Adaptive Proofs of Knowledge in the Random Oracle Model

David Bernhard[1], Marc Fischlin[2], Bogdan Warinschi[1]

[1]Dept. of Computer Science, University of Bristol, Woodland Road, Bristol, U.K.
[2]Dept. of Computer Science, Technische Universität Darmstadt, Darmstadt, Germany

We define a notion of adaptive proofs of knowledge (PoKs) in the Random Oracle Model (ROM). These are proofs where the malicious prover can adaptively issue multiple statements and proofs, and where the extractor is supposed to extract a witness for each statement. We begin by studying the traditional notion of zero-knowledge PoKs in the ROM and then show how to extend it to the case of adaptive adversaries and to simulation soundness, where the adversary can also learn simulated proofs.

Our first main result is negative. Under common assumptions we can show that the well-known Fiat-Shamir-Schnorr proof system is *not* adaptively secure. As for the second result we prove that an existing construction due to Fischlin (Crypto 2005) yields adaptively secure simulation-sound PoKs in the ROM. Since the purpose of this work is to motivate and introduce adaptive proofs, we only briefly discuss some applications to other areas, for example that adaptive proofs seem to be exactly what one requires to construct CCA-secure public-key encryption from IND-CPA secure schemes.

1. Introduction

Proofs of knowledge [1, 2, 3, 4] are a generic tool to ensure correct operation in many cryptographic constructions, including voting protocols, e-cash systems, or group signatures. More generally, they can turn passively secure multi-party protocols into actively secure ones. The value of proofs of knowledge in security arguments is that whenever a participant makes a proof of knowledge on some statement as part of a protocol, one can "hop" into an alternate, virtual world in which the participant outputs the witness along with the statement. This approach of pretending that each proof makes its witness available in a security argument relies on the extractor that exists by definition of a proof of knowledge: when a participant outputs a proof, we "freeze" the protocol, and invoke the extractor to get the witness. This extraction is usually carried out by rewinding the party and branching into another protocol execution. Then we resume the protocol with the witness now being available.

The problem with the "freeze-extract-resume" approach is that its implementation can easily become expensive. Each extraction and its rewinding can double the running time of a reduction such that, if a participant makes a badly nested "chain" of n proofs, a naive approach ends up with an exponential running time of 2^n to get all the witnesses. This is certainly true for interactive proofs of knowledge, but also in the case of non-interactive proofs of knowledge in the random or-

acle model. Such random-oracle based proofs are paramount if efficiency is vital, especially in the form of Fiat-Shamir transformed Sigma protocols a.k.a. "Schnorr-type" proofs. In this context the rewinding problem was first mentioned explicitly by Shoup and Gennaro [5], who required nested proofs in the random oracle model for the construction of CCA-secure public-key encryption from IND-CPA secure encryption via the encrypt-then-prove approach (e.g., for signed ElGamal). In this case, since the adversary against the CCA-secure version may ask for decryptions of adaptively chosen ciphertexts, one requires the notion of adaptive PoKs in order to be able to simulate decryption queries. In contrast to the non-rewinding solutions based on common reference strings, the more efficient PoKs in the ROM do not seem to concord well with this case. In fact, it is currently unclear which adaptivity notions are appropriate for such scenarios.

1.1. Our Contributions

A DEFINITION OF ADAPTIVE PROOFS OF KNOWLEDGE. In this work we define adaptive proofs of knowledge in the random oracle model (ROM). We first study the traditional notions of proofs of knowledge (PoKs) and simulation-sound PoKs (ss-PoKs) and formalise them in a way that reflects their use in security proofs. We then define adaptive proofs using the same principles and formalism. Our definition of adaptive proofs works equally well with or without simulation soundness, but in the ROM simulation soundness is usually less of a problem. All our definitions are cast in the ROM, and we focus on non-interactive proofs, which we consider to be the most relevant case in practice.

Adaptive proofs address the case where one has to extract from multiple proofs chosen adaptively by a possibly malicious prover. This is the case for example when using an extractor to answer decryption queries in the CCA security game: one must decrypt the adversary's first ciphertext before he sends the second ciphertext to be decrypted (this is exactly what makes CCA security stronger than non-malleability (NM-CPA), as shown by Bellare and Sahai [6]).

We view the (non-adaptive) PoK property as a game between a prover (or adversary) and an extractor. The prover wins the game by making a valid proof for which the extractor cannot return a witness. The extractor wins if it can always return a witness to any proof, or the prover does not make a valid proof. A scheme is a PoK if there is an extractor that can win against any prover. Of course the PoK property can be specialised by considering only efficient provers or demanding an overwhelming success probability relative to some security parameter.

The ss-PoK property is an extension of the above game in which the prover can additionally make any number of simulation queries, similar to in the zero-knowledge notion.

Our adaptive versions of the games give the prover many attempts at making proofs — instead of returning a proof to the extractor, the prover now makes extraction queries, to which the extractor must reply with a valid witness. The feature that makes this definition adaptive is that the extractor returns the witness to the prover, who can use it to construct the next query. The prover can make any number of extraction and simulation queries in any order. The prover wins if the extractor ever fails to extract a witness or the simulator fails to produce a proof; the prover loses if it does not provide a valid proof in an extraction query or a valid witness in a simulation query or it asks an extraction query on a simulated proof. A scheme is called an n-proof if there is a simulator/extractor pair such that the extractor can answer the first n extraction queries from any prover (there is no bound on simulation queries); it is called an adaptive proof if it is a n-proof for any n polynomially bounded in the security parameter.

FEASABILITY AND INFEASABILITY RESULTS. We formalise adaptive proofs and show two main results.
(1.) *Simulation-sound adaptive proofs exist.* We discuss that the construction of straight-line proofs of knowledge by Fischlin [7] satisifies our notion. Fischlin's transformation is an alternative to the common Fiat-Shamir transformation and allows any Sigma protocol with unique responses to be turned into a non-interactive proof.
(2.) *Fiat-Shamir-Schnorr is not adaptively secure.* The main technical result in this paper is that the most common and efficient construction of proofs of knowledge via the Fiat-Shamir transformation [8] is not adaptively secure. Our proof constructs a prover who makes a "chain" of n Fiat-Shamir-Schnorr statement/proof pairs, following the ideas of Shoup and Gennaro [5]. We then show that any extractor that wins the adaptive proof game against this prover either reduces to breaking the one-more discrete logarithm problem or launches $\Omega(2^n)$ copies of the prover. The key technical tools in the proof are the meta-reduction paradigm and a technique which we term coin splitting.

Coin splitting allows us to perform a kind of hybrid argument on attackers which have rewinding black-box access to several copies of the same algorithm. We can change these copies individually as long as we can argue that the attacker does not notice the difference. Coin splitting is a technique to show that some changes which we make to individual copies are indeed indistinguishable to an attacker who cannot break a more basic security assumption. The idea of this technique originates in papers on resettable zero-knowledge [9].

1.2. Related Work

A high-level overview and some details regarding the above contributions appear in the conference version of this paper [24], which also sketches how to apply the concept of adaptive proofs in the context of the encrypt-then-prove approach. The focus of this paper is on the notion of adaptive proofs for which we provide a detailed treatment. In particular, we spell out the proof of insecurity of the Fiat-Shamir-Schnorr transform which has led to follow-up work on Signed ElGamal [10].

Historically, the theory of NIZK proofs began with common reference string schemes, the most relevant ones to our notions being [11, 12, 13, 14, 15, 16] for their treatment of simulation soundness. ROM NIZK and simulation soundness was defined in a line of works starting with Bellare and Rogaway [17, 18, 19]. The first formal definitions of (non-adaptive) simulation-sound PoKs in the ROM were concurrently given by Bernhard et al. [20] and Faust et al. [21], and inspired the current work.

2. Zero-Knowledge Proofs

In this section we discuss zero-knowledge proofs of knowledge and simulation soundness in the random oracle model (ROM). The central idea for zero-knowledge and its extension of simulation soundness is captured by a game between two players, a malicious prover $\widehat{\mathcal{P}}$ and an extractor $\mathcal{K}$. The prover's aim is to produce a statement and a proof that verifies such that the extractor cannot extract a witness from this proof. The extractor's goal is to extract a witness from the proof that the prover provides.

We use a code-based game-playing model à la Bellare and Rogaway [17] to define adaptive proofs of knowledge. The game mediates between all involved players, e.g., between the adversarial prover and the extractor and possibly also the simulator in case of simulation soundness. The game starts by running the initialisation algorithm which ends by specifying to which player

it wishes to transfer control to first. Executions are token-based: at any time, either the game or one of the players is running ("holding the token") and a call from one participant to another yields control. All game variables are global and persist between calls so the game may maintain state between calls. The game eventually declares the winner among the prover and the extractor.

To ensure termination, we assume strict polynomial-time provers and extractors (in the size of implicit parameters such as the size of the groups over which the proofs are constructed). Our notions could also be achieved by having the game "time out" if one player does not reply in a bounded number of time steps, though this would require a more involved model of concurrency. The important property is in any case that all players in the game must, on receiving an input, eventually produce an output. In particular, a prover cannot prevent a "perfect" extractor from winning a game by entering an infinite loop instead of producing a proof.

2.1. Proof Schemes

A cryptographic group $\mathbb{G}$ consists of a group (in the sense of group theory) of prime order q with a distinguished generator g. A group generation algorithm $\mathsf{GrpSetup}$ is an algorithm that takes a security parameter $\lambda \in \mathbb{N}$ (in unary encoding, written 1^λ) and outputs a cryptographic group $\mathbb{G}(\lambda)$. A relation over cryptographic groups is a relation R' parametrised by the "interface" of a group. For example, in any cryptographic group $\mathbb{G}$ one can define the discrete logarithm relation $R'(\mathbb{G}) := \{(x, w) \mid x \in \mathbb{G} \wedge x = g^w\}$ where g is the distinguished generator of $\mathbb{G}$. For a group generator $\mathsf{GrpSetup}$and a relation R' over groups one can define the relation $R = \bigcup_{\mathbb{G}(\lambda)} R'(\mathbb{G}(\lambda))$ for a sequence of groups $\mathbb{G}(\lambda)$, $\lambda \in \mathbb{N}$; proof schemes typically prove relations R of this kind that are in class NP.[1] An algorithm over cryptographic groups is given by pseudocode that uses the "interface" of groups. For example, the algorithms $\mathcal{P}$ and $\mathcal{V}$ below are algorithms over groups: they can be implemented for any given group[2] $\mathbb{G}$.

Definition 1. A non-interactive proof scheme for a relation R over groups consists of two algorithms $(\mathcal{P}, \mathcal{V})$ over groups. $\mathcal{P}$ may be randomised, $\mathcal{V}$ must be deterministic. For any pair $(x, w) \in R$, if $\pi \leftarrow \mathcal{P}(x, w)$ then $\mathcal{V}(x, \pi)$ must output "true".

The elements of the relation R are called statement and witness. $\mathcal{P}$ is called the prover and its outputs π are called proofs. $\mathcal{V}$ is called the verifier and a statement/proof pair on which $\mathcal{V}$ outputs "true" is called a valid proof. In the random oracle model, both $\mathcal{P}$ and $\mathcal{V}$ may call the random oracle but the relation R' itself must be independent of any oracles. The last condition in the definition of proof schemes is called correctness and says that proofs produced by the genuine prover are valid. In the random oracle model, the prover and verifier in the definition of the correctness property have access to the same random oracle, i.e. the oracle answers consistently for both algorithms.

Our definitions of properties for proof schemes are centered around a game with multiple interfaces to which various parties such as provers, extractors or simulators may connect. We give our games as collections of algorithms where each algorithm has both a name and a description of the interface to which it applies. A `return` statement in an algorithm terminates the algorithm and outputs the return value on the same interface that called the algorithm. Where an algorithm should terminate and send a value on a different interface, we use the keyword `send` instead. The keyword `halt` in the code of a game terminates not just an algorithm but the entire game — when this occurs, the game will announce a winner.

[1] Note that it would be pointless to ask whether R' is in class NP or not, as $R'(\mathbb{G})$ is a finite object for any finite group $\mathbb{G}$.

[2] If we wanted to be really precise we could write $\mathcal{P}(\mathbb{G})(x, w)$ to capture the dependency on a group. We choose not to use subscripts in order to avoid subscripted subscripts when the group itself depends on a security parameter.

2.2. Zero-Knowledge

A proof scheme is called zero-knowledge if there is an algorithm $\mathcal{S}$, called the simulator, which is able to produce proofs indistinguishable from genuine ones after seeing only the statement but no witness. Informally, $\pi \leftarrow \mathcal{P}(x, w)$ and $\pi' \leftarrow \mathcal{S}(x)$ should be indistinguishable for any pair $(x, w) \in R$.

In the (programmable) random oracle model we define zero-knowledge in such a way that the simulator is responsible for the random oracle (if present). Formally, we treat the prover $\mathcal{P}$ as an interactive algorithm that may issue oracle queries and get responses, and that eventually outputs a proof. A simulator is a stateful interactive algorithm $\mathcal{S}$ that can respond to two kinds of queries: a prove query which takes a value x as input and should produce a proof π as output, and a ro query which takes an arbitrary $x \in \Sigma^*$ as input and returns a $y \in \Sigma^*$ (throughout the paper we assume $\Sigma = \{0, 1\}$). A simulator does not have access to a random oracle, but simulates its own random oracle towards its "clients". A proof scheme is zero-knowledge in the random oracle model if the following two games are indistinguishable:

- The first game internally runs a random oracle RO . On input a pair (x, w), if $R(x, w)$ does not hold then the game returns $\perp$ and halts. If the relation holds, the game runs $\pi \leftarrow \mathcal{P}(x, w)$ and returns π. The prover $\mathcal{P}$ uses the game's random oracle. The adversary may then query the game's random oracle (which $\mathcal{P}$ used) directly, as often as she wants.
- The second game does not run a random oracle. On input a pair (x, w), again if $R(x, w)$ does not hold the game returns $\perp$ and halts. Otherwise, the game runs $\pi \leftarrow \mathcal{S}(x)$ and returns π to the adversary. The adversary may then issue random oracle queries which the game delegates to the simulator's random oracle.

We specify this property using pseudocode in Figure 1. We use the following notation: An oracle is a stateful process which other processes can access via a well-defined set of queries. If $\mathcal{O}$ is an oracle and q is one if its supported queries then we write $\mathcal{O}.\mathsf{q}(x)$ to denote the invocation of this query with parameter x. We write $x \leftarrow\!\!\$\ S$ for selecting x uniformly at random from the set S and $y \leftarrow\!\!\$\ \mathcal{A}^{O_1,\ldots,O_n}(x)$ for calling the (potentially randomised) algorithm $\mathcal{A}$ on input x to get output y. The superscripts denote the oracles that $\mathcal{A}$ can use while it is running. Sometimes, we will allow these oracles to call each other directly (for example if several oracles need access to a random oracle) and to issue a command halt that halts the entire execution.

To maintain random oracle queries in later definitions we write $[\,]$ for the empty list and $L :: l$ to concatenate element l to list L. When L is a list of pairs, we define $L(x)$ to be y such that (x, y) is the first element in L of the form $(x, \cdot)$. If no such element exists then $L(x)$ is defined to be $\perp$.

In Figure 1 we give the games G_1 and G_2 and the methods that the adversary can call. Since it will be helpful later on to give each kind of query a name, we call the adversary's initial query with parameters (x, w) a prove query. Similarly, we call the two operations that a simulator $\mathcal{S}$ admits prove and ro queries.

At the moment, our code may seem like an unnecessarily complicated way of stating a simple property. This level of formalism will become necessary when we move on to adaptive proofs however.

Definition 2. A proof scheme $(\mathcal{P}, \mathcal{V})$ is zero knowledge in the random oracle model for a relation R if there exists a simulator $\mathcal{S}$ satisfying the following condition. For any security parameter λ let $\delta(\lambda)$ be the distinguishing advantage of any efficient adversary between the games G_1 and G_2 of Figure 1 and for relation R and simulator $\mathcal{S}$. Then $\delta(\lambda)$ is negligible as a function of λ.

Game G_1	**Game** G_2
$\underline{\mathsf{initialise}()}$:	$\underline{\mathsf{initialise}()}$:
//potentially generate parameters	//potentially generate parameters
$\mathcal{A}$ issues $\underline{\mathsf{prove}(x,w)}$:	$\mathcal{A}$ issues $\underline{\mathsf{prove}(x,w)}$:
if $\neg R(x,w)$ then return $\bot$	if $\neg R(x,w)$ then return $\bot$
$\pi \leftarrow \mathcal{P}^{\mathsf{RO}}(x,w)$	$\pi \leftarrow \mathcal{S}.\mathsf{prove}(x)$
return π	return π
$\mathcal{A}$ issues $\underline{\mathsf{ro}(x)}$:	$\mathcal{A}$ issues $\underline{\mathsf{ro}(x)}$:
return $\mathsf{RO}(x)$	return $\mathcal{S}.\mathsf{ro}(x)$

Fig. 1: Games for zero-knowledge (ZK) in the random oracle model. A scheme $(\mathcal{P},\mathcal{V})$ is ZK if the two games G_1 and G_2 are indistinguishable. The adversary $\mathcal{A}$ may issue prove once and ro any number of times. RO is a random oracle.

2.3. Proofs of Knowledge

A proof scheme is a proof of knowledge if there is an extractor $\mathcal{K}$ such that for any prover $\widehat{\mathcal{P}}$ which can make a statement/proof pair that verifies, $\mathcal{K}$ can deliver an associated witness. Formalising this statement requires that we not only take care of random oracles but also the extractor's ability to "fork" the prover.

We first consider the non-rewinding case as a warm-up. A prover $\widehat{\mathcal{P}}$ is a randomized interactive algorithm that may make random oracle queries and eventually outputs a pair (x,π). A non-rewinding extractor $\mathcal{K}$ is an algorithm that takes a pair (x,π) as input, may make random oracle queries and eventually outputs a value w. We consider the game G that runs a random oracle RO internally and connects a prover and an extractor as in Figure 2. A proof scheme is an R-proof of knowledge if there is an extractor $\mathcal{K}$ such that for every prover $\widehat{\mathcal{P}}$, the game mediating between the two algorithms outputs "$\mathcal{K}$ wins" with overwhelming probability.

The game as in Figure 2, in which both the prover $\widehat{\mathcal{P}}$ and the extractor $\mathcal{K}$ can access a random oracle and where the extractor is supposed to find a witness for a valid proof produced by the prover, is actually too demanding to be useful: It basically says that anyone is able to extract a witness from the proof. To derive some sensible notion we give the extractor some advantage and allow it to inspect the random oracle queries made by the prover. That is, the extractor $\mathcal{K}$ can make an extra query list in response to which the game G returns the list H. This gives us a notion of straight-line proofs in the random oracle model which is actually sufficient for capturing the approach used by Fischlin [7].

Definition 3. A proof scheme $(\mathcal{P},\mathcal{V})$ is a straight-line proof of knowledge in the ROM w.r.t. a relation R if there is an extractor $\mathcal{K}$ such that for any prover $\widehat{\mathcal{P}}$, the game in Figure 2 augmented with a list query that allows $\mathcal{K}$ to see the list H returns "$\mathcal{K}$ wins" with overwhelming probability.

The above definition is less general than the one first proposed by Bellare and Goldreich [4]. There the authors relate the extractor's success probability to that of the prover (in producing a valid proof), whereas our definition lets the extractor win by default if the prover does not make a proof. However, our notion generalises more easily to the adaptive setting where the probability of the prover making a valid proof is no longer well-defined, since it also depends on the extractor's response to earlier proofs.

initialise:
$H \leftarrow [\,]$
start $\widehat{\mathcal{P}}$

$\widehat{\mathcal{P}}$ issues $\mathsf{ro}(x)$:
$y \leftarrow \mathsf{RO}(x)$
$H \leftarrow H :: (x, y)$
return y to $\widehat{\mathcal{P}}$

$\mathcal{K}$ issues $\mathsf{ro}(x)$:
$y \leftarrow \mathsf{RO}(x)$
return y to $\mathcal{K}$

$\widehat{\mathcal{P}}$ outputs (x, π):
if $\neg \mathcal{V}^{\mathsf{RO}}(x, \pi)$ then
halt with output "$\mathcal{K}$ wins"
$X \leftarrow x$
send (x, π) to $\mathcal{K}$

$\mathcal{K}$ outputs w:
if $R(X, w)$ then
halt with output "$\mathcal{K}$ wins"
else
halt with output "$\widehat{\mathcal{P}}$ wins"

$\mathcal{K}$ issues list :
return H

Fig. 2: The game G defining proofs of knowledge in the random oracle model. Capital-X is part of the game's internal state that persists between calls (so that the extractor's witness is verified against the same statement that the prover provided earlier).

2.4. Rewinding Extractors

The next standard notion that we formalise in our game-based model is that of a rewinding extractor in the ROM, running the prover multiple times. We model the extractor's rewinding ability by giving the extractor access to further copies of the prover, initialised with the same random string as the main incarnation of the prover's algorithm which connects to the proof of knowledge game. We call these further copies "rewound copies". Although all copies of the prover share the same random string, this string is not available to the extractor directly. This prevents the extractor from just simulating the prover on its own and reading off any witness used to make a proof.

The rewound copies of the prover connect to the extractor directly as sketched in Figure 3. In particular, the extractor is responsible for answering the random oracle queries for the rewound copies and can use this ability to "fork" them at any point. In order to apply the forking strategy to proofs made by the main prover, the extractor may make use of the list H that records all random oracle queries and answers for the main execution.

The game itself is the same as for non-rewound provers. For example, for the prover in Schnorr's protocol, one extraction strategy is to start a rewound copy of the prover and run it up until the point that it asks the oracle query which the main prover used to make its proof. Then, the extractor gives the rewound copy a different answer to the oracle query and hopes to obtain a new proof with the same commitment, in order to extract via special soundness. If the main prover made other oracle queries before the "target" one, then the extractor looks these up in the list H and gives the rewound copy the same answers when it makes the relevant queries.

Definition 4. A proof scheme is a rewinding proof of knowledge in the ROM if it satisfies the conditions of Definition 3 (proof of knowledge) for an extractor $\mathcal{K}$ that has black box access to further copies of the main prover with the same random string.

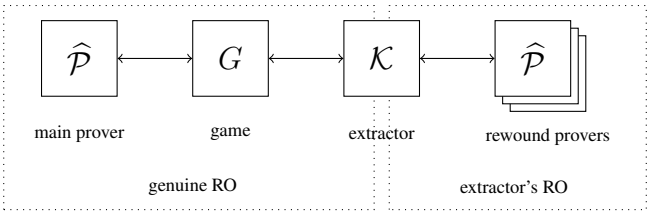

Fig. 3: Extending the straight-line proof of knowledge game to the rewinding case.

2.5. Simulation Soundness and Extractability

Sahai [11] introduced simulation soundness for the security proof of the Naor-Yung transformation. A proof scheme is simulation sound if even a prover who has seen some simulated proofs from the zero-knowledge simulator (possibly on false statements) cannot produce a new proof of a false statement.

Simulation soundness does not guarantee that an extractor will still work after a prover has seen simulated proofs, but this property is exactly what the encrypt-then-prove principle requires. Groth [14] called this property Simulation Sound Extractability (SSE). It simultaneously implies simulation soundness and extractability/PoK. SSE is still not adaptive in our sense, though: the prover may be able to see many simulated proofs, but still only produces a single proof for the extractor.

In the Random Oracle Model, SSE is easy to achieve but subtle to define (Groth [14] originally defined it for CRS proofs): the usual Sigma protocol simulator only works if it controls the random oracle but the extractor needs to access the oracle as well. The crux of the definition is that the extractor must work w.r.t. the simulator's random oracle.

We present the SSE game in Figure 4. The prover has access to a simulation oracle that replies to `prove` queries, just as in the ZK game. The prover's RO queries are handled by the simulator as well. The list Π keeps track of the simulator's replies as the prover is banned from returning a simulated proof to the extractor. Since both the prover and extractor may ask RO queries, the game has to do some extra bookkeeping (using the state C) to return the simulator's RO replies to the correct party. The state H and the list query allow the extractor to observe the prover's RO queries and launch and further copies of the prover, replaying RO answers. In addition, the game itself needs RO access to run the verifier; we write $\mathcal{V}^{\mathcal{S}.\mathsf{ro}}$ to mean that the game runs the verifier $\mathcal{V}$ and delegates any RO queries to the simulator's oracle. This means that even the notion of a "valid proof" now depends on the simulator.

In addition to the main game G, we define an auxiliary game $\widehat{G}$ that sits between the extractor $\mathcal{K}$ and its rewound provers (there is one copy of $\widehat{G}$ for each rewound prover). The task of $\widehat{G}$ is to "sanitize" prove queries made by rewound provers. When a rewound prover makes such a query, the extractor must play the role of the simulator — after all, the extractor is already simulating the rewound prover's random oracle. (The extractor may run a copy of the simulator $\mathcal{S}$ internally.) However, provers make prove queries containing both a statement x and a witness w whereas the simulator only ever gets to see x. The auxiliary game $\widehat{G}$ strips the witness from these proof queries. Otherwise, $\widehat{G}$ acts as a channel between $\mathcal{K}$ and a rewound copy of $\widehat{\mathcal{P}}$. This is slightly tedious to write in our notation; we make the convention that $\widehat{G}$ prefixes a string to every message from $\widehat{\mathcal{P}}$ to $\mathcal{K}$ to indicate whether the value is meant to be a random oracle, extraction or proof query. Messages

initialise:
$H \leftarrow [\,]$; $\Pi \leftarrow [\,]$
start $\widehat{\mathcal{P}}$

$\widehat{\mathcal{P}}$ issues $\mathsf{ro}(x)$:
$C \leftarrow$ "prover"; $I \leftarrow x$
send x to $\mathcal{S}.\mathsf{ro}$

$\mathcal{K}$ issues $\mathsf{ro}(x)$:
$C \leftarrow$ "extractor"
send x to $\mathcal{S}.\mathsf{ro}$

$\mathcal{S}.\mathsf{ro}$ returns a value y:
if $C =$ "prover" then
$H \leftarrow H :: (I, y)$
send y to $\widehat{\mathcal{P}}$
else
send y to $\mathcal{K}$

$\mathcal{K}$ calls list:
return (H, Π)

$\widehat{\mathcal{P}}$ outputs (x, π):
if $\neg \mathcal{V}^{\mathcal{S}.\mathsf{ro}}(x, \pi)$ or $(x, \pi) \in \Pi$ then
halt with output "$\mathcal{K}$ wins"
$X \leftarrow x$
send (x, π) to $\mathcal{K}$

$\mathcal{K}$ outputs w:
if $R(X, w)$ then
halt with output "$\mathcal{K}$ wins"
else
halt with output "$\widehat{\mathcal{P}}$ wins"

$\widehat{\mathcal{P}}$ issues $\mathsf{prove}(x, w)$:
if $\neg R(x, w)$ then
halt with output "$\mathcal{K}$ wins"
$X' \leftarrow x$
send x to $\mathcal{S}.\mathsf{prove}$

$\mathcal{S}.\mathsf{prove}$ returns π:
$\Pi \leftarrow \Pi :: (X', \pi)$
send π to $\widehat{\mathcal{P}}$

Fig. 4: The game G defining SSE in the random oracle model. It has three interfaces for the main prover $\widehat{\mathcal{P}}$, the extractor $\mathcal{K}$ and the simulator $\mathcal{S}$. The list H tracks the prover's random oracle queries. The state C (for caller) tracks who made the last random oracle query so that the result can be returned to the initiator. I (input) tracks the prover's oracle inputs to update H correctly. X and X' track the prover's outputs and proof requests for verification in a later query.

(responses) flowing in the other direction can always be passed on unchanged — the prover will hopefully remember what its last query was when it gets a response.

Definition 5. A proof scheme $(\mathcal{P}, \mathcal{V})$ is simulation sound in the ROM for a relation R if it satisfies the following conditions. An s-prover is an algorithm $\widehat{\mathcal{P}}$ that can ask random oracle and proof queries and eventually outputs an extraction query containing a statement/proof pair.

- The proof scheme is zero-knowledge w.r.t. R for a simulator $\mathcal{S}$ and a proof of knowledge w.r.t. R for an extractor $\mathcal{K}$.
- For every s-prover $\widehat{\mathcal{P}}$, if we connect $\mathcal{K}$ to $\widehat{\mathcal{P}}$ through the game G of Figure 4 and give $\mathcal{K}$ access to further rewound copies of the prover (with the same random string) through the auxiliary game $\widehat{G}$ of Figure 5 then with overwhelming probability the game G returns "$\mathcal{K}$ wins".

3. Adaptive Proofs of Knowledge

Given our game-centric view of proofs of knowledge we can extend the approach to adaptive proofs of knowledge. An adaptive proof is simply a proof scheme where the extractor can still win if the prover is given multiple turns to make proofs. The adaptive part is that the game hands the extractor's witness in each turn back to the prover before the prover must take her next turn.

$\mathcal{K}$ calls $\widehat{\mathcal{P}}$ for the first time:	$\widehat{\mathcal{P}}$ calls $\mathsf{ro}(x)$:
start $\widehat{\mathcal{P}}$	send ("ro", x) to $\mathcal{K}$
$\mathcal{K}$ sends a value z:	$\widehat{\mathcal{P}}$ outputs (x, π):
send z to $\widehat{\mathcal{P}}$	send ("extract", x, π) to $\mathcal{K}$
	$\widehat{\mathcal{P}}$ calls $\mathsf{prove}(x, w)$:
	send ("prove", x) to $\mathcal{K}$

Fig. 5: The auxiliary game $\widehat{G}$ for SSE. It acts mostly as a channel between $\mathcal{K}$ and a rewound prover $\widehat{P}$ except that it strips witnesses from proof queries. We use the convention that $\widehat{G}$ indicates to $\mathcal{K}$ whether a value is for a random oracle, extraction or proof query by prefixing a string.

Should a prover be able to produce a proof for which she does not know the witness, she could then use the extractor's ability to find a witness to help make her next proof. The intuition is essentially the same for the cases with and without simulation soundness. We first introduce adaptive proofs formally without simulation soundness using so-called n-proofs, where n is a parameter describing the number of rounds the prover plays. In a later step we add simulation soundness.

3.1. Adaptive Proofs and n-proofs

Fig. 6: The adaptive proof game and the queries that the various algorithms can exchange.

Let $(\mathcal{P}, \mathcal{V})$ be a proof scheme for a relation R. An adaptive prover $\widehat{\mathcal{P}}$ in the ROM is a component that can make two kinds of queries, repeatedly and in any order. The first are random oracle queries; these are self-explanatory. The second are extraction queries which take a statement and a proof as parameters. The response to an extraction query is a witness. (Correctness conditions will be enforced by the game, not the prover.) Adaptive provers may also halt. For example, a non-adaptive prover can be seen as an adaptive prover that halts after its first extraction query.

An adaptive extractor $\mathcal{K}$ is a component that can make list and random oracle queries and receive and process extraction queries from an adaptive prover. In addition, an extractor may have black-box access to further rewinding copies of the adaptive prover (with the same random string) and answer all of their queries.

The n-proof game takes a parameter n as input and connects an adaptive prover and extractor. It runs up to n rounds in which the prover may make a statement and proof and the extractor must return a witness. The extractor wins if it can deliver all n witnesses or if the prover halts earlier than this, or fails to make a valid proof. The extractor loses if it does not supply a valid witness to one of the first n extraction queries.

initialise(n):
  $H \leftarrow [\,]$
  $K \leftarrow 0$
  start $\widehat{\mathcal{P}}$

$\widehat{\mathcal{P}}$ issues $\mathsf{ro}(x)$:
  $y \leftarrow \mathsf{RO}(x)$
  $H \leftarrow H :: (x, y)$
  return y to $\widehat{\mathcal{P}}$

$\mathcal{K}$ issues $\mathsf{ro}(x)$:
  $y \leftarrow \mathsf{RO}(x)$
  return y to $\mathcal{K}$

$\mathcal{K}$ issues list :
  return H to $\mathcal{K}$

$\widehat{\mathcal{P}}$ halts: halt with output "$\mathcal{K}$ wins"

$\widehat{\mathcal{P}}$ issues $\mathsf{extract}(x, \pi)$:
  if $\neg \mathcal{V}^{\mathsf{RO}}(x, \pi)$ then
    halt with output "$\mathcal{K}$ wins"
  $X \leftarrow x$
  send (x, π) to $\mathcal{K}$

$\mathcal{K}$ outputs w:
  if $\neg R(X, w)$ then
    halt with output "$\widehat{\mathcal{P}}$ wins"
  $K \leftarrow K + 1$
  if $K = n$ then
    halt with output "$\mathcal{K}$ wins"
  else
    send w to $\widehat{\mathcal{P}}$

Fig. 7: The game G for adaptive proofs with parameter n.

Definition 6. A proof scheme is an n-proof in the ROM for a relation R if there exists an extractor $\mathcal{K}$ such that for every adaptive prover $\widehat{\mathcal{P}}$ the game G of Figure 7 when connected to $\widehat{\mathcal{P}}$ and $\mathcal{K}$ returns "$\mathcal{K}$ wins" with overwhelming probability.

If $\mathcal{K}$ also has access to further copies of $\widehat{\mathcal{P}}$ with the same random string then we call the proof scheme a rewinding n-proof, otherwise we call it a straight line n-proof.

If for every polynomial $p(x)$ there is an extractor $\mathcal{K}_{p(x)}$ making a particular scheme a $p(n)$-proof, we say that the scheme is an adaptive proof.

3.2. Simulation-Sound Adaptive Proofs

Adding simulation soundness to adaptive proofs works the same way as for non-adaptive proofs. Adaptive s-provers may make random oracle, proof and extraction queries (the simulation sound n-proof game only limits the number of extraction queries, not proof queries). We give the new algorithms in Figure 8. Random oracle calls from the main prover go to the simulator; simulated proofs are logged and provided on request to the extractor (via a list query) and are banned from extraction queries. The rewinding copies of the prover are connected to the extractor through the same games $\widehat{G}$ as in the non-adaptive case: extraction queries and witnesses found by the extractor are simply passed back and forth. Only the witnesses in prove queries are stripped out.

Consider a proof scheme $(\mathcal{P}, \mathcal{V})$ that is zero-knowledge for a relation R with simulator $\mathcal{S}$. The simulation sound n-proof experiment for this scheme, an adaptive s-prover $\widehat{\mathcal{P}}$ and an extractor $\mathcal{K}$ is the following experiment. Connect the prover $\widehat{\mathcal{P}}$, the simulator $\mathcal{S}$ and the extractor $\mathcal{K}$ to the game G of Figure 8. Let $\mathcal{K}$ have black box access to further copies of $\widehat{\mathcal{P}}$ mediated by $\widehat{G}$ as in Figure 5 that forwards all messages in both directions except that it strips witnesses from proof queries.

Definition 7. Let $(\mathcal{P}, \mathcal{V})$ be a proof scheme for a relation R that is zero-knowledge with simulator $\mathcal{S}$. The scheme is a simulation sound n-proof if there is an extractor $\mathcal{K}$ such that for any adap-

initialise(n):
  $H \leftarrow [\,]$; $\Pi \leftarrow [\,]$
  $K \leftarrow 0$
  start $\widehat{\mathcal{P}}$

$\widehat{\mathcal{P}}$ issues $\mathsf{ro}(x)$:
  $C \leftarrow$ "prover"; $I \leftarrow x$
  send x to $\mathcal{S}.\mathsf{ro}$

$\mathcal{K}$ issues $\mathsf{ro}(x)$:
  $C \leftarrow$ "extractor"
  send x to $\mathcal{S}.\mathsf{ro}$

$\mathcal{S}.\mathsf{ro}$ returns a value y:
  if $C =$ "prover" then
    $H \leftarrow H :: (I, y)$
    send y to $\widehat{\mathcal{P}}$
  else
    send y to $\mathcal{K}$

$\mathcal{K}$ issues list:
  return (H, Π)

$\widehat{\mathcal{P}}$ halts:
  halt with output "$\mathcal{K}$ wins"

$\widehat{\mathcal{P}}$ issues $\mathsf{extract}(x, \pi)$:
  if $\neg \mathcal{V}^{\mathcal{S}.\mathsf{ro}}(x, \pi)$ or $(x, \pi) \in \Pi$ then
    halt with output "$\mathcal{K}$ wins"
  $X \leftarrow x$
  send (x, π) to $\mathcal{K}$

$\mathcal{K}$ outputs w:
  if $\neg R(X, w)$ then
    halt with output "$\widehat{\mathcal{P}}$ wins"
  $K \leftarrow K + 1$
  if $K = n$ then
    halt with output "$\mathcal{K}$ wins"
  else
    send w to $\widehat{\mathcal{P}}$

$\widehat{\mathcal{P}}$ issues $\mathsf{prove}(x, w)$:
  if $\neg R(x, w)$ then
    halt with output "$\mathcal{K}$ wins"
  $X' \leftarrow x$
  send x to $\mathcal{S}.\mathsf{prove}$

$\mathcal{S}.\mathsf{prove}$ returns π:
  $\Pi \leftarrow \Pi :: (X', \pi)$
  send π to $\widehat{\mathcal{P}}$

Fig. 8: Simulation sound n-proofs in the random oracle model.

tive s-prover $\widehat{\mathcal{P}}$, the simulation sound n-proof experiment returns "$\mathcal{K}$ wins" with overwhelming probability. If the extractor works for all polynomially bounded n, we call the scheme an adaptive simulation sound proof.

3.3. Discussion

We can classify the types of proofs that we have introduced with the following table. "poly" means polynomially many and means that the number of queries is not bounded by the games involved but an efficient algorithm will be able to launch at most polynomially many queries (in some initial input or security parameter) in the first place. We see no reason to study explicit bounds on the number of proof queries as our simulators are non-rewinding.

type	# extraction queries	# proof queries
PoK	1	none
ss-PoK	1	poly
n-proof	n	none
ss-n-proof	n	poly
adaptive proof	poly	none
ss adaptive proof	poly	poly

The adaptive property of our proofs is strictly stronger than allowing the prover to make one extraction query with a vector of many statement/proof pairs. Namely, the multiforking lemma of Bagherzandi et al. [22] shows that Fiat-Shamir-Schnorr proofs are extractable for one such "parallel" query yet we will show that such proofs are not adaptive proofs (under the one-more discrete logarithm assumption). This separation has an analogue in the world of public-key encryption that we will discuss soon. Sahai [11] shows that security against one "parallel" decryption query is equivalent to non-malleability (NM-CPA) which is known to be strictly weaker than CCA security where adaptive decryption queries are allowed. It is also known that adding a proof of knowledge (of the message and randomness) to an IND-CPA secure encryption scheme can be used to construct non-malleable encryption yet this technique is not guaranteed to achieve CCA security.

After establishing that simulation sound adaptive proofs exist by studying a construction due to Fischlin [7], we show two main results. The negative result is that the Fiat-Shamir transformed Schnorr protocol is not adaptively secure.

Adaptive proofs bear an interesting relationship to CCA security of encryption. Although adaptivity is an interesting question for many applications in particular in multi-party computation, almost all known constructions of CCA secure public-key encryption use some form of non-interactive proof of knowledge in their ciphertexts.[3] This is sometimes known as the encrypt-then-prove construction.

4. Fischlin proofs are adaptively secure

In this section we establish that simulation-sound adaptive proofs in the random oracle model exist. An existing construction due to Fischlin [7] is adaptively secure. Fischlin gives a transformation of Sigma protocols to non-interactive proof schemes as an alternative to the more common Fiat-Shamir transformation. Below we describe this transformation and show that it yields adaptively secure proofs.

THE PROTOCOL. The basic idea of Fischlin's transformation is to select the verifier's challenge in a Σ-protocol such that a number of low-order bits in the hash of the proof are all zero, or at least "small enough". This forces the prover to ask repeated values of potential challenges to the random oracle until he finds one with a suitable response; from any two such queries, which must appear in the list of hash queries made by the prover, one can extract a witness using special soundness.

We begin with a Σ-protocol with a challenge space of ℓ bits size, such that $\ell = \mathcal{O}(\log \lambda)$ where λ is the security parameter. We can associate the following algorithms to a Σ-protocol:

Commit takes some inputs and produces a commitment $comm$ and auxiliary information aux (to generate the response later). This algorithm models what the honest prover does at the be-

[3] The exception is the original construction via the Naor-Yung transformation [12] but this comes at the cost of having to encrypt twice.

ginning of the protocol. We assume that the commitment has superlogarithmic min-entropy, which is easy to satisfy by appending random bits to the value.

$\mathsf{Respond}(ch, aux)$ produces the response that the honest prover gives after receiving challenge ch. All other information that he needs is contained in aux. We assume that Respond is deterministic.

$\mathsf{Verify}(y, comm, ch, resp)$ verifies a protocol execution with respect to some initial input y. This algorithm is run by the verifier at the end of the protocol. We assume that Verify is deterministic.

The transformation relies on parameters b, r, S, t subject to the following conditions.

b is the bit-size of the range of a random oracle $H : \{0,1\}^* \to \{0,1\}^b$. We require $b = \mathcal{O}(\log \lambda)$ such that searching for a compliant answer can be done efficiently by the prover.

r is the number of repetitions of the basic Σ-protocol. We require $r = \mathcal{O}(\log \lambda)$ and $b \cdot r = \omega(\log \lambda)$. The latter ensures that a malicious prover cannot find a "low-weight" proof with non-negligible probability.

S is the maximum sum of hash values for an accepting proof. We require $S = \mathcal{O}(r)$.

t is the size of challenges. We require $t = \mathcal{O}(\log \lambda)$, $b \leq t \leq \ell$ and $2^{t-b} = \omega(\log \lambda)$. These choices ensure that the honest prover can find a proof with overwhelming probability.

The prover and verifier execute the following algorithms which we write $(\mathsf{Prove}, \mathsf{Verify})$ instead of $(\mathcal{P}, \mathcal{V})$ in this section, to be consistent with the original. By $i \leftarrow \mathsf{argmin}_{j \in I} f(j)$ we mean iterate over all values of I and set i to be the value of $j \in I$ for which $f(j)$ is minimal. If the minimum is attained for several values, pick the first such one.

procedure Prove (x, w)

100 for $i = 1, \ldots, r$ do

101 $\quad (com_i, aux_i) \leftarrow \mathsf{Commit}(x, w)$

102 for $i = 1, \ldots, r$ do

103 $\quad ch_i \leftarrow \mathsf{argmin}_{j \in [0, 2^t - 1]} H(x, (com_k)_{k=1}^r, i, j, \mathsf{Respond}(j, aux_i))$

104 $\quad resp_i \leftarrow \mathsf{Respond}(ch_i, aux_i)$ // We assumed that Respond is deterministic so we get the same value as above.

105 $\quad \pi \leftarrow (com_i, ch_i, resp_i)_{i=1}^r$

procedure Verify (x, π)

200 for $i = 1, \ldots, r$ do // Verify each sigma proof individually.

201 $\quad$ if $\mathsf{Verify}(x, com_i, ch_i, resp_i) = 0$ then

202 $\quad\quad$ return 0

203 if $\sum_{i=1}^r H(x, (com_k)_{k=1}^r, i, ch_i, resp_i) \leq S$ then return 1 else return 0

For the full security proofs of this scheme we refer the reader to the original paper [7]; we repeat the main points here according to our terminology. As shown in [7] the protocol has a straight-line (non-rewinding) extractor: given only the hash queries made by any prover that produced an accepting proof, with overwhelming probability there are two queries $H(x, (com_k)_{k=1}^r, i, ch_i, resp_i)$ and $H(x, (com_k)_{k=1}^r, i, ch_i', resp_i')$ such that $ch_i \neq ch_i'$. The original Σ-protocol is required to have

special soundness, allowing to extract a witness from two accepting proofs for the same commitment, and to satisfy the property of unique responses, which says that for any malicious prover it is infeasible to find x, com, ch together with two valid responses $resp, resp'$. This suffices to compute a witness w efficiently.

The following theorem establishes that Fischlin's transformation yields simulation sound adaptive proofs in the random oracle model.

Theorem 1. Let $(\mathsf{Commit}, \mathsf{Respond}, \mathsf{Verify})$ be a Σ-protocol with special soundness and unique responses. Then the Fischlin transformation of this protocol is a simulation sound adaptive proof in the random oracle model.

Proof. We show that the system in question is an adaptive proof. The extractor $\mathcal{K}$ stores all hash input/output pairs that the main invocation of the adversary makes. The extractor does not launch any other invocations. When the extractor $\mathcal{K}$ receives an extraction query, it searches for values in the hash list from which it can extract a witness to the given proofs using special soundness and the fact that responses are unique. That is, to extract for a statement x, algorithm $\mathcal{K}$ issues a list request and then searches for entries $H(x, (com_k)_{k=1}^r, i, ch_i, resp_i)$ and $H(x, (com_k)_{k=1}^r, i, ch_i', resp_i')$ such that $ch_i \neq ch_i'$. If it finds such entries then it invokes the special-soundness extractor for the underlying Σ-protocol to compute a witness. If there are no such entries then $\mathcal{K}$ returns $\perp$.

COMPLETENESS. Before showing extractability, we first argue along the reasoning in [7] that the scheme is complete, i.e., allows an honest prover to efficiently find a proof with overwhelming probability.[4] For this Fischlin first shows that the probability of the prover obtaining an extremely large sum of hash values of value rS is at most $re^{-(S+1)2^{t-b}}$. By the choice of parameters, r being logarithmic and 2^{t-b} being superlogarithmic, this probability is negligible. Next, Fischlin shows that obtaining any sum T between $S+1$ and rS is also negligible, namely, $e^{r+(S+1)(r-2^{t-b})}$. It follows again by the choice of the parameters r and 2^{t-b} that this is negligible.

ZERO-KNOWLEDGE. The zero-knowledge simulator in [7], programming the random oracle on some points when giving a proof via $\mathcal{S}.\mathsf{prove}(x)$ and otherwise answering with random values for "other" random oracle queries $\mathcal{S}.\mathsf{ro}(x)$, works as follows. When simulating a proof for each $i = 1, 2, \ldots, r$ and each $ch_i \in \{0,1\}^t$ the simulator first picks random hash values (denoted h_{i,ch_i}). For each i it picks the lexicographic first one ch_i with the smallest hash value (when truncated to b bits). It then calls the zero-knowledge simulator for the Σ-protocol to create a proof $com_i, ch_i, resp_i$ for the given challenge.[5] For each i the simulator now implicitly sets the random oracle at $H(x, (com_k)_{k=1}^r, i, ch_i^*, resp_i^*)$ for valid proofs to h_{i,ch_i^*} for all $ch_i^* \in \{0,1\}^t$. In particular, for each i the hash value $H(x, (com_k)_{k=1}^r, i, ch_i, resp_i)$ for the created proof equals the lexicographic first among the smallest ones over all challenges.

In [7] it is now discussed that the proof is perfectly zero-knowledge, unless the simulator $\mathcal{S}.\mathsf{prove}$ resets a previously defined hash value, or if the malicious prover asks the random oracle about two tuples $(x, (com_k)_{k=1}^r, i, ch_i, resp_i)$, $(x, (com_k)_{k=1}^r, i, ch_i, resp_i')$ for $resp_i \neq resp_i'$ for valid responses. The former happens with negligible probability by the superlogarithmic min-entropy of the commitments, the latter would contradict the property of unique responses. For details see [7].

SIMULATION SOUNDNESS. Recall that the extractor works by searching the list of hash queries for two accepting proofs for some x, com_i, ch_i. In our setting, we are concerned with the malicious

[4] As pointed out in [7], it can be turned into a perfectly complete proof scheme by letting the prover output a witness in clear in case it does not manage to find an accepting proof.

[5] It was shown in [7] that any Σ-protocol with logarithmic challenge size is special zero-knowledge in the sense that one can pre-determine a challenge value and the zero-knowledge simulator creates a corresponding proof.

prover outputting some valid pair $(x, \pi) \notin \Pi$ not in the list of previously proven statements, such that the extractor fails (see Figure 8).

As in [7] we will first discuss the case that the prover $\widehat{P}$ re-uses x as in a previously simulated proof, but for a fresh proofπ. First note that if the prover uses some simulated proof containing tuples $(x, (com_k)_{k=1}^r, i, ch_i, resp_i)$, but where it leaves $x, (com_k)_{k=1}^r, i$ and now uses a fresh challenge $ch_i' \neq ch_i$ for some i and augments it by a valid response $resp_i'$, then our extractor is able to find a witness by the special soundness. This is independent of the fact that the first prove is simulated only.

Next, Fischlin shows that, for any tuple $(x, (com_k)_{k=1}^r, i, ch_i, resp_i)$ in the proof π, the adversary cannot have made hash queries $(x, (com_k)_{k=1}^r, i, ch_i, resp_i')$ for valid responses $resp_i' \neq resp_i$, by the property of unique responses. If the adversary, on the other hand, made only the calls $(x, (com_k)_{k=1}^r, i, ch_i, resp_i)$ to the random oracle, then he proves that the probability of finding a proof with sum at most S in Q hash queries (for $S = \mathcal{O}(r)$) is at most $(Q+1) \cdot (S+1) \cdot 2^{(\log(ec)-b)r}$, which is negligible by the choice of parameters. Put differently, for a single extraction query the extractor finds a witness with overwhelming probability.

The final step is now to argue that the probability of success remains non-negligible even after polynomially many repetitions, whether in sequence or in parallel. More precisely, let $q(\lambda)$ be a bound on the number of proofs the extractor must extract from in total. Let $\varepsilon(\lambda)$ be the probability that the extractor fails on a single proof. We know that $\varepsilon(\lambda)$ is negligible, so from some λ_0 onwards we can assume $q(\lambda) < 1/\varepsilon(\lambda)^2$. But $\lim_{\lambda \to \infty} (1 - 1/\lambda^2)^\lambda = 1$, so the success probability of the extractor converges to 1. From some λ_0 onwards we can therefore assume that the extractor succeeds with constant (greater than zero) and hence non-negligible probability.[6] □

5. Fiat-Shamir-Schnorr proofs are not adaptively secure

Our second result is negative. It shows that Fiat-Shamir Schnorr is an example that separates proofs of knowledge from adaptive proofs.

Theorem 2. The Fiat-Shamir-Schnorr (FSS) proof scheme is not adaptively secure under the one-more discrete logarithm assumption. Specifically, for any n there is a prover $\widehat{\mathcal{P}}$ who makes a sequence of n FSS proofs. For any extractor $\mathcal{K}$ who can win the adaptive proof experiment against $\widehat{\mathcal{P}}$, either $\mathcal{K}$ calls at least 2^n rewinding copies of $\widehat{\mathcal{P}}$ or there is a reduction that solves the one-more discrete logarithm problem in the underlying group with a comparable success rate to $\mathcal{K}$.

We start with a high-level overview of our approach and then provide the details. The prover in question follows the same ideas as Shoup and Gennaro's CCA attacker [5]. While the cited work gave the attacker as an example why the "obvious" proof fails, it did not show any inherent limitation of the Fiat-Shamir technique; it did not show that this limitation cannot be overcome by using a different proof technique. Our paper is the first to give a proof that Fiat-Shamir transformed sigma protocols have an unavoidable limitation.

The core of the proof is a metareduction. For any extractor that wins the adaptive proof game against our prover, we construct a second "meta-extractor" that finds a discrete logarithm to one of the Schnorr challenges involved if the extractor's rewinding strategy satisfies certain conditions that we call "event E". We then replace the prover by a one-more discrete logarithm challenger

[6] The proof so far can be extended to any non-interactive proof scheme with a straight-line extractor that takes the transcript of all queries made by the adversary so far as input and has an overwhelming probability of extraction.

procedure initialise (λ):
$\mathbb{G} \leftarrow \mathsf{GrpSetup}(\lambda)$;
$i, j \leftarrow 0; C \leftarrow \emptyset$;
return $\mathbb{G}$

procedure finalise $(\{x_i\}_{i\in[n]})$:
If $n > j$ and $g^{x_i} \in C$ for all i
then return 1
else return 0

oracle challenge $()$:
$i++; X_i \leftarrow_{\$} \mathbb{G}$;
$C \leftarrow C \cup \{X_i\}$;
return X_i

oracle dlog (X):
$j++$;
return $dlog_g(X)$

Fig. 9: Experiment for defining the one-more discrete logarithm problem. We demand that $x_i \neq x_j \mod |\mathbb{G}|$ for all $i \neq j$.

to deal with the other proofs that may arise from rewinding copies. We reuse our coin splitting technique to argue that this game hop is undetectable by the extractor.

We use a combinatorial argument to show that any extractor who does not make at least 2^n calls to different rewinding copies of the prover must trigger event E. The gist of the argument is that we map the prover's n queries to a path in a depth-n binary tree where each node represents a discrete-logarithm problem. Each query that the extractor answers successfully reveals certain nodes in this tree and answering all n queries implies that the entire tree is revealed. We treat each rewinding operation that the extractor performs as exploring a path in the tree. If the extractor explores the complete tree we get the claimed 2^n rewinding operations. The event E is that the extractor "jumps" from one leaf in the tree to another without exploring the path in between. If this occurs, our reduction solves the one-more discrete logarithm problem. It is impossible, even for a computationally unbounded extractor, to reveal the whole tree (answer all n queries) without either exploring the whole tree (taking exponential time) or performing at least one jump (solving a one-more discrete logarithm instance).

Before we proceed with the details of the proof, we recall the one-more discrete logarithm (OMDL) assumption. We first recall this assumption and then present our proof.

5.1. The One-More Discrete Logarithm Problem

Formally, we consider the experiment $\mathbf{Exp}^{\mathsf{omdl}}_{\mathcal{A},\mathbb{G}}(\lambda)$ defined in Figure 9. The notation follows the code-based game-playing model of Bellare and Rogaway [23]: there are procedures called initialise and finalise and oracles called chal and dlog. The experiment begins with the game running the initialisation algorithm and handing its return value to the adversary. The adversary may then call the two oracles in any order and as many times as she wishes before she returns an output value which is passed to the finalisation algorithm — in our case a set of candidate discrete logarithms for the challenges. The output of the finalisation is taken to be the game result. The adversary's aim is to make the game result beome 1 and the OMDL assumption holds in a family of groups if no efficient adversary can make the game return 1 with more than negligible probability.

The experiment involves an adversary $\mathcal{A}$ that works against group $\mathbb{G}$ defined by GrpSetup. The adversary has access to two oracles, one that provides fresh random group elements, and one that returns the discrete logarithm for aribtrary group elements. For the OMDL experiment, the game output is the boolean value that indicates whether the number of challenges obtained from the challenge oracle is strictly greater than the number of discrete logarithm queries made, and that the adversary has returned the discrete logarithms of all of the challenges she has received.

procedure $\hat{P}_n^{\Psi}(\mathbb{G})$:

$c_0 \leftarrow 0$
for $i \leftarrow 1, \ldots, n$ do
  $x_i \leftarrow\!\!\$\ \Psi(1, c_0, c_1, \ldots, c_{i-1})$
  $a_i \leftarrow\!\!\$\ \Psi(2, c_0, c_1, \ldots, c_{i-1})$
  $X_i \leftarrow g^{x_i}; A_i \leftarrow g^{a_i}$
  $c_i \leftarrow \mathsf{ro}(X_i, A_i)$
for $i \leftarrow n, \ldots, 1$ do
  $s_i \leftarrow a_i + c_i x_i$
  $w_i \leftarrow \mathsf{extract}(X_i, (A_i, s_i))$
  halt if $g^{w_i} \neq X_i$ //check answer w_i
halt //give up, extractor wins

procedure $\mathcal{A}_n^{\mathsf{challenge}_\Psi, \mathsf{dlog}}(\mathbb{G})$:

$c_0 \leftarrow 0$
for $i \leftarrow 1, \ldots, n$ do
  $X_i \leftarrow \mathsf{challenge}_\Psi(1, c_0, \ldots, c_{i-1})$
  $A_i \leftarrow \mathsf{challenge}_\Psi(2, c_0, \ldots, c_{i-1})$
  $c_i \leftarrow \mathsf{ro}(X_i, A_i)$
for $i \leftarrow n, \ldots, 1$ do
  $s_i \leftarrow \mathsf{dlog}(A_i \cdot X_i^{c_i})$
  $w_i \leftarrow \mathsf{extract}(X_i, (A_i, s_i))$
  halt if $g^{w_i} \neq X_i$ //check answer w_i
halt //give up, extractor wins

Fig. 10: Adversary $\hat{P}$ is against the adaptive property of the Fiat-Shamir-Schnorr. Adversary $\mathcal{A}_n$ simulates $\hat{P}$ using the oracles from the one more discrete logarithm experiment.

For an adversary $\mathcal{A}$ we define its advantage against the one more discrete logarithm problem by $\mathbf{Adv}^{\mathsf{omdl}}_{\mathcal{A},\mathbb{G}}(\lambda) = \mathbf{Pr}[\mathbf{Exp}^{\mathsf{omdl}}_{\mathcal{A},\mathbb{G}}(\lambda) = 1]$ and we say that the problem is hard with respect to a group generator GrpSetup if for any efficient adversary $\mathcal{A}$, its advantage is negligible.

5.2. Overview

We begin by defining a particular adversary $\hat{P}$ which makes a chain of n proofs following the ideas of Shoup and Gennaro [5]. Our adversary uses a random function Ψ but this is only for illustration purposes (one could also replace Ψ with a pseudorandom function).

Next, we construct a reduction $\mathcal{R}$ which simulates multiple copies of the prover $\hat{P}$. We claim that no extractor $\mathcal{K}$ in the adaptive proof game can distinguish whether it is interacting with $\mathcal{R}$ or multiple copies of $\hat{P}$. The reduction $\mathcal{R}$ does not require a random function. $\mathcal{R}$ reduces to the one-more discrete logarithm problem and the main challenge in this step is ensuring that $\mathcal{R}$ can simulate a potentially unbounded number of copies of $\hat{P}$ using only one OMDL challenger.

In the adaptive proof game between $\mathcal{K}$ and $\mathcal{R}$ (the latter playing all copies of the adversary at once), we define an event NZP. If this event occurs, $\mathcal{R}$ can solve the one-more discrete logarithm instance that is is interacting with.

For any execution of the game that $\mathcal{K}$ wins against $\mathcal{R}$ without event NZP occuring, we show via a combinatorial argument that $\mathcal{K}$ must have interacted with at least 2^n copies of the adversary (simulated by $\mathcal{R}$). Since an efficient extractor (in the sense of the adaptive proof definition) must win the game with overwhelming probability but cannot take exponential time, we conclude that such an extractor always triggers event NZP and therefore we have an efficient OMDL adversary which always succeeds.

5.3. Construction

In Figure 10 we give the prover $\hat{P}$ and a "hypothetical" OMDL reduction $\mathcal{A}_n$. We cannot use this reduction directly as the extractor has the ability to rewind its adversary, but a reduction cannot

rewind the challenger. The reduction $\mathcal{A}_n$ is intended only as an illustration of how our reduction $\mathcal{R}$ will work later.

The prover $\hat{P}$ runs two loops. First, it builds up a chain of n Schnorr statement/proof pairs by picking statements, commitments and challenges (X_i, A_i, c_i). Whenever a random value is required, $\hat{P}$ uses its random function on the entire "history" of random oracle queries so far. This has the effect that if an extractor runs two copies of $\hat{P}$ and "forks" them at any point by giving them different answers to a random oracle query, they will behave from then on as if they had independent sources of randomness.

In the second loop, $\hat{P}$ completes the proofs by computing the responses s_i and asks extraction queries in reverse order (note that the loop counter j runs from n down to 1). Whenever the extractor returns a witness w_i, $\hat{P}$ checks this and halts if it is incorrect. While the main prover does not need to check witnesses (the adaptive proof game does this already), if the extractor tries to give a bad witness to a rewinding copy of the prover then this copy halts and refuses to divulge any further responses s_i which the extractor could use to apply special soundness.

The algorithm $\mathcal{A}_n$ shows the idea behind our reduction $\mathcal{R}$. The Schnorr proof statements and commitments become OMDL challenges and we use one dlog query to compute the response. We can already see that if the extractor correctly answers any extract query of the main copy of the adversary without launching any other copies of the adversary then we must win the OMDL game since we used two challenges but only one discrete logarithm query for the proof in question and the extractor's reply is the second discrete logarithm. We can open all other proofs with one extra dlog query in which case we have obtained $2n$ challenges and solved them all with only $2n - 1$ dlog queries, which means that we win the OMDL game. The reduction $\mathcal{R}$ in the next step keeps track of OMDL queries across all copies of the adversary so that we can make a similar argument to win OMDL whenever there are fewer than 2^n copies of the adversary in play.

5.4. Reduction

The key idea in our reduction $\mathcal{R}$ is to track the history H of all challenges (random oracle responses) in each copy of the adversary. We give the code of the reduction in Figure 11. The calls to ro and extract pass control to the extractor which may choose to activate a different copy of the adversary before returning a value[7].

The reduction shares state between the copies of the adversary that it simulates through the OMDL challenger and two global maps L and Φ, which are shared between all copies of the adversary simulated by $\mathcal{R}$. In the first loop, the adversary makes a chain of n Schnorr statement/-commitment pairs. The map L tracks the pair that the adversary makes when its history is H and we write $L[H]$ for the value stored at key H, if any. This allows the reduction to simulate several copies of the adversary consistently: whenever a copy with history H is supposed to make a statement/commitment pair, it looks first in the list H if another copy has already made the required pair and re-uses the same pair if this is the case.

If a copy of the adversary needs to make a fresh statement/commitment pair because it is the first copy to reach history H, it calls the newchallenge subroutine. This draws two OMDL challenges, records them in L and makes an entry in Φ which we will describe in a moment.

In each pass through the second loop, each adversary copy completes a proof, asks and verifies an extraction query. Since the aim of the OMDL game is to open all challenges with fewer dlog calls than challenge calls, consider a potential Φ defined as the number of challenge calls minus

[7]For those with knowledge of software engineering terminology: the copies of the adversary simulated by $\mathcal{R}$ are coroutines with shared state and the ro and extract calls are "yield" calls.

procedure adversary(id):

$H \leftarrow [\,]$
for $i \leftarrow 1, \ldots, n$ do
  if $(X_i, A_i) \leftarrow L[H]$ then
    $c_i \leftarrow \mathsf{ro}(X_i, A_i)$
    $H \leftarrow H :: c_i$
  else
    $(X_i, A_i) \leftarrow \mathsf{newchallenge}(H)$
    $c_i \leftarrow \mathsf{ro}(X_i, A_i)$
    $H \leftarrow H :: c_i$
for $i \leftarrow n, \ldots, 1$ do
  $(\phi, c, s, x, a) \leftarrow \Phi[(X_i, A_i)]$
  if $\phi = 2$ then
    $s_i \leftarrow \mathsf{firstproof}(X_i, A_i, c_i)$
  elseif $\phi = 1$ then
    $s_i \leftarrow \mathsf{secondproof}(X_i, A_i, c_i)$
  else
    $s_i \leftarrow a + c \cdot x$
  $w_i \leftarrow \mathsf{extract}(X_i, (A_i, s_i))$
  if $g^{w_i} \neq X_i$ then halt

procedure newchallenge(H):

$X \leftarrow \mathsf{challenge}()$
$A \leftarrow \mathsf{challenge}()$
$L[H] \leftarrow (X, A)$
$\Phi[(X, A)] \leftarrow (2, ?, ?, ?, ?)$
return (X, A)

procedure firstproof(X, A, c):

$s \leftarrow \mathsf{dlog}(A \cdot X^c)$
$\Phi[(X, A)] \leftarrow (1, c, s, ?, ?)$
return s

procedure secondproof(X, A, c):

$(\phi, c', s', x', a') \leftarrow \Phi[(X, A)]$
if $c = c'$ then
  return s'
else
  $x \leftarrow \mathsf{dlog}(X)$
  $a \leftarrow s' - c' \cdot x$
  $s \leftarrow s' + (c - c') \cdot x$
  $\Phi[(X, A)] \leftarrow (0, c, s, x, a)$
  return $a + c \cdot x$

Fig. 11: The reduction $\mathcal{R}$ simulating a copy of $\hat{P}$. The reduction $\mathcal{R}$ itself consists of multiple copies of this algorithm each with a unique identifier id. The variables C, L, Φ are shared between all copies.

the number of discrete logarithm calls made so far at any point in the execution of the reduction $\mathcal{R}$. We will see that this potential can never become negative. If we ever manage to collect all discrete logarithms while the potential is strictly positive, we can win the OMDL game.

The potential Φ can be expressed as the sum of the local potentials ϕ over all pairs (X, A) of statements and commitments made by the reduction. The map Φ tracks this potential and some extra bookkeeping information. Whenever the newchallenge procedure draws two new OMDL challenges, it adds a new entry for them in Φ. Each such entry is a 5-tuple (ϕ, c, s, x, a) where the first element ϕ is the local potential of the pair and the last four elements can take the special value $?$ (undefined). When a pair is first created, two OMDL challenges have been used to create it and no discrete logarithms for this pair are known yet so the local potential is set to $\phi = 2$ and the other entries are undefined.

The first time we make a proof on a pair (X, A), we use one dlog query to get the required response s and drop the local potential to $\phi = 1$. We also record the challenge c and response c used in the map Φ.

If we need to make a proof on a pair (X, A) at local potential $\phi = 1$, there must have been a previous proof on this pair (or the potential would still be at 2). We have two cases: if we have been given the same challenge as in the previous proof, we just replay the same response. This is why we record challenge/response pairs in the map Φ. If we are given a fresh challenge, this means that the extractor has "forked" two copies of the adversary on this proof and is about to

obtain the witness by special soundness. In this case, we drop the potential to 0 with a dlog query to get x, the actual witness on which we are making our Schnorr proof, and use the previously stored information to find a.

If the extractor forks multiple copies of the adversary on the same proof more than once (i.e. it gives three different copies three different challenges for the same statement/commitment pair) then after the second proof, the local potential will be at 0 but we will have recovered x and a already. Therefore, we do not need any further dlog queries to complete the third proof but can just compute the response the usual way.

Lemma 1. For any extractor $\mathcal{K}$ connected to the adaptive proof game, our reduction $\mathcal{R}$ is statistically indistinguishable from multiple copies of the adversary $\hat{P}$.

Proof. The proof is by induction over all queries sent to the extractor $\mathcal{K}$. Before the first time the extractor receives anything, it obviously cannot distinguish anything. The reduction receives a random oracle query from the adversary whenever a rewinding copy of the adversary passes through its first loop. The elements in these queries are completely characterised by the following description:

- If the history of the current copy (making the random oracle query) is a prefix of any other copy's history, then the current copy returns the exact same query as the previous one. After all, all copies run the same algorithm on the same initial random string.
- If the current copy's history is "fresh" — either the current copy has advanced further than any other copy or the extractor has forked it — then the two elements in the random oracle query are uniformly random group elements independent of any previous elements. This is because the elements are drawn using a random function on a fresh input.

By inspection of the code of the reduction, we see that it has the same loop structure and meets the same invariants. The list L ensures the first condition that copies with the same history return the same elements and fresh copies trigger challenge calls which return fresh, uniform group elements by definition of the OMDL problem. Therefore, the pattern of random oracle queries is statistically indistinguishable between the reduction and the adversary copies. The argument for the main copy is identical although the extractor only gets to see the oracle queries in response to list queries to the adaptive game rather than immediately.

For extraction queries the induction argument is even simpler: statement/commitment pairs of each copy are exactly the ones asked earlier in the random oracle queries (in reverse order) and the responses s_i are completely determined by the statement, commitment and challenge (which came from the extractor). □

5.5. The event NZP

We write NZP for the event that throughtout its execution, $\mathcal{R}$ receives a correct response to an extract query while the local potential of the associated statement/commitment pair is $\phi = 1$ (it can never be 2 as the reduction had to drop it to 1 just to get the response s to ask the extract query in the first place).

Lemma 2. In any execution, the moment event NZPoccurs, the reduction $\mathcal{R}$ can immediately win the OMDL game that it is playing.

The only way that the extractor can ever answer an extraction query without triggering event NZP is if it has interacted with a different copy of the adversary, giving it a different challenge for

the same pair. This triggers the secondproof algorithm which drops the local potential to 0. This is exactly extraction by forking and special soundness. In other words, event NZP is the event that the extractor finds a witness by some other means than special soundness.

Proof. When the reduction receives the correct w for a pair (X, A) at potential 1, the entry $\Phi[(X, A)]$ is of the form $(1, c, s, ?, ?)$ such that (X, A, c, s) is a correct Schnorr transcript. Since w is the discrete logarithm of X, the reduction can compute $a \leftarrow s - c \cdot x$ and has both discrete logarithms of a pair of challenges at potential 1. The reduction next opens all other challenge pairs: pairs at potential 0 are already opened; for pairs (X', A') at potential 1 the reduction asks $x' \leftarrow \mathsf{dlog}(X')$ and proceeds as before to get a'; for pairs (X'', A'') at potential 2 it just asks $\mathsf{dlog}(X'')$ and $\mathsf{dlog}(A'')$. The result is that the reduction has made exactly one fewer dlog query than challenge query and has all discrete logs, which wins the OMDL game. □

Note that the reduction can check the correctness of a candidate w from the extractor itself. If the extractor passes the reduction's "main adversary" an incorrect witness, the reduction loses the adaptive proof game. If a rewinding adversary simulated by the reduction gets an incorrect witness, event NZP is not triggered but this copy halts and so is of no further use to the extractor.

There is one other case that lets the reduction win OMDL immediately: if two challenges returned from the OMDL challenger ever collide. We could simply ignore this event as it happens with negligible probability but even then, since one discrete logarithm call in this case gives the answer to two challenges, the reduction can open all further challenges at a cost of one discrete logarithm call each and win the OMDL game.

5.6. Combinatorial argument

Lemma 3. Suppose that the extractor $\mathcal{K}$ wins an execution of the adaptive proof game against the reduction $\mathcal{R}$ without $\mathcal{R}$ winning its OMDL game (whether by event NZP or by a collision in the challenger). Then the extractor must have interacted with at least 2^n copies of the adversary (simulated by $\mathcal{R}$).

Proof. Consider an arbitrary execution of $\mathcal{K}$ with $\mathcal{R}$ in which $\mathcal{K}$ wins the adaptive proof game. We identify each instance of the adversary that $\mathcal{R}$ simulates through the sequence of answers it received to its random oracle calls. For example, the main adversary is identified through $\mathcal{I}_0(c_0, c_1, \ldots, c_n)$: since $\mathcal{K}$ was successful it returned answers to all of the extraction queries (and in particular to all of the random oracle queries). Different random oracle answers imply that $\mathcal{K}$ talks to different copies of the adversary. At the same time, we ignore duplicates: if $\mathcal{K}$ had induced identical executions in two different copies then we count them as one.

Since we consider an execution where $\mathcal{R}$ cannot solve the OMDL problem, by Lemma 2 it holds that event NZP does not occur. Then we can construct the following complete binary tree: the nodes of the tree are of the form $(\mathcal{I}, k)$ where $\mathcal{I}$ is an identifier (for a copy of the adversary) and $1 \leq k \leq n$. The nodes of the tree satisfy the invariant that if $(\mathcal{I}, k)$ is present in the tree, then copy $\mathcal{I}$ must have run up to the point where it made its k-th extraction query and received a valid answer. We set the root of the tree to be $(\mathcal{I}_0, n)$: the main adversary completed so $\mathcal{K}$ must have answered all of the n extraction query of $\mathcal{I}_0$.

For each node $(\mathcal{I}, k)$ that occurs in the tree and for which $k > 1$ we recursively add two children: the first child of $(\mathcal{I}, k)$ is $(\mathcal{I}, k-1)$. If an invocation $\mathcal{I}$ asked k extraction queries then it certainly also asked $k-1$ queries and got answers to all previous ones, so the first child meets the required invariant. For the second child of $(\mathcal{I}, k)$, consider the pair of OMDL challenges (X, A)

that $\mathcal{I}$ made its k-th extraction query about. Since event NZP has not occurred, there must have been some invocation $\mathcal{I}'$ in which a second extraction query was made on the same (X, A) (as otherwise the potential of (X, A) would be one).

With overwhelming probability the other execution $\mathcal{I}'$ must have been about the same $n-k$ first hash queries (and answers) as $\mathcal{I}$, or else the value (X, A) would not occur in the random oracle call made by $\mathcal{I}'$. Furthermore, $\mathcal{I}'$ can only reduce the potential for (X, A) if it has run up to the k-th extraction query, which implies that it received valid answers for the first $k-1$ queries. The invariant is satisfied, so we can add $(\mathcal{I}', k-1)$ as the second child of $(\mathcal{I}, k)$ to our binary tree.

To summarize, if a node $N' = (\mathcal{I}', k')$ is a child of a node $N = (\mathcal{I}, k)$ then $k' = k-1$ and $\mathcal{I}'$ is an execution that received identical answers to the first $n-k$ hash queries as $\mathcal{I}$. Furthermore, the two children of $N = (\mathcal{I}, k)$ correspond to adversaries that differ in the responses to their $n-k+1$-st hash query. Therefore, for any node $N = (\mathcal{I}, k)$ at level $k > 1$, all identities appearing in the subtree rooted at the first child of N shared the first $n-k+1$ hash answers with $\mathcal{I}$; all identities appearing in the subtree rooted at the second child of N shared the first $n-k$ hash answers with $\mathcal{I}$ but got a different answer to the $n-k+1$-st hash query.

The leaves of the binary tree will thus be of the form $(\mathcal{I}_i, 1)$, where each different $\mathcal{I}_i$ corresponds to a different adversary that $\mathcal{R}$ simulates. The tree has 2^n leaves and it only remains to show that the identities in the leaves are pairwise distinct. Suppose that two of these identities $\mathcal{I}$ and $\mathcal{I}'$ in different leaves are identical. There is a unique path in the binary tree that connects these two leaves and a unique node $N = (\mathcal{I}'', k'')$ on this path with a highest index k'' among all nodes on this path. Without loss of generality, the path from N to $(\mathcal{I}, 1)$ passes through the first child of N, and that to $(\mathcal{I}', 1)$ through the second child of N. But this is a contradiction: all identities in the subtree rooted at the first child of N differ from those in the second at least in the response to the $n-k''+1$-st hash query, therefore they are distinct. We conclude that an extractor $\mathcal{K}$ which does not trigger event NZP must access 2^n distinct copies of the adversary. In other words, for any efficient $\mathcal{K}$ the probability of winning the adaptive proof game against our adversary without triggering event NZP is negligible. □

5.7. Proof

Suppose that Fiat-Shamir-Schnorr is an adaptive proof with respect to a group generator GrpSetup and the discrete logarithm relation. Then there is an efficient extractor $\mathcal{K}$ that can win the adaptive proof game with overwhelming probability against $\hat{P}$ for $n = \lambda$ in the groups $\mathbb{G}(\lambda)$ created by GrpSetup. By Lemma 1 the reduction $\mathcal{R}$ is statistically indistinguishable to an adversary from (many copies of) $\hat{P}$, so with overwhelming probability, $\mathcal{K}$ still wins the adaptive proof game against $\mathcal{R}$ (w.r.t. GrpSetup).

Since $\mathcal{K}$ is efficient, from some λ_0 onwards $\mathcal{K}$ makes strictly fewer than 2^λ queries; in particular it launches fewer than 2^λ copies of the adversary. However, by Lemma 3, since $\mathcal{K}$ triggers event NZP in $\mathcal{R}$ whenever it wins the adaptive proof game with fewer than 2^λ copies of the adversary, we conclude that $\mathcal{R}$ (interacting with $\mathcal{K}$ and the adaptive proof game) wins the OMDL game with overwhelming probability w.r.t GrpSetup.

Actually, since the probability that $\mathcal{R}$ solves OMDL is negligibly close to the probability of $\mathcal{K}$ winning the adaptive proof game (for $\lambda > \lambda_0$), our Theorem even holds for extractors with only non-negligible success probability and yields an OMDL adversary with a comparable success probability. This concludes the proof of Theorem 2.

6. Conclusion

It has been known for a long time [5] that Fiat-Shamir proofs can be problematic in adaptive settings. Our notion of adaptive proofs captures the kind of proofs we would like to have in such scenarios (and which are achievable via the Fischlin transformation). Since the publication of the first version of this work [24], the chain-of-proofs technique has been developed further to prove that CCA security of Signed ElGamal cannot be reduced to IND-CPA of plain ElGamal [10] (unless Schnorr proofs leak the witness, in which case any use of Signed ElGamal is questionable). This does not follow directly from the theorems of this work here which do not exclude an extractor tailored specifically to Signed ElGamal that uses the ElGamal ciphertext as well as the attacked PoK. The cited work validates our intuition that adaptive proofs are a useful tool to study the security of real-world public-key constructions.

Furthermore, it is known that allowing only a single, parallel extraction query results in a proof system which is sufficient to construct NM-CPA encryption, and that the common Fiat-Shamir transformation on Sigma-protocols meets this notion [25]. An intriguing question is to study other restriction on the power of malicious prover. In particular such restrictions may lead to new constructions for encryption scheme that satisfy stronger notions than NM-CPA yet weaker than IND-CCA, e.g. those put forth in [25].

Acknowledgements

We thank the current and earlier reviewers of this paper for their helpful comments. This work has been supported in part by the European Union under the Seventh Framework Programme (FP7/2007-2013), grant agreement 609611 (PRACTICE) and the ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO. Marc Fischlin is supported by the Heisenberg grant Fi 940/3-2 of the German Research Foundation (DFG).

7. References

[1] Goldwasser, S., Micali, S. and Rackoff, C.: The Knowledge Complexity of Interactive Proof Systems. SIAM J. Comput., volume 18, number 1, pages 186–208, 1989.

[2] Feige, U. and Shamir, A.: Zero-knowledge proofs of identity. Journal of Cryptology, volume 1, number 2, pages 77–94, 1988.

[3] Tompa, M. and Woll, H.: Random Self-Reducibility and Zero Knowledge Interactive Proofs of Possession of Information, FOCS, pages 472–482, 1987.

[4] Bellare, M. and Goldreich, O.: On Defining Proofs of Knowledge. In: Advances in Cryptology (CRYPTO' 92), LNCS 740, pages 390–420, 1992.

[5] Shoup, V. and Gennaro, R.: Securing Threshold Cryptosystems against Chosen Ciphertext Attack. J. Cryptology, volume 15, number 2, Springer, pages 75-96, 2002.

[6] Bellare, M. and Sahai, A.: Non-Malleable Encryption: Equivalence between Two Notions, and an Indisinguishability-Based Characterization. In: Advances in Cryptology (Crypto '99), LNCS 1666, Springer, pages 519–536, 1999.

[7] Fischlin, M.: Communication-Efficient Non-Interactive Proofs of Knowledge with Online Extractors. In: Proceedings of the 25th annual international cryptology conference on advances in cryptology (CRYPTO '05), pages 152–168, 2005.

[8] Fiat, A. and Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Proceedings on advances in cryptology (CRYPTO '86), pages 186–194, 1986.

[9] Canetti, R., Goldreich, O., Goldwasser, S. and Micali, S.:. Resettable zero-knowledge, STOC, ACM Press, pages 235–244, 2000.

[10] Bernhard, D., Fischlin, M., and Warinschi, B.: On the hardness of proving CCA security of Signed ElGamal. In: PKC 2016, LNCS 9614, pages 47 – 69 , Springer 2016. `eprint.iacr.org/2015/649`.

[11] Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: Proceedings of the 40th annual symposium on foundations of computer science (FOCS '99), pages 543–553, 1999.

[12] M. Naor, M. and Yung, M.: 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.

[13] De Santis, A. and Persiano, G., Zero-Knowledge Proofs of Knowledge Without Interaction (Extended Abstract), FOCS, pages 427-436, 1992.

[14] Jens Groth, J.: Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures. ASIACRYPT, Lecture Notes in Computer Science, volume 4284, Springer, pages 444-459, 2006.

[15] Chase, M. and Lysyanskaya, A.: On Defining Signatures of Knowledge. In: Advances in Cryptology (Crypto '06), LNCS 4117, pages 78–96, 2006.

[16] Dodis, Y., Haralambiev, K., López-Alt, A. and Wichs, D.:, Efficient Public-Key Cryptography in the Presence of Key Leakage, ASIACRYPT, Lecture Notes in Computer Science, volume 6477, Springer, pages 613-631, 2010.

[17] Bellare. M. and Rogaway, P.. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. ACM Conference on Computer and Communications Security, pages 62-73, ACM, 1993.

[18] Wee, H.: Zero Knowledge in the Random Oracle Model, Revisited. ASIACRYPT, Lecture Notes in Computer Science, volume 5912, Springer, pages 417-434, 2009.

[19] Fouque, P. and Pointcheval, D.: Threshold Cryptosystems Secure against Chosen-Ciphertext Attacks. In: Advances in Cryptology (Asiacrypt '01), LNCS 2248, pages 351–368, 2001.

[20] Bernhard, D., Pereira, O., and Warinschi, B:. 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.

[21] Faust, S., Kohlweiss, M., Marson, G., and Venturi, D.: On the Non-malleability of the Fiat-Shamir Transform. Indocrypt, Lecture Notes in Computer Science, Springer, 2012.

[22] Bagherzandi, A., Cheaon, J. H., and Jarecki, S.: Multisignatures Secure under the Discrete Logarithm Assumption and a Generalized Forking Lemma. In: CCS '08, pages 449–458, ACM press 2008.

[23] Bellare, M. and Rogaway, P.: Code-Based Game-Playing Proofs and the Security of Triple Encryption. Full version of November 27, 2008 (Draft 3.0) at `eprint.iacr.org/2004/331`. Originally published in: Advances in Cryptology (Eurocrypt '06), LNCS 4004, Springer, pages 409–426, 2006.

[24] Bernhard, D., Fischlin, M. and Warinschi, B.: Adaptive Proofs of Knowledge in the Random Oracle Model. In: PKC 2015, LNCS 9020, pages 629–649, Springer 2015.

[25] Coretti, S., Dodis, Y., Tackman, B. and Venturi, D.. Non-Malleable Encryption: Simpler, Shorter, Stronger. In: Theory of Cryptography (TCC '16), LNCS 9012, pages 306–335, 2016.