious" proof. (Shoup and Gennaro discussed a scheme called TDH0 which is the threshold version of signed ElGamal. This does not affect the argument about lack of chosen-ciphertext security).

Consider an adversary that makes a chain of n ciphertexts-with-proofs, somehow using the ith ciphertext to create the $(i+1)$st each time (for example, the adversary could encrypt a hash of the ith ciphertext as the $(i+1)$st message; we will give the exact adversary later). Then, the adversary asks decryption queries on these ciphertexts in reverse order. Now consider the

extractor tasked with extracting from the ciphertext in the last decryption query (which contains the first ciphertext that the adversary made). To apply special soundness, our extractor invokes a second copy of the adversary and gives it a different answer to the random oracle query for the challenge in question. The copied adversary will not only make a new proof but all subsequent ciphertexts and proofs in the copied adversary will differ from the original ones; before the copied adversary gives the extractor the new nth ciphertext, it expects decryptions of the new $n-1$ other ciphertexts. Our extractor is now forced to deal with two adversaries each making $n-1$ ciphertexts before seeing the pair of proofs that will allow it to extract the message that the extractor was supposed to find in the first place. The result is that extracting from such an n-chain of proofs with this technique takes on the order of 2^n runs of the adversary: our rewinding extractor is no longer efficient.

Proving or disproving CCA security of signed ElGamal without resorting to the generic group model has been an open question. In this section we show that there can be no reduction proving CCA security of signed ElGamal assuming only the random oracle model and reasonable computational assumptions, but not the generic group model. Our proof is similar to the proof that Fiat-Shamir-Schnorr on its own is not a multi-proof but with two key differences. The first is the assumption we (meta-)reduce to. In the multi-proof game, the prover can challenge the extractor with proofs and expect witnesses back; the metareduction uses these witnesses to break the one-more discrete logarithm assumption. In the CCA game, the adversary "challenges" with decryption queries but only gets messages back. This is a weaker notion: one can compute a message from a ciphertext and witness in the Encrypt+PoK construction, but not the other way round. Indeed, in the Fiat-Shamir-Schnorr construction obtaining such a witness would imply the ability to take discrete logarithms. To avoid this problem, we will introduce a new assumption that we call "interactive ElGamal-Schnorr" or IES.

Secondly, the execution environment for a CCA–to–IND-CPA reduction does not check the validity of decryptions returned to the main copy of the adversary any more (there is no longer a "main" copy at all) and our metareduction cannot check the correctness of these decryptions itself either. We need another tool to ensure that the reduction cannot "cheat" by returning false answers to rewinding copies of the adversary which would spoil the success probability of the metareduction. We cannot "open" instances to check them either as this would break the argument that the reduction must make exponentially many queries to avoid helping the metareduction break its assumption. We cannot even hop from a game with checks into a game that omits them and use a "bad event" argument since there is no longer a main copy of the adversary: the reduction can always test how any copy of the adversary would react to an invalid

decryption, then "rewind" if it does not like the outcome. The only solution we have found is to sacrifice some soundness to regain the ability to check decryptions.

5.7.1 The IES assumption

We present a new assumption that we call interactive ElGamal-Schnorr or IES (Assumption 5.45). Recall that interactive Schnorr proofs are honest-verifier zero-knowledge but not known to be zero-knowledge (for large challenge spaces). We could assume that Schnorr proofs, while perhaps not zero-knowledge, are not completely insecure in the sense that a verifier interacting once with a prover cannot extract the witness immediately. Breaking this assumption would imply breaking the one-more discrete logarithm assumption too using techniques from the proof of Theorem 5.32 (essentially we are dealing with a single pair of instances at potential 1). The interactive ElGamal-Schnorr assumption applies the above idea to the signed ElGamal scheme instead of to Schnorr proofs directly.

Definition 5.44 (IES, informal). You are given an ElGamal public key and play the verifier on a single interactive Schnorr proof on an ElGamal ciphertext for a random message. Then you cannot compute the message.

This informal description suffices to compare IES to some existing assumptions:

- IES is a weaker assumption than claiming the interactive Schnorr protocol to be zero-knowledge against malicious verifiers.
- IES is stronger than CDH, since the adversary gets the extra "Schnorr challenge" on the CDH instance she is trying to solve to break ElGamal.
- IES is weaker than asking that the adversary be unable to extract the witness from the Schnorr protocol (the randomness used to encrypt) — IES only asks for the message.

Since even an ElGamal decryptor with the secret key cannot recover the randomness from a ciphertext, we require a weaker assumption than a full witness-extractor for our proof that signed ElGamal cannot be shown CCA secure; in particular we cannot use the one-more discrete logarithm assumption anymore. The following is the precise definition of the IES assumption.

Assumption 5.45 (IES). Let $\mathbb{G}$ be a group generated by G. The interactive ElGamal-Schnorr (IES) assumption is given by the following game.

```
oracle initialise
  sk ↞ Z_|G|, pk ← G^sk
  m ↞ Z_|G|, M ← G^m
  r ↞ Z_|G|, A ← G^r, B ← M · pk^r
  u ↞ Z_|G|, U ← G^u
  return pk, A, B, U

oracle challenge(c)
  // may be called once
  return u + cr

oracle finalise(M′)
  if M = M′ then
    return 1
  else
    return 0
  end if
```

5.7.2 Variations on IES

Our proof that signed ElGamal cannot be shown CCA secure reduces to the IES assumption but for better readability, we introduce some intermediate assumptions that we will use in the final proof and reduce them to IES.

Verifiable IES. First, we deal with the issue that decrypted messages are not "verifiable". We introduced proofs of knowledge for $\mathbf{NP}$ relations where a witness allows one to efficiently verify language membership of the claimed instance. In the proof that Fiat-Shamir-Schnorr is not a multi-proof, it is crucial that our adversary verify all witnesses returned by the extractor in order to force an exponential amount of rewinding. The IES assumption considers a Schnorr proof but only asks to obtain the encrypted message, not the randomness involved in encryption. If ElGamal is IND-CPA secure then correctness of the message cannot be efficiently verified; the relation we are effectively taking a proof over is not in $\mathbf{NP}$:

$$R'((pk, c), m) := \text{“}c \text{ is an ElGamal ciphertext for } m \text{ under key } pk\text{”}$$

We overcome this issue by introducing a new assumption that we call verifiable IES or V-IES. The difference is that the adversary gets many attempts at guessing the message; formally we introduce a new oracle for the adversary to check messages.

Assumption 5.46 (V-IES). The verifiable IES assumption is derived from the IES assumption by adding the extra oracle check given below. The adversary may call challenge once and afterwards, check any number of times.

```
oracle check(M')
  if M = M' then
      return 1
  else
      return 0
  end if
```

The V-IES assumption reduces to the IES assumption with a loss in soundness of a factor $q + 1$ where q is the number of checks made by the adversary. To see this, consider an efficient adversary with probability p of winning the V-IES game and let q be a (polynomial) bound on the number of checks the adversary makes. Then with probability p, one of the following $q + 1$ events occur: E_i for $1 \leq i \leq q$ is the event that the adversary makes at least i checking queries and the i-th check contains the correct message; E_0 is the event that the adversary never makes a checking query on the correct message but still calls the finalisation oracle with the correct message.

Our reduction to IES guesses $i \twoheadleftarrow \{0, 1, \ldots, q\}$ uniformly at random and simulates as follows: forward the initial data and the challenge query between the adversary and IES challenger, for $i > 0$ answer the first $i - 1$ checking queries with 0 and pass the result of the i-th checking query to the IES finalisation oracle directly, aborting the adversary at this point. For $i = 0$ answer 0 to all the adversary's checking queries and forward the adversary's output to the finalisation oracle. If the adversary does not make i checking queries or in case 0 makes no output, abort.

If event E_i occurs then the reduction for case i will break IES. Since we assumed the adversary to succeed with probability p, at least one of the events will occur with probability $p/(q + 1)$ as the $q + 1$ events are a partition of the event that the adversary succeeds. Since the reduction chooses i uniformly, we conclude that it succeeds against IES with probability $p/(q + 1)$.

One-more verifiable IES. For our metareduction we will use a one-more variation of IES. This is purely for an easier presentation and to be more aligned with the proof of Theorem 5.32 that

Fiat-Shamir-Schnorr is not a multi-proof. Unlike the one-more discrete logarithm assumption however, the one-more IES assumption reduces to "basic" IES. We give the one-more assumption and reduction for verifiable IES; the same reduction holds for the non-verifiable case.

The one-more assumption works as follows. The adversary may obtain and open a number of IES "instances"; her aim is to guess the message on an unopened instance. The initialisation oracle produces a public key shared between all instances. The instance oracle creates a fresh message, ElGamal ciphertext and Schnorr commitment; it marks these instances with a flag f where 0 means fresh, 1 means used (in a challenge) and 2 means opened. The challenge oracle allows the adversary one challenge per instance to which it gives the Schnorr response. The open oracle reveals the random values used in the encryption Schnorr proof. The checking oracle allows the adversary to verify a guess on the message in any instance; such checking does not mark an instance as opened. To win, the adversary must decrypt a message in an unopened instance. Our one-more version is "asymmetric" in that the adversary need only extract the message from an unopened instance to win the game but when opening an instance, she gets the random values used in the proof as well.

Assumption 5.47 (one-more verifiable interactive ElGamal-Schnorr). Let $\mathbb{G}$ be a group generated by G. The one-more verifiable interactive ElGamal-Schnorr (OM-V-IES) assumption is given by the game in Figure 5.48.

The reader may be asking why they should have any confidence that an assumption as complex as OM-V-IES should be hard. V-IES and OM-V-IES derive their justification solely from the fact that they reduce to IES: they are intermediate steps to make our main proof easier, not assumptions in their own right that we ask anyone to believe in. We gave the justification for basic IES when we introduced it: a single Schnorr proof should not completely break the security of ElGamal encryption.

The reason that the one-more version reduces to the simple one is that the instances are independent in the sense that the adversary cannot perform a challenge query that "touches" more than one instance.

```
1  oracle initialise
2    sk ↞ Z_|G|, pk ← G^sk
3    i ← 0, inst ← [ ]
4    return pk
5
6  oracle instance
7    m ↞ Z_|G|, M ← G^m
8    r ↞ Z_|G|, A ← G^r
9    B ← M · pk^r
10   u ↞ Z_|G|, U ← G^u
11   i ← i + 1
12   inst[i] ← (M, u, r, 0)
13   return A, B, U
14
15 oracle check(j, M′)
16   if j > i then
17       return ⊥
18   end if
19   (M, u, r, f) ← inst[j]
20   if M = M′ then
21       return 1
22   else
23       return 0
24   end if
25
26 oracle open(j)
27   if j > i then
28       return ⊥
29   end if
30   (M, u, r, f) ← inst[j]
31   inst[j] ← (M, u, r, 2)
32   return u, r
33
34 oracle challenge(j, c)
35   if j > i then
36       return ⊥
37   end if
38   (M, u, r, f) ← inst[j]
39   if f = 0 then
40       inst[j] ← (M, u, r, 1)
41       return u + cr
42   else
43       return ⊥
44   end if
45
46 oracle finalise(j, M′)
47   if j > i then
48       return ⊥
49   end if
50   (M, u, r, f) ← inst[j]
51   if M = M′ and f < 2 then
52       return 1
53   else
54       return 0
55   end if
```

Figure 5.48: The OM-V-IES game.

Lemma 5.49. There is a reduction from OM-V-IES to IES that loses a factor $O(q^2)$ in soundness where q is a bound on the number of queries made by the adversary.

It suffices to reduce OM-V-IES to verifiable IES with a loss of $O(q)$. Given an upper bound q on the number of instances an adversary can create, pick $n \leftarrow \{1, \ldots, q\}$ at random and use the verifiable IES challenger for the n-th instance. This requires using the challenger's public key for all other instances which is not a problem as the secret key is never required anyway; we can simulate, check and open all other instances in the same way as the one-more verifiable IES challenger. If the adversary tries to open the n-th instance the reduction aborts; since n was uniform we know that for an adversary with success probability p there is a probability of at least p/q of this event not occurring. *q.e.d.*

5.7.3 Admissible reductions

Before we state our main theorem we mention two restrictions we place on the class of reductions that we show cannot exist unless IES is easy. The reductions we consider are from CCA security of signed ElGamal to IND-CPA security of plain ElGamal.

1. Reductions may make only black-box use of the CCA adversary and while they may have access to many copies of the adversary with the same random source, they may not access the random source directly.

   This restriction seems to be necessary in order for the adversary to "hide" anything from the reduction, since a reduction that knows the adversary's code and random string could recompute anything the adversary "knows". (The CCA proof for signed ElGamal using the generic group model [SJ00] makes use of exactly this concept.)

2. Reductions must be key-preserving in the sense that they start by obtaining a public key from the IND-CPA challenger and pass this to the adversary unchanged.

   While we are not aware of any reason that it might be useful for a reduction to start otherwise than passing the challenger's public key on directly, we admit that we would like to remove this limitation in future work. The problem is not so much that the reduction might rerandomise the public key (which we could handle) but that it may choose to give some copies of the adversary a public key for which it knows the secret key.

5.7.4 Main theorem and proof

Theorem 5.50. Let $\mathbb{G}$ be a finite cyclic group with generator G. Suppose that R is an efficient reduction from CCA security of signed ElGamal over $\mathbb{G}$ to IND-CPA security of plain ElGamal over $\mathbb{G}$ that makes black-box use of multiple copies of the CCA adversary and preserves public keys, i.e. passes the public key from the challenger to all copies of the adversary.

Then there is either an efficient metareduction that breaks OM-V-IES with a non-negligible success probability or an efficient attack on DDH in $\mathbb{G}$ with a non-negligible advantage.

Proof. We consider the IND-CPA adversary consisting of R with black-box access to the CCA adversary in Figure 5.51 who operates in three phases. First, she does the same as the adversary from Theorem 5.32: build up a chain of proofs (in signed ElGamal ciphertexts) using a random function. Secondly, she asks decryption queries on them in reverse order. This gets messages rather than Schnorr witnesses back. Our adversary also checks that the reduction is returning correct messages (unlike in the multi-proof game, no-one is vouching for this). Thirdly, the adversary makes a challenge on two random messages and takes a discrete logarithm to decrypt it, then guesses the CCA challenger's bit correctly with probability 1.

The CCA adversary is inefficient using both a random function F and taking discrete logarithms however this does not matter since R only has black-box access to the adversary and we will construct an efficient simulation of the adversary. Later in the proof we will argue that R cannot progress the adversary to the point that it uses its discrete logarithm capability without breaking IES, allowing us to eliminate this part of the adversary.

In the proof we will make three case distinctions.

1. R answers the IND-CPA challenger's query without any copy of the adversary reaching phase 3. In this case, we can simulate all copies of the adversary by lazily sampling the random function to obtain an IND-CPA adversary that wins its game with the same probability as R, given access to a CCA adversary that always guesses correctly.

2. R answers a decryption query on a ciphertext without using special soundness. This case is the equivalent of the reduction in Theorem 5.32 answering an extraction query at potential

```
1  algorithm adversary(pk)
2    z_0 ↞ Z_{|G|}
3    // *** PHASE 1 ***
4    for i = 1 to n do
5        r_i ← F(z_{i-1}, 0, 1), a_i ← F(z_{i-1}, 0, 2), m_i ← F(z_{i-1}, 0, 3)
6        (C_i, D_i, A_i) ← (G^{r_i}, G^{m_i} · pk^{r_i}, G^{a_i})
7        c_i ← ro(pk, C_i, D_i, A_i)
8        z_i ← F(z_{i-1}, c_i, 4)
9    end for
10   // *** PHASE 2 ***
11   for i = n to 1 step −1 do
12       s_i ← a_i + c_i · r_i
13       M_i ← decrypt(C_i, D_i, A_i, c_i, s_i)
14       if not M_i = G^{m_i} then
15           halt // reduction cheated!
16       end if
17   end for
18   // *** PHASE 3 ***
19   // We will argue that the reduction never gets this far.
20   m*_0 ← F(z_n, 0, 1), m*_1 ← F(z_n, 0, 2)
21   (C*, D*, A*, c*, s*) ← challenge(G^{m_0}, G^{m_1})
22   r* ← dlog(C*) // by brute force
23   m* ← D*/pk^{r*}
24   if m* = m*_0 then
25       return 0
26   else
27       return 1
28   end if
```

Figure 5.51: The IND-CCA adversary.

$\phi = 1$. We build a metareduction to OM-V-IES where this case corresponds to R giving us the message for an unopened instance.

3. Neither of the above cases occur. In this case one copy of the adversary we are simulating proceeds to the point where it would have to use its discrete logarithm capability hence it must have got answers to all n decryption queries. By exactly the same reasoning as in Theorem 5.32, if the metareduction has not broken IES by this point then the reduction must have invoked $\Omega(2^n)$ copies of the adversary, contradicting the reduction's efficiency.

5.7.5 Case 1: the reduction solves DDH by itself

If the reduction answers the IND-CPA challenge without getting any copy of the adversary to run to phase 3 then the reduction must be breaking indistinguishability "by itself". In this case we can just simulate the adversary efficiently for as long as needed.

Lemma 5.52. Let E be the event that the reduction R calls its challenger's finalisation oracle without any copy of the adversary reaching phase 3. There is a metareduction M_1 that breaks DDH in $\mathbb{G}$ with advantage $\alpha_M = \mathbf{Pr}[E]\alpha_E/2$ where α_E is the advantage of R (with access to our adversary) given that E has occurred.

Proof. Metareduction M_1 simulates all the copies of our adversary in phases 1 and 2 and the random function by lazy sampling, once for all copies of the adversary. If an adversary reaches phase 3 or R aborts, M_1 outputs a random guess. Writing $\sigma_E := \mathbf{Pr}[R \text{ guesses correctly } \mid E]$ and $\alpha_E := (2\sigma_E - 1)$ we compute the advantage of M_1 as $\mathbf{Pr}[E] \cdot \alpha_E$. By Lemma 3.7 this gives an adversary against DDH with advantage $\mathbf{Pr}[E]\alpha_E/2$. *q.e.d.*

5.7.6 Case 2: The reduction breaks IES

If the reduction R does get a copy of the adversary to phase 3, we can hope that it solves IES for us along the way. We define a metareduction M as follows. We suggest that the reader familiarise themselves with the proof of Theorem 5.32 first: our metareduction is essentially the same as the three-layered reduction from said theorem that memoizes calls to the challenger, in this case instance and challenge queries. (The syntax is slightly adjusted for the challenger's instance indexing system.)

Each copy of the adversary in Figure 5.53 stores its challenge history in a list L that it uses in calls to the memoizer. The memoizer in Figure 5.54 contains the following data: I stores instances indexed by histories, n holds the number of instances created so far and N maps histories to instance numbers. Finally, Ch stores the challenges and responses computed so far and the associated potentials ϕ.

Our metareduction M simulates both the adversary and challenger interfaces towards the reduction R and interacts with an OM-V-IES challenger. On the challenger interface M passes the challenger's public key. By assumption, R is key-preserving so although the simulated adversaries formally receive a public key from R we could equally well have the metareduction provide them with this key directly. If the reduction ask a challenge query, we just simulate this challenge query (picking a random bit b); since we have already dealt with case 1 we can ignore the reduction returning a guess to the challenger for now.

layer 1 — adversary

```
1  algorithm adversary(pk)
2    L ← [ ]
3    for i = 1 to n do
4        (C_i, D_i, A_i) ← m_instance(L)
5        c_i ← ro(pk, C_i, D_i, A_i)
6        L ← L :: c_i
7    end for
8    for i = n to 1 step −1 do
9        s_i ← m_challenge(L, i)
10       M_i ← decrypt(C_i, D_i, A_i, c_i, s_i)
11       if not m_check(D_i, M_i, L, i) then
12           halt  // reduction cheated
13       end if
14   end for
15   halt
```

Figure 5.53: The adversary on layer 1.

Unlike in the proof of Theorem 5.32, the elements the adversary receives in response to her queries — plaintexts rather than discrete logarithms — do not suffice to check that the reduction has not cheated. This forces us to perform an explicit checking query on each plaintext.

The memoizer in Figure 5.54 differs from the one in Theorem 5.32 principally in the addressing scheme (using histories and prefixes pre to find the index j that the OM-V-IES challenger requires) and in the extra oracle `m_check` for the adversary to check her decryptions.

Lemma 5.55. The metareduction M above working with an OM-V-IES challenger produces identically distributed elements as a collection of multiple copies of the

layer 2 — memoizer

```
16 oracle m_instance(L)
17   if I[L] = ⊥ then
18       n ← n + 1
19       N[L] ← n
20       I[L] ← instance()
21       Ch[L] ← (2, ⊥, ⊥)
22   end if
23   return I[L]
24
25 oracle m_challenge(L, i)
26   pre ← (L[1], ..., L[i − 1])
27   c ← L[i]
28   j ← N[pre]
29   if j = ⊥ then
30       halt // cannot happen
31   end if
32   (φ, u, v) ← Ch[pre]
33   if φ = 2 then // fresh instance
34       s ← challenge(j, c)
35       Ch[pre] ← (1, c, s)
36       return c
37   else if φ = 1 then
38       if c = u then
39           // repeated query
40           return v
41       else
42           // forked on u, c
43           (a, r) ← open(j)
44           Ch[pre] ← (0, a, r)
45           return a + cr
46       end if
47   else // φ = 0
48       return u + cv
49   end if
50
51 oracle m_check(D, M, L, i)
52   pre ← (L[1], ..., L[i − 1])
53   j ← N[pre]
54   if not check(j, M) then
55       return false
56   else
57       return true
58   end if
```

Figure 5.54: The memoizer on layer 2.

adversary given at the beginning of this theorem, until the point where a copy of the adversary would reach phase 3.

Proof. We observe that the one-more verifiable IES challenger creates instances the same way as our adversary used to: it picks ciphertexts for uniformly random messages. Everything in the memoizer is "bookkeeping" that does not affect the metareduction's outputs to the reduction, namely decryption and random oracle queries. The IND-CPA advantage of the reduction thus does not decrease when we replace the copies of the adversary by the metareduction. *q.e.d.*

Lemma 5.56. Let E be the event that the reduction answers a decryption query correctly on an instance at potential $\phi = 1$. If event E occurs then the metareduction breaks OM-V-IES.

The proof of this lemma is essentially the same as that of Lemma 5.38. Since instances at potential 1 are unopened, the correct decryption wins the OM-V-IES game using the instance in question.

(We assume w.l.o.g. that instances from the challenger do not collide: in case of a collision we can again proceed as in Lemma 5.38 and win the OM-V-IES game immediately.)

5.7.7 Case 3: The reduction takes exponential time

Lemma 5.57. Let E be the event that the reduction answers a decryption query correctly on an instance at potential $\phi = 1$. If a copy of the adversary advances to phase 3 without event E having occurred then the reduction must have launched $\Omega(2^n)$ copies of the adversary.

This lemma is "isomorphic" to Lemma 5.39 in the proof that Fiat-Shamir-Schnorr is not a multi-proof and the proof of this lemma is the same too. We repeat only the main argument: the whole chain of instances in the copy A that advanced to phase 3 must be at potential 0 and the only way that its i-th instance can drop to potential 0 is if some copy A' of the adversary triggers an opening query on this instance. Translating the conditions for an opening query into copies' histories, we find that A and A' must share the first $i-1$ oracle responses and fork on the i-th one. A combinatorial argument shows that these conditions lead to a complete binary tree of depth n with a different copy of the adversary at each leaf.

5.7.8 Conclusion

If R is an efficient key-passing black-box reduction from CCA security of signed ElGamal to IND-CPA of plain ElGamal, with a non-negligible advantage α when given a CCA adversary with advantage 1, then we combine the above cases as follows.

Assume w.l.o.g. that the reduction R always outputs a guess in time bounded by some polynomial $p(n) < 2^n$: if R aborts, we pick a bit at random and return this. This change cannot decrease the advantage α. Next, let E be the event that the reduction answers some copy of the adversary's query at potential $\phi = 1$. We know from Lemma 5.57 that such a reduction R can never advance an adversary to phase 3 without E occurring, as this would take at least $\Omega(2^n)$ steps just to launch all the required copies.

Suppose we abort the reduction if event E occurs and return a random bit here too — this means we never have to simulate any adversaries advancing to phase 3. The advantage of our reduction drops by $2 \cdot \mathbf{Pr}[E]$.

If E occurs with some probability $p_E > 0$, we could run the whole construction against IES as in Lemma 5.56 and win with probability $p_E/2q(q+1)$ where q is a bound on the number of queries made to the challenger. We can bound q from above by $p(n)$; the factor $2q(q+1)$ is the reduction from OM-V-IES to plain IES. We have therefore constructed efficient adversaries with advantage α_{DDH} against DDH in the underlying group $\mathbb{G}$ and with success probability $\sigma_{\mathtt{IES}}$ such that

$$\alpha_{\mathsf{DDH}} + 4q(q+1)\sigma_{\mathtt{IES}} \geq \alpha$$

where α was the advantage of our reduction against DDH given a CCA adversary with advantage 1. If α is non-negligible, so is at least one of the two quantities α_{DDH} and $\sigma_{\mathtt{IES}}$ on the left-hand side. *q.e.d.*

5.7.9 Discussion

Is signed ElGamal CCA secure? Unfortunately, the only answer we can give with absolute certainty is a clear "it depends". To see why, consider what form an answer could take. A positive answer could be a proof that under some assumptions, any attack on CCA security can be efficiently transformed into an attack on some underlying assumption. A negative answer could be an attack that breaks CCA security with a non-negligible advantage under some assumptions, either generically or for a particular instantiation in a given family of groups.

When Shoup and Gennaro [SG98] first pointed out the problem with the "obvious" proof, it was not yet clear whether one just needed a cleverer argument to make the proof go through or whether there was a fundamental problem with the approach. We view our result (Theorem 5.50) as evidence that there is a fundamental problem with the use of rewinding-based proof schemes in adaptive scenarios such as CCA security. In other words, a positive answer using any of the usual techniques in game-based security is unlikely if not impossible.

eSince Signed ElGamal uses the Fiat-Shamir-Schnorr proof scheme, we need some model in which to reason about this scheme and the best model we know of is the random oracle model. Even more fundamentally, since Signed ElGamal requires a group over which to operate, we need some assumption or model of the group itself or rather of a family of groups. The DDH assumption is one example of such and assumption; the generic group model is another. We know that in some families of groups (such as additive groups modulo primes) any form of ElGamal is insecure. We also know that in the generic group model there are proofs of CCA security of Signed ElGamal [TY98, SJ00]. The answer to our question therefore must depend to some extent on the model involved; if we ask whether one should use Signed ElGamal as a CCA secure scheme in the "real world" this raises the question of which models we trust.

The problem preventing us from turning our negative results into a straightforward attack on the CCA game for Signed ElGamal lies with the nature of proof schemes: we declare a scheme to be secure (zero-knowledge, proof of knowledge etc.) by showing the existence of some hypothetical parties such as simulators or extractors under model-specific conditions that never occur in real uses of the proof scheme. If we can show that it is impossible to construct a simulator under conditions that are impossible in the first place (such as being able to program a hash function), it says more about the inadequacy of our understanding of zero-knowledge than about the Signed ElGamal scheme itself. We cannot exclude the possibility that someone will design a new model on which to base reductionist cryptography, which eventually becomes accepted alongside the random oracle and generic group models or even replaces them, and that Signed ElGamal turns

out to be CCA secure in this model. One could for example include a "knowledge assumption" on sigma protocols or an equivalent statement as a new axiom, similar to the approach already taken by Tsiounis and Yung [TY98].

We believe that the general approach known variously as reductionist or "provable" security is a good model in which to reason about the "real" security of encryption schemes, whatever that means; within this model we are willing to accept the random oracle model and its applications such as Fiat-Shamir transformed sigma protocols. For the generic group model on the other hand, while we consider it useful as a "sanity check" on schemes and assumptions and a way to disprove security of particular constructions, we have less confidence in the generic group model as a foundation to argue that a scheme is secure. The question here is not whether or not a scheme can be shown to be secure if one assumes the generic group model — this is usually not a matter of opinion — but whether a generic group security argument can be sufficient to conclude security in the "real world".

For the question at hand, we note that the generic group proofs [TY98, SJ00] rely on a straight-line extractor that provides a witness along with each Fiat-Shamir-Schnorr proof. Yet Seurin and Treger [ST13] showed that a straight-line extractor for Signed ElGamal in the random oracle model alone implies an attack on the CDH assumption. To make a statement on the security of Signed ElGamal we are forced to pick one of two models with contradictory results and we choose to place our confidence in the random oracle model without generic groups.

Our own result indicates that even a non-straight-line, i.e. rewinding extractor in the random oracle model would yield an attack albeit on a stronger assumption. We interpret this result as showing that the intuition behind the informal argument why Signed ElGamal should be CCA secure (at the start of Section 5.2) is incorrect when applied to rewinding-based proof schemes. This leaves us with neither an intuitive nor a formal argument that would convince us that Signed ElGamal is, or should be, CCA secure.

Based on our results in this work and the published literature we are familiar with, our present opinion therefore is that Signed ElGamal, for practical purposes, should be viewed as a non-malleable yet *not* CCA secure scheme. If CCA security is required in the proof of a particular protocol, depending on whether homomorphic features or ciphertext size are more important, we recommend TDH2 or Chaum-Pedersen Signed ElGamal. We would be happy to reconsider this stance however if anyone presents us with new evidence, informal or formal, in favour of CCA security of Signed ElGamal.

6 Cryptography for voting

We come to a practical application of our theory of proof schemes: the Helios [A06, A08] voting system. Our main notion in this chapter is that of ballot privacy (Definition 6.5) which we first formulated in a paper at Esorics in 2011 [BC+11]; the version in this paper improves on some details that were left ambiguous in the original. Ballot privacy applies to single-pass voting systems, a class of systems that we defined to generalise Helios (Definition 6.3). Our main result of this chapter is Theorem 6.12 which for the first time proves ballot privacy of "full" Helios, i.e. considering the complete ballot structure. This is important because several of the issues raised by Cortier and Smyth [CS11, CS13] apply to full ballots rather than individual ciphertexts within ballots.

6.1 Introduction to voting

Voting looks like an almost trivial task to run electronically. Surely all there is to do is to count some votes! Nothing could be further from the truth. Schneier [Sch01] and Adida [A06] refute the claim that, as we already use computers to fly planes full of people and run the worldwide financial industry, trusting computers with both our lives and our money, then computers are also good enough to run elections. Voting poses a set of challenges not encountered anywhere else. As Schneier says, you can't redo an election. If something goes wrong, unlike in the financial industry, it can't be fixed with a "refund". As Adida elaborates, it is also much harder to tell if something has gone wrong in an election in the first place. Planes do not crash without anyone noticing that they are missing. One cannot steal money from a bank account without leaving the owner a clue on their balance that some money has been taken. Yet how can we ever have a guarantee that an election was held fairly? To understand the problems surrounding elections better, we give a brief overview of some points in the history of voting and then explain what we think cryptography can, and cannot, contribute.

Voting BC ("before cryptography"). Adida [A06], in his PhD thesis, gives a brief history of voting. Early elections had voters publicise their votes by writing them on ballot papers (earlier still, clay shards), announcing them to election officials who recorded them on lists or by a show of hands — a system still in widespread use today. The main advantages of these systems are their simplicity and verifiability: anyone present can in principle satisfy themselves that the announced results correspond to the votes cast by voters. However, such systems do not scale particularly well — a show of hands is much harder to trust for someone who was not in the same room at the time.

In the 19th century, the secret ballot was introduced, originally called the Australian ballot after the first country to use it. Secret ballots have become standard in political elections nowadays because as soon as the stakes are high enough, any system in which voters' votes are public becomes open to abuse. Voters can be bribed or intimidated so while public voting guarantees an accurate tally of the votes cast, it allows for considerable incentive for voters to deviate from their intentions. The downside of the secret ballot is that correctness of the result now relies on trust in the vote counters, whether human, mechanical or electronic.

The voting paradox. Choosing between public and secret ballot, we face a problem that we call the voting paradox: we want to be sure that the count is correct yet at the same time we do not want to reveal how anyone voted. It may seem that we are faced with a spectrum as illustrated in Figure 6.1. At one end, we have a secret ballot where we have to take the election results on trust but may be able to provide reasonable security against vote-buying. At the other end, we can verify the results but have to trust that they were not obtained by bribery or coercion. A point in the interior of the spectrum might be an election with paper ballots that can be recounted by anyone who wishes but these ballot papers are assumed not to reveal their authors — for example, they are only marked with crosses rather than handwriting. Such a system might still be vulnerable to ballots being added or removed.

Figure 6.1: A non-cryptographer's view of voting?

There has been much debate where on the spectrum the optimal point lies. The secret ballot is widely practiced in the western world nowadays yet not without opposition. For example, US activist Lynn Landes called to "scrap the secret ballot" in an article and a report to the US Congress [L05, L07].

Enter the machines. If vote counts need to be accurate yet private, surely the counting process is an ideal candidate for mechanisation or even computerisation? In practice, while mechanical voting machines have been used particularly in the US for some time, they failed spectacularly in the 2000 presidential election. The presumed remedy was to switch to computerised "direct recording electronic" systems — no more hanging chads! Yet these machines, if anything, have performed even worse: numerous examples exist of failures on all levels, from votes being incorrectly recorded to impossible results for the number of votes actually cast. As a reliable indicator of the scale of the problems, we mention the results of the "top to bottom review" carried out in California in 2007 [CA07], finding that all audited machines had serious problems and leading to them losing their certification to be used in official elections. Among other problems, these electronic machines lacked a key feature present in all previous systems: a full paper audit trail that could if necessary be examined in court. A disputed punch card with a "hanging chad" is still better than a dispute where there is no card at all. Indeed, since the decertification a "voter-verified paper audit trail" has been retrofitted onto some electronic voting machines; such machines can currently be used in California under certain circumstances if supplemented by a manual recount of votes using the paper records [CA07].

Internet voting. Using cryptography in elections was first suggested by Chaum [C81]. Unfortunately, the ideas that originated in his paper and that we address in this chapter are often misnamed as "electronic voting" or, worse still in our opinion, "Internet voting". For lack of a more concise term, we suggest *cryptographic* voting. Voting over the Internet, as has recently been tried or introduced in several countries, comes with challenges and opportunities of its own that have little to do with the subject of this chapter. In particular, a shift from voters being physically present in a polling station to casting votes online opens up new possibilities (such as allowing expatriates to vote more easily) but comes with new challenges for voter authentication. Further, voting from your personal computer/smartphone/internet-enabled home appliance raises concerns such as malware, phishing or the sale of voting credentials. Cryptography can help address these concerns to some extent but not overcome them on its own; in any case such issues are outside the scope of this thesis. Cryptographic voting, on the other hand, can be deployed in both polling stations and Internet-based elections.

For example, the UK currently uses a paper-based voting system that is neither cryptographic nor uses the internet; Estonia has an internet voting system that is not cryptographic in our sense of the term; Israel uses the cryptographic Wombat [W] voting system for primaries which uses paper ballots and is not internet-based and Norway has a cryptographic internet voting scheme. All possible combinations of cryptographic and internet voting are represented here!

Cryptographic voting. So what *did* Chaum and other cryptographers propose? Our selling point is that privacy and verifiability in elections are simultaneously achievable. Modern cryptography has much more to offer than just encryption. Commitment schemes are the cryptographic equivalent of sealed envelopes: others cannot see what is inside yet you still cannot change what you sealed in the envelope. Zero-knowledge protocols allow Alice to prove that she has filled in a valid ballot without showing anyone the ballot or allowing her to change her mind later; or she can prove that she has counted a stack of votes correctly without revealing the votes. The situation we hope to achieve is that of Figure 6.2.

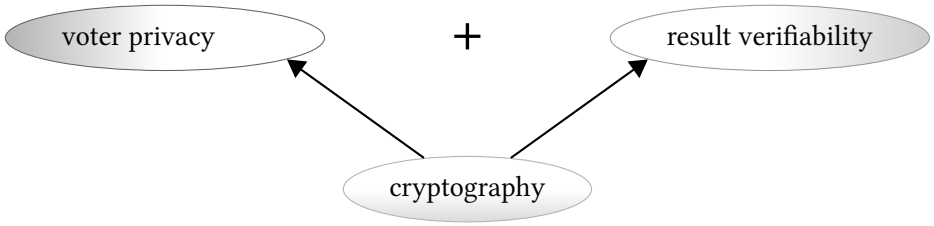


Figure 6.2: Cryptographers' view of voting?

We view cryptography as a layer that can be placed on top of election protocols to achieve extra security properties, not as a way to replace voting as we know it by something new. Cryptography does not dictate a shift from paper ballots placed in locked boxes at a polling station to voting from home on a computer connected to the Internet, any more than it dictates whether one should vote using first-past-the-post, instant runoff or approval voting.

Indeed, the two principal instances we know of where true cryptographic voting has been used in political elections — in the sense that we understand the term, to provide verifiability — both use a system where voters fill out a paper ballot. These two examples are the Wombat voting system [W] in Israel and mayoral elections in Takoma Park, Florida using Scantegrity which we analysed for privacy [BCPW12]. Both these systems allow a manual count using the paper ballots.

Practical considerations. All uses of cryptography in voting come at a cost. We concede that to perform cryptographic tallying and verification of tallies, computers are required at some stage in the process (although elections may still be tallied independently by hand). While election officials can be assumed to have some technical knowledge necessary to operate a cryptographic voting system, we view the usability of a system from a voter's point of view as one of the most important properties of a voting system. We thus exclude from practical consideration any system where voters are required to manage their own cryptographic keys or create and store secret data for any length of time. (This excludes, apart from systems where everyone is assumed to possess a certified key pair as part of a universal public-key infrastructure, also systems based on commitments where voters commit to their vote during the election proper and are asked to decommit at a later time.) In addition, voters should not have to communicate with each other (to perform multi-party computation). We will propose a model of voting that we call single-pass: voters have to interact only with one, publicly known voting authority (which could be a website or a polling station) and interact with this authority only once to cast their vote. Voters can choose to additionally audit their vote and the tally but this is not required.

Some cryptographic voting systems aim to use cryptography to ensure coercion resistance, for which formal definitions and schemes implementing them exist. The idea in one scheme due to Juels et al. [JCJ05] is that voters hold one true credential and can create ballots for any credential they like; ballots with fake credentials are not counted. A voter under coercion can thus use a fake credential to create useless ballots while secretly casting her actual vote. Unfortunately, achieving coercion-resistance (in the formal sense) requires knowledge and skill on the part of voters beyond the requirements of simply ballot-private schemes. We are not aware of any use of such coercion-resistant cryptographic schemes in practical elections and believe that achieving coercion-resistance cryptographically is impractical; we would prefer physical measures such as ballot boxes and booths to be used for this purpose.

Finally, no new proposal for a voting system can succeed unless it is trusted by the public, who overwhelmingly are not mathematicians let alone cryptographers. At the very least, we should ensure that cryptography is not used in a way that adds an additional point of failure to an election process. We therefore propose that any practical voting scheme for high-stake political elections should be able to fall back to a non-electronic mode of operation: first, voters should either fill in paper ballots directly or possibly fill in ballots digitally and obtain a paper printout on which their choices are clearly visible, before casting this into a ballot box. Secondly, it must be possible to count the paper ballots by hand; for the time being we would even suggest that such a count be mandatory in addition to any cryptographic tallying.

Outline of this chapter and future work. The rest of this chapter is structured as follows. First, we introduce our model of elections that we call single-pass voting and give a "toy" example. Next, we introduce the security notion of ballot privacy and the Helios voting scheme. Finally, we prove that Helios satisfies ballot privacy assuming that the zero-knowledge proofs employed are strong.

Since the selling point for cryptographic election schemes is their verifiability, the reader may be asking where to find a formal notion of verifiability and a proof that Helios satisfies this notion. Such a model and proof do not yet exist. Helios was built, in our opinion, on sound principles by competent cryptographers yet it was never proven secure in any model by its authors, neither with respect to privacy nor verifiability. Cortier and Smyth [CS11, CS13] discovered some privacy issues which we have rectified together with them and our supervisor Warinschi and Pereira, one of the authors of Helios. The model of single-pass elections in this thesis is essentially that of our joint paper at Esorics [BC+11] which has undergone only minor changes since. This thesis is the first work to contain a ballot privacy proof for "full" Helios (Theorem 6.12), i.e. taking into account the ballot format with multiple ciphertexts which gave rise to some of the Cortier-Smyth attacks. Our proof refers to Helios patched as we suggested in a paper with Pereira and Warinschi [BPW12b] which will be implemented (according to Pereira) in the upcoming version 4 of Helios. In the same work, we also found some potential problems with verifiability in previous versions of Helios. While we have every reason to believe that these issues are fixed too, no proof of this fact exists to date — this will be an obvious opportunity for future work after we have completed this thesis.

6.2 Single-pass voting

Single-pass voting is a published model of ours [BC+11, BPW12a, BPW12b] for election or polling schemes in which the following rules limit voters' interaction with the scheme:

- Voters may have a private input representing their identity or credentials; how they obtain this is outside the scope of the single-pass model.
- Voters' only interaction during the voting process is with a public bulletin board.
- Before voting, voters may read a message off the board containing public information (such as election rules, candidate lists and cryptographic information).

- Voters cast a ballot by submitting it to the board. They are not required to interact with the voting system any further.
- The bulletin board displays successfully cast ballots unchanged.

Bulletin boards. A bulletin board, for our purposes, satisfies the following requirements.

1. All parties involved in a voting protocol know how to interact with the bulletin board.
2. All parties may post messages to the board, which can accept or reject messages. We assume that communication to the board is via "authentic channels".
3. The bulletin board runs a publicly known algorithm; one can therefore predict the outcome of sending a message to the board.
4. The board maintains as its state the list of all messages accepted so far. It appends all accepted messages to its state; no state is ever overwritten.
5. All parties may read the state of the board at any time.

Result functions. Mathematically speaking, the purpose of an election is to compute some function $\rho : (V \cup \{\bot\})^n \to R$ where n is the number of voters, V is the space of valid votes (with $\bot$ denoting abstention) and R is some result space; we call this the result function of an election.

The notion of a result function has several purposes although we can only hint at them in this thesis. Above all, we would like to use result functions to define verifiability in future work, in a similar manner to their use in the correctness property below. For privacy purposes, we will use the result function in our proof of the minivoting scheme which we then extend to a proof for Helios.

The output of an election protocol we call the tally. Given a tally, we assume there is an unambiguous and efficient way to extract the result contained in the tally. The tally in a verifiable election scheme will typically contain more than the result, for example zero-knowledge proofs of correct tallying.

6.2.1 Formal definition

We give a version of single-pass voting that builds upon our earlier notions [BC+11, BPW12a, BPW12b] but with some changes in presentation. For example, we will present ballot privacy

using code-based games, our presentation of the output of the ballot validation algorithm is new and we leave the algorithm to extract results from tallies implicit.

Definition 6.3 (Single-pass voting). A single-pass voting scheme (SPS) is a scheme meeting the conditions given in this section.

Participants. A single-pass scheme is a scheme for a set $\mathbb{V}$ of voters, a set $\mathbb{A}$ of administrators and a bulletin board *bb*. We assume that all participants can read the state of the board at any time and can send messages to the board, which the board can either append to its state or reject.

Sets. The algorithms and protocols in an SPS operate over the following sets.

V	votes	Y	setup parameters
B	ballots	X	administrator private data
$\mathbb{B}$	bulletin board states	T	tallies
ID	voter identities	R	results

Constants. We assume the following are fixed in advance: non-cryptographic election specifications such as candidate lists and voting rules, the sets of voters and administrators and voter identities. Further we assume a result function $\rho : (V \cup \{\perp\})^n \to R$ is given, where $n = |\mathbb{V}|$ is the number of voters and we use the symbol $\perp$ here to denote abstention.

Algorithms. A single-pass scheme is defined by the following algorithms and protocols.

`Setup` is an interactive, randomised protocol for the administrators to set up an election. It takes no input and produces a single public output $y \in Y$ and a private output in X for each administrator. The "signature" of `Setup` viewed as an algorithm when the number of administrators is m is thus $\twoheadrightarrow Y \times X^m$.

`Vote` : $V \times ID \times Y \twoheadrightarrow B$ is a randomized algorithm with which voters create their ballots.

`Validate` : $B \times \mathbb{B} \to \{0, 1\}$ checks ballots for validity. The bulletin board runs this algorithm. It takes a new submission and the current state of the bulletin board as inputs.

`Tally` is an interactive protocol for the administrators. Each administrator has a private input from the setup and all administrators have the bulletin board *bb* as public input. The

algorithm outputs a tally $\tau \in T$. (The "signature" is thus $\mathbb{B} \times X^m \twoheadrightarrow T$ where m is the number of administrators).

`Verify` : $\mathbb{B} \to \{0,1\}$ The verification algorithm takes a board after an election has completed and accepts or rejects the election.

Phases. A single-pass scheme runs in three phases.

1. *Setup phase.* The administrators jointly run the `Setup` subprotocol and post its public output $y \in Y$ to the bulletin board.

2. *Voting phase.* Each voter may read the board to get the public information y and create a ballot $s \leftarrow \texttt{Vote}(v, id, y)$ where v is her vote and id her identity or credential information.

   On receiving a submission s, the board runs $\texttt{Validate}(s, bb)$ where bb is its current state. If this succeeds (returns 1) then the board appends s to bb to give its new state, otherwise it discards s.

3. *Tallying phase.* The administrators first simulate the board's actions in the voting phase (e.g. in particular checking that the board never added any data that failed the `Validate` test). Next, they jointly run `Tally` and append the tally to the board.

   After the administrators have posted the tally, anyone can run `Verify` on the board.

Data formats. We assume that given a bulletin board (state) it is possible to distinguish individual submissions and determine which submissions correspond to which phases. Further, we assume that a tally $\tau \in T$ decomposes into two parts, a result ρ and auxiliary data a.

Correctness. An SPS is correct with respect to a result function ρ if the following property holds. In this case, we call it simply "an SPS for ρ".

For any assignment of votes or abstentions to voters (formally, a map $\nu : \mathbb{V} \to V \cup \{\perp\}$), if all parties execute the SPS as described above and the voters cast their assigned votes, then the result $r \in R$ of the SPS (appearing on the final board) corresponds to the result function ρ applied to the votes directly, i.e. $r = \rho([\nu(v)]_{v \in \mathbb{V}})$.

6.2.2 Minivoting

We present the minivoting construction, essentially taking a public-key encryption scheme and calling it a single-pass voting scheme. This scheme is not verifiable and so of little practical use but serves as an example SPS, a tool for analysing ballot privacy and a step towards Helios.

Definition 6.4 (minivoting). The minivoting scheme is the following SPS for one administrator and a single yes/no question. Suppose E is a public-key encryption scheme. Then `minivoting`(E) consists of

`Setup` The administrator runs the `KeyGen` algorithm, sets her private data to be the secret key and the public output of the setup algorithm to be the public key.

`Vote` The voter encrypts 1 if her vote is "yes", otherwise 0 under the public key of the election (that is on the bulletin board).

`Validate` Given a submission s, if the exact same s already appears on the bulletin board then reject it, otherwise accept it.

`Tally` Decrypt all ballots (discard any that decrypt to $\perp$), count the number of 0 and 1-votes and return these counts.

`Verify` Just return 1 (this scheme is not verifiable).

6.3 Ballot privacy

Ballot privacy is the property that a run of an election does not reveal more than it is supposed to via the result function. For example, if the result function outputs the number of "yes" and "no" votes and two voters Alice and Bertha vote for opposing choices, the run of the election should not reveal which of the two voted "yes". However, if the result is a unanimous "yes" with no abstentions then no protocol can prevent anyone from learning that Alice voted "yes".

We model ballot privacy as a game in which the adversary interacts with the voting protocol on behalf of some corrupt parties and wishes to obtain knowledge about the votes of honest voters. Borrowing from the security notions for encryption, actually we let the adversary choose two votes for each honest voter, a "left" and a "right" one; her task is to guess which of the two votes the election used. The adversary may corrupt as many voters as she likes, even adaptively (i.e. choose whether a particular voter is to be corrupted or not after she has let other voters interact with the voting protocol).

When the adversary has let all voters she wishes to interact with the protocol, she obtains the result of the election. Of course, we cannot give the result of the ballots on the board or

we would immediately reveal whether we used the "left" or "right" votes so we always give the result of the "left" votes. To write this formally, we actually maintain two boards, a left and a right one. For each honest voter, the adversary may pick two votes and a ballot for the left vote gets added to the left board and vice versa. The adversary can ask to read the board at any time and gets the *visible* board back; the two games between which the adversary should distinguish differ principally in which of the two boards is visible, the left or the right one.

For each dishonest voter, the adversary may choose an arbitrary ballot that gets submitted first to the visible board, then if successful to the invisible board. The reason for this ordering is that the adversary may try and copy an honest voter's ballot and submit it on behalf of a dishonest voter in which case the visible board should reject it as a duplicate, yet on the invisible board the original ballot will not exist but we still want to reject the copied ballot or else the adversary could trivially distinguish the two games. Otherwise, the invisible board would end up with one ballot more than the visible board yet the adversary always gets the tally of the left board and could tell from the total number of votes reported, which of the two boards the game tallied.

Definition 6.5 (ballot privacy). An SPS protocol has ballot privacy if no adversary can distinguish the two games of Figure 6.6 with a non-negligibly greater advantage than $1/2$.

The adversary may call oracles `board`, `vote` and `ballot` arbitrarily many times and in any order; she may call `tally` once after which all oracles except `board` become inactive.

Ballot privacy of minivoting. Minivoting has ballot privacy under the following conditions.

Encryption	Origin of result
IND-CCA2	[BC+11]
NM-CPA	[BPW12a]

We give the theorem and proof for the NM-CPA case as a step towards proving security of Helios.

Oracles in both "L" and "R" games.

```
1  oracle initialise
2    L ← [ ],  R ← [ ]
3    (x⃗, y) ← Setup()
4    L ← L :: y,  R ← R :: y
5
6  oracle tally
7    τ ← Tally(L, x⃗)
8    r ← r(τ) // extract result
9    return r
10 oracle vote(id, v_L, v_R)
11   b_L ← Vote(v_L, id, y)
12   b_R ← Vote(v_R, id, y)
13   if Validate(b_L, L) then
14       L ← L :: b_L
15   end if
16   if Validate(b_R, R) then
17       R ← R :: b_R
18   end if
```

Oracles for game "L" only.

```
19 oracle board
20   return L
21
22 oracle ballot(s)
23   if Validate(s, L) then
24       L ← L :: s
25       if Validate(s, R) then
26           R ← R :: s
27       end if
28   end if
```

Oracles for game "R" only.

```
29 oracle board
30   return R
31
32 oracle ballot(s)
33   if Validate(s, R) then
34       R ← R :: s
35       if Validate(s, L) then
36           L ← L :: s
37       end if
38   end if
```

Figure 6.6: The ballot privacy game.

Theorem 6.7. Minivoting with NM-CPA secure encryption has ballot privacy.

We give a reduction from ballot privacy of minivoting to IND-1-CCA security of encryption, which we know by Lemma 3.10 to be equivalent to NM-CPA. First, we need to extend the IND-1-CCA notion from one to many challenge queries. This technique, called a hybrid argument, works for any IND notion.

Lemma 6.8. Consider a version of the IND-1-CCA game in which the adversary may call the challenge oracle many times and each challenge query uses the same random bit b; when asking the decryption query she receives $\perp$ as an answer if she includes any of the challenge ciphertexts in the vector of ciphertexts to be decrypted.

There is an efficient reduction taking an adversary against this game that makes up to n challenge queries and achieves advantage α and produces an adversary against the original IND-1-CCA game with advantage α/n.

Proof. For this lemma it is helpful to view the multi-challenge IND-1-CCA game as a pair of games, a left one G_L for the case $b = 0$ and a right one G_R for $b = 1$. Consider the sequence $G_0, \ldots, G_n$ of $n + 1$ games where game G_i answers the first i challenges as if $b = 1$ (i.e. it encrypts the second of the two input messages); game G_i answers and all further challenges as if $b = 0$, i.e. it encrypts the first message. We have $G_L = G_0$ and $G_R = G_n$ for an adversary making up to n challenge queries. Let $p(i)$ for $0 \leq i \leq n$ be the probability that the adversary outputs 1 in game G_i and $\delta_i := p(i) - p(i-1)$ for $1 \leq i \leq n$. We wish to show that all δ_i are negligible unless there is an adversary with a non-negligible advantage against the single-challenge game; the best adversary's advantage against the multi-challenge game is then $p(n) - p(0) = \sum_{i=1}^{n} \delta_i$ by "telescoping" the sum, which is still negligible.

Given an adversary A_i (for $1 \leq i \leq n$) with advantage δ_i in distinguishing games G_i and G_{i-1}, we construct the following reduction to the one-challenge IND-1-CCA game: obtain a public key from the challenger, answer the first i challenge queries by selecting the first message m_0, pass the ith challenge on to the IND-1-CCA challenger and answer all subsequent queries

using the second message m_1. Forward the decryption query and its results between the adversary and challenger. If $b = 0$ then the challenger will answer the ith query using m_0, which is exactly what G_{i-1} does; if $b = 1$ then it will answer this query using m_1 which is the same as G_i. So our adversary's advantage in distinguishing the two games translates into our advantage against the IND-1-CCA game.

Suppose there is an adversary with advantage α against the multi-challenge game. Since $\alpha = p(n) - p(0) = \sum_{i=1}^{n} \delta_i$ for the definitions of these quantities above, there must be a δ quantity of size at least α/n which yields an adversary against IND-1-CCA with the same advantage. *q.e.d.*

The reduction from minivoting ballot privacy to multi-challenge IND-1-CCA is mostly a matter of renaming the oracles involved. The reduction operates as follows and is given in Figure 6.9. It maintains a single board B, initially empty, which it returns in response to `board` queries. It also maintains lists L and R to record all honest votes and S to record all dishonest ballots.

- The public setup information placed on the boards is the public key obtained from the IND-1-CCA challenger.
- `vote` queries become `challenge` queries: the reduction records the two votes, passes them to the challenge oracle and places the result on the single board B.
- To `tally`, the reduction uses its one "parallel" decryption query to decrypt all dishonest ballots. The reduction now has all votes and can compute the result function (based on the "left" votes) and place the result on the board.
- The adversary's guess at which board she is seeing becomes the reduction's guess at the challenge bit.

If the challenge bit b is 0, the adversary sees (through `board` queries) a board on which the reduction placed ballots of "left" votes, just like in the "left" ballot privacy game; ditto for $b = 1$ and "right" votes. The tally is still the left one in both cases.

By $\tau \leftarrow \rho(L, \sigma)$ in the tallying oracle we mean "compute the tally using the result function on input the honest votes (stored in list L) and the dishonest votes stored in σ". Since minivoting does not use auxiliary data, the tally is exactly the result.

Controlled malleability. The exact same theorem and proof that minivoting has ballot privacy hold if we replace non-malleable encryption with controlled-malleable encryption modulo an equivalence relation R as in Definition 3.11 and have the `Validate` algorithm reject not only

```
1  oracle initialise(pk)
2    L ← [ ], R ← [ ]
3    S ← [ ]
4    B ← [ ]
5    B ← B :: pk
6
7  oracle board
8    return B
9
10 oracle vote(id, v_L, v_R)
11   b ←challenge(v_L, v_R)
12   // correctness guarantees that
13   // this won't fail except with
14   // negl. prob.
15   if Validate(b, B) then
16       // record votes for later
17       L ← L :: v_L, R ← R :: v_R
18       B ← B :: b
19   end if
20 oracle ballot(s)
21   if Validate(s, B) then
22       S ← S :: s // record for later
23       B ← B :: s
24   end if
25
26 oracle tally
27   // invariant of minivoting: Valid-
28   // ate rejects exact duplicates,
29   // so none of these is the result
30   // of a challenge query.
31   σ ←decrypt(S)
32   // compute result now that we have
33   // all votes.
34   τ ← ρ(L, σ)
35   return τ
36
37 oracle finalise(g)
38   finalise(g) // to challenger
```

Figure 6.9: Reduction for ballot privacy of minivoting.

duplicates but ballots in relation R with any previous ones. For correctness, we have to make the further assumption that creating a fresh ballot with the `Vote` algorithm produces a ballot that with overwhelming probability is not in relation R with any previous ballot.

Theorem 6.10. Given an encryption scheme that is controlled-malleable with respect to an equivalence relation R let $\texttt{minivoting}(E, R)$ be the derived minivoting scheme with the modification that $\texttt{Validate}(s, bb)$ rejects any submission s in relation R with any previous ballot on the board bb. Then $\texttt{minivoting}(E, R)$ has ballot privacy.

The proof is identical to that of Theorem 6.7; the reason that the tally algorithm is allowed to decrypt all adversarial ballot is still that any ciphertext excluded from decryption would have been caught by the validation algorithm but the set of excluded ciphertexts now contains not just the challenge ciphertext but all ciphertexts in relation R with the challenge ciphertext too.

The obvious application we have in mind is minivoting with multi-ciphertext ballots. Suppose that there are two questions in a poll and we build a minivoting scheme where each ballot consists of a pair of ciphertexts from a non-malleable encryption scheme, one for each question. If we view this pair construction as a new encryption scheme (transforming a scheme for message space V into one for message space $V \times V$) then we unfortunately lose non-malleability: given a ciphertext $c = (c_1, c_2)$ one can always form $c' = (c_2, c_1)$. Applied to voting, this is one of the Cortier-Smyth attacks [CS11, CS13].

All is not lost however as non-malleability of the original scheme guarantees that we cannot do much more than permute ciphertexts, yielding a controlled-malleable scheme in which "pair" ciphertexts are related if they share any "small" ciphertexts. We formulate this as a theorem for vectors of any length.

Theorem 6.11. Let E be an encryption scheme with message, ciphertext and key spaces M, C, PK, SK. The scheme E^n with spaces M^n, C^n, PK, SK is defined as using the same key generation algorithm as E and encrypting/decrypting the vector of messages component-wise with the same key.

If E is non-malleable (IND-1-CCA) then E^n is non-malleable modulo relation R defined as

$$R(\vec{c}, \vec{c}') :\iff \exists i, j \leq n : c_i = c'_j$$

Proof. The proof is straightforward. Given an adversary against IND-1-CCA modulo R of E^n we reduce to IND-1-CCA for E. We can forward the public key from the challenger to the adversary. Since our adversary submits two vectors of challenge messages, we switch to an IND-1-CCA game with n challenges for a loss of a factor n in soundness as in Lemma 6.8. When the adversary makes her parallel decryption query, it will contain some number m of ciphertext-vectors each with n elements which we can treat as a total of $m \cdot n$ individual E-ciphertexts. Relation R guarantees that none of these ciphertexts will match one of the challenges so we can simply make a parallel decryption query on all these ciphertexts to our challenger and return the result to the adversary. Finally, we forward the adversary's guess to our challenger to win the IND-1-CCA game with an advantage of α/n where n is the length of ciphertext vectors in scheme E^n and α was the adversary's advantage against the IND-1-CCA modulo R game. *q.e.d.*

6.4 Helios

Helios [A08] is a cryptographic voting scheme developed principally by Adida. In this thesis we mainly use "Helios" to mean version 3 which is cryptographically identical to version 2 and uses homomorphic tallying. Helios version 1 used a mix-net, version 4 under development may offer both tallying options.

Helios has been used to elect the student government at the university of Princeton [HP] in 2009, the board of the IACR since 2010 [HeIACR] and the president of the Catholic University of Louvain [AMPQ09] in 2009.

In 2011, Cortier and Smyth [CS11] published a paper questioning whether Helios satisfies ballot privacy. (We use the term "ballot privacy" for the property in question. It is sometimes also called "ballot secrecy" in the literature, for example in the title of the aforementioned paper.) During a formal analysis of Helios ballots, they discovered that one could in some cases resubmit ballots, make minor irrelevant tweaks i.e. to whitespace then resubmit or permute individual ciphertexts in ballots to create a ballot for a different choice (in a yes/no referendum this creates a ballot for the opposite choice to the original one). Cortier and Smyth further showed how the ability to copy ballots can be detrimental to privacy and gave a case study for French legislative

elections. The basic argument is as follows. Consider three voters, Alice, Bertha and Carol. Carol votes last and copies Alice's ballot as her own: she does not mind who gets elected, but would like to know how Alice voted. When the election result is announced, whoever got two votes was Alice's choice; moreover Bertha's vote is immediately revealed to Carol too.

We have analysed the security of "idealised" versions of Helios in several papers and given arguments that it satisfies privacy if the encryption scheme used is good enough: first [BC+11] we argued security if IND-CCA2 secure encryption were deployed and suggested how this could be achieved; next [BPW12a] we weakened this restriction to NM-CPA, which we thought that Helios achieved at the time. Unfortunately, we later discovered [BPW12b] a flaw in the encryption scheme used (the weak Fiat-Shamir transformation, Section 5.1.1 of this thesis) meaning that it in fact achieves only IND-CPA security and does not satisfy our notions of ballot privacy. This flaw is easy to overcome and a fix will appear in the future version 4 of Helios.

Theorem 6.12 in this thesis proves that Helios has ballot privacy assuming our fix has been deployed and supersedes the ballot privacy theorems in the cited papers. This thesis is also the first work to prove ballot privacy for "full" Helios ballots that may contain any number of vote-ciphertexts.

6.5 Overview of Helios

Helios consists of several components. All components, protocols and data structures are open source and freely available through the website http://heliosvoting.org. Helios is licenced under the GPL version 3.

The server allows one to set up an election and create credentials for voters and keys for election officials, as well as functionality to open, close and tally an election. The server also maintains a bulletin board. The client, written in JavaScript, allows voters to cast a vote and encrypt it to create a ballot in their browser.

The client employs a "Benaloh challenge": after seeing their ballot but before casting it, voters can opt to save a copy and ask the client to reveal the randomness used, with which they can verify integrity of the ballot using an external program if they wish. If the voter chooses to reveal the randomness in a ballot, the client refuses to submit it and asks the voter to create another ballot. In principle, it is possible to write a custom Helios client and use it to cast ballots.

The verifier is a web application into which voters can paste bulletin boards of completed elections and verify them; several people not on the Helios development team have written external verifiers.

While Helios allows voters to audit the board and check for presence of their ballot and anyone to audit an election, procedures for raising and handling complaints if any of these checks fail are up to election officials for individual elections. Similarly, authentication and eligibility issues are matters for the administrators of an election to address. Helios has its own username/-password authentication option built in but is primarily designed to operate with any external authentication mechanism. In Leuven for example [AMPQ09], authentication was handled by the university's single sign-on system and upon casting a ballot, voters received a digitally signed PDF file containing their ballot that they could present to election officials in case of a dispute.

Technical details. Cryptography in Helios uses a group $\mathbb{G}$, defined in version 3 by parameters (G, p, q) as the q-order subgroup of $\mathbb{Z}_p^*$ with generator G, where p, q are primes. New parameters can be chosen for each election. Data structures exchanged between components are in the JSON (JavaScript Object Notation) format. This includes election specifications, keys, ballots and bulletin boards.

All administrators create their own ElGamal keypair and combine public keys by group addition to create an n-out-of-n shared election key. In addition, administrators prove knowledge of their respective secret key shares. The election public key along with the administrators' individual public keys and proofs of knowledge of secret keys are published in the election specifications. Generalisations to threshold schemes may be implemented in future versions of Helios.

6.6 Helios ballots

For simplicity, we consider an election or poll with only one question. Helios allows multiple questions in a poll, in which case a ballot contains one of the structures mentioned below per question.

Ballot structure. A Helios ballot consists of a number of ciphertext/proof pairs and possibly an extra overall proof. To aid homomorphic tallying, each ciphertext is an encryption of either 0 or 1 and there are as many ciphertexts as choices, so a yes/no question would contain two ciphertexts per ballot. Each ciphertext is accompanied by an individual proof that it indeed encodes 0 or 1.

The overall proof, if required, shows that the sum of all votes in the ciphertexts is in a given range. Typical examples of this range are

- $[0, 1]$: voters may choose at most one choice or abstain.
- $[1, 1]$: voters must pick exactly one choice. ("I abstain" may be one of the choices).

- No overall proof: for example, in approval voting.

Technical details. The encryption scheme used in Helios is ElGamal with the vote in the exponent (i.e. one encrypts G^v where G is the group generator) to achieve additive homomorphism. The individual and overall proofs are disjunctive Chaum-Pedersen proofs made non-interactive with the Fiat-Shamir transformation. A ciphertext/individual proof pair can be analysed as an Encrypt+PoK construction (Section 5.2). The overall proof, if present, applies to the homomorphic sum of all ciphertexts; this sum can be recomputed from the individual ciphertexts and so is not included in the ballots.

Ballot encoding and hashing. Helios ballots are encoded as JavaScript Object Notation (JSON) objects, containing lists of key/value pairs where the values are encodings of the group elements forming the ciphertexts, hashes and proofs. When we say ballots are encryptions of votes, we are being a bit imprecise - ballots are JSON objects that contain ciphertexts for votes as some of their components.

Helios creates hashes of all ballots it receives and stores these along with the ballots themselves. Each voter is given (for example by e-mail) the hash of her ballot after she has voted; she can check that her vote has been counted, and not tampered with, by looking for this hash in the transcript of the tallying process which is posted to the board at the end of an election. This process prevents Helios from accepting exact duplicates of ballots. However, one could still try and modify the JSON of a ballot in ways that do not affect the content to obtain an equivalent ballot, for example by adding whitespace or reordering the keys. Helios prevents this by normalising the JSON representation before hashing a ballot.

6.7 The Cortier-Smyth attacks

Cortier and Smyth [CS11, CS13] found a number of ways in which Helios ballots could be mauled, allowing a dishonest voter to submit a ballot related to an earlier one on the board. The creates the following privacy problem. Suppose Alice and Bertha have voted, and Eve submits a copy of Alice's ballot as her own. The result is announced as "2 yes, 1 no". Eve now knows exactly how everyone voted. Contrast this with the situation where Eve casts a "yes" ballot and obtains the same result: she now cannot tell whether it was Alice or Bertha that cast the other "yes". Cortier and Smyth argued that Helios was vulnerable to privacy breaches in realistic settings, by analysing French legislative elections and estimating the cost to expose an individual voter by bribing other voters to submit related ballots.

We sketch some of the Cortier-Smyth attacks [CS11, CS13]. Most of these can be traced to the use of the weak Fiat-Shamir transformation. All of these attacks are detectable as we will soon argue; previous versions of Helios did not prevent such ballots from being cast however.

- If a ballot contains multiple ciphertexts, one can permute these ciphertexts along with their individual proofs within a ballot. This may result in a ballot for a different vote to the original but with the relation between the two known to the adversary casting this ballot. Overall proofs do not prevent this attack as they check only the sum of ciphertexts, which is invariant under permutation.

- A bug in modular arithmetic allowed the same value to be represented in multiple ways in some versions of Helios. In detail, Helios used an order-p cyclic group represented as a subgroup of $\mathbb{Z}_q^*$ where p, q are primes with $p = (q-1)/2$. Some values in the exponent group $\mathbb{Z}_p$ were only normalised modulo q instead of modulo p, allowing an attacker to add p to such a value to create a different representation with the same functionality.

- The pair $(1, 1)$ is a valid ElGamal ciphertext for $m = 0$ with randomness $r = 0$ for any generator G and public key Y in Helios. Instead of permuting ciphertexts, an adversary can select a particular individual ciphertext/proof pair from a previous ballot and fill up the remaining positions with such dummy ciphertexts and corresponding proofs; if the overall proof is for the interval $[0, 1]$ the attacker can now reuse copied individual proof as the overall proof in her ballot since all other ciphertexts contain the group's neutral element 1 which contributes nothing to the sum. The attacker also choose less "obvious" ciphertexts for the remaining positions by ensuring that their sum cancels out.

- One can rerandomise ciphertexts in ballots and their associated weak proofs, keeping only the hash of the weak Fiat-Shamir proof constant. Some care needs to be taken not to jeopardise overall proofs but this too can be accomplished by making sure that all randomisers added to a ballot cancel out modulo the group order.

6.8 Verifying ballots

To check a ballot, one checks that each ciphertext is accompanied by a valid individual proof; if required by the election specification, one also checks the overall proofs. To check for duplicates, it is not sufficient to check for exact copies since Cortier and Smyth [CS11, CS13] and ourselves

[BPW12b] have discovered various ways in which ballots are malleable but all of these are detectable assuming the strong Fiat-Shamir transformation has been used (the same checks also catch most reused-ballot attacks in existing Helios elections using the weak transformation).

Based on our latest work [BPW12b] we recommend extracting all the hash values used in Fiat-Shamir transformed proofs and rejecting any ballot where any of the hash values has been used earlier, even if in a different context (for example, a hash from a previous individual proof now appearing in an overall proof). This detects all known ballot-copying attacks although it does not prove the absence of as yet unknown ones. Further, one should check for group neutral elements or zero values (which are sometimes not even valid group elements) where random group elements would be expected in ciphertexts or proofs. As always when working with representations of cryptographic groups, it is essential to check that every value is in fact a representation of a valid element (i.e. zero is not an element of $\mathbb{Z}_q^*$, but can still be used in places for certain ballot-copying techniques). In an earlier work [BC+11] we also discovered that one of Cortier and Smyth's techniques hinted at a bug in Helios (some elements were only reduced mod q when they should be mod p, leading to multiple representations). This attack will already be caught by the hash-check however.

With these "low-level" checks in place, it is not necessary to normalise ballots' JSON representation and compare hashes of entire ballots, as previously implemented in Helios. Comparing such entire-ballot hashes was never sufficient to detect all Cortier-Smyth attacks.

6.9 Tallying in Helios

Before election officials tally an election, they must check that all ballots are well-formed and there are no duplicates on the board. This is required for our ballot privacy proof to apply. The board should also perform these checks itself and reject ballots not meeting these criteria but election officials should not trust the board in this matter.

The actual tallying is simply homomorphic addition of ciphertexts for each option and thus publicly verifiable. The final ciphertexts are jointly decrypted by the administrators, each one performing a partial decryption and a proof that their partial decryption is correct with respect to the secret key for their public key share (which is published in the election specification). All decryption shares can be publicly combined to obtain, for each option, the value G^r where r is the result (number of votes cast) for this option. This is an artefact of the use of ElGamal "in the exponent". Helios then computes a table of values $G^0, G^1, \ldots G^n$ where n is the number of voters and recovers r.

Denial of service. One attack on Helios we found recently [BPW12b] allows voters to submit a malformed ciphertext for one option containing a random value r in $[0, q-1]$, causing the intermediate result G^r for this option to be a random group element. Helios at the time of writing did not detect such ciphertexts, the table lookup of G^r failed with a null value leading Helios to display `NONE` as the option's tally (NONE is the python language's word for "null value"). While this attack does not cause a false result to be obtained and it is obvious when this attack has taken place, it does disrupt an election until the officials know how to handle it. Our proposed checks for unexpected zero/neutral values in ballots catch this attack: the malformed ballots involved are easy enough to spot. However, to make our security arguments apply it is important that these ballots are checked for and eliminated before the election officials perform a decryption with their secret keys, rather than having to decrypt the sum of the remaining ballots a second time.

6.10 Verifying an election

To verify a completed election, one should obtain the bulletin board and check all the following properties. If the verifier is a voter, she can additionally check that her ballot appears on the board.

1. The election specification corresponds to the correct format and all the proofs of knowledge of secret key shares are valid with respect to the public key shares contained in the specification, further the election public key is the sum of the public key shares.

2. Each ballot is well-formed and all the contained proofs verify and are of the format decreed by the election specification. No neutral or zero values appear where random group elements or exponents are expected.

3. No hash value used in a Fiat-Shamir transformation is ever reused between ballots or within a ballot.

4. The values on which the partial decryptions are taken are the homomorphic sums of the ciphertexts in the ballots for the corresponding options and the proofs associated to partial decryptions all verify. Finally, the decrypted intermediate results (G^r) are the group values obtained by raising the claimed result (r) to the group generator in the election specification.

6.11 Ballot privacy in Helios

In this section we state and prove our main theorem concerning Helios.

Theorem 6.12. Helios (the version described in this thesis using the strong Fiat-Shamir transformation) satisfies ballot privacy according to Definition 6.5 under the DDH assumption.

Proof. The proof is essentially that of the minivoting construction taking into account the layer of proofs that Helios adds for verifiability. Under the DDH assumption, signed ElGamal from Definition 5.19 is NM-CPA by Theorem 5.21. The individual proofs in Helios ballots are disjunctive Chaum-Pedersen proofs since they must additionally certify that the encrypted vote is 0 or 1. These proofs are still SSE: we proved this for our sigma protocol template which covers Chaum-Pedersen proofs in Theorem 5.14; the same holds for disjunctive proofs.

Therefore, the encryption scheme defined by an ElGamal ciphertext coupled with an individual proof in Helios is non-malleable. For Helios ballots without overall proofs, Theorem 6.11 shows that Helios ballots are controlled-malleable (viewing ballot creation as a form of encryption) where two ballots are considered related if they share any individual encrypted vote, i.e. a pair consisting of an ElGamal ciphertext and its individual proof. We proposed to weed for repeated ballots by comparing hash values in the Fiat-Shamir proofs. This technique will certainly catch repeated encrypted votes.

To deal with overall proofs, we note that these proofs are made on a ciphertext that is the homomorphic sum of all other ciphertexts in a ballot yet is not included in the ballot for space reasons. If we added these sum-ciphertexts back into ballots, we would see that we could treat them and their associated overall proofs as just another ciphertext/proof pair and weed repeated ballots as before to obtain controlled malleability. (The underlying "encryption scheme" might be slightly different as overall proofs can have a different range than $[0, 1]$ but this does not affect non-malleability.) We have proposed to weed for repeated ballots by looking for repeated Fiat-Shamir hash values in both the individual and overall proofs. This catches repeated ciphertexts whether the sum-ciphertexts are contained in a ballot or not so we do not need to include them. Although Helios tallies contain further proofs in addition to the result, the ballot privacy game is not concerned with these. Formally, we could perform a Helios tally of the left board and extract the result, then return this to the adversary in the `Tally` oracle. *q.e.d.*

Threshold privacy. The ballot privacy game in our presentation in this thesis manages the entire tallying process which could be seen as the case where all election officials are honest. Helios however guarantees privacy as long as at least one official (that holds a decryption key share) is honest and could easily be generalised to any threshold. We only sketch the security argument as it is orthogonal to the main purpose of this thesis.

We can set up a variant of the ballot privacy game where the game plays the honest official(s) and the adversary all others — we assume that unlike voters, officials cannot be corrupted adaptively but instead their honesty or lack thereof is fixed from the outset. We also assume that the game has black-box access to the dishonest decryptors. The setup and tallying phases will now consist of interactive protocols between the game and the dishonest officials.

In the setup phase, since all Helios decryptors must create a (strong Fiat-Shamir) proof of knowledge of their secret key share, the game can rewind all dishonest decryptors and obtain their key shares. In the tallying phase the game can now obtain the result as in the non-threshold proof and since at least one decryptor is honest, the game can compute a decryption share for this decryptor that gives the desired result (in particular the "left" result on the "right" board); the proof of correctness this share is simulated as before. The dishonest decryptors cannot simulate the proofs of their decryption shares however so any cheating here would immediately give an attack on soundness of the proofs involved.

7 Conclusion

We have studied the theory of zero-knowledge proof schemes and some practical applications in the Helios voting system. Our first contribution was to identify weak proofs (Definition 5.1) and problems that arise from their use. Next, we formalised strong proofs (Definition 5.5), showed how they could be obtained (Theorem 5.8) and proved that they really do strengthen encryption when deployed in the common Encrypt+PoK construction (Theorem 5.21).

In 1998, Shoup and Gennaro [SG98] discovered an apparent limitation of sigma protocol-based proof schemes: an "obvious" proof of chosen-ciphertext security for a scheme known as Signed ElGamal (our Definition 5.19) breaks down when one tries to write it out in detail. Shoup and Gennaro were unable to either restore the proof by other means nor to prove that this construction fails to achieve the desired notion, leaving the problem as an open question. In this thesis, we have answered this question negatively. We have introduced a notion of multi-proofs (Definitions 5.28,5.30) and shown that the "obvious" proof of chosen-ciphertext security does hold in the presence of simulation sound multi-proofs (Theorem 5.40). The proof is anything but obvious however: it is to our knowledge the first proof of chosen-ciphertext security using schemes that rely on a technique known as rewinding, the very technique that caused problems for Shoup and Gennaro. To complete our work on the theory of zero-knowledge proofs, we have shown two separation results. First, we have proved that sigma-protocol based schemes (whether weak or strong) cannot be multi-proofs (Theorem 5.32). We have introduced an assumption that we call IES (Assumption 5.45), failure of which would raise serious questions about the security of the Signed ElGamal scheme. Under this assumption, our second separation theorem (Theorem 5.50) is that Signed ElGamal cannot be proven chosen-ciphertext secure, answering the open question to our satisfaction.

We close by taking a step back to observe the big picture and a step forward to consider possible future work. Looking at the big picture of "provable security", we believe that there are several attributes of our field of research in need of improvement. We have a jungle of methods (game-based and simulation-based security, Dolev-Yao etc.), models (CRS, random oracle etc.) and assumptions (discrete logarithms, Diffie-Hellman etc.). Particularly when dealing with zero-

knowledge schemes, unified definitions are still lacking that capture the essence of the notion at hand but are as far as possible agnostic of a particular model. At the same time, the tedious low-level work that the different models and methods require can tempt cryptographers to gloss over steps in proofs or settle for "proof sketches", increasing the chance of mistakes.

We have started to search for a solution to these issues in this thesis and will continue on this path in the coming years. We have, for example, presented our definitions of a strong proof (Definition 5.5) and multi-proof (Definition 5.28) first in a more model-independent manner and then, for the rewinding random oracle model, spelled them out in detail using code-based games. Our hope in future work is to formulate the models in such a way that the model-specific definitions become functions of the generic definition and a model description.

To lessen the chance of mistakes in proofs, we have high hopes for automated proof-verification techniques and hope to study such techniques in the near future and if feasible, extend them to cover the applications that we require. A key problem here will be how to deal with rewinding, which turns a two-party system into a multi-party one.

Another problem that appears regularly concerns addressing conventions in (token-based) concurrent systems. Some models based on universal composability introduce elaborate conventions here. We appreciate their precision but would hope for a method to eliminate as many such details as possible — without sacrificing rigour, of course — especially when these addressing conventions in the security argument do not have a counterpart in the real scheme under consideration. Considering the amount of research effort already spent in this area however, we do not have any hope of a quick solution.

Finally, while we have established not only the notion of multi-proofs but also found a scheme (by Fischlin [F05]) that meets this notion, it is still an open question whether there exist useful multi-proofs with some amount of inherent rewinding (the Fischlin scheme is "straight line"), or whether Fiat-Shamir-Schnorr could be sensibly extended to become a multi-proof. Since we know that multi-proofs can be used to yield CCA encryption, we have also asked ourselves whether existing CCA schemes (Naor-Yung [NY90], TDH2 [SG98] and Chaum-Pedersen Signed ElGamal [ST13]) can be viewed as Encrypt+multi-proof instances. At a first glance, this seems to be the case but the statements being proved in this case depend on extra keys and so are not in $\mathbf{NP}$. This breaks our current theory since a reduction that verifies such proofs would not be efficient. We hope to extend the theory to be able to handle these cases and uncover the "essence" of chosen-ciphertext security in these schemes.

Bibliography

[A06] B. Adida. Advances in Cryptographic Voting Systems. PhD thesis, MIT August 2006.

[A08] B. Adida. Helios: Web-based open-audit voting. In: 17th USENIX security symposium, Pages 335-348, 2008. Helios website: http://heliosvoting.org paper: http://www.usenix.org/events/sec08/tech/full_papers/adida/adida.pdf

[ACM] Association of Computing Machinery. 2012 Turing award statement, at http://www.acm.org/press-room/awards/turing-award-12.

[AF01] M. Abadi and C. Fournet. Mobile values, new names and secure communication. In: ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages (POPL '01), pages 104–115, ACM Press 2001.

[AMPQ09] B. Adida, O. de Marneffe, O. Pereira and J.-J. Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of Helios. In: Proceedings of the 2009 conference on Electronic voting technology/workshop on trustworthy elections.

[AR00] M. Abadi and P. Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). In: Journal of Cryptology, volume 15 no. 2, pages 103–127, 2002.

[B09] D. Bernhard. Zero-Knowledge Interactive Proofs. Master Thesis, Department of Mathematics, ETH Zürich.

[BBS04] D. Boneh, X. Boyen and H. Schacham. Short group signatures. In: Advances in Cryptology — CRYPTO '04, LNCS 3152, pages 41–55, 2004.

[BCJ08] A. Bagherzandi, J. H. Cheaon and S. Jarecki. Multisignatures Secure under the Discrete Logarithm Assumption and a Generalized Forking Lemma. In: CCS '08, pages 449–458, ACM press 2008.

[BC+11] D. Bernhard, V. Cortier, O. Pereira, B. Smyth and B. Warinschi. Adapting Helios for Provable Ballot Secrecy. In: ESORICS '11.

[BCPW12] D. Bernhard, V. Cortier, O. Pereira and B. Warinschi. Measuring Vote Privacy, Revisited. In: Proceedings of ACM CCS '12, pages 941–952, 2012.

[BDPR98] M. Bellare, A. Desai, D. Pointcheval and P. Rogaway. Encryption Schemes. Relations Among Notions of Security for Public-Key In: Advances in Cryptology — CRYPTO '98, LNCS 1462, pages 26–45, 1998.

[BFG13] D. Bernhard, G. Fuchsbauer and E. Ghadafi. Efficient Signatures of Knowledge and DAA in the Standard Model. In: ACNS '13, LNCS 7954, pages 518–533, 2013.

[BF+11] D. Bernhard, G. Fuchsbauer, E. Ghadafi, N. P. Smart and B. Warinschi. Anonymous attestation with user-controlled linkability. In: Int. Journal of Information Security, pages 1–31, 2011.

[BFM88] M. Blum, P. Feldman and S. Micali. Non-Interactive Zero-Knowledge and its Applications. In: Proceedings of the twentieth annual ACM symposium on computing (STOC '88), pages 103–112, 1988.

[BG92] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. In: Advances in Cryptology — CRYPTO' 92, LNCS 740, pages 390–420, 1992.

[BGP11] P. Bulens, D. Giry and O. Pereira. Running mixnet-based elections with Helios. In: Electronic Voting Technology Workshop/Workshop on Trustworthy Elections, Usenix 2011.

[BN06] M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In: Proceedings of ACM Conference on Computer and Communications Security, pages 390–399, 2006.

[BNV13] D. Bernhard, S. Neumann and M. Volkamer. Towards a Practical Cryptographic Voting Scheme Based on Malleable Proofs. In: E-Voting and Identity (Vote-ID 2013), LNCS 7985, pages 176–192, 2013.

[BPW12a] D. Bernhard, O. Pereira and B. Warinschi. On Necessary and Sufficient Conditions for Private Ballot Submission. Eprint, eprint.iacr.org/2012/236

[BPW12b] D. Bernhard, O. Pereira and B. Warinschi. How Not to Prove Yourself: Pitfalls of Fiat-Shamir and Applications to Helios. In: Advances in Cryptology — Asiacrypt '12, LNCS 7658, pages 626–643, 2012.

[BR93] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: Proceedings of the 1st ACM conference on Computer and communications security (CCS '93), pages 62–73, 1993.

[BR06] M. Bellare and P. Rogaway. Code-based game-playing proofs and the security of triple encryption. In: Advances in Cryptology — Eurocrypt '06, LNCS 4004, pages 409–426, 2006. The title cited is from the latest version on eprint at http://eprint.iacr.org/2004/331.

[BS99] M. Bellare and A. Sahai. Non-Malleable Encryption: Equivalence between Two Notions, and an Indisinguishability-Based Characterization. In: Advances in Cryptology — CRYPTO '99, LNCS 1666, pages 519–536, 1999.

[BS13] D. Bernhard and B. Smyth. Ballot secrecy and ballot independence coincide. In: Computer Security — ESORICS '13, LNCS 8134, pages 463–480, 2013.

[C71] S. A. Cook. The complexity of theorem proving procedures. In: STOC '71, pages 151–158, 1971.

[C81] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. In: Communications of the ACM vol. 24.2, pages 84–88, 1981.

[C96] R. Cramer. Modular Design of Secure yet Practical Cryptographic Protocols. PhD thesis, University of Amsterdam, 1996.

[C98] J. Camenisch. Group signature schemes and payment systems based on the discrete logarithm problem. PhD thesis, ETH Zürich, 1998.

[C01] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In: FOCS '01, pages 136–145, 2001. Revised version of December 2005 on eprint, eprint.iacr.org/2000/067.

[CA07] Secretary of State of California D. Bowen. Top-to-bottom review of voting systems certifies for use in California. Multiple documents, available at http://www.sos.ca.gov/voting-systems/oversight/top-to-bottom-review.htm.

[CKLM12] M. Chase, M. Kohlweiss, A. Lysyanskaya and S. Meiklejohn. Malleable Proof Systems and Applications. In: Eurocrypt '12, LNCS 7237, pages 281–300, 2012.

[CKN03] R. Canetti, H. Krawczyk and J. B. Nielsen. Relaxing chosen-ciphertext security. In: CRYPTO '03, LNCS 2729, pages 565–582, 2003.

[CKY09] J. Camenisch, A. Kiayias and M. Yung. On the portability of generalized schnorr proofs. In: Advances in Cryptology — Eurocrypt '09, LNCS 5479, pages 425–442, 2009.

[Coq] The Coq proof assistant, http://coq.inria.fr/.

[CP92] D. Chaum and T. P. Pedersen. Wallet Databases with Observers. In: Advances in Cryptology — CRYPTO' 92, LNCS 740, pages 89–105, 1992.

[CS97b] J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. Technical report TR 260, Institute for theoretical computer science, ETH Zürich, 1997.

[CS97c] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In: Advances in Cryptology — CRYPTO '97, pages 410–424, 1997.

[CS08] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Advances in Cryptology — CRYPTO '98, pages 13–25, 2008.

[CS11] V. Cortier and B. Smyth. Attacking and Fixing Helios: An Analysis of Ballot Secrecy. Eprint, report 2010/625.

[CS13] V. Cortier and B. Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. In: Journal of Computer Security, volume 21(1), pages 89–148, 2013.

[DDN91] D. Dolev, C. Dwork and M. Naor. Non-malleable cryptography. In: Proc. of the 23rd ACM symposium on theory of computing, pages 542–552, 1991.

[DF89] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In: Advances in Cryptology — CRYPTO '89, LNCS 435, pages 307–315, 1989.

[DH76] W. Diffie and M. Hellman. New Directions in Cryptography. In: IEEE Transactions on Information Theory, vol. 22, no. 6, pages 644–654, 1976.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. In: IEEE transactions on information theory, vol. 29, pages 198–208, 1983.

[E85] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In: IEEE transactions on information theory, Pages 469-472, Volume 31, 1985.

[EC] Easycrypt toolkit, www.easycrypt.info

[F05] M. Fischlin. Communication-Efficient Non-Interactive Proofs of Knowledge with Online Extractors. In: Advances in Cryptology — CRYPTO '05, pages 152–168, 2005.

[FFS88] U. Feige, A. Fiat and A. Shamir. Zero-knowledge proofs of identity. In: Journal of Cryptology, vol. 1.2, pages 77–94, 1988.

[FP01] P.-A. Fouque and D. Pointcheval. Threshold cryptosystems secure against chosen-ciphertext attacks. In: Advances in Cryptology — Asiacrypt '01, LNCS 2248, pages 351–358, 2001.

[FS86] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In: Advances in Cryptology — CRYPTO '86, pages 186–194, 1986.

[FS90] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In: Proc. of the 22nd ACM symposium on computing, pages 416–426, 1990.

[G04] J. Groth. Evaluating Security of Voting Schemes in the Universal Composability Framework. In: Applied Cryptography and Network Security (ACNS '04), pages 46–60, 2004.

[G06] J. Groth. Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures. In: Advances in Cryptology — Asiacrypt '06, LNCS 4284, pages 444–459, 2006.

[GJKR99] R. Gennaro, S. Jarecki, H. Krawczyk and T. Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In: Advances in Cryptology — Eurocrypt '99, LNCS 1592, pages 295–310, 1999.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. In: Journal of computer and system sciences, vol. 28.2, pages 270–299, 1984.

[GMR85] S. Goldwasser, S. Micali and C. Rackoff. The knowledge complexity of interactive proof systems. In: 17th ACM symposium on Theory of Computation, 1985 (citation according to [FS86]).

[GMR89] S. Goldwasser, S. Micali and C. Rackoff. The knowledge complexity of interactive proof systems. In: SIAM Journal on computing, vol. 18.1, pages 186–208, 1989.

[GS08] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In: Advances in Cryptology — Eurocrypt '08, LNCS 4965, pages 415–432, 2008.

[GMW91] O. Goldreich, S. Micali and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. In: Jounral of the ACM, vol. 38.3, pages 690–728, 1991.

[H05] S. Halevi. A plausible approach to computer-aided cryptographic proofs. Eprint, report 2005/181, eprint.iacr.org/2005/181.

[HeIACR] International association for cryptologic research. Election page at http://www.iacr.org/elections/2010

[HK07] D. Hofheinz and E. Kiltz. Secure Hybrid Encryption from Weakened Key Encapsulation. In: Advances in Cryptology — CRYPTO '07, LNCS 4622, pages 553–571, 2007.

[HP] Helios Headquarters, Princeton University Undergraduate Student Government. http://usg.princeton.edu/officers/elections-center/helios-headquarters.html

[JCJ05] A. Juels, D. Catalano and M. Jakobsson. Coercion-Resistant Electronic Elections. In: Proceedings of the 4th Workshop on Privacy in the Electronic Society (WPES'05), pages 61–70, 2005.

[K72] R. M. Karp. Reducibility among combinatorial problems. In: Complexity of Computer Computations, pages 85–103, 1972.

[L03] Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. In: Journal of Cryptology, volume 16.3, pages 143–184, 2003.

[L05] L. Landes. Scrap the secret ballot. Version of November 2005 as posted on http://www.dissidentvoice.org/Nov05/Landes1105.htm, author's page at http://www.lynnlandes.com/.

[L07] L. Landes. The Landes report to Congress. At http://www.thelandesreport.com/ToCongress1.htm

[NY90] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In: Proceedings of the twenty-second annual ACM symposium on theory of computing (STOC '90), pages 42–437, 1990.

[P91] T. Pedersen. Non-Interactive and Information-Theoretic Secure Verfiable Secret Sharing. In: Advances in Cryptology — CRYPTO '91, LNCS 576, pages 129–140, 1991.

[PR07] M. Prabhakaran and M. Rosulek. Rerandomisable RCCA encryption. In: Advances in Cryptology — CRYPTO '07, LNCS 4622, pages 517–534, 2007.

[RS91] C. Rackoff and D. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In: Advances in Cryptology — CRYPTO '91, LNCS 576, pages 433–444, 1991.

[RS11] M. D. Ryan and B. Smyth. Applied pi calculus. Tutorial, available at http://www.bensmyth.com/publications/2011-Applied-pi-calculus/

[RSA78] R. Rivest, A. Shamir and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. In: Communications of the ACM vol. 21.2, pages 120–126, 1978.

[S91] C. P. Schnorr. Efficient signature generation for smart cards. In: Journal of Cryptology, Volume 4, Pages 161-174, 1991.

[S99] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: Proceedings of th 40th annual symposium on foundations of computer science (FOCS '99), pages 543–553, 1999.

[S01] V. Shoup. A proposal for an ISO standard for public-key encryption (v2.1). Eprint 2001/112, https://eprint.iacr.org/2001/112.

[S05] B. Snow. We Need Assurance! Distinguished Practicioner address at IEEE Computer Security Applications Conference (ACSSC '05), available at www.acsac.org/2005/papers/Snow.pdf.

[Sch01] B. Schneier. Internet Voting vs. Large-Value e-Commerce. Online journal entry at: https://www.schneier.com/crypto-gram-0102.html#10

[SG98] V. Shoup and R. Gennaro. Securing Threshold Cryptosystems Against Chosen-Ciphertext Attack. In: Advances in Cryptology — Eurocrypt '98, LNCS 1403, pages 1–16, 1998.

[SJ00] C.P. Schnorr and M. Jakobsson. Security of Signed ElGamal Encryption. In: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology — ASIACRYPT '00, pages 73–89, 2000.

[ST13] Y. Seurin and J. Treger. A robust and plaintext-aware variant of signed elgamal encryption. In: Topics in Cryptology — CT-RSA '13, LNCS 7779, pages 68-83, 2013.

[TY98] Y. Tsiounis and M. Yung. On the security of ElGamal-based encryption. In: International Workshop on Practice and Theory in Public Key Cryptography (PKC '98), pages 117–134, 1998.

[W08] D. Wikström. Simplified Submission of Inputs to Protocols. In: Security and Cryptography for Networks, 6th International Conference, SCN 2008, pages 293–308, 2008.

[W] The wombat voting system, www.wombat-voting.com