Verifiable Random Oracles

Master's Thesis of

Karsten Diekhoff

at the Department of Informatics
KASTEL – Institute of Information Security and Dependability

Reviewer:	Prof. Dr. Jörn Müller-Quade
Second reviewer:	Prof. Dr. Thorsten Strufe
Advisor:	M. Sc. Michael Klooß

01. September 2020 – 01. June 2021

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

. . . . . .
(Karsten Diekhoff) (Place, Date)

Abstract

The goal of this thesis is to instantiate random oracles without losing security proven in the random oracle model. It is well known that this is not possible with any function family, as first shown by Halevi et al. (IACR'1998). Therefore, we will resort to interaction, but reduce the overhead created by it as well as we can.

In order to keep the interaction needed to a minimum, we introduce a novel ideal model called verifiable random oracle model. In addition to a random oracle RO, this model also includes a verification oracle, which for input (x, h) returns 1 if $\mathrm{RO}(x) = h$ and 0 otherwise. We then continue to present two concrete instantiations for verifiable random oracles, one of which does not use trusted parties. Additionally, we reduce the network overhead (i.e. combined message size needed).

When used with the Fiat-Shamir transformation, these instantiations preserve the simulation-sound extractability property. The prover of the Fiat-Shamir transformation unfortunately loses its non-interactivity. However, the verifier remains non-interactive, as the instantiations for the verify oracle are non-interactive. The proofs for these claims make up a significant part of this thesis.

Zusammenfassung

Ziel dieser Arbeit ist es, Random Oracle zu instanziieren, ohne dabei Sicherheit zu verlieren, die im Random Oracle Modell bewiesen wurde. Das dies mit Funktionsfamilien nicht geht ist eine wohl bekannte Aussage, die zuerst von Halevi et al. (IACR'1998) gezeigt wurde. Wir werden aus diesem Grund auf Interaktion zurückgreifen, aber versuchen, den erzeugten Overhead möglichst zu reduzieren.

Um möglichst wenig zu Interagieren führen wir ein neues ideales Modell mit Namen Verifiable Random Oracle ein. Dieses Modell bietet zusätzlich zum Random Oracle ein Verifikations-Orakel, welches bei Eingabe (x, h) 1 ausgibt, falls $\mathsf{RO}(x) = h$ und anderenfalls 0. Wir stellen danach zwei konkrete Instanziierungen für Verifiable Random Oracle vor, von denen eine keine vertrauenswürdige Party benötigt. Zusätzlich reduzieren wir den Netzwerk-Overhead (also die Gesamtgröße der verwendeten Nachrichten).

Wenn wir unsere Instanziierungen zusammen mit der Fiat-Shamir Transformation verwenden, bleibt die Simulation-Soundness Extractability Eigenschaft erhalten. Der Beweiser der Fiat-Shamir Transformation verliert leider seine nicht-Interaktivität. Der Verifizierer bleibt jedoch Nicht-interaktiv, da die Instanziierungen des Verifikations-Orakels nicht-interaktiv sind. Die Beweise für diese Behauptungen bilden einen signifikanten Teil dieser Arbeit.

Contents

1 Introduction

Our steps to find an instantiation for random oracles are motivated in section 1.1. Afterwards, in section 1.2 we set a concrete, detailed goal. We present existing research with a similar focus in section 1.3, and summarize our contribution in section 1.4.

1.1 Motivation

The random oracle model was first explicitly mentioned in 1993 [3]. Whether or not security in the random oracle model can indicate security in practice has been discussed ever since. In this section we will explain how these discussions, and the problem with random oracles at their core, motivate this thesis.

The Random Oracle Model A random oracle implements a function randomly drawn from all functions with a suitable domain and codomain, usually $\{0,1\}^* \to \{0,1\}^l$ for some length l depending on the context. In the *random oracle model* (ROM), all parties have access to a random oracle. In order to translate a scheme from the ROM to other models, the random oracle in most cases is replaced by a family of hash functions. Because of this, random oracles are often viewed as over-idealisations of the functionality of cryptographic hash functions.

The Discussion about the Random Oracle Model The existence of a random oracle is a very strong assumption. In fact Halevi et al. demonstrated the existence of schemes secure in the ROM but insecure for any family of functions replacing the random oracle [20]. Critics of the ROM therefore argue that the ROM is an unfit heuristic for the security of cryptographic hash functions, and can at most be used as a sanity check. Advocates reply that in 20 years of practice not a single scheme has been successfully attacked as a result of this discrepancy [23]. They argue that all schemes broken because of their use of a random oracle in their proof of security are artificially constructed. For example, the first such scheme [20] has a case that can never occur in the ROM designed to publish the private key.

Programmability On top of that, there are many cases where random oracles have been given additional properties. The most common of these properties is *programmability*. In security reductions for chosen inputs the output of a programmable random oracle can be dictated by an adversary or other designated party. There are cases where hash functions with limited programmability provide security in the standard model [31, 21]. However, programmability cannot be achieved in the standard model in general, which worsens the discrepancy between security in the ROM and standard model.

Our Approach All negative results about bringing security from the ROM to the standard model (e.g. [20, 1, 18]) are based on replacing the random oracle with a family of functions. Fortunately there are other structures that can be used. In this thesis we analyse the use of interactive protocols to sidestep these negative results. This is often undesired, as it entails the use of a public party and the necessity to be online. However this can still be relevant in cases where proven security is of high importance.

The Fiat-Shamir Transformation The Fiat-Shamir transformation [16] was introduced as an efficient identification scheme supporting multiple users and without requiring interaction or shared public keys. Since then, different versions of the Fiat-Shamir transformation have been used in various other scenarios. In fact there are so many results even a quick search yields research about

- security of the scheme, including for post-quantum settings [6, 15, 11, 29, 14, 24],
- signature schemes with additional functionality or properties [4, 2, 26, 27],
- smartcard and passport implementations of the scheme [12, 22],
- variants and similar schemes [17, 5, 28].

Especially the use as efficient non-interactive, zero-knowledge proof system gave the Fiat-Shamir transformation appeal across a wide variety of use cases.

Security of the Fiat-Shamir Transformation A variant of the Fiat-Shamir transformation is secure in the random oracle model [6] with programmability. However, at least some instances are not secure for any function family used[1] [18]. Because of its popularity and lack of proven security in the standard model the Fiat-Shamir transformation is a fitting scenario for us to further investigate. We will use the notion of simulation-sound extractability as a benchmark for our results. All protocols introduced in chapter 3 of this thesis will preserve this property when replacing the random oracle. A formal definition of all important notions can be found in chapter 2.

Optimization One of the important benefits of the Fiat-Shamir transformation is its non-interactivity. We will unfortunately lose this trait, when instantiating the random oracle with an interactive protocol. However, we will keep the interaction as limited as possible. In order to do so, we make it possible to verify values returned by the random oracle without interaction. Formally, we will add a second ideal functionality to the random oracle model, and introduce the novel notion of *verifiable random oracles*. This change helps to reduce the number of calls to the RO. We will further reduce the amount of interaction by reducing the size of messages sent.

[1] Strictly speaking these results concern different notions of security (namely simulation-sound extractability and simulation soundness). The negative result still raises the question whether or not the security of the Fiat-Shamir transformation should be trusted in practice.

1.2 Goal

The goal of this thesis is to find an instantiation of the random oracle used in the simulation-sound extractability proof of the Fiat-Shamir transformation (see section 2.3). In the first step we will introduce verifiable random oracles and use them to replace random oracles. This step is taken to make more efficient instantiations possible. We then try to find an instantiation which preserves the security of the original ROM proof with only small adjustments to the proof. A noteworthy challenge of this goal is the modelling of programmability for the instantiation. These instantiations will contain one or more additional parties, which provide the functionality of the verifiable random oracle. The remainder of this section describes additional desired qualities. After presenting the first instantiation we will adapt it in order to optimize these qualities. We will also prove any claims made about the security of our instantiations.

Weaken Assumptions A common way to describe a random oracle is through a lazy-evaluating, trusted party. This is technically a secure instantiation of a random oracle. However, having a trusted party is an utopian assumption. Therefore, our goal is to find an instantiation with assumptions which are as weak as possible. We will still rely on trusted parties at first, in order to have a simpler example. The combination of programmability and the possibility of compromised parties was one of the harder challenges to figure out and adds the most complexity to the instantiation.

Reducing the Network Overhead As described before, we want to minimize the amount of network overhead created by our instantiation. This not only includes the amount of queries to the verifiable random oracle. It also means minimizing

- the amount of rounds needed to answer one query,
- the combined size of messages sent, and
- the amount of messages between different parties of the verifiable random oracle, and the combined size of these messages.

We will also reason about lower bounds for these quantities, though not in a formal manner.

1.3 Related Work

In general, random oracles cannot be instantiated with function families as proven by Halevi et al. [20]. Apart from this negative result, and the surrounding discussion, there are surprisingly few results about instantiating random oracles. In this section we discuss related research.

Verifiable Random Functions *Verifiable random functions* were first introduced by Micali et al. [25] and first implemented by Dodis [13]. Their core idea is very similar to our notion of verifiable random oracles. Verifiable random functions can be used to instantiate random oracles with the help of an additional party, and there are distributed variants [13] to tackle corruption. However, due to the deterministic nature of verifiable random functions it is not possible to program them.

Simulatable Verifiable Random Functions There are also verifiable random functions that include a simulator which can generate real looking proofs [9]. With these simulated proofs it is possible to convince other parties of values different from the honest computed ones. This can in some scenarios replace programmability, but requires a setup with trusted party.

Trusted Party Implementation There are rare cases, where random oracles can be realized with the use of a trusted party. For example, a trusted platform module could provide a (pseudo) random oracle to the system it is part of [19]. This is interesting, as out instantiations could be used in similar setups.

1.4 Contribution

We introduce a novel notion of *verifiable random oracles* which are a variant of random oracles that output a proof of correct evaluation alongside the random query result. This notion is then used to reduce overhead of queries to instantiations of the oracle. Next, we present instantiations of a verifiable random oracle which preserve security proven in the ROM, even if programmability is required. New proofs of security can be gained by slightly adjusting the ROM proof to account for the different structure of the verifiable random oracle.

Our final instantiation uses four parties and is secure if at most one of them is corrupted (or faulty). This instantiation can be generalized to allow for any number n of corrupted parties, but requires a significant amount of total parties ($3n + 1$). We also demonstrate the capability of our instantiations by providing proofs for the simulation-sound extractability property of the Fiat-Shamir transformation. These proofs only need small adjustment to the respective proof in the ROM. In general, simulation-sound extractability could previously not be transferred to the standard model.

2 Preliminaries

In this chapter we will explain any formal notations and notions needed to understand the content of this thesis. We begin in section 2.1 with the foundations, and introduce the Fiat-Shamir transformation which is central to the thesis afterwards in section 2.2. In section 2.3 we explain and prove an important property of the Fiat-Shamir transformation, namely simulation-sound extractability. This property and its proof is used throughout this thesis as important use case.

2.1 Basic Definitions and Notation

This section explains the mathematical foundations used in this thesis, and covers important notations. Anyone familiar with theoretical computer science, especially cryptography, can likely skip this section.

Security Parameter Throughout the paper we will often implicitly talk about a security parameter λ. The security parameter is a natural number which indicates the asymptotic security of a cryptographic scheme. The higher λ is, the lower the probability of any adversary breaking the security of the scheme. Use of such a parameter, as well as omitting it, is common practice. Usually, cryptographic schemes have a setup algorithm $\mathsf{Setup}(\lambda)$ dependent on λ, that outputs descriptions of all other algorithms of the scheme. We also sometimes chose a concrete member f_λ of a family of e.g. functions F with λ. We can arbitrarily fix the security parameter and with it the algorithms output by Setup. Security properties might still be dependent on λ, e.g. through the use of negligible functions as described in the next paragraph.

Polynomial, Negligible, and Overwhelming We call a function f *polynomial* (in the security parameter), if there is a polynomial P such that $f(\lambda) \leq P(\lambda)$ for any $\lambda \in \mathbb{N}$. We call f *negligible* (in the security parameter), if there exists a point n so that for all $\lambda \geq n$ it holds that $f(\lambda) \leq \frac{1}{P(\lambda)}$. Furthermore, we call f *overwhelming* (in the security parameter), if there exists a negligible function g so that $f(\lambda) \geq 1 - g(\lambda)$ for any $\lambda \in \mathbb{N}$. It will become apparent in the rest of this section how we use these three properties.

Algorithms In our model of computation all algorithms are Turing machines. However we will abstract from that fact, as is common in literature. Algorithms, like functions, have an input and an output. However, algorithms may use randomness which may result in different outputs for identical inputs. Most algorithms in this thesis run in polynomial time, meaning the maximum amount of steps taken is restricted from above by a polynomial in λ. We will call such algorithms *probabilistic polynomial time* (PPT) algorithms. The notation for the result of an algorithm $\mathcal{A}$ with input x is $\mathcal{A}(x)$, similar to the notation of functions. We will define algorithms through step-by-step instructions called their code.

Parties and Interaction All entities taking part in a cryptographic scheme are modelled as parties. A party knows one or more algorithms which they can execute, and has a memory shared between all of them. We model the memory as a map and write $\mathbf{mem}[x]$ for access to storage location x. Parties can provide interfaces to other parties to define possible interactions. Other parties can call such an interface by sending a message. The providing party then executes an algorithm linked to the interface with the input specified by the call message, and sends a message with the answer back to the caller. We often do not distinguish between a party, an interface, and an algorithm, unless it is necessary to prevent ambiguity. Otherwise we write $\mathcal{P}.\mathcal{A}(x)$ for the interface executing algorithm $\mathcal{A}$ with input x on party $\mathcal{P}$.

Protocols We also allow two algorithms of different parties to send messages between each other before making an output. Behaviour of such codependent algorithms is defined in protocols. The sequence of all messages of a protocol is called a *transcript*. We write $\langle \mathcal{A}(x), \mathcal{B}(y) \rangle$ for the output of the protocol between algorithms $\mathcal{A}$ and $\mathcal{B}$ with inputs x and y respectively. This notation will also be used with sets of algorithms, simply meaning all algorithms of the set take part in the protocol. During a protocol, we also write $x \leftarrow \mathcal{B}(y)$ meaning y is sent to $\mathcal{B}$, and the answer is stored in x. We use $\leftarrow$ instead of = to clarify the call is part of a protocol.

Game Based Security Notions of security are often described with the help of games. Games define a situation in which multiple parties interact. They include an arbitrary but fixed adversary and define a condition in which the adversary wins. We will present games in the form of algorithms executing the specified interaction, returning 1 if the adversary won, and 0 otherwise. At the beginning of a game, the randomness of all participating parties is uniformly chosen. We denote a uniform choice for x from elements of the set S as $x \xleftarrow{\$} S$. A scheme is secure if for all adversaries the probability (taken over all randomness) to win the game is negligible.

Game Hops Sometimes it is easier to prove security for a game that is slightly different then the one stated in the security notion. In such a case, one can make an additional proof showing that the probability of different outcomes in the two games is negligible. This strategy is called making game hops, as you can create a sequence of these games and "hop" from one to the next. We use game hops to take complexity out of long proofs.

Black Box Access and Oracles The sort of interaction, where a party provides an input and receives an output without knowing the process in between is called *black box access*. For example, parties providing interfaces provide black box access to the associated algorithm. We denote that $\mathcal{A}$ has black box access to $\mathcal{B}$ as $\mathcal{A}^{\mathcal{B}}$. Black boxes can be described with the properties of the provided functionality rather than concrete code. This is possible, even if no code has the specified properties. For this reason, we call entities providing black box access to such a functionality an *oracle*. Oracles are useful tools in security proofs.

Rewindability In addition to regular access to a black box, a party can have *rewinding access* to a black box. This allows a party to undo steps of the execution of the black box (without it noticing or remembering). In this thesis, we model this ability by allowing to restart the interaction. These definitions are equivalent, as the interaction can be replayed

from the start to any point[1]. The intention is to have control over the algorithm in the black box without knowing its code.

Random Oracles and Programmability The most common oracle in cryptography is the *random oracle*. Random oracles behave like a function randomly drawn from the set of all functions with a suitable domain and codomain, and are often used to idealize cryptographic hash functions. In the *random oracle model* (ROM) all parties have access to the same random oracle. Sometimes, a party is allowed to control a random oracle by dictating its output. We refer to this as *programming* the oracle. In most cases a programming party acts exactly like the random oracle with only few exceptions. Therefore, we specify the behaviour of a programmed random oracle by specifying the difference to the non-programmed random oracle. We write $\mathsf{RO}.\mathsf{prgm}(x, h)$ if the result of RO on input x is changed to h.

Trusted Parties and Corruption Some security notions can give the adversary the ability to *corrupt* one or more parties. Corruption happens once at the beginning of a game and follows rules defined by the security notion. The only rules used in this paper limit the number of corrupted parties. A corrupted party shares all information with the adversary. It does not make decisions, but passes all messages to the adversary, who can dictate the answers. Parties which cannot be corrupted are called *trusted* parties. It is possible, but not necessary for an untrusted party to be corrupted.

Bottom and Timeouts An algorithm can return a special symbol $\perp$ (named bottom) to signal it is not able to make a successful output. When defining algorithms, we often ignore the possibility of receiving $\perp$ as result of calls. If an algorithm would do any computation with $\perp$, it implicitly returns $\perp$ as well, or 0 if it is some kind of verification algorithm. This makes algorithms easier to understand by focusing on their intended behaviour. We also want to consider the possibility of parties not giving an answer due to being corrupted or otherwise faulty. In practice such behaviour would be handled by timeouts. We instead model receiving no answer as another special symbol **timeout**. The error handling for **timeout** works the same as for $\perp$.

2.2 The Fiat-Shamir Transformation

This section introduces sigma protocols and zero-knowledge proofs. It explain how the Fiat-Shamir transformation changes sigma protocols to be zero-knowledge.

Sigma Protocols Sigma protocols (or Σ-protocols) [10] are a way for a prover P to convince a verifier V of a statement x. The important part is that P does not give V enough information to prove x to other parties. In order to achieve this, P gains an additional input w called witness for x. In the protocol, P sends a commitment a to V to receive a challenge c. P will then send a response z to V, who outputs either 1 to accept the proof, or 0 to reject the proof. The reason, why V cannot reuse the proof to convince other parties is that these parties will most likely choose another challenge c' that V is not able to answer. Formally,

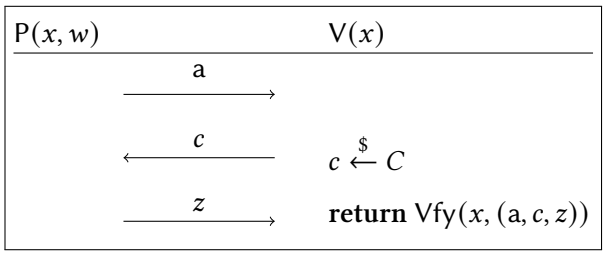
[1] This may take polynomially more time then undoing one or more steps, which is of no concern in this paper.

this property is described as a simulator Sim with the ability to create real looking proof conversations.

Definition 1 (Sigma Protocol)**.** *Let*

- $R \subseteq X \times W$ *be a witness relation,*
- $L_R = \{x \in X \mid \exists w \in W.(x, w) \in R\}$ *be the associated language,*
- P *and* V *be interactive, stateful* PPT *algorithms, and*
- C *be a set called challenge space.*

For public input $x \in X$*, and input* $w \in W$ *to* P*, the interaction between the two algorithms has the following form:*



where Vfy *takes a potential word* x *and a transcript as inputs and maps them to* 1 *or* 0*, accepting or rejecting the proof respectively. The tuple* (P, V) *is called* Σ-protocol *for* R*, if the following properties are fulfilled:*

- Completeness: *For all input values* $(x, w) \in R$ *the honest prover will always convince the honest verifier i.e.* $\langle \mathsf{P}(x, w), \mathsf{V}(x) \rangle = 1$.
- Special Soundness: *There exists a polynomial-time extractor* Ext *that given a word* x *and two accepting transcripts* (a, c, z) *and* (a, c', z') *with equal commitments but different challenges* $(c \neq c')$ *always outputs a witness* $w \in W$ *such that* $(x, w) \in R$.
- Special Honest-Verifier Zero-Knowledge: *There exists a* PPT *simulator* Sim *that, given* $x \in X$ *and* $c \in C$*, always outputs* a *and* z *such that* (a, c, z) *has the same distribution as transcripts between the honest* P *and* V.

We will also often require Σ-protocols to have a super-polynomial challenge space and the *strict soundness* property as introduced by Unruh [30], meaning for all x, a, and c there is at most one verifying z i.e.

$$\forall x, \mathsf{a}, c.\mathsf{Vfy}(x, (\mathsf{a}, c, z)) \wedge \mathsf{Vfy}(x, (\mathsf{a}, c, z')) \Rightarrow z = z'.$$

Zero-Knowledge One potential shortcoming of the Σ-protocol is that the simulator only works for the honest verifier. Use of malicious verifiers may break the zero-knowledge property and give additional information about x (other than $x \in L_R$). A protocol needs to have a simulator for each verifier in order to be zero-knowledge.

Definition 2 (Zero-Knowledge)**.** *Let*

- *R and L_R be defined as in definition 1, and*
- P *and* V *be interactive, stateful* PPT *algorithms with the completeness property of definition 1.*

(P, V) *is called a* zero-knowledge *proof system* (ZK) *if for each malicious* PPT *verifier* V^* *there exists a* PPT *simulator* $\mathsf{Sim}_{\mathsf{V}^*}$ *that, given x, always outputs a transcript that has the same distribution as transcripts between* P *and* V^**.*

The upcoming Fiat-Shamir transformation changes a Σ-protocol in a way that makes it zero-knowledge in the ROM when giving $\mathsf{Sim}_{\mathsf{V}^*}$ programming access to the random oracle.

The Fiat-Shamir Transformation Fiat and Shamir were the first to transform a Σ-protocol to create an one-round (non-interactive) zero-knowledge proof system [16]. Since then, some variants and similar transformations like the Fischlin transformation [17] have been published. In this thesis we will use the strong Fiat-Shamir transformation as described by Bernhard et al. [6]. It replaces the challenge from the Σ-protocol by querying (x, a) to a random oracle, instead of V drawing it. As P can now compute the challenge itself, no interaction with V is needed anymore.

Definition 3 (Fiat-Shamir Transformation)**.** *Let*

- $\Sigma = (\mathsf{P}_\Sigma, \mathsf{V}_\Sigma)$ *be a Σ-protocol,*
- Sim_Σ *be the special honest-verifier zero-knowledge simulator of Σ,*
- $\mathsf{RO} : X \times A \to C$ *be a random oracle, and*
- Vfy *be the algorithm* V_Σ *uses to verify.*

The Fiat-Shamir transformation of Σ ($FS(\Sigma)$) *in the random oracle model is defined by these two polynomial time algorithms:*

$\mathsf{P}(x, w)$*:*

1 : $\mathsf{a} \leftarrow \mathsf{P}_\Sigma(x, w)$
2 : $c = \mathsf{RO}(x, \mathsf{a})$
3 : $z \leftarrow \mathsf{P}_\Sigma(c)$
4 : **return** (a, c, z)

$\mathsf{V}(x, (\mathsf{a}, c, z))$*:*

1 : **if** $c \neq \mathsf{RO}(x, \mathsf{a})$*:*
2 : **return** 0
3 : **return** $\mathsf{Vfy}(x, (\mathsf{a}, c, z))$

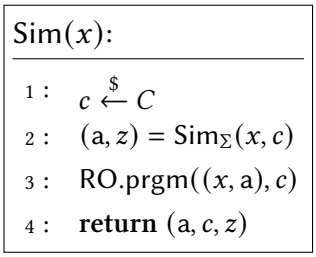


Figure 2.1: The zero-knowledge simulator for the Fiat-Shamir transformation.

In definition 3, P and V are not described through their interaction, as there is only a single message from P to V. Instead, V receives the output of P as an input. This non-interactive nature of the Fiat-Shamir transformation is an important advantage over interactive schemes, as it can be utilized in more settings. It is also the reason why it is ZK, as the verifier does not make decisions anymore, and therefore malicious verifiers cannot diverge from the honest one. However, the simulator needs programming access to RO in order to work, as shown in figure 2.1.

2.3 Simulation-Sound Extractability

A very strong property of the Fiat-Shamir transformation is *simulation-sound extractability* [6]. It is a combination of the two properties *simulation soundness* and *extractability*. Here are intuitive descriptions of them, before we go into the formal definition of simulation-sound extractability:

Extractability. A ZK proof system is *extractable* if there exists an algorithm called extractor that can output witnesses using access to any successful malicious prover. You may think about extractability this way: Any successful malicious prover has the ability to compute witnesses.

Simulation Soundness. A ZK proof system is *simulation sound*, if no PPT adversary with access to the ZK-simulator can forge proofs for false statements with more then negligible probability.

Simulation-sound extractability combines these properties as follows: The extractor of the extractability definition uses an adversary for simulation soundness instead of a malicious prover. The formal definition is phrased in a way that the adversary only wins if it is not possible to extract a witness from it:

Definition 4 (Simulation-Sound Extractability in the ROM)**.** *Let*

- RO *be a random oracle,*
- Π *be a* ZK *proof system, and*
- Sim *be the* ZK*-simulator of* Π*.*

Π *is* simulation-sound extractable (SSE) *in the random oracle model if there exists an extractor* Ext *for which no* PPT *adversary* $\mathcal{A}$ *can win the following game with non-negligible probability, and the expected runtime of* Ext *in the game is polynomial.*

Game SSE*:*

1 : $(x, \pi) = \mathcal{A}^{\mathsf{Sim},\mathsf{RO}}()$
2 : $M = \{(x, \pi) \mid \mathsf{Sim}(x)$ *returned* π *to* $\mathcal{A}\}$
3 : $w = \mathsf{Ext}^{\mathcal{A},\mathsf{RO}}(x, \pi)$
4 : **return** $\mathsf{Vfy}(x, \pi) \wedge (x, \pi) \notin M \wedge (x, w) \notin R$

In this game, Ext *has rewinding access to* $\mathcal{A}$ *and can program* RO*. Additionally, whenever* $\mathcal{A}$ *is being rewound,* Sim *rewinds with it, and both algorithms use the same randomness as before.*

A successful adversary needs to forge a new proof for some x in a way that the extractor is unable to find a witness for x. Again, you might think about this property the following way: There is no successful attack using a zero knowledge simulator that could not also find a witness. This directly implies *soundness*, as finding a witness w for a value not in the language $x \notin L$ is not possible.

Simulation-Sound Extractability of the Fiat-Shamir Transformation Over the course of the thesis, we will refer back to the following theorem 1 and its proof, as we use it as an example for the application of our results. To be more precise, we will replace the random oracle in the proof with our verifiable random oracle instantiations and prove the preservation of simulation-sound extractability. The main ideas for the proof come from the forking lemmas of Bellare et al. [2] and Bootle et al. [8]. However, as there are many similar, but non-equivalent notions of simulation-extractability, it was necessary to create a detailed proof for the version we use.

Theorem 1. *The Fiat-Shamir transformation of a strictly sound* Σ*-protocol with super-polynomial challenge space is simulation-sound extractable in the random oracle model.*

The Extractor The extractor Ext used in the following proof utilizes the special soundness extractor Ext_Σ required of the underlying Σ-protocol. When given a statement x with two accepting transcripts with the same commitment and different challenges, Ext_Σ outputs a witness for x. The first of the transcripts used by Ext to obtain the witness is the one the adversary outputs in the initial run. The second transcript is gained from additional executions of $\mathcal{A}$ after carefully reprogramming the random oracle as described in figure 2.2. For technical reasons of the proof of theorem 1, Ext tries at most $\sqrt{|C|}$ times to find a second accepting transcript. The set M contains all outputs of Sim to $\mathcal{A}$ in the initial run, as in **Game** SSE. Ext can compute M by simply restarting $\mathcal{A}$ and Sim and executing the initial run again. Before every execution of line 6, Ext restarts $\mathcal{A}$ and Sim, so the only difference to the initial run is the value of $\mathsf{RO}(x, \mathrm{a})$. Therefore, all runs of $\mathcal{A}$ diverge only after $\mathsf{RO}(x, \mathrm{a})$ is queried for the first time. Ext finds the required second transcript by executing $\mathcal{A}$ with different values for $\mathsf{RO}(x, \mathrm{a})$ until a second challenge is answered successfully. It remains to be shown that Ext runs in expected polynomial time and the probability of any $\mathcal{A}$ succeeding is negligible.

$\mathsf{Ext}^{\mathcal{A},\mathsf{RO}}(x, (\mathsf{a}, c, z))$:

1 : **if** $\neg \mathsf{V}(x, (\mathsf{a}, c, z)) \vee (x, (\mathsf{a}, c, z)) \in M$:
2 : **return** $\perp$ # $\mathcal{A}$ failed
3 : **do** (max $\sqrt{|C|}$ times):
4 : $\tilde{c} \xleftarrow{\$} C$
5 : $\mathsf{RO}.\mathsf{prgm}((x, \mathsf{a}), \tilde{c})$
6 : $(x', (\mathsf{a}', c', z')) = \mathcal{A}^{\mathsf{Sim},\mathsf{RO}}()$
7 : **until** $x' = x \wedge \mathsf{a}' = \mathsf{a} \wedge \mathsf{V}(x', (\mathsf{a}', c', z'))$
8 : **return** $\mathsf{Ext}_{\Sigma}(x, (\mathsf{a}, c, z), (\mathsf{a}, c', z'))$

Figure 2.2: The extractor used in the SSE proof for the Fiat-Shamir transformation.

Some additional Setup In the two following proofs, we will fix the randomness used by $\mathcal{A}$ and Sim to arbitrary values. All probabilities are taken over the randomness of RO and the choice of $\tilde{c}$ in line 4 of the extractor. Let c_i be the result of the i-th query to RO. Because RO is uniformly random, its randomness can be depicted as a sequence of unique random choices $\mathcal{R} = (c_1, \ldots, c_{\mathbf{P}})$, one for each query[2]. $\mathbf{P}$ is the maximum number of queries made to RO, and is polynomial, as all algorithms able to send queries are polynomially time bound. For any choice c_i, we can then split the probability of any event E by explicitly going over one or more of the $\mathbf{P}$ choices:

$$\Pr[\mathsf{E}] = \sum_{c \in C} \Pr[c_i = c] \cdot \Pr[\mathsf{E} \mid c_i = c]$$

We will do so later for all but one i, which is why we introduce some notation for it now. Let

- $\mathcal{R}_i = (c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_{\mathbf{P}})$ be the randomness of RO except for the choice of c_i,
- $\mathsf{E}^{\mathsf{init}}_{\mathcal{A}}$ is successful in the initial run of **Game** SSE i.e. $\mathsf{Vfy}(x, \pi) \wedge (x, \pi) \notin M$ (see definition 4), and
- $\mathsf{E}^{\mathsf{init}}_{\mathcal{A},i}$ be the event of $\mathcal{A}$ being successful in the initial run and answering challenge c_i.

Lemma 1. Ext *runs in expected polynomial time.*

Proof. The maximum amount of time any line of Ext takes to execute is polynomial. With exception of the loop, every line is executed at most once. Therefore, it suffices to prove that the expected number of loop iterations $\mathbb{E}[\mathsf{I}]$ is polynomial. The loop is only reached if $\mathcal{A}$ is successful in the initial run. We can therefore state

$$\mathbb{E}[\mathsf{I}] = \Pr\left[\mathsf{E}^{\mathsf{init}}_{\mathcal{A}}\right] \cdot \mathbb{E}\left[\mathsf{I} \,\middle|\, \mathsf{E}^{\mathsf{init}}_{\mathcal{A}}\right].$$

[2]If a query (x, a) is made to RO multiple times, only the first is represented by a choice c_i.

We can split this expression into a sum over the index of the successfully answered challenge c_i, and then over all randomness of RO except for c_i[3]:

$$\begin{aligned}\mathbb{E}[\mathsf{I}] &= \sum_{i=1}^{\mathbf{P}} \Pr\left[\mathsf{E}^{\text{init}}_{\mathcal{A},i}\right] \cdot \mathbb{E}\left[\mathsf{I} \;\middle|\; \mathsf{E}^{\text{init}}_{\mathcal{A},i}\right] \\ &= \sum_{i=1}^{\mathbf{P}} \sum_{\mathbf{C}_i \in C^{\mathbf{P}-1}} \Pr[\mathcal{R}_i = \mathbf{C}_i] \cdot \Pr\left[\mathsf{E}^{\text{init}}_{\mathcal{A},i} \;\middle|\; \mathcal{R}_i = \mathbf{C}_i\right] \cdot \mathbb{E}\left[\mathsf{I} \;\middle|\; \mathcal{R}_i = \mathbf{C}_i \wedge \mathsf{E}^{\text{init}}_{\mathcal{A},i}\right] \end{aligned} \tag{2.1}$$

The value $\mathbb{E}\left[\mathsf{I} \;\middle|\; \mathcal{R}_i = \mathbf{C}_i \wedge \mathsf{E}^{\text{init}}_{\mathcal{A},i}\right]$ is the expected number of loop iterations in the case where most randomness of RO is fixed ($\mathcal{R}_i = \mathbf{C}_i$) and $\mathcal{A}$ successfully answered c_i in the initial run ($\mathsf{E}^{\text{init}}_{\mathcal{A},i}$). Executions of $\mathcal{A}$ initiated by Ext will use the same randomness as the initial run. The only difference is that c_i gets programmed randomly. This situation can be described by $\mathcal{R}_i = \mathbf{C}_i$. Part of the loop exit condition ($x' = x \wedge \mathsf{a}' = \mathsf{a}$) can be interpreted as $\mathcal{A}$ answering c_i. The reason for this is the fixed randomness. We know that runs can only diverge after the i-th query to RO has been answered with c_i, as this is the first step in the execution with unfixed randomness. The rest of the exit condition, i.e. $\mathsf{V}(x', (\mathsf{a}', c', z'))$, ensures that the reply to c_i was successful. As there is a second break condition (max $\sqrt{|C|}$ iterations), we conclude that in each iteration, the probability to exit the loop is greater or equal to $\Pr\left[\mathsf{E}^{\text{init}}_{\mathcal{A},i} \;\middle|\; \mathcal{R}_i = \mathbf{C}_i\right]$. The expected number of loop iterations can be expressed recursively using the probability to exit the loop by

$$\mathbb{E}\left[\mathsf{I} \;\middle|\; \mathcal{R}_i = \mathbf{C}_i \wedge \mathsf{E}^{\text{init}}_{\mathcal{A},i}\right] \leq 1 + \left(1 - \Pr\left[\mathsf{E}^{\text{init}}_{\mathcal{A},i} \;\middle|\; \mathcal{R}_i = \mathbf{C}_i\right]\right) \cdot \mathbb{E}\left[\mathsf{I} \;\middle|\; \mathcal{R}_i = \mathbf{C}_i \wedge \mathsf{E}^{\text{init}}_{\mathcal{A},i}\right],$$

which simplifies to

$$\mathbb{E}\left[\mathsf{I} \;\middle|\; \mathcal{R}_i = \mathbf{C}_i \wedge \mathsf{E}^{\text{init}}_{\mathcal{A},i}\right] \leq \frac{1}{\Pr\left[\mathsf{E}^{\text{init}}_{\mathcal{A},i} \;\middle|\; \mathcal{R}_i = \mathbf{C}_i\right]}.$$

By substituting this result into equation 2.1 we obtain

$$\begin{aligned}\mathbb{E}[\mathsf{I}] &\leq \sum_{i=1}^{\mathbf{P}} \sum_{\mathbf{C}_i \in C^{\mathbf{P}-1}} \Pr[\mathcal{R}_i = \mathbf{C}_i] \cdot \Pr\left[\mathsf{E}^{\text{init}}_{\mathcal{A},i} \;\middle|\; \mathcal{R}_i = \mathbf{C}_i\right] \cdot \frac{1}{\Pr\left[\mathsf{E}^{\text{init}}_{\mathcal{A},i} \;\middle|\; \mathcal{R}_i = \mathbf{C}_i\right]} \\ &= \sum_{i=1}^{\mathbf{P}} \sum_{\mathbf{C}_i \in C^{\mathbf{P}-1}} \Pr[\mathcal{R}_i = \mathbf{C}_i] \\ &= \sum_{i=1}^{\mathbf{P}} 1 \\ &= \mathbf{P}.\end{aligned}$$

As $\mathbf{P}$ is polynomial, so is the expected runtime of Ext. □

[3]This argument makes the implicit assumption that c used by the answer of $\mathcal{A}$ in the initial run was received from RO and not guessed. We will attend to the corner case, in which this is not the case, after we finish the rest of the proof for theorem 1 at the end of this section.

Lemma 2. *The probability of $\mathcal{A}$ succeeding in* ***Game*** SSE *is negligible.*

Proof. In order for $\mathcal{A}$ to win **Game** SSE it has to succeed the initial run, and Ext has to fail in finding a witness. Ext can only fail, if

1. the loop is exited because $\sqrt{|C|}$ iterations have been reached, or
2. the loop is exited with "colliding challenges" $\tilde{c} = c$.

Let us consider for a moment an extractor Ext′ that does not stop after $\sqrt{|C|}$ loop iterations. The prior proof implies that the expected number of loop iterations $\mathbb{E}[\mathsf{I}']$ for Ext′ is also polynomial. By Markov's inequality we can derive that

$$\Pr\left[\mathsf{I}' \geq \sqrt{|C|}\right] \leq \frac{\mathbb{E}[\mathsf{I}']}{\sqrt{|C|}}.$$

This is the exact probability of Ext exiting the loop because $\sqrt{|C|}$ iterations have been reached. It is negligible, as $\mathbb{E}[\mathsf{I}']$ is polynomial and $\sqrt{|C|}$ is super-polynomial. It remains to be shown that the probability of a collision $\tilde{c} = c$ is negligible. This probability is at most the probability of drawing c out of C with $\sqrt{|C|}$ tries. As each try has a success chance of $\frac{1}{|C|}$ we get a probability of at most

$$\frac{\sqrt{|C|}}{|C|} = \frac{1}{|C|}$$

for such a collision. Because the challenge space is super-polynomial, we can derive that this probability is also negligible. Therefore, the probability of $\mathcal{A}$ winning **Game** SSE is negligible. □

In the proofs of both lemmas, we made the implicit assumption, that $\mathcal{A}$ only answers with challenges it queried from RO. As $\mathcal{A}$ has to guess a challenge when not querying it, the probability of $\mathcal{A}$ is $\frac{1}{|C|}$, which in negligible as C is super-polynomial. We can conclude that $FS(\Sigma)$ is simulation-sound extractable in the random oracle model for all strictly sound sigma protocols Σ with super-polynomial challenge space C, as stated by theorem 1.

3 Verifiable Random Oracles

In this chapter we will introduce the novel notion of verifiable random oracles, and demonstrate how they can replace random oracles. Section 3.1 explains the idea behind the ideal verifiable random oracle model and defines it. We then present a framework for instantiation in section 3.2 and multiple security properties. Afterwards, in section 3.3, we instantiate a verifiable random oracle with the use of a trusted party, and in section 3.4, we present an instantiation without need for a trusted party. For all instantiations, we also provide a proof that they preserve simulation-sound extractability when instantiating a random oracle.

3.1 The Verifiable Random Oracle Model

As explained in section 1.1, we want to find a way to instantiate random oracles while preserving security. In our case, security means simulation-sound extractability of the Fiat-Shamir transformation. However, we are confident that our instantiations will work in other contexts as well. In this section, we will recap the motivation for the new notion of verifiable random oracles, and define their ideal functionality. We will also prove that the Fiat-Shamir transformation remains simulation-sound extractable in the verifiable random oracle model in order to prepare later proofs for instantiations.

Motivation Recap In general, it is not possible to instantiate random oracles with any function family [20]. We can bypass this negative result by using interaction in instantiations. This is unfortunate, as our use case (the Fiat-Shamir transformation) and many other scenarios aim to be non-interactive. Even though it is not possible to eliminate the interaction completely, we can reduce it by providing additional functionality to verify random oracle values without a query. For the Fiat-Shamir transformation, this implies that we can at least make the verifier non-interactive. As a prover might have the same proof verified by multiple verifiers, this can significantly reduce the network interaction of the scheme, depending on the use case. Formally defining a random oracle with this functionality leads to the new notion of verifiable random oracles.

Changes to the Random Oracle Model In the *verifiable random oracle model*, all parties gain access to these two oracles:

- $\mathsf{H}_{\mathsf{VRO}}$, which acts exactly like a random oracle, and
- $\mathsf{Vfy}_{\mathsf{VRO}}$, which checks if $\mathsf{H}_{\mathsf{VRO}}$ maps a given query value to a given result value.

We will later make sure to instantiate $\mathsf{Vfy}_{\mathsf{VRO}}$ non-interactively (meaning with a function all parties can compute locally). The instantiation of $\mathsf{H}_{\mathsf{VRO}}$ has to include interaction though, as mentioned in the last paragraph.

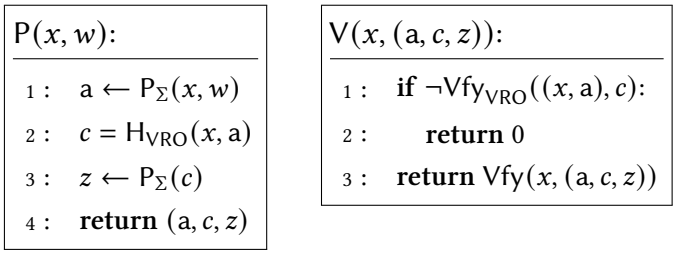


Figure 3.1: The prover and verifier of the Fiat-Shamir transformation, adapted to be used with the ideal verifiable random oracle.

Definition 5 (Ideal Verifiable Random Oracle)**.** *Let* $\mathsf{H}_{\mathsf{VRO}} : D \to C$ *be a* (programmable) random oracle*, and* $\mathsf{Vfy}_{\mathsf{VRO}} : D \times C \to \{1, 0\}$ *be the function defined by*

$$\mathsf{Vfy}_{\mathsf{VRO}}(x, h) = \begin{cases} 1 & \textbf{if } \mathsf{H}_{\mathsf{VRO}}(x) = h \\ 0 & \textbf{else} \end{cases} .$$

We call the tuple $\left(\mathsf{H}_{\mathsf{VRO}}, \mathsf{Vfy}_{\mathsf{VRO}}\right)$ *an* ideal (programmable) verifiable random oracle.

Note that in this ideal scenario, programming $\mathsf{H}_{\mathsf{VRO}}$ automatically changes the output of $\mathsf{Vfy}_{\mathsf{VRO}}$ accordingly, by definition.

Adapting Schemes for the Verifiable Random Oracle When translating schemes from the random oracle model to the verifiable random oracle model, one must decide which RO queries to replace with $\mathsf{H}_{\mathsf{VRO}}$ queries, and which with $\mathsf{Vfy}_{\mathsf{VRO}}$ computations. It might also be necessary to send a VRO query result to a party so that the party does not have to query it, but can verify it instead. These changes require decisions and can unfortunately not be automated. However, this should be straightforward in most cases. In the Fiat-Shamir transformation (see definition 3), we only need to replace

- the explicit comparison $c \neq \mathsf{RO}(x, \mathsf{a})$ in the verifier with $\mathsf{Vfy}_{\mathsf{VRO}}((x, \mathsf{a}), c)$, and
- the occurrence of RO in the prover with $\mathsf{H}_{\mathsf{VRO}}$,

as shown in 3.1.

Security in the Verifiable Random Oracle Model Because schemes have to be translated manually, as argued in the last paragraph, we also need to adjust any proofs given in the random oracle model by hand. This includes completeness, which is very easy to prove in our case: P has not really changed, we only renamed RO to $\mathsf{H}_{\mathsf{VRO}}$. Their definition is the same. V uses $\mathsf{Vfy}_{\mathsf{VRO}}$ instead of an explicit check. However, the check being replaced is the exact definition of $\mathsf{Vfy}_{\mathsf{VRO}}$, so nothing changes. Overall, the scheme remains complete. We also need to adjust the simulation-sound extractability proof:

Theorem 2. *The Fiat-Shamir transformation of a strictly sound* Σ*-protocol with super-polynomial challenge space is simulation-sound extractable in the verifiable random oracle model.*

Game SSE and Ext are both the same as in theorem 1, with the following exceptions:

- The prover and verifier are updated as described above,
- any calls to RO.prgm made by Ext are replaced by VRO.prgm, and
- $\mathcal{A}$ and Ext now have access to $\mathsf{H}_{\mathsf{VRO}}$ and $\mathsf{Vfy}_{\mathsf{VRO}}$ instead of RO.

After a game hop, the following proof reflects the proof of theorem 1:

Proof. Let

- **Game 0** be **Game** SSE in the VROM, and
- **Game 1** be the same game, except that $\mathsf{Vfy}_{\mathsf{VRO}}(x, \mathsf{a}, c)$ only returns 1 if (x, a) was queried to $\mathsf{H}_{\mathsf{VRO}}$ (and $\mathsf{H}_{\mathsf{VRO}}(x, \mathsf{a}) = c$).

We will now show that $\mathcal{A}$ behaves the same in both games, except for a negligible probability. $\mathcal{A}$ is only able to behave differently if it finds x, a, and c such that $\mathsf{Vfy}_{\mathsf{VRO}}(x, \mathsf{a}, c) = 1$ without querying $\mathsf{H}_{\mathsf{VRO}}(x, \mathsf{a}) = c$. As this is only true for one c out of the super-polynomial challenge space, $\mathcal{A}$ has only a negligible probability of guessing correct even with a polynomial amount of tries. As the probability of $\mathcal{A}$ behaving differently is at most negligible, so is the difference in the success probability of $\mathcal{A}$ in the two games. In **Game 1** we can use Ext and the argumentation for theorem 1:

- If the initial run is successful, Ext fixes randomness in a way that one of the challenges queried is for (x, a) from the initial run.
- When $\mathcal{A}$ is successful, it answers one of the **P** challenges queried, where **P** is bound by the polynomial runtime of $\mathcal{A}$.
- In each loop iteration, the probability that $\mathcal{A}$ answers the challenge for (x, a) is therefore $\frac{1}{|C|}$.
- This implies the expected amount of loop iterations is $|C|$, making the expected runtime of Ext polynomial.
- Using this and Markov's inequality yields that the probability of exiting the loop because $\sqrt{|C|}$ tries have been reached is negligible.
- The probability of drawing the same challenge c as in the initial run for (x, a) in $\sqrt{|C|}$ tries is also negligible.
- These are the only two ways $\mathcal{A}$ can win **Game 1**, so the probability of that is negligible as well.

We can conclude that the Fiat-Shamir transformation is simulation-sound extractable in the VROM. □

The proof is only outlined here, because it follows the exact same argumentation as the proof for theorem 1.

3.2 A Framework for Instantiation

In this section we define a framework for the instantiation of verifiable random oracles. The main advantage for such a framework over directly defining concrete instantiation is the unification of security proofs. We define what algorithms and protocols an instantiations consists of, and present properties for such instantiations. Afterwards, we provide a proof that any instantiation with these properties preserve simulation-sound extractability of the Fiat-Shamir transformation. It then suffices to prove these properties for our two instantiations in sections 3.3 and 3.4.

Proof of Correct Evaluation There is a fundamental problem when trying to instantiate verifiable random oracles: As explained before, the goal is to instantiate $\mathsf{Vfy}_{\mathsf{VRO}}$ with an algorithm $\mathcal{V}$ that can be executed locally by anyone. At the same time, we want to be able to program the implementation of $\mathsf{H}_{\mathsf{VRO}}$. There is no $\mathcal{V}$ that can accomplish this required behaviour without additional information. We will provide this information in the form of a *proof of correct evaluation* $\sigma \in S$. This value will be a side output of instantiations of $\mathsf{H}_{\mathsf{VRO}}$ and an additional input to $\mathcal{V}$.

Allowing for more complex Interaction As discussed before, the instantiation of $\mathsf{H}_{\mathsf{VRO}}$ will be interactive. For this reason, it consists of a query algorithm $\mathcal{Q}$ that can be executed by anyone, and one or more algorithms $\mathcal{H}_i$ each provided by a different party $\mathcal{P}_i$. We instantiate $\mathsf{H}_{\mathsf{VRO}}$ with the protocol $\langle \mathcal{Q}, \mathcal{P}_1.\mathcal{H}_1, \ldots, \mathcal{P}_n.\mathcal{H}_n \rangle$ where n is the amount of parties specified by the concrete instantiation. For brevity, we will write $\mathcal{H}_i$ instead of $\mathcal{P}_i.\mathcal{H}_i$. The types of these new algorithms are:

$$\langle \mathcal{Q}(\cdot), \mathcal{H}_1, \ldots, \mathcal{H}_n \rangle : D \to C \times S$$
$$\mathcal{V} : D \times C \times S \to \{1, 0\}$$

It is often helpful to talk about the collective as a single algorithm $\mathcal{H} = (\mathcal{H}_1, \ldots, \mathcal{H}_n)$. In those cases you may think of $\mathcal{H}$ as wrapper which only passes messages between $\mathcal{Q}$ and all $\mathcal{H}_i$.

A Setup Algorithm Concrete instantiations may require a setup e.g. to generate keys. Therefore, we will add an additional setup algorithm $\mathcal{S}$. This setup computes information known to all parties realised as one public key pk, and for each $\mathcal{P}_i$ information only known to that party realised as a secret key sk_i. We can now define the interface for instantiation of verifiable random oracles.

Definition 6 (Verifiable Random Oracle Instantiation)**.** *Let*

- *n be the number of parties implementing the verifiable random oracle,*
- *$\mathcal{S}$ be a* PPT *setup algorithm which outputs a public key* pk *and n secret keys* $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$*,*
- *$\mathcal{H}_{\mathsf{pk},\mathsf{sk}} = (\mathcal{H}_{1,\mathsf{pk},\mathsf{sk}_1}, \ldots, \mathcal{H}_{n,\mathsf{pk},\mathsf{sk}_n})$ be a non-empty tuple of interactive* PPT *algorithms each provided by a different party $\mathcal{P}_i$,*

- $\mathcal{q}_{\mathsf{pk}}$ *be an interactive* PPT *algorithm known to all parties,*
- $D \to C \times S$ *be the type of the protocol* $\langle \mathcal{q}_{\mathsf{pk}}(\cdot), \mathcal{H}_{\mathsf{pk,sk}} \rangle$*, and*
- $\mathcal{v}_{\mathsf{pk}} : D \times C \times S \to \{1, 0\}$ *be a non-interactive* PPT *algorithm known to all parties.*

Then $(\mathcal{g}, \mathcal{q}, \mathcal{H}, \mathcal{v})$ *is called* verifiable random oracle*, if the following properties are met for all* $(\mathsf{pk}, \mathsf{sk}) = \mathcal{g}()$*:*

- Completeness: $\mathcal{v}_{\mathsf{pk}}$ *always verifies outputs of* $\langle \mathcal{q}_{\mathsf{pk}}(\cdot), \mathcal{H}_{\mathsf{pk,sk}} \rangle$ *i.e.*

$$\forall x.(h, \sigma) = \langle \mathcal{q}_{\mathsf{pk}}(x), \mathcal{H}_{\mathsf{pk,sk}} \rangle \Rightarrow \mathcal{v}_{\mathsf{pk}}(x, h, \sigma).$$

- Pseudo Determinism: *The same input* x *results in the same output value* h*, i.e.*

$$\forall x.(h_1, \sigma_1) = \langle \mathcal{q}_{\mathsf{pk}}(x), \mathcal{H}_{\mathsf{pk,sk}} \rangle \wedge (h_2, \sigma_2) = \langle \mathcal{q}_{\mathsf{pk}}(x), \mathcal{H}_{\mathsf{pk,sk}} \rangle \Rightarrow h_1 = h_2.$$

- Pseudorandomness: h *is undistinguishable from uniform randomness, i.e. there exists no stateful* PPT *adversary* $\mathcal{A}$ *that wins the following game with more then negligible probability.*

Game PR*:*

1 : $(\mathsf{pk}, \mathsf{sk}) = \mathcal{g}()$
2 : $x \leftarrow \mathcal{A}^{\langle \mathcal{q}_{\mathsf{pk}}(\cdot), \mathcal{H}_{\mathsf{pk,sk}} \rangle, \mathcal{v}_{\mathsf{pk}}}()$
3 : $M = \{x \mid \mathcal{A}\ \textit{queried}\ x\ \textit{to}\ \langle \mathcal{q}_{\mathsf{pk}}(\cdot), \mathcal{H}_{\mathsf{pk,sk}} \rangle\}$
4 : $(h_0, \sigma_0) = \langle \mathcal{q}_{\mathsf{pk}}(x), \mathcal{H}_{\mathsf{pk,sk}} \rangle$
5 : $h_1 \xleftarrow{\$} C$
6 : $b \xleftarrow{\$} \{0, 1\}$
7 : $b' \leftarrow \mathcal{A}(h_b)$
8 : **return** $b' = b \wedge x \notin M$

Forgery of Proofs The proof of correct evaluation introduced by this framework is a new potential weakpoint. If an adversary can forge such a proof it can virtually program the verifiable random oracle. For this reason, we define a security property for verifiable random oracles similar to EUF-CMA for signature schemes:

Definition 7 (Weak Unforgeability)**.** *A verifiable random oracle* $\mathsf{VRO} = (\mathcal{q}, \mathcal{H}, \mathcal{v})$ *is* weakly unforgeable*, if there exists no* PPT *adversary* $\mathcal{A}$ *that can win the following game with non-negligible probability:*

Game wUF*:*

1 : $(x, h, \sigma) \leftarrow \mathcal{A}^{\mathcal{H}}()$
2 : $M = \{(x, h) \mid \langle \mathcal{q}_{\mathsf{pk}}(x), \mathcal{H}_{\mathsf{pk,sk}} \rangle\ \textit{returned}\ (h, \sigma)\ \textit{to}\ \mathcal{A}\}$
3 : **return** $\mathcal{v}_{\mathsf{pk}}(x, h, \sigma) \wedge (x, h) \notin M$

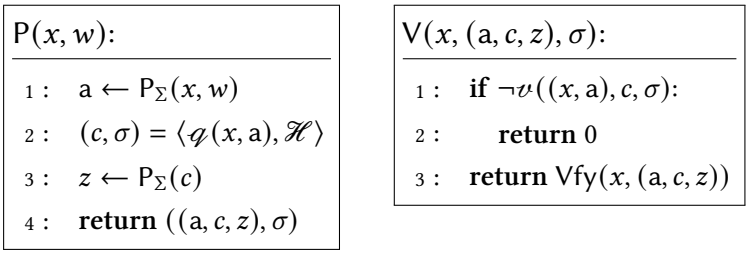


Figure 3.2: The prover and verifier of the Fiat-Shamir transformation, adapted to be used with a verifiable random oracle instantiation.

A Programming Interface Our notation to program random oracles $\mathsf{RO.prgm}(x, h)$ reminds of the notation for an interface. Programming interfaces are usually more of a way to intuitively explain programmability. However, for concrete instantiations we need a concrete way to program it. We will describe how to program an instantiation with pseudocode, thus taking the idea of programming interfaces a little more literally. To define the exact properties of such an interface is intrinsically hard and not the subject of this thesis. We will instead define what algorithms make up the interface, and what properties we require of them in the simulation-sound extractability setting.

Definition 8 (Programming Interface)**.** *Let*

- $\mathsf{VRO} = (\mathfrak{z}, q, \mathscr{H}, v)$ *be a verifiable random oracle (with weak unforgeability), and*
- $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ *be the parties providing the interfaces in* $\mathscr{H}$*.*

We now modify VRO *as described below. Let*

- $\mathcal{P}$ *be the party with programming access,*
- $\mathsf{prgm}_{\mathsf{pk}}$ *be an interactive* PPT *algorithm known only to* $\mathcal{P}$*, and*
- $\mathsf{PRGM}_{\mathsf{pk},\mathsf{sk}} = \left(\mathsf{prgm}_{1,\mathsf{pk},\mathsf{sk}_1}, \mathsf{prgm}_{n,\mathsf{pk},\mathsf{sk}_n}\right)$ *be programming interfaces, each provided only to* $\mathcal{P}$ *by a different* $\mathcal{P}_i$*.*

Then (prgm, PRGM) *is called* programming interface *of* VRO *for* $\mathcal{P}$*.*

The properties of VRO from definition 8 do not necessarily hold if the programming interface is used. In fact, you would expect the programming of an oracle to break both the pseudo determinism and the pseudorandomness of VRO. However, in cases where the programming interface is not used, the properties hold. We will later take advantage of this fact to proof preservation of simulation-sound extractability.

Using the Instantiation One again, we need to slightly adapt the Fiat-Shamir transformation to the form of the verifiable random oracle instantiation. Specifically, the proof of correct evaluation has to be shared in order for queries to be verified, as shown in 3.2. We also need to change **Game** SSE in the following way:

- Any programming calls from the extractor are made to $\langle \mathsf{prgm}(\cdot,\cdot), \mathsf{PRGM}\rangle$,
- $\mathcal{A}$ and Ext gain access to q, $\mathscr{H}$, and v instead of $\mathsf{H}_{\mathsf{VRO}}$ and $\mathsf{Vfy}_{\mathsf{VRO}}$.

With these changes, we can go back to the simulation-sound extractability proof.

Simulation-Sound Extractability with Instantiations Before we begin with the simulation-sound extractability proof, we will require another two properties from our programming interface:

- In the executions of $\mathcal{A}$ initiated by Ext, with the exception of query (x, a), queries to the verifiable random oracle are answered identical to queries in the initial run, except for a negligible probability.
- $\mathcal{A}$ cannot distinguish between the initial run and being executed by Ext.

We will call programming interfaces with these properties *suitable*. Here is a outline for the proofs, that our instantiations preserve simulation-sound extractability of the Fiat-Shamir transformation.

Proof Outline. We make use of several game hops.

- Let **Game 0** be the **Game** SSE using an instantiated verifiable random oracle $\mathsf{VRO} = (\mathcal{S}, q, \mathscr{H}, v)$ with weak unforgeability and suitable programming interface $(\mathsf{prgm}, \mathsf{PRGM})$.
- In **Game 1**, we use a changed version of VRO with uniform randomness instead of pseudorandomness, and show that $\mathcal{A}$ cannot distinguish this change. How exactly VRO has to be changed depends on the concrete instantiation.
- In **Game 2** $\mathcal{A}$ loses when outputting a forged proof of correct evaluation, which only changes the success probability of $\mathcal{A}$ negligibly.

Using the suitability of the programming interface, we can follow the steps of the proof for theorem 2 to prove security in **Game 2**. □

We will use this outline to prove security with our instantiations in the sections 3.3 and 3.4.

3.3 Instantiation with a Trusted Party

In this section, we will showcase the first instantiation of a verifiable random oracle. This instantiation relies on a trusted party. After explaining a simple idea for instantiation, we optimize it to have less overhead. We also eliminate the use of real randomness in order to be closer to practice. Finally, we prove that the instantiation preserves simulation-sound extractability of the Fiat-Shamir transformation.

Lazy Sampling The instantiation presented in this section is inspired by the lazy sampling interpretation of a random oracle. We only use a single trusted party $\mathcal{P}_1$ to implement the functionality. $\mathcal{P}_1.\mathcal{h}_1$ draws a random h uniformly for each new query x, saves it, and returns it. If the same x is queried again, $\mathcal{h}_1$ returns the saved h. In addition, $\mathcal{h}_1$ outputs a *signature* for the tuple (x, h) as proof of correct evaluation. This only yields a valid method to prove the correct evaluation, if $\mathcal{P}_1$ is a trusted party. In the following paragraphs, we will further customize this instantiation to have less network overhead.

Input Hashing The instantiation suggested in the previous paragraph uses the minimal number of messages possible (a single message to answer a query). If we want to reduce overhead further we need to shorten these messages. The length of the random value h is dictated by the scheme that makes use of the instantiation, meaning we have no influence on its length. We can reduce the length of the signature to a certain extent by choosing a suitable signature scheme. Apart from that, we can use a cryptographic hash function H in $\mathcal{q}$ and send $\mathsf{H}(x)$ instead of x to $\mathcal{h}_1$. This can reduce the size of queries to be linear in the security parameter[1] with negligible security loss (as proven later for theorem 3).

Using Pseudorandomness As of now, the values output by the verifiable random oracle are chosen uniformly random. Although this is allowed in theory, we cannot rely on the ability to draw random values uniformly in practice. Therefore we will use a *pseudorandom function* to chose random values. As any adversary will only notice this difference with negligible probability, the instantiation remains secure (as proven later for theorem 4). This also eliminates the need to store the random values as $\mathcal{P}_1$ can compute them again.

Programming We still need to demonstrate how the instantiated version of the oracle can be programmed. In order to program $\langle \mathcal{q}(x), \mathcal{h}_1 \rangle$ to be equal to h in this instantiation, the programming party can "just tell" $\mathcal{P}_1$ what values to use. As $\mathcal{P}_1$ only interacts with hashed values of x, prgm computes $x' = \mathsf{H}(x)$ and calls $\mathcal{P}_1.\mathsf{prgm}_1(x', h')$. The tuple is saved by $\mathcal{P}_1$, which will use h' whenever $\mathcal{h}_1(x')$ is queried. Otherwise, $\mathcal{P}_1$ proceeds normally. We can now formally define the described instantiation:

Theorem 3 (Trusted Party Instantiation)**.** *Let*

- $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ *be a* EUF-CMA*-secure signature scheme,*
- PRF *be a keyed pseudorandom function,*
- H *be a keyed, collision resistant, publicly known hash function,*
- *the setup algorithm* $\mathcal{s}$ *be defined as*

[1] This optimization assumes queries of big values, e.g. polynomial in the security parameter. For schemes that query only small values, this optimization might even increase the length of query messages and can be skipped.

$\mathcal{I}()$:
1 : $\mathsf{pk}_{\mathsf{H}} \xleftarrow{\$} \mathsf{Rand}$
2 : $\mathsf{sk}_{\mathsf{PRF}} \xleftarrow{\$} \mathsf{Rand}$
3 : $(\mathsf{pk}_{\mathsf{SIG}}, \mathsf{sk}_{\mathsf{SIG}}) = \mathsf{Gen}()$
4 : $\mathsf{pk} = (\mathsf{pk}_{\mathsf{H}}, \mathsf{pk}_{\mathsf{SIG}})$
5 : $\mathsf{sk}_1 = (\mathsf{sk}_{\mathsf{PRF}}, \mathsf{sk}_{\mathsf{SIG}})$
6 : **return** $(\mathsf{pk}, (\mathsf{sk}_1))$

,

- *the algorithms* $\mathcal{q}$, $\mathcal{h}_1$, *and* $\mathcal{v}$ *be defined as*

$\mathcal{q}_{\mathsf{pk}}(x)$:
1 : $x' = \mathsf{H}_{\mathsf{pk}_{\mathsf{H}}}(x)$
2 : **return** $\mathcal{h}_1(x')$

$\mathcal{h}_{1,\mathsf{pk},\mathsf{sk}_1}(x')$:
1 : **if** $x' \in$ **mem**:
2 : $\quad h = \mathbf{mem}[x']$
3 : **else**:
4 : $\quad h = \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF}}}(x')$
5 : $\sigma = \mathsf{Sign}_{\mathsf{sk}_{\mathsf{SIG}}}(x', h)$
6 : **return** (h, σ)

$\mathcal{v}_{\mathsf{pk}}(x, h, \sigma)$:
1 : $x' = \mathsf{H}(x)$
2 : **return** $\mathsf{Vfy}_{\mathsf{pk}_{\mathsf{SIG}}}((x', h), \sigma)$

, *and*

- *the programming interfaces* prgm *and* prgm_1 *be defined as*

$\mathsf{prgm}_{\mathsf{pk}}(x, h')$:
1 : $x' = \mathsf{H}_{\mathsf{pk}_{\mathsf{H}}}(x)$
2 : $\mathsf{prgm}_1(x', h')$

$\mathsf{prgm}_{1,\mathsf{pk},\mathsf{sk}_1}(x', h')$:
1 : **mem**$[x'] = h'$

.

Then $(\mathcal{I}, \mathcal{q}, (\mathcal{h}_1), \mathcal{v})$ *is a verifiable random oracle with weak unforgeability and suitable programming interface* $(\mathsf{prgm}, (\mathsf{prgm}_1))$.

We will prove this theorem by proving that $(\mathcal{I}, \mathcal{q}, (\mathcal{h}_1), \mathcal{v})$ (without the programming interface) is a verifiable random oracle, and afterwards, that it is weakly unforgeable, and that the programming interface is suitable.

Lemma 3. $(\mathcal{I}, \mathcal{q}, (\mathcal{h}_1), \mathcal{v})$ *is a verifiable random oracle.*

Proof. Without the programming interface, the greyed out branch in $\mathcal{h}_1$ will never be taken. We can therefore conclude, that all queries get answered as follows:

$$\begin{aligned}(h, \sigma) &= \left\langle \mathcal{q}_{\mathsf{pk}}(x), \mathcal{h}_{1,\mathsf{sk}_1} \right\rangle \\ &= \mathcal{h}_{1,\mathsf{sk}_1}(\mathsf{H}_{\mathsf{pk}_{\mathsf{H}}}(x)) \\ &= \left(\mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF}}}(\mathsf{H}_{\mathsf{pk}_{\mathsf{H}}}(x)),\ \mathsf{Sign}_{\mathsf{sk}_{\mathsf{SIG}}}(\mathsf{H}_{\mathsf{pk}_{\mathsf{H}}}(x), \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF}}}(\mathsf{H}_{\mathsf{pk}_{\mathsf{H}}}(x)))\right) \\ &= \left(h, \mathsf{Sign}_{\mathsf{sk}_{\mathsf{SIG}}}(\mathsf{H}_{\mathsf{pk}_{\mathsf{H}}}(x), h)\right)\end{aligned}$$

Passing these values to υ results in

$$\begin{aligned}\upsilon_{\mathsf{pk}}(x, h, \sigma) &= \mathsf{Vfy}_{\mathsf{pk}_{\mathsf{SIG}}}\left(\left(\mathsf{H}_{\mathsf{pk}_{\mathsf{H}}}(x), h\right),\ \mathsf{Sign}_{\mathsf{sk}_{\mathsf{SIG}}}\left(\mathsf{H}_{\mathsf{pk}_{\mathsf{H}}}(x), h\right)\right)\\ &= 1.\end{aligned}$$

We can conclude *completeness*. For the same input x, the output h is always computed by $\mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF}}}(\mathsf{H}_{\mathsf{pk}_{\mathsf{H}}}(x))$, which is a function. Therefore, we have *pseudo determinism*. An adversary $\mathcal{A}$ to the *pseudorandomness* has to either break the pseudorandomness of PRF, or the collision resistance of H. Therefore, $\mathcal{A}$ is successful only with negligible probability. With this, all requirements of a verifiable random oracle are met. □

Lemma 4. *$(\mathcal{S}, \mathcal{q}, (\mathcal{h}_1), \upsilon)$ has weak unforgeability.*

We prove this lemma by contradiction. We assume a successful adversary exists and construct a successful EUF-CMA adversary with it:

Proof. Let

- $\mathcal{A}$ be any successful adversary to the weak unforgeability of $(\mathcal{S}, \mathcal{q}, (\mathcal{h}_1), \upsilon)$, and
- $\mathsf{O}_{\mathsf{Sign}}$ the signing oracle of **Game** EUF-CMA of the used signature scheme.

We can use $\mathcal{A}$ to break EUF-CMA security by constructing an adversary like this:

$\mathcal{A}()$		$\mathcal{A}_{\mathsf{Sign}}()$
do:	$\xrightarrow{\mathcal{h}_1(x_i')?}$	$h_i = \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF}}}(x_i)$
until done	$\xleftarrow{(h_i, \sigma_i)}$	$\sigma_i = \mathsf{O}_{\mathsf{Sign}}(x_i', h_i)$
	$\xrightarrow{(x, h, \sigma)}$	$x' = \mathsf{H}_{\mathsf{pk}_{\mathsf{H}}}(x)$
		return $((x', h), \sigma)$

In this setup, $\mathcal{A}_{\mathsf{Sign}}$ answers to $\mathcal{h}_1$ queries from $\mathcal{A}$, and uses H and PRF from theorem 3. If $\mathcal{A}$ is successful, we know that $\mathsf{Vfy}(\mathsf{pk}, (x', h), \sigma) = 1$ by definition of υ. $\mathcal{A}_{\mathsf{Sign}}$ can only be unsuccessful in such a case if it queried (x', h) from $\mathsf{O}_{\mathsf{Sign}}$ before. In this case, $\mathcal{A}$ found a hash collision for H. The probability of this is negligible due to the collision resistance of H. Because we required the signature scheme to be EUF-CMA secure, no such $\mathcal{A}_{\mathsf{Sign}}$ exists, and therefore, $\mathcal{A}$ cannot exist. □

Lemma 5. *$\left(\mathsf{prgm}, \left(\mathsf{prgm}_1\right)\right)$ is a suitable programming interface for $(\mathcal{S}, \mathcal{q}, (\mathcal{h}_1), \upsilon)$ in the simulation-sound extractability setting.*

Proof. Ext only ever calls the programming interface for (x, a). Therefore, the only storage location used is $\mathcal{P}_1.\mathbf{mem}[\mathsf{H}_{\mathsf{pk}_{\mathsf{H}}}(x, \mathsf{a})]$. The only deviations from the original run occur if $x' = \mathsf{H}_{\mathsf{pk}_{\mathsf{H}}}(x, \mathsf{a})$ is queried to $\mathcal{h}_1$. This happens either if $\langle \mathcal{q}(x, \mathsf{a}), \mathcal{H} \rangle$ is called, or in the negligible case that $\mathcal{A}$ found a hash collision for H. Therefore, the probability of queries

not being answered identically as in the initial run, with the exception of query (x, a), is negligible.

It remains to be shown that $\mathcal{A}$ cannot distinguish between the initial run and runs started by Ext, except with negligible probability. We know that the probability of finding any difference beside the value of $(c, \sigma) = \langle q(x, \mathsf{a}), \mathcal{H} \rangle$ is negligible. In the initial run, c is a pseudorandom value gained by $\mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF}}}(\mathsf{H}_{\mathsf{pk}_{\mathsf{H}}}(x, \mathsf{a}))$. In runs started by $\mathcal{A}$, it is a uniformly random value out of the challenge space. Distinguishing between those is negligible because of the pseudorandomness of PRF. If σ would help to distinguish between runs, the signature scheme could be used to break the pseudorandomness of PRF. Overall, the probability for $\mathcal{A}$ distinguishing the runs is negligible. □

In conclusion, $(\mathcal{S}, q, (h_1), v)$ is a verifiable random oracle with weak unforgeability and a suitable programming interface $(\mathsf{prgm}, (\mathsf{prgm}_1))$, as stated in theorem 3. We will now proceed to prove simulation-sound extractability of the Fiat-Shamir transformation when used with this instantiation.

Theorem 4. *The Fiat-Shamir transformation of a strictly sound Σ-protocol with super-polynomial challenge space is simulation-sound extractable if used with the verifiable random oracle defined in theorem 3.*

The proof for theorem 4 follows the outline presented at the end of section 3.2.

Proof. Let

- $\mathsf{VRO} = (\mathcal{S}, q, (h_1), v)$ be the verifiable random oracle with the programming interface $(\mathsf{prgm}, (\mathsf{prgm}_1))$ defined in theorem 3,
- h_1' be a version of h_1 that uses uniform randomness instead of PRF to gain h in line 4,
- **Game 0** be **Game** SSE using VRO,
- **Game 1** be the same game, except it uses $\mathsf{VRO}' = (\mathcal{S}, q, (h_1'), v)$ (with unchanged programming interface) instead of VRO, and
- **Game 2** be like **Game 1** except $\mathcal{A}$ loses when outputting a forged proof.

Any $\mathcal{A}$ noticing a difference between **Game 1** and **Game 2** could be used to break the pseudorandomness of PRF. Therefore, the difference in probabilities of $\mathcal{A}$ succeeding in **Game 0** and **Game 1** is negligible. VRO′ is a verifiable random oracle with weak unforgeability and the suitable programming interface $(\mathsf{prgm}, (\mathsf{prgm}_1))$ for the same reasons as VRO. Because of the weak unforgeability of VRO′ the probability of $\mathcal{A}$ acting different in **Game 1** and **Game 2** is negligible. This also implies the difference in success probabilities for the two games is negligible. With help of the suitable behaviour of $(\mathsf{prgm}, (\mathsf{prgm}_1))$, we can repeat the argumentation for theorem 2 to prove that the Fiat-Shamir transformation is simulation-sound extractable in **Game 2**, and therefore in **Game 0**. □

3.4 Reducing Trust in Parties

In section 3.3, we have seen how to construct a verifiable random oracle using a trusted party. This is a strong assumption, as trusting a party in this context not only means trusting it to be honest. It also implies trusting that there will never be any successful attack against the party. This section introduces the concept of corruption and presents a verifiable random oracle instantiation which is safe as long as no more than one party is corrupted. We will prove this instantiation is suitable to be used with the Fiat-Shamir transformation by preserving simulation-sound extractability. Afterwards, we demonstrate that the instantiation can be generalized to allow up to any number of corrupted parties. However, this comes with a significant increase of needed parties and size of messages.

Corruption Corruption is a tool to model parties getting "hacked" or otherwise compromised. A party corrupted by an adversary shares all information with this adversary. If the adversary only gets information this way, but does not otherwise influence the corrupted party, we speak of *passive corruption*. If the adversary can control the corrupted party by dictating its outputs, it is *active corruption*. For now, we will focus on active corruption. In settings that allow corruption, security has to be proven for any possible set of corrupted parties. A party is *untrusted*, if it is part of any possible set of corrupted parties.

Distributing Functionality A common strategy to tackle the problem of corruption is to distribute the functionality among multiple parties. We could for example have three parties implementing the exact same functionality. A query would then consist of separate queries to each of the parties and use the output given by a majority of the parties. If a single corrupted party among them changes its output, the majority of outputs and therefore the effective value does not change. This naive idea is undesirable for multiple reasons, as we will soon realize.

Corruption vs. Programming Corruption in the simulation-sound extractability setting entails an additional challenge: The extractor should be able to program the VRO without the adversary noticing. Without this ability, simulation-sound extractability cannot be accomplished. If the extractor changes any output of a corrupted party, the adversary will notice the programming attempt. Therefore, the extractor has to know which parties are corrupted. It will gain this information as an additional input.

Finding an Instantiation Distributing functionality by copying it three times fails in our case. The corrupted party would notice programming attempts as it can compute the honest output by itself. In fact, any instantiation in which an untrusted party can compute the honest result alone will fail for this reason. To avoid this problem, we will change the way we derive the output value. Thus far, given an input x, we used a pseudorandom function PRF to calculate the random value $h \leftarrow \mathsf{PRF}(x)$. A simple solution to our problem is to use multiple such functions and combine their output with a bitwise exclusive OR:

$$h = \bigoplus_i \mathsf{PRF}_i(x)$$

We then make sure that no party knows all PRF_i, and use the majority for each individual PRF_i as described before. The minimum number of parties required for this strategy is

four, as one party is required not to know some PRF_i, and three others are required in order for majority to work. Let all parties $\mathcal{P}_i$ know all pseudorandom functions except for PRF_i. By giving each $\mathcal{P}_i$ a unique signature, and letting them sign $(x, \mathsf{PRF}_n(x))$ for all $n \neq i$, we can make this system verifiable. We write $n \neq i$ as a shorthand for $n \in \{1..4\} \setminus \{i\}$ when n and i are indices of parties.

How to Program It is still possible to freely program this instantiation if only one party $\mathcal{P}_n$ is corrupted. In this case, the extractor can program all other parties to change the result of PRF_n without $\mathcal{P}_n$ noticing. To change the overall output for input x to h', the extractor can set

$$\mathsf{PRF}_n(x) = h' \oplus \bigoplus_{i \neq n} \mathsf{PRF}_i(x).$$

We will now give a formal definition for this instantiation.

Theorem 5 (Distributed Instantiation)**.** *Let*

- $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vfy})$ *be a* EUF-CMA*-secure signature scheme,*
- PRF *be a keyed pseudorandom function,*
- $\mathsf{Corr} \subseteq \{1..4\}$ *be the indices of all corrupted parties,*
- *the setup algorithm* $\mathcal{S}$ *be defined as*

$\mathcal{S}()$:

1 : **for** $i \in \{1..4\}$:
2 : $\quad \mathsf{sk}_{\mathsf{PRF},i} \xleftarrow{\$} \mathsf{Rand}$
3 : $\quad \left(\mathsf{pk}_{\mathsf{SIG},i}, \mathsf{sk}_{\mathsf{SIG},i}\right) = \mathsf{Gen}()$
4 : $\mathsf{pk} = \left(\mathsf{pk}_{\mathsf{SIG},i} \mid i \in \{1..4\}\right)$
5 : **for** $i \in \{1..4\}$:
6 : $\quad \mathsf{sk}_i = \left(\mathsf{sk}_{\mathsf{SIG},i}, \left(\mathsf{sk}_{\mathsf{PRF},n} \mid n \neq i\right)\right)$
7 : $\mathsf{sk} = \left(\mathsf{sk}_i \mid i \in \{1..4\}\right)$
8 : **return** $(\mathsf{pk}, \mathsf{sk})$

,

- *the helper algorithm* extract*, which computes h from* σ*, be defined as*

$\mathsf{extract}_{\mathsf{pk}}(\sigma)$:

1: $\left(\left(h_{i,n}, \sigma_{i,n}\right) \,\middle|\, n \in \{1..4\} \wedge i \neq n\right) = \sigma$
2: **for** $i \in \{1..4\}$:
3: **if** $\max_{h'} \left|\left\{n \,\middle|\, \mathsf{Vfy}_{\mathsf{pk}_{\mathsf{SIG},n}}\left(\left(x, h_{i,n}\right), \sigma_{i,n}\right) \wedge h_{i,n} = h'\right\}\right| < 2$:
4: **return** $\perp$
5: $h_i = \arg\max_{h'} \left|\left\{n \,\middle|\, \mathsf{Vfy}_{\mathsf{pk}_{\mathsf{SIG},n}}\left(\left(x, h_{i,n}\right), \sigma_{i,n}\right) \wedge h_{i,n} = h'\right\}\right|$
6: **return** $\bigoplus_{i \in \{1..4\}} h_i$

,

- *the algorithms* $\mathcal{q}$, $\mathscr{H} = (\mathcal{h}_n \mid n \in \{1..4\})$, *and* $\mathcal{v}$ *be defined as*

$\mathcal{q}_{\mathsf{pk}}(x)$:

1: **for** $n \in \{1..4\}$:
2: $\sigma_n = \mathcal{h}_n(x)$
3: $\sigma = (\sigma_n \mid n \in \{1..4\})$
4: $h = \mathsf{extract}_{\mathsf{pk}}(\sigma)$
5: **return** (h, σ)

$\mathcal{h}_{n,\mathsf{pk},\mathsf{sk}_n}(x)$:

1: **for** $i \neq n$:
2: **if** $(i, x) \in \mathbf{mem}$:
3: $h_{i,n} = \mathbf{mem}[i, x]$
4: **else**:
5: $h_{i,n} = \mathsf{PRF}_{\mathsf{sk}_i}(x)$
6: $\sigma_{i,n} = \mathsf{Sign}_{\mathsf{sk}_{\mathsf{SIG},n}}(x, h_{i,n})$
7: **return** $\left(\left(h_{i,n}, \sigma_{i,n}\right) \,\middle|\, i \neq n\right)$

$\mathcal{v}_{\mathsf{pk}}(x, h, \sigma)$:

1: $h' = \mathsf{extract}_{\mathsf{pk}}(\sigma)$
2: **return** $h = h'$

,*and*

- *the programming interfaces* prgm *and* $\mathsf{PRGM} = \left(\mathsf{prgm}_n \,\middle|\, n \in \{1..4\}\right)$ *be defined as*

$\mathsf{prgm}_{\mathsf{pk},\mathsf{Corr}}(x, h')$:

1: **if** $|\mathsf{Corr}| > 1$:
2: **return** $\perp$
3: $\{i\} = \mathsf{Corr}$
4: $h'_i = h' \oplus \bigoplus_{n \neq i} h_n$
5: **for** $n \neq i$:
6: $\mathsf{prgm}_n(i, x, h'_i)$

$\mathsf{prgm}_{n,\mathsf{pk},\mathsf{sk}_n}(i, x, h'_i)$:

1: $\mathbf{mem}[i, x] = h'_i$

,

where prgm *can acquire all* h_n *values with* $\mathcal{h}_n(x)$ *queries to uncorrupted parties. Then* $(\mathcal{s}, \mathcal{q}, \mathscr{H}, \mathcal{v})$ *is a verifiable random oracle with weak unforgeability and suitable programming interface* (prgm, PRGM)*, if* $|\mathsf{Corr}| \leq 1$.

We will again split theorem 5 into three lemmas.

Lemma 6. $\mathsf{VRO} = (\mathcal{s}, \mathcal{q}, \mathscr{H}, \mathcal{v})$ *is a verifiable random oracle if* $|\mathsf{Corr}| \leq 1$.

Proof. Let $\mathcal{P}_1$ be the only corrupted party without loss of generality, i.e. Corr = {1}. This implies, that all algorithms except for $\hbar_1$ behave as defined. Without the programming interface, the greyed out branch in each $\hbar_i$ will never be taken. Therefore, for a call $\langle q(x), \mathscr{H}\rangle$, we have the following knowledge for $h_{i,n}$ values in σ:

$$\begin{aligned} h_{1,2} = h_{1,3} = h_{1,4} &= \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},1}}(x), \\ h_{2,3} = h_{2,4} &= \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},2}}(x), \\ h_{3,2} = h_{3,4} &= \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},3}}(x), \text{ and} \\ h_{4,2} = h_{4,3} &= \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},4}}(x). \end{aligned}$$

We also know, that all respective $\sigma_{i,n}$ are verifying. Overall, this implies that h returned by extract is equal to $\bigoplus_{i\in\{1..4\}} \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},i}}$. This is a function, meaning VRO is *pseudo deterministic*. As υ uses the same σ to extract h', we know that $h = h'$, and that VRO is *complete*. We also know that the bitwise exclusive OR of pseudorandom functions is itself a pseudorandom function. Therefore, VRO is *pseudorandom*, and we can conclude it is a verifiable random oracle. □

Lemma 7. VRO = $(\mathcal{I}, q, \mathscr{H}, \upsilon)$ *is weakly unforgeable if* $|\mathsf{Corr}| \leq 1$.

Proof. To forge a new proof σ', it is necessary to change at least one h_i as calculated in extract from a known σ. This requires to change two signatures $\sigma_{i,a}$ and $\sigma_{i,b}$ for (x, h_i). Let $\mathcal{P}_a$ be corrupted by $\mathcal{A}$ without loss of generality. If $\mathcal{A}$ is successful in breaking the weak unforgeability of VRO, it can be used to break EUF-CMA of SIG, by forging $\sigma'_{i,b}$. This probability is negligible, so $\mathcal{A}$ cannot be successful, and we can conclude the weak unforgeability of VRO. □

Lemma 8. (prgm, PRGM) *is a suitable programming interface for* $(\mathcal{I}, q, \mathscr{H}, \upsilon)$ *in the simulation-sound extractability setting, if* $|\mathsf{Corr}| \leq 1$.

Proof. Again, let $\mathcal{P}_1$ be the only corrupted party without loss of generality. Ext only ever calls the programming interface for (x, a). Therefore, $\mathcal{P}_i.\mathbf{mem}[1, (x, \mathrm{a})]$ for $i \neq 1$ are the only storage locations used. The only deviations from the original run occur if (x, a) is queried to any $\mathcal{P}_i$ for $i \neq 1$. This can only influence the result of $\langle q(x, \mathrm{a}), \mathscr{H}\rangle$. Any other queries have identical results.

It remains to be shown that $\mathcal{A}$ can only distinguish between the initial run and runs started by Ext with negligible probability. We know that there are no differences, except for the call to $\langle q(x, \mathrm{a}), \mathscr{H}\rangle$. The result of this computation in the initial run is the pseudorandom value $\bigoplus_{i\in\{1..4\}} \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},i}}(x, \mathrm{a})$ as demonstrated before. In a programming call $\langle \mathsf{prgm}((x, \mathrm{a}), h'), \mathsf{PRGM}\rangle$, the interface prgm_1 of the corrupted party is never called and therefore does not give anything away. After a call to $\langle \mathsf{prgm}((x, \mathrm{a}), h'), \mathsf{PRGM}\rangle$, we know about the associated σ that

$$h_{1,2} = h_{1,3} = h_{1,4} = h' \oplus \bigoplus_{i\neq 1} \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},i}}(x, \mathrm{a}), \text{ and thus,}$$

$$\begin{aligned}\langle q(x,\mathrm{a}),\mathscr{H}\rangle &= h' \oplus \bigoplus_{i\neq 1} \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},i}}(x,\mathrm{a}) \oplus \bigoplus_{i\neq 1} \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},i}}(x,\mathrm{a}) \\ &= h'.\end{aligned}$$

This value was drawn with uniform randomness by Ext. Distinguishing it from the honest output would require breaking the pseudorandomness of $\bigoplus_{i\in\{1..4\}} \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},i}}(x,\mathrm{a})$. Overall, the probability for $\mathcal{A}$ to distinguish the runs is negligible. □

In conclusion, $(\mathcal{I}, q, \mathscr{H}, \upsilon)$ is a verifiable random oracle with weak unforgeability and suitable programming interface (prgm, PRGM), as stated by theorem 5. We will now proceed to prove simulation-sound extractability of the Fiat-Shamir transformation when used with this instantiation.

Theorem 6. *The Fiat-Shamir transformation of a strictly sound Σ-protocol with super-polynomial challenge space is simulation sound extractable if used with the verifiable random oracle defined in theorem 5, and at most one party of the oracle is corrupted.*

We will once more follow the outline presented at the end of section 3.2 to prove theorem 6.

Proof. Let

- $\mathcal{P}_1$ be the only corrupted party without loss of generality,
- $\mathsf{VRO} = (\mathcal{I}, q, \mathscr{H}, \upsilon)$ be the verifiable random oracle with the programming interface (prgm, PRGM) defined in theorem 5,
- $\mathscr{H}' = \left(\hbar_1, \hbar'_2, \hbar'_3, \hbar'_4\right)$ be a version of $\mathscr{H}$ where all parties with access to $\mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF},1}}$ use the same uniform randomness instead,
- **Game 0** be **Game** SSE using VRO,
- **Game 1** be the same game except it uses $\mathsf{VRO}' = (\mathcal{I}, q, \mathscr{H}', \upsilon)$ instead of VRO,
- **Game 2** be like **Game 1** except $\mathcal{A}$ loses when outputting a forged proof.

Any $\mathcal{A}$ noticing a difference between **Game 0** and **Game 1** could be used to break the pseudorandomness of PRF, specifically the one used with key $\mathsf{sk}_{\mathsf{PRF},1}$. Therefore, the difference in probabilities of $\mathcal{A}$ succeeding in **Game 0** and **Game 1** is negligible. VRO′ is a verifiable random oracle with weak unforgeability and the suitable programming interface (prgm, PRGM) for the same reasons as VRO. Because of the weak unforgeability of VRO′ the probability of $\mathcal{A}$ acting differently in **Game 1** and **Game 2** is negligible. This also implies that the difference in success probabilities for the two games is negligible. With the help of suitable behaviour from (prgm, PRGM), we can repeat the argumentation of theorem 2 to prove that the Fiat-Shamir transformation is simulation-sound extractable in **Game 2**, and therefore in **Game 0**. □

An Untrusted Setup The setup algorithm $\mathcal{S}$ presented in theorem 5 requires a trusted party to generate and distribute keys. We can replace it with a secure multi-party computation protocol between all $\mathcal{P}_i$. However, finding a suitable protocol is not the subject of this thesis.

Timeouts Theorems 5 and 6 still hold if the corrupted party is faulty in some other way. This includes not answering at all. Remember that we modelled this behaviour with a special error symbol **timeout**. As **timeout** is never valid, all other parties always have a majority over the faulty one. However, the pseudocode does not reflect this behaviour explicitly, as error handling would decrease the readability.

Optimizations The instantiation presented in this section can easily be optimized. We begin with the same trick as in section 3.3 to hash x in $\mathcal{G}$, and use the hashed value for queries to any $\mathcal{H}_i$. As before, this change will result in queries with size linear in λ, with negligible security loss. In the next three paragraphs, we reduce the size of σ by a significant amount.

Redundancy There is a lot of redundant information in σ: For all i, a majority of $h_{i,n}$ in any σ have to be equal to h_i (as computed by extract). Any deviating value will not be used in the computation of the final h and could be omitted from σ. Additionally, two accepting $\sigma_{i,n}$ are enough to confirm any h_i. The third is not needed, regardless if it verifies or not. Therefore, we can reduce the size of σ by restructuring it so that it contains each h_i once, and exactly two accepting signatures for each. This optimization requires the possibility of requesting only specific information from each $\mathcal{P}_i$ and a much more complex algorithm $\mathcal{G}$ to query them. The complexity stems from all the special cases, in which parties send non-verifying responses.

Coordination We can further optimize scenarios where calls from $\mathcal{G}$ to any $\mathcal{H}_i$ are more costly than calls between two parties in $\mathcal{H}$. $\mathcal{G}$ can query the whole σ from any $\mathcal{H}_i$, which in turn requests missing information from the other $\mathcal{H}_n$. The only scenario in which $\mathcal{G}$ needs to make a second query is if the first obtained σ doesn't verify. This can happen when making a request to a corrupted or otherwise faulty party.

Aggregate Signatures By using aggregate signatures [7], we can continue to reduce the size of a result σ coordinated by $\mathcal{H}$ and returned to $\mathcal{G}$ even further. Multiple aggregate signatures (even of different parties and for different messages) can be aggregated into a single signature that is sufficient to replace all original signatures. With this optimization, σ contains only the four h_i values and a single signature, down from twelve $h_{i,n}$ and twelve signatures. This step concludes our optimization.

More Corrupted Parties The instantiation in this section can be generalized to be able to handle up to n corrupted parties. We will now compute the total number of parties and total number of PRFs needed in this situation. For each possible set of corrupted parties, there has to be a PRF not known by any party in the set. Now there exists a situation in which all n corrupted parties know a concrete PRF. In this scenario, there are still n other parties that do not know PRF. To be able to form a majority against the corrupted parties, we need another $n+1$ uncorrupted parties which do know PRF. Therefore, we need a total

of $3n + 1$ parties if up to n parties can be corrupted. The number of PRFs needed is the amount of different n party subsets from the set of all $3n + 1$ parties. This value amounts to

$$\binom{3n+1}{n} = \frac{(3n+1)!}{n!(2n+1)!} > 2^n$$

which is unfortunately super-polynomial in the amount of corrupted parties. This also increases the size of proofs of correct evaluation accordingly, as all PRF_i values have to be included. It would be interesting to see if instantiations with less overhead exist for situations with more corrupted parties.

4 Conclusion

In this thesis, we have demonstrated that instantiation of random oracles is possible with the help of interaction. This chapter summarizes what we did in order to achieve this goal.

Verifiable Random Oracles In section 3.1, we introduced a novel ideal model called verifiable random oracle model which is closely related to the random oracle model. This step was not necessary to find an instantiation. However, it helped to reduce the interaction in protocols instantiating the random oracle, by adding a second oracle to verify random values gained from the random oracle. Instantiations for this new verification oracle can be non-interactive.

Instantiations In section 3.2, we presented a framework to instantiate verifiable random oracles, and defined properties we later used in security proofs. We then used this framework to define two concrete instantiations. One of them needs only one party, but requires it to be trusted (section 3.3), the other is secure as long as at most one party is corrupted, but requires four parties (section 3.4).

Security Throughout the thesis, we have demonstrated the viability of our approach by proving that it is suitable to be used with the Fiat-Shamir transformation. To be more accurate, we have proven that versions of the Fiat-Shamir transformation using the ideal verifiable random oracle model, the framework with a variety of assumed properties, the instantiation with one trusted party, or the distributed instantiation respectively remain simulation-sound extractable. We are confident that other properties in other scenarios will be preserved from the random oracle model by the use of these instantiations as well.

Bibliography

[1] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. "An uninstantiable random-oracle-model scheme for a hybrid-encryption problem". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2004, pp. 171–188.

[2] Mihir Bellare and Gregory Neven. "Multi-signatures in the plain public-key model and a general forking lemma". In: *Proceedings of the 13th ACM conference on Computer and communications security*. 2006, pp. 390–399.

[3] Mihir Bellare and Phillip Rogaway. "Random oracles are practical: A paradigm for designing efficient protocols". In: *Proceedings of the 1st ACM conference on Computer and communications security*. 1993, pp. 62–73.

[4] Mihir Bellare and Sarah Shoup. "Two-Tier Signatures, Strongly Unforgeable Signatures, and Fiat-Shamir Without Random Oracles". en. In: *Public Key Cryptography – PKC 2007*. Ed. by Tatsuaki Okamoto and Xiaoyun Wang. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 201–216. ISBN: 978-3-540-71677-8. DOI: 10.1007/978-3-540-71677-8_14.

[5] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. *Interactive Oracle Proofs*. Tech. rep. 116. 2016. URL: https://eprint.iacr.org/2016/116 (visited on 04/03/2020).

[6] David Bernhard, Olivier Pereira, and Bogdan Warinschi. "How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios". en. In: *Advances in Cryptology – ASIACRYPT 2012*. Ed. by David Hutchison et al. Vol. 7658. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 626–643. ISBN: 978-3-642-34960-7 978-3-642-34961-4. DOI: 10.1007/978-3-642-34961-4_38. URL: http://link.springer.com/10.1007/978-3-642-34961-4_38 (visited on 07/02/2020).

[7] Dan Boneh et al. "Aggregate and verifiably encrypted signatures from bilinear maps". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2003, pp. 416–432.

[8] Jonathan Bootle et al. *Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting*. Tech. rep. 263. 2016. URL: https://eprint.iacr.org/2016/263 (visited on 11/04/2020).

[9] Melissa Chase and Anna Lysyanskaya. "Simulatable VRFs with applications to multi-theorem NIZK". In: *Annual International Cryptology Conference*. Springer, 2007, pp. 303–322.

[10] Ronald Cramer. “Modular design of secure yet practical cryptographic protocols”. In: *Ph. D. Thesis, CWI and University of Amsterdam* (1996).

[11] Dana Dachman-Soled, Abhishek Jain, and Yael Tauman Kalai. “On the (In)security of the Fiat-Shamir Paradigm, Revisited”. en. In: (), p. 27.

[12] Yvo Desmedt, Claude Goutier, and Samy Bengio. “Special Uses and Abuses of the Fiat-Shamir Passport Protocol (extended abstract)”. en. In: *Advances in Cryptology — CRYPTO ’87*. Ed. by Carl Pomerance. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1988, pp. 21–39. ISBN: 978-3-540-48184-3. DOI: 10.1007/3-540-48184-2_3.

[13] Yevgeniy Dodis. “Efficient construction of (distributed) verifiable random functions”. In: *International Workshop on Public Key Cryptography*. Springer, 2003, pp. 1–17.

[14] Jelle Don et al. “Security of the Fiat-Shamir Transformation in the Quantum Random-Oracle Model”. en. In: *Advances in Cryptology – CRYPTO 2019*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 356–383. ISBN: 978-3-030-26951-7. DOI: 10.1007/978-3-030-26951-7_13.

[15] Sebastian Faust et al. “On the non-malleability of the Fiat-Shamir transform”. In: *International Conference on Cryptology in India*. Springer, 2012, pp. 60–79.

[16] Amos Fiat and Adi Shamir. “How To Prove Yourself: Practical Solutions to Identification and Signature Problems”. en. In: *Advances in Cryptology — CRYPTO’ 86*. Ed. by Andrew M. Odlyzko. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1987, pp. 186–194. ISBN: 978-3-540-47721-1. DOI: 10.1007/3-540-47721-7_12.

[17] Marc Fischlin. “Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors”. en. In: *Advances in Cryptology – CRYPTO 2005*. Ed. by Victor Shoup. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 152–168. ISBN: 978-3-540-31870-5. DOI: 10.1007/11535218_10.

[18] Shafi Goldwasser and Yael Tauman Kalai. “On the (in) security of the Fiat-Shamir paradigm”. In: *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* IEEE, 2003, pp. 102–113.

[19] Vandana Gunupudi and Stephen R. Tate. “Random oracle instantiation in distributed protocols using trusted platform modules”. In: *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW’07)*. Vol. 1. IEEE, 2007, pp. 463–469.

[20] Shai Halevi, Oded Goldreich, and Ran Canetti. *The Random Oracle Methodology, Revisited*. Tech. rep. 011. 1998. URL: https://eprint.iacr.org/1998/011 (visited on 04/03/2020).

[21] Dennis Hofheinz and Eike Kiltz. “Programmable Hash Functions and Their Applications”. en. In: *Advances in Cryptology – CRYPTO 2008*. Ed. by David Wagner. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 21–38. ISBN: 978-3-540-85174-5. DOI: 10.1007/978-3-540-85174-5_2.

[22] Hans-Joachim Knobloch. “A Smart Card Implementation of the Fiat-Shamir Identification Scheme”. en. In: *Advances in Cryptology — EUROCRYPT ’88*. Ed. by D. Barstow et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1988, pp. 87–95. ISBN: 978-3-540-45961-3. DOI: 10.1007/3-540-45961-8_8.

[23] Neal Koblitz and Alfred J. Menezes. “The random oracle model: a twenty-year retrospective”. In: *Designs, Codes and Cryptography* 77.2-3 (2015). Publisher: Springer, pp. 587–610.

[24] Qipeng Liu and Mark Zhandry. “Revisiting Post-quantum Fiat-Shamir”. en. In: *Advances in Cryptology – CRYPTO 2019*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 326–355. ISBN: 978-3-030-26951-7. DOI: 10.1007/978-3-030-26951-7_12.

[25] Silvio Micali, Michael Rabin, and Salil Vadhan. “Verifiable random functions”. In: *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE, 1999, pp. 120–130.

[26] Kazuo Ohta and Tatsuaki Okamoto. “A digital multisignature scheme based on the Fiat-Shamir scheme”. en. In: *Advances in Cryptology — ASIACRYPT ’91*. Ed. by Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1993, pp. 139–148. ISBN: 978-3-540-48066-2. DOI: 10.1007/3-540-57332-1_11.

[27] Kazuo Ohta, Tatsuaki Okamoto, and Kenji Koyama. “Membership Authentication for Hierarchical Multigroups Using the Extended Fiat-Shamir Scheme”. en. In: *Advances in Cryptology — EUROCRYPT ’90*. Ed. by Ivan Bjerre Damgård. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1991, pp. 446–457. ISBN: 978-3-540-46877-6. DOI: 10.1007/3-540-46877-3_40.

[28] H. Ong and C. P. Schnorr. “Fast Signature Generation with a Fiat Shamir — Like Scheme”. en. In: *Advances in Cryptology — EUROCRYPT ’90*. Ed. by Ivan Bjerre Damgård. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1991, pp. 432–440. ISBN: 978-3-540-46877-6. DOI: 10.1007/3-540-46877-3_38.

[29] Dominique Unruh. “Post-quantum Security of Fiat-Shamir”. en. In: *Advances in Cryptology – ASIACRYPT 2017*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 65–95. ISBN: 978-3-319-70694-8. DOI: 10.1007/978-3-319-70694-8_3.

[30] Dominique Unruh. “Quantum proofs of knowledge”. In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2012, pp. 135–152.

[31] Brent Waters. “Efficient identity-based encryption without random oracles”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2005, pp. 114–127.