Study of Correlation Intractable Hash Functions

Endrit VORFAJ

School of Computer and Communication Sciences

Semester Project

June 2024

Responsible
Prof. Serge Vaudenay
EPFL / LASEC

Supervisor
Abdullah TALAYHAN
EPFL / LASEC

1 Introduction

Non-interactive zero-knowledge (NIZK) proofs enable a prover to convince a verifier of the validity of an NP-statement with just one round of communication, where a single message is sent from the prover to the verifier. One of the most prominent methods for creating non-interactive proofs is the Fiat-Shamir (FS) transform. This transformation takes a sigma-protocol and converts it into a NIZK proof.

We briefly mention interactive proof protocols and we consider the definitions of sigma-protocols which are a specific type of three-round public-coin interactive proof conducted between a prover P and a verifier V.

In these protocols, the only interaction between the prover P and the verifier V happens in the second round where the verifier with a challenge e to the initial message of the prover (denoted as a). The FS transform makes a sigma-protocol non-interactive by allowing the prover to generate the challenge itself. Specifically, the prover computes $e \leftarrow H(a)$, where H is a hash function. The security of this construction can be argued by modeling H as a Random Oracle, introduced in [BR93]. However, recent research ([CX23], [Can+18a]) has demonstrated that if the hash function is correlation-intractable (CI) for certain relations, then it is possible to construct a NIZK protocol using only the CI property instead of the random oracle model. Informally, the CI property ensures that given a random hash key k, it is computationally difficult to find any input x such that $(x, H_k(x)) \in R$ for a particular relation R.

In greater detail, [Can+19] demonstrated that the FS transform remains secure if the hash function is CI for efficiently searchable relations. Their results apply to a specific class of sigma-protocols known as trapdoor sigma-protocols. These protocols are defined in the Common Reference String (CRS) model and possess three main properties: honest verifier zero-knowledge (HVZK), optimal soundness, and a bad-challenge extractor.

The HVZK property is quite standard, ensuring the existence of a simulator that, given the challenge (the second round), produces a transcript indistinguishable from one generated by an honest prover and verifier. Optimal soundness guarantees that for any statement $x \notin L$ and any first-round message a, there is at most one challenge e that would make the verifier accept the transcript (a, e, z) for some third-round message z. This unique challenge e is referred to as the bad-challenge. Lastly, the bad-challenge extractor is an algorithm that, given a false statement x, a valid first-round message a, and some trapdoor information τ, can efficiently compute the bad-challenge e.

By exploring these foundational concepts, this paper aims to provide a comprehensive study of correlation intractability, focusing on the theoretical underpinnings and practical implementations of these functions, and providing insights into their role in modern cryptography. We then examine the failures of the random oracle model as described in [CGH04], discussing its limitations in the face of real-world implementations and security. From there, we discuss more formally the nature and utility of CI hash functions and we mention a CI construction that employs shiftable shift-hiding functions as outlined in [LV20]. Our objective is to define basic cryptographic primitives as pedagogically as possible, aiming to produce a report that is both self-contained and comprehensive. This document is designed to serve as a foundational basis for individuals embarking on research in correlation intractable hash functions, equipping them with the necessary tools and knowledge to advance the field.

2 Preliminaries

2.1 Notations and terminology

We denote $\mathbb{R}, \mathbb{Z}, \mathbb{N}$ as the set of reals, integers, and natural numbers. Let $\mathbb{Z}_N$ denote $\mathbb{Z}/N\mathbb{Z}$ (the set of all numbers modulo N). For $n \in N$, we let $[n]$ denote $\{1, 2, ..., n\}$.

In cryptography, the security parameter (in this paper denoted as λ) is a variable that is used to parameterize the computational complexity of a cryptographic algorithm or protocol from which we can then define the probability of an adversary breaking the security of the algorithm or protocol in question. We say that an algorithm is *efficient* if it runs in probabilistic polynomial time over λ, often denoted as PPT.

Probabilistic algorithms may occasionally require a coin toss to decide an outcome. For ease, we denote this by $\mathcal{A}(x; r)$, indicating that algorithm $\mathcal{A}$ operates on input x with a sequence of random coins r. Here, r is distinguished from standard inputs by a semicolon and in some instances depending in the context, r might be implicit. To define the terms 'easy' or 'hard' computation, we often use the concept of a polynomially bounded algorithm. A task is considered easy if it can be executed in time $O(\lambda^n)$ for some integer n and security parameter λ. Typically, 'polynomially bounded' relates to a polynomial based on the input size. We express λ in unary notation as 1^λ to ensure its length is λ rather than $log_2(\lambda)$. Thus, we usually write $\mathcal{A}(1^\lambda, x; r)$, though it's more practical to assume 1^λ and exclude it from the notation. Another related concept is that of negligible functions. We consider a function $f(\lambda)$ negligible (implicitly as λ approaches infinity) if, for any integer n, $f(\lambda) = O(\lambda^{-n})$.

We consider probability spaces defined over executions of probabilistic machines. Specifically, we consider the probability that an output from a probabilistic machine M_1 meets a criterion involving another probabilistic machine M_2. Specifically,

$$Pr[y \leftarrow M_1(x), |y| = |x| \text{ and } M_2(y) = 1]$$

represents the probability that on input x, M_1 yields a string of length $|x|$ accepted by M_2. Here, y is a random variable in $\{0, 1\}^*$, and we evaluate the probability that y satisfies these conditions in the distribution of $M_1(x)$.

2.2 The Random Oracle Model

The random oracle is a heavily abstract and theoretical version of hash functions introduced by [BR93]. In the *Random Oracle Model* (ROM), all participants in a cryptographic protocol can query an oracle $\mathcal{O}$ to receive responses, while being unable to see the queries made by others. The oracle responds randomly, but consistently, meaning that while the response to a new query will be random, identical queries will yield the same response. Essentially, a random oracle simulates a deterministic function chosen randomly at the beginning of the protocol. The responses are of a pre-determined length depending on the security parameter λ. The key advantage of the ROM is that it allows reductions to simulate the random oracle, making it appear as though it is a real random oracle with hidden but useful information.

The Fiat-Shamir transform, which converts an interactive proof into a non-interactive one, often relies on a hash function h to eliminate the need for verifier interaction. In many cryptographic schemes, a random oracle is used to represent h. However, since the random oracle model is an idealized abstraction that does not exist in practice, a cryptographic hash function is typically employed to concretely instantiate a random oracle. This practical instantiation aims to maintain the same security properties as those assured by the ROM. By doing so, the transform leverages the properties of hash functions to simulate the unpredictable yet consistent behavior of a random oracle, thus preserving the security guarantees of the protocol under realistic conditions.

2.3 Function Ensembles

For simplicity, let's define a length function $\ell_{\text{out}} : \mathbb{N} \to \mathbb{N}$, which determines the length of the output from the random oracle and its potential implementations. We assume that these length functions are super-logarithmic and polynomially bounded. Specifically, this means that $\omega(\log(\lambda)) \leq \ell_{\text{out}}(\lambda) \leq \text{poly}(\lambda)$. An oracle using a length function ℓ_{out} is called an ℓ_{out}-oracle. Each response from this oracle is a string with a length of $\ell_{\text{out}}(\lambda)$.

Definition 2.1 (Function ensembles)**.** Let $\ell_{out} : N \to N$ be a length function. An ℓ_{out}-ensemble is a sequence of $\mathcal{F} = \{F_\lambda\}_{\lambda \in N}$ function families where $F_\lambda = \{f_s : \{0,1\}^* \to \{0,1\}^{\ell_{out}(\lambda)}\}_{s \in \{0,1\}^\lambda}$ so that the following hold:

- **Length requirement:** For every $s \in \{0,1\}^\lambda$ and every $x \in \{0,1\}^*$, $|f_s(x)| = \ell_{out}(\lambda)$
- **Efficiency requirements:** There exists a PPT algorithm $EVAL$ so that for every $s, x \in \{0,1\}^*$, it holds that $EVAL(s,x) = f_s(x)$.

s is often called the *description* or the *seed* of the function f_s.

2.4 Interactive Proofs

Interactive proof systems, first introduced by Goldwasser, Micali, and Rackoff [GMR19], have emerged as a highly influential and versatile tool in both cryptography and computer science. These systems have paved the way for major advancements in theoretical cryptography, complexity theory, and quantum computing. They are central to significant innovations in practical cryptography, especially within the realms of blockchains and cryptocurrencies.

We take the following description from [Vau24] so that we can maintain notation consistency when defining sigma protocols. In an interactive proof system, we consider:

- an alphabet Z, i.e., a set of letters;
- the set Z^* of finite strings composed of elements from Z, i.e., the set of all words;
- the subsets of Z^*, which are called languages, i.e., sets of words.

Given a language L and a word x, we consider the problem of determining whether x belongs to L. This is known as the membership problem.

Languages for which the membership problem can be decided by a deterministic algorithm within polynomial time relative to the length of the string x ($|x|$), are called P languages.

Sometimes, we treat x as a statement and L as the language of true statements. True statements can be proven by a proof w, referred to as a witness. Given a predicate $R(x,w)$ that checks whether w is a correct proof for x, the language L is defined by:

$$L = \{x \in Z^*; \exists w \in Z^* \ R(x,w)\}$$

(For convenience, proofs are encoded into a word so that the witness is also a word.)

Languages where R can be evaluated in polynomial time relative to $|x|$, and where the witness must also be polynomially bounded, are called NP languages. The complement of an NP language is called a co-NP language.

Definition 2.2. An interactive machine is an algorithm A that takes as input some x, a list of incoming messages $m_1, \ldots, m_n$ of variable length, and a (sufficiently long) sequence of random coins r, and computes an outgoing message $A(x, m_1, \ldots, m_n; r)$. The tuple $(x, m_1, \ldots, m_n; r)$ is called the partial view of A.

We assume a special symbol in the alphabet for terminal messages. If m_n is a terminal message, then $A(x, m_1, \ldots, m_n; r)$ is a terminal message as well.

If $A(x, m_1, \ldots, m_n; r)$ is a terminal message, $(x, m_1, \ldots, m_n; r)$ is called the final view of A.

A pair of interactive machines $(\mathcal{A}, \mathcal{B})$ (with $\mathcal{A}$ called the initiator) is an interactive system. An experiment $\exp = (\mathcal{A}(r_A) \leftrightarrow \mathcal{B}(r_B))$ is characterized by an input x and the coins r_A and r_B for each participant. It consists of iteratively defining:

$$a_i = \mathcal{A}(x, b_1, \ldots, b_{i-1}; r_A)$$
$$b_j = \mathcal{B}(x, a_1, \ldots, a_j; r_B)$$

for $i = 1, \ldots, n_A$, where n_A is the smallest i such that a_i is a terminal message, and $j = 1, \ldots, n_B$, where n_B is the smallest j such that b_j is a terminal message. $\mathcal{A}$ initiates the interaction with the message $a_1 = \mathcal{A}(x; r_A)$ to $\mathcal{B}$. Then, $\mathcal{B}$ sends the message $b_1 = \mathcal{B}(x, a_1; r_B)$ to $\mathcal{A}$. $\mathcal{A}$ continues with $a_2 = \mathcal{A}(x, b_1; r_A)$, and so on. We define the outputs of both participants $\mathrm{Out}_{\mathcal{A}}(\exp) = a_{n_A}$ and $\mathrm{Out}_{\mathcal{B}}(\exp) = b_{n_B}$, and the final views $\mathrm{View}_{\mathcal{A}}(\exp) = (x, a_1, \ldots, a_{n_A}; r_A)$ and $\mathrm{View}_{\mathcal{B}}(\exp) = (x, b_1, \ldots, b_{n_B}; r_B)$.

We are now ready to define an interactive proof.

Definition 2.3 (Interactive Proof System)**.** Given a language L over an alphabet Z, an interactive proof system is an interactive system $(\mathcal{P}, \mathcal{V})$, where $\mathcal{P}$ is called the prover and $\mathcal{V}$ is called the verifier. There exists a polynomial P and some real numbers α and β such that $0 \leq \beta < \alpha \leq 1$ and:

- (**termination**) For any x and every set of random coins, the experiment $\mathcal{P} \leftrightarrow \mathcal{V}$ terminates within a complexity bounded by $P(|x|)$;
- (α**-completeness**) For any $x \in L$, the experiment $\mathcal{P} \leftrightarrow \mathcal{V}$ makes $\mathcal{V}$ output "accept" with probability at least α (the probability is taken over the random coins);
- (β**-soundness**) For any $x \notin L$ and any interactive machine $\mathcal{P}^*$, the experiment $\mathcal{P}^* \leftrightarrow \mathcal{V}$ makes $\mathcal{V}$ output "accept" with probability at most β (the probability is taken over the random coins).

This means that a prover $\mathcal{P}$ can convince a verifier $\mathcal{V}$ that $x \in L$ with probability at least α, and that no malicious prover $\mathcal{P}^*$ can convince the verifier when this is not true, with probability larger than β. We note that we assume no complexity bound on $\mathcal{P}$ or $\mathcal{P}^*$. We often consider $\alpha = 1$ in which case we say we have perfect completeness.

2.5 Computationally Sound (CS) Proofs

In traditional cryptographic proof systems, the prover is often considered to have unbounded computational resources. This is too idealized and impractical for real-world applications. In contrast, Computationally Sound (CS) proofs constrain the prover to operate within Polynomial-time Probabilistic (PPT) limits. This restriction reflects realistic capabilities, ensuring that the security of the proof system holds against adversaries that are also limited by polynomial-time constraints. Such a design not only aligns more closely with practical scenarios but also ensures that the system is realizable since no party has infinite computational power.

Definition 2.4 (Non-interactive CS proofs in the Random Oracle Model)**.** A CS-proof system consists of a deterministic polynomial-time prover P and verifier V where:

- On input $(1^\lambda, \langle M\rangle, x, 1^t)$ and access to an oracle $\mathcal{O}$, the prover computes a proof $\pi = P^{\mathcal{O}}(1^\lambda, \langle M\rangle, 1^t)$ such that $|\pi| \leq \text{poly}(\lambda, |\langle M\rangle|, |x|, \log(t))$
- On input $(1^\lambda, \langle M\rangle, x, t, \pi)$ with t in binary and access to an oracle $\mathcal{O}$, the verifier decides whether to accept or reject a proof π.

The CS-proof system has the following properties where the probabilities are taken over the choice of the random oracle $\mathcal{O}$:

- **Completeness:** For any M, x, t such that the machine M accepts the string x within t steps and for any λ:
$$\Pr_{\mathcal{O}}\left[\pi \leftarrow P^{\mathcal{O}}(1^\lambda, \langle M\rangle, x, 1^t),\ V^{\mathcal{O}}(1^\lambda, \langle M\rangle, x, t, \pi) = \text{accept}\right] = 1$$
- **Computational Soundness:** For any polynomial time oracle machine denoted as BAD and any input $w = (\langle M\rangle, x, 1^t)$ such that M **does not** accept x within t steps, it holds that:
$$\Pr_{\mathcal{O}}\left[\pi \leftarrow \text{BAD}^{\mathcal{O}}(1^\lambda, \langle M\rangle, x, 1^t),\ V^{\mathcal{O}}(1^\lambda, \langle M\rangle, x, t, \pi) = \text{accept}\right] \leq \frac{\text{poly}(\lambda + |w|)}{2^\lambda}$$

Furthermore, the running time of the prover P on input $(1^\lambda, \langle M\rangle, x, 1^t)$ is polynomially related to the actual running time of $M(x)$ rather than the global upper bound t. Thus, there exists a fixed polynomial $p(\cdot)$ such that:

$$T_P((1^\lambda, \langle M\rangle, x, 1^t)) \leq p(\lambda, min\{t, T_M(x)\})$$

where $T_M(x)$ denotes the running time of a machine M on input x.

For security proof of our construction , we need a different soundness condition than the computational soundness used above. Specifically, we need to ensure that given the machine M (and the complexity bound t), it is hard to find any pair (x, π) such that M does not accept x within t steps, and yet Ver will accept π as a valid CS-proof to the contrary. One way to achieve this soundness property from the original one is by postulating that when the verifier is given a proof for an assertion $w = (\langle M\rangle, x, t)$, it uses the security parameter $\lambda + |w|$ (rather than just λ). Using a straightforward counting argument, we get:

Proposition 2.5. *Let* (Prv, Ver) *be a CS-proof system. Then for every polynomial-time oracle machine BAD, there exists a polynomial* $q(\cdot)$*, such that for every* λ *it holds that*

$$\begin{aligned}\epsilon_{bad}(\lambda) = \Pr_{\mathcal{O}}\Big[&(w, \pi) \leftarrow BAD^{\mathcal{O}}(1^\lambda),\\ &\textit{where } w = (\langle M\rangle, x, t),\ \textit{such that } M \textit{ does not accept } x \textit{ within } t \textit{ steps}\\ &\textit{and yet } Ver^{\mathcal{O}}(1^{\lambda+|w|}, w, \pi) = accept\Big] \leq \frac{q(\lambda)}{2^\lambda}\end{aligned}$$

This notion of Computationally Sound proofs serves as the foundational basis for later sections where we make use of them and we look closely into their application. Specifically, we will utilize CS proofs to examine and demonstrate the inherent limitations and failures of the Random Oracle Model.

2.6 Sigma Protocols

In this section, we present sigma protocols as described in [Vau24], which are constructed based on 3-round interactive protocols. This is a specific instantiation of interactive proof systems and we examine simple protocols that operate in three phases: the prover sends an initial message a, the verifier responds with a random challenge e chosen from a set E, and then the prover replies with an answer z. Finally, the verifier decides whether to accept or reject. With additional properties, these protocols are classified as Σ-protocols.

Definition 2.6. Given a language $L \in \mathsf{NP}$ over an alphabet Z defined by a relation R, a Σ-protocol for L is a pair (P, V) of interactive machines such that:

- V is polynomially bounded.
- 3-move structure (*Figure 1*): P initiates with a message a, V responds with a challenge $e \in_U E$, P concludes with a response z, and V accepts (for $x \in L$) or rejects based solely on (x, a, e, z).
- Special soundness: there exists a polynomially bounded algorithm $\mathcal{E}$ called the extractor such that for any x, if $(x, a, z; r)$ and $(x, a, z'; r')$ are two accepting interactions for V with $e \neq e'$ where $e = V(x, a; r)$ and $e' = V(x, a; r')$, then $\mathcal{E}(x, a, e, z, e', z')$ produces w such that $R(x, w)$ holds.
- Special honest-verifier zero-knowledge (HVZK): there exists a polynomially bounded algorithm S called the simulator such that for any $x \in L$ and e, the transcript (a, e, z) from the interaction $P(r_p) \leftrightarrow V(r_v)$, given e, has the same distribution as $S(x, e; r)$.

To fully specify a Σ-protocol, we need:

- A relation R defining the language.
- A function $a = P(x, w; r_P)$.
- A samplable domain E for e.
- A function $z = P(x, w, e; r_P)$.
- A verification relation $V(x, a, e, z)$.
- An extraction function $\mathcal{E}(x, a, e, z, e', z')$.
- A simulation function $S(x, e; r)$.

The following properties must be satisfied:

1. $R, P, V, \mathcal{E}, S$, and sampling are all polynomially computable in $|x|$.
2. $\forall (x, w) \in R, \forall r_P, \forall e \in E$, we have $V(x, a, e, z)$, with
$$a = P(x, w; r_P) \text{ and } z = P(x, w, e; r_P)$$
;

3. $\forall x,\ \forall e, e' \in E,\ \forall a, z, z',$ if $e \neq e'$ and $(V(x,a,e,z) = 1 \ \wedge \ V(x,a,e',z') = 1) \implies R(x, \mathcal{E}(x,a,e,z,e',z'))$;
4. $\forall (x,w) \in R,\ \forall e \in E,$

$$\text{distrib}_{r_P}(a,e,z) = \text{distrib}(S(x,e;r))$$

with a and z defined by $a = P(x,w;r_P)$ and $z = P(x,w,e;r_P)$;

$$\begin{array}{lcl}
\underline{\textbf{Prover } P(x,w;1^\lambda)} & & \underline{\textbf{Verifier } V(x;1^\lambda)} \\
a \leftarrow P(x,w) & \xrightarrow{a} & \\
 & & e \leftarrow V(x,a) \\
 & \xleftarrow{e} & \\
z \leftarrow P(a,e,x,w) & \xrightarrow{z} & \text{Accept iff } V(x,a,e,z) = 1
\end{array}$$

Figure 1: A Σ-protocol for a language L.

3 Fiat-Shamir Transformation

3.1 Description

The Fiat-Shamir heuristic is originally designed [FS86] to transform 3-round identification schemes into efficient signature schemes. Given that the Σ-protocols are an extension of 3-round schemes, the Fiat-Shamir transform applies to any variant of the Σ-protocols. The main goal of the Fiat-Shamir transform is to eliminate the interaction between a prover P and a verifier V by replacing the challenge c in the second round (*Figure 1*) by a hash value $H(a,x)$ computed by the prover P where a is the *commitment* of P in the first round. H is a hash function that is modelled as a random oracle. From this modification of the second round, we obtain a non-interactive protocol (P,V).

$$\begin{array}{lcl}
\underline{\textbf{Prover } \mathcal{P}(x,w;1^\lambda)} & & \underline{\textbf{Verifier } \mathcal{V}(x;1^\lambda)} \\
a \leftarrow \mathcal{P}(x,w) & & \\
e \leftarrow H(x,a) & & \\
z \leftarrow \mathcal{P}(a,e,x,w) & \xrightarrow{a,z} & \text{Accept iff } \mathcal{V}(x,a,H(x,a),z) = 1
\end{array}$$

Figure 2: Non-interactive version of the Σ-protocol using Fiat-Shamir Transformation

Generalizing the heuristic above, we show that any language that has an interactive proof can have its proof efficiently transformed into a non-interactive zero- knowledge one, as described in [BR93]. The model of computation is that all parties, including cheating provers, are afforded only polynomially many queries to the random oracle.

3.2 NIZK from interactive zero-knowledge

The description for this transformation has been heavily adapted for this paper from [BR93]. To keep the same notations coherent with the rest of the report we now redefine some of the primitives and properties already mentioned in the sigma protocols. Let (P', V') be a zero-knowledge (ZK) proof for a language $L \subseteq \text{NP}$, utilizing the standard model (devoid of a random oracle) with an error probability of $\frac{1}{2}$. Let $k(n) = \omega(\log n)$. We aim to construct a non-interactive ZK (NIZK) proof (P, V) in the random oracle model which achieves an error $\epsilon(n) = 2^{-k(n)}$, while only increasing computing time and the number of communicated bits by a factor of at most $O(k(n))$.

For a language $L \in NP$, we say that (P, V) is an interactive proof for a language L in the random oracle model with an error function $\epsilon(n)$ under the following two conditions:

1. **Completeness:** If $x \in L$, then for all witnesses w to the membership of x in L, it must be that $\Pr[(V, x, a, e, z) \text{ accepts }] = 1$.
2. **Soundness:** For all probabilistic polynomial-time (PPT) cheating prover $\hat{P}$ and input $x \notin L$, it should hold that $\Pr[(V, x, a, e, z) \text{ accepts}] \leq \epsilon(|x|)$.

Consider a zero-knowledge proof (ZKP) for a language $L \in \text{NP}$, utilizing a random oracle to achieve an error probability of $\frac{1}{2}$. Let $k(n) = \omega(\log n)$ represent a given function. Our goal is to develop a non-interactive zero-knowledge (NIZK) proof (P, V) in the random oracle model that achieves an error $\epsilon(n) = 2^{-k(n)}$ and only increases the computational and communication overhead by a factor of $O(k(n))$. Typically, such ZKP systems involve three moves protocols as described above. The essence of zero-knowledge is captured by an algorithm S' which, given x, e, produces a, e, z such that $\Pr[(V, x, a, e, z) \text{ accepts }] = 1$ and the ensembles:

- $\{(a, e, z) | e \leftarrow \{0, 1\}, (a, e, z) \leftarrow S'(x, e)\}_{x \in L}$
- $\{(a, e, z) | e \leftarrow \{0, 1\}, a \leftarrow P'(x), z \leftarrow P'(x, a, e)\}_{x \in L}$

are indistinguishable. Notice: in the original paper [BR93], they denote the conversation in a single variable, for example we could say aez to say that the prover has sent a, it got e by the verifier and then finally the prover sent z. For simplicity, we keep these interactions separate.

Let H be a random hash function from the family $\mathcal{H}$. Recall that we require H to have output length denoted by ℓ_{out}. The new prover, now denoted as P^H computes:

- $a_1 \leftarrow P(x); ...; a_{\ell_{out}} \leftarrow P(x)$
- it sets the challenge e_i to the i-th bit of the $H(a_1, ..., a_{\ell_{out}})$
- Computes $z_1 \leftarrow P(x, a_1 e_1); ...; z_{\ell_{out}} \leftarrow P(x, a_{\ell_{out}} e_{\ell_{out}})$
- It sends $(a_1, ..., a_{\ell_{out}}, z_1, ..., z_{\ell_{out}})$ to the verifier who accepts if $V(a_i, e_i, z_i) = 1$ for all i.

It is clear that this protocol is non-interactive as claimed.

For honest prover and honest verifier the protocol behaves as interned, thus completeness is clear. For soundness, we can show that if a cheating prover $\hat{P}^H$ makes $T(n)$ oracle queries then $\Pr[V(x, a, e, z) = 1] \leq T(n) * 2^{-2*k(n)}$ which is at most $2^{-k(n)}$ for sufficiently large n. To prove the ZK, we only need to simulate the view of the honest verifier V. Given an $x \in L$, the simulator S chooses $e_1 \leftarrow \{0, 1\}; ...; e_{\ell_{out}} \leftarrow \{0, 1\}$ and it takes computes $(a_i, e_i, z_i) \leftarrow S'(x, e_i)$. It sets $T = (a_1, ..., a_{\ell_{out}}, e_1, ..., e_{\ell_{out}})$ and outputs (x, T). When T is passed to the random oracle, the output is subject to constraints that $H(a_1, ..., a_{\ell_{out}}) = e_1, ..., e_{\ell_{out}}$, which based on the indistinguishability assumption on S', is ZK.

Below is a concrete instantiation of the FS transformation applied to a single key signing algorithm for which we later provide security proofs as shown in [PS96].

- **Key Generation:** for a security parameter λ, we choose two large primes p, q and we compute $N = p * q$. We also pick a random hash function $f_\lambda \in H$ (sometimes we might write f instead of f_λ for simplicity) and a random $s \in \mathbb{Z}/N\mathbb{Z}$. From this we calculate $v = s^2 \mod N$. N, f are public while p, q, s are kept secret.
- **Signing:** to sign a message m, we generate λ random numbers, $r_i \in \mathbb{Z}/N\mathbb{Z}$ for $i = 1, ..., \lambda$ and set $x_i = r_i^2 \mod N$. We compute the challenge $h = (e_1, ..., e_k) = f(m, x_1, ..., x_\lambda)$. We can now set $y_i = r_i \cdot s^{e_i} \mod N$ and we output $\sigma_1 = (x_1, ..., x_\lambda)$ and $\sigma_2 = (y_1, ..., y_\lambda)$. The signature is $(\sigma_1, h, \sigma_2$.
- **Verification:** for a given message m and signature (σ_1, h, σ_2), we check whether $h = f(m, \sigma_1)$ and if $y_i = r_i s^{e_i} \mod N$ for all i.

To prove the soundness of the Fiat-Shamir transformation in the ROM, we first mention the following lemma:

Lemma 3.1 (**Forking Lemma**). *Let $\mathcal{A}$ be a PPT Turing machine, given only the public data as input. If $\mathcal{A}$ can find with non-negligible probability a valid signature $(m, \sigma_1, h, \sigma_2)$, then with non-negligible probability, a replay of this machine, with the same random tape and a different oracle, produces two valid signatures $(m, \sigma_1, h, \sigma_2)$ and $(m, \sigma_1', h, \sigma_2')$ such that $h \neq h'$. [PS96]*

To prove the soundness of FS in the ROM we need to first mention :

Lemma 3.2. *Let $A \subset X \times Y$ such that $Pr[A(x, y)] \geq \epsilon$, then $\exists \Omega \subset X$ such that :*

1. *$Pr[x \in \Omega] \geq \frac{\epsilon}{2}$*
2. *whenever $a \in \Omega$, $Pr[A(a, y)] \geq \frac{\epsilon}{2}$*

Using this lemma, we can split X into a subset Ω that consists of "*good*" x's which provide a non-negligible probability of success over y and a second subset which is its complement. Proof [PS96]

3.3 Fiat-Shamir security under the Random Oracle Model

When considering the security of a Fiat-Shamir transformed signature scheme, we consider an adversary A that tries to forge a proof for some input $x \in X$. If x is fixed, then the adversary is called *static* otherwise if the adversary A chooses x along with a for the case of the Σ protocol and Π for the case of FS transform, then the adversary is *adaptive.*

Theorem 3.3. *We consider a no-message attack in the random oracle. If there is an adversary $\mathcal{A}$ that is successful in existential forgery against Fiat Shamir with non-negligible probability, then $\mathcal{A}$ can factor RSA moduli in PPT.*

Proof: *Let $N \in \mathbb{N}$ be the integer to factor (RSA moduli). We choose $s \in_R \mathbb{Z}/N\mathbb{Z}$ and let $v = s^2 \mod N$. If an attacker $\mathcal{A}_1$ can break the FS signature scheme, then by using the forking lemma he can obtain two valid signatures $(m, \sigma_1, h, \sigma_2)$ and $(m, \sigma_1', h', \sigma_2')$ such that $h \neq h'$. From this we then get i such that $h_i \neq h_i'$, say $h_i = 0$ and $h_i' = 1$. We get $(y_i)^2 \equiv x_i \pmod N$ and $(y_i')^2 \equiv x_i \cdot v \pmod N$. If we let $z = y_i' y_i^{-1} \pmod N$, then $z^2 \equiv v \equiv s^2 \pmod N$.*

Since the algorithm cannot distinguish s from other roots, $gcd(z - s, N)$ provides a factor of N with probability $\frac{1}{2}$.

4 Failures of random oracle

This section demonstrates that the security of a cryptographic scheme in the Random Oracle Model does not always imply its security under any specific "good hash function" used to implement the random oracle. To illustrate this, we construct signature and encryption schemes that are secure in the Random Oracle Model, but become insecure with any implementation of the random oracle using a function ensemble. In essence, while the ideal scheme is secure, any real-world implementation is inherently insecure.

The idea is to start with a scheme secure in the Random Oracle Model, but easily broken when the random oracle is replaced by any ensemble. This is achieved using sparse relations. The modified scheme attempts to find an input that, together with its output, forms a pair in the sparse relation. If successful, the scheme performs an insecure action (e.g., revealing the secret key). Otherwise, it behaves like the original secure scheme. In the Random Oracle Model, finding such a pair is rare, thus maintaining security. However, any implementation of the random oracle will allow an adversary to always find such pairs, making the scheme insecure.

We begin with signature schemes and present the construction in three steps [CGH04]:

- In the first step, we simply show that for any function ensemble F, there exists a signature scheme that is secure in the Random Oracle Model but insecure when implemented using F. This implies that no function ensemble can securely implement any cryptographic scheme shown to be secure in the Random Oracle Model. However, it does not rule out the possibility that for any secure scheme in the Random Oracle Model, there exists some secure implementation (via a different function ensemble).
- In the second step, we use diagonalization techniques to reverse the order of quantifiers. We demonstrate that there exists a signature scheme that is secure in the Random Oracle Model, but any implementation (using any function ensemble) results in an insecure scheme. This step involves constructing a scheme with signing and verification procedures that run in (slightly) super-polynomial time.
- In the third step, we use CS-proofs to eliminate the super-polynomial running time of the legitimate procedures, resulting in a standard signature scheme that is secure in the Random Oracle Model but has no secure implementation. Specifically, CS-proofs are used to "diagonalize against all polynomial-time ensembles in polynomial time."

4.1 First Step

We let $S = (G, S, V)$ be a signature scheme and we let R be any binary sparse relation with respect to the length function ℓ_{out}. By $S_R^{\mathcal{O}}$ we denote the signature scheme S that has access to a random oracle $\mathcal{O}$, with the following modifications:

- **Modified signature** $S_R^{\mathcal{O}}(sk, msg)$:
  1. If $(msg, \mathcal{O}(msg)) \in R$ then output (sk, msg)
  2. Otherwise, output $S^{\mathcal{O}}(sk, msg)$
- **Modified verification** $V_R^{\mathcal{O}}(pk, msg, \sigma)$:
  1. If $(msg, \mathcal{O}(msg)) \in R$ accept

2. Otherwise output $V^{\mathcal{O}}(pk, msg, \sigma)$.

The purpose of this modification is to illustrate how security in the Random Oracle Model (ROM) might not translate to real-world security when the random oracle is instantiated with any function ensemble. In the ROM, finding pairs $(msg, \mathcal{O}(msg)) \in R$ is rare due to the random nature of $\mathcal{O}$. Therefore, the scheme $S_R^{\mathcal{O}}$ remains secure in this model.

However, when replacing the random oracle with any deterministic function ensemble, an adversary might be able to consistently find such pairs, triggering the insecure action of revealing the secret key. This demonstrates that while ROM provides a useful abstraction for analyzing security, it may not always guarantee security in practical implementations.

If we use $\mathcal{F} = \{f_s\}$ an ℓ_{out}-ensemble instead of $\mathcal{O}$, then we can define the binary relation:

$$R^{\mathcal{F}} = \bigcup_{\lambda} \left\{ (s, f_s(s)) : s \in \{0,1\}^{\lambda} \right\}$$

Proposition 4.1. *Suppose that R is sparse (with respect to ℓ_{out}) and that S is existentially unforgeable under a chosen message attack in the Random Oracle Model. Then S_R is also existentially unforgeable under a chosen message attack in the Random Oracle Model.*

Proof. The idea behind the proof is that since R is sparse, it is infeasible for an adversary to find a message m such that $(m, O(m)) \in R$. Therefore, any successful forgery of the modified scheme S_R must be due to the second condition, which contradicts the security of the original scheme S.

Formally, let A_R be an adversary conducting an adaptive chosen message attack on S_R, with a success probability of obtaining an existential forgery (in the Random Oracle Model) denoted by $\epsilon_{\text{frg}} = \epsilon_{\text{frg}}(\lambda)$, where λ is the security parameter. Assume, for contradiction, that ϵ_{frg} is not negligible with respect to λ.

Define the event REL as the event where, during an execution of A_R, it generates a message m for which $(m, O(m)) \in R$ (either as a query to the signer during the chosen message attack, or as the message used to produce a forgery). Let $\epsilon_{\text{rel}} = \epsilon_{\text{rel}}(\lambda)$ be the probability of this event. Given that R is sparse, we now aim to prove that ϵ_{rel} is negligible with respect to λ. Assume, for contradiction, that ϵ_{rel} is not negligible.

If ϵ_{rel} is not negligible, we can efficiently find pairs $(x, O(x)) \in R$ by selecting a key-pair for S and simulating the attack, acting as both the signer algorithm and the adversary A_R. With a probability of ϵ_{rel}, one of A_R's messages will satisfy $(m, O(m)) \in R$. Randomly selecting one of these messages and outputting it yields a success probability of ϵ_{rel}/q, where q is the number of different messages (queries) used in the attack. If ϵ_{rel} is not negligible, then neither is ϵ_{rel}/q, which contradicts the sparsity of R.

Excluding the event REL, the execution of A_R against the original scheme S would be identical to its execution against S_R. Therefore, the probability that A_R successfully produces an existential forgery against S is at least $\epsilon_{\text{frg}} - \epsilon_{\text{rel}}$. Since ϵ_{rel} is negligible and ϵ_{frg} is not, this implies that the probability of A_R producing an existential forgery against S is not negligible, contradicting the assumed security of S.

Therefore, S_R is existentially unforgeable under a chosen message attack in the Random Oracle Model. □

Claim 4.2. *For every efficiently computable ℓ_{out}-ensemble $\mathcal{F}$, there exists a signature scheme that is existentially unforgeable under a chosen message attack in the Random Oracle Model, yet when implemented with $\mathcal{F}$, the resulting scheme is totally breakable under an adaptive chosen message attack, and existentially forgeable under a key-only attack.*

Proof. The modification to S enables us to break the modified scheme S_R when implemented with a real ensemble $\mathcal{F}$, in the case where R is the relation R^F.

Consider now what happens when we use the ensemble $\mathcal{F}$ to implement the random oracle in the scheme S_R, we obtain the following real scheme (which we denote $S'_R = (G', S'_R, V'_R)$):

- $G'(1^\lambda)$: Uniformly pick $s \in \{0,1\}^\lambda$, set $(sk, pk) \leftarrow G^{f_s}(1^\lambda)$, and output $((sk, s), \langle pk, s\rangle)$.
- $S'_R((sk, s), msg)$: Output $S_R^{f_s}(sk, msg)$.
- $V'_R(\langle pk, s\rangle, msg, \sigma)$: Output $V_R^{f_s}(pk, msg, \sigma)$.

Consider now what happens when we use the ensemble $\mathcal{F}$ to implement the scheme S_{R^F}. Since R^F is evasive, then we infer that the S_{R^F} is secure in the Random Oracle Model. However, when we use the ensemble $\mathcal{F}$ to implement the scheme, the seed s becomes part of the public verification-key, and hence is known to the adversary. The adversary can simply output the pair $(s, \epsilon = f_s(s))$, that will be accepted by V'_{R^F} as a valid message-signature pair (since $(s, f_s(s)) \in R^F$). Hence, the adversary achieves existential forgery (of S'_{R^F} under key-only attack. Alternatively, the adversary can ask the legitimate signer for a signature on s, hence obtaining the secret signing-key (i.e., total forgery). □

4.2 Second Step

Enumeration. In this step, we need to enumerate all efficiently computable function ensembles. This enumeration is achieved by considering all polynomial-time algorithms that can evaluate such ensembles. However, several technical challenges arise. Enumerating all polynomial-time algorithms is difficult because there is no single polynomial that bounds the running time of all these algorithms. To address this, we fix an arbitrary super-polynomial complexity function $t : \mathbb{N} \to \mathbb{N}$ (e.g., $t(n) = n^{\log n}$), and enumerate all algorithms with running times bounded by t. We achieve this by enumerating all possible algorithms and adding a timeout mechanism that halts the execution if more than t(input size) steps are taken. This modification does not affect the polynomial-time algorithms. Additionally, since we are interested in ℓ_{out}-ensembles, we modify each function by treating its seed as a pair $\langle s, x\rangle$ (using a standard parsing rule) and adjusting its output length to $\ell_{\text{out}}(|s|)$. This modification also does not affect the ℓ_{out}-ensembles.

Let us denote by $\mathcal{F}^i$ the i-th function ensemble according to the above enumeration, and denote by f_s^i the function indexed by s from the ensemble $\mathcal{F}^i$. For parsing a string α, we use a standard rule to interpret it as a pair $\langle i, s\rangle$ and view it as a description of the function f_s^i.

Universal Ensemble. Let $\mathcal{U} = \{U_k\}_{\lambda \in \mathbb{N}}$ be the "universal function ensemble" derived from the enumeration described above. Specifically, $U_k = \{u_{\langle i,s\rangle}\}_{\langle i,s\rangle \in \{0,1\}^\lambda}$ and $u_{\langle i,s\rangle}(x) = f_s^i(x)$. There exists a machine that computes the universal ensemble $\mathcal{U}$ and operates in slightly super-polynomial time t.

Universal Relation. Define $R^{\mathcal{U}}$ as the universal relation associated with the universal ensemble $\mathcal{U}$, analogous to how R^F is defined for any ensemble $\mathcal{F}$. Formally:

$$R^{\mathcal{U}} = \bigcup_k \left\{ (\langle i, s\rangle, f_s^i(\langle i, s\rangle)) : \langle i, s\rangle \in \{0,1\}^\lambda \right\}$$

In other words:

$$(x, y) \in R^{\mathcal{U}} \iff y = u_x(x) \quad \text{(i.e., } x = \langle i, s\rangle \text{ and } y = f_s^i(x)\text{)}$$

Modified Signature Scheme

Let $S = (G, S, V)$ be a signature scheme (as above). We then denote by $S_u = (G, S_u, V_u)$ the modified signature scheme that is derived by using $R^{\mathcal{U}}$ in place of R in the previous construction. Specifically:

- $S_u^O(sk, msg) =$
  1. If $(msg, \mathcal{O}(msg)) \in R^{\mathcal{U}}$, output (sk, msg)
  2. Otherwise, output $S^{\mathcal{O}}(sk, msg)$.
- $V_u^O(pk,\ msg\ , \sigma) =$
  1. If $(msg, \mathcal{O}(msg)) \in R^{\mathcal{U}}$ then accept.
  2. Otherwise, output, $V^O(pk, \text{msg}, \sigma)$.

We note that since these signature and verification algorithms need to compute $\mathcal{U}$, they both run in time $O(t)$, which is slightly super-polynomial.

Proposition 4.3. *Suppose that S is existentially unforgeable under a chosen message attack in the Random Oracle Model. Then S_u is also existentially unforgeable under a chosen message attack in the Random Oracle Model, but implementing it with any function ensemble yields a scheme that is totally breakable under a chosen message attack and existentially forgeable under a key-only attack.*

Proof. Since $R^{\mathcal{U}}$ is evasive, then it follows that S_u is secure in the Random Oracle Model. On the other hand, suppose that one tries to replace the random oracle in the scheme by an ensemble $\mathcal{F}^i$ (where i be the index in the enumeration). An adversary, given a seed s of a function in $\mathcal{F}^i$ can then set msg $= \langle i, s\rangle$ and output the pair (msg, ϵ), which would be accepted as a valid message-signature pair by V_u. Alternatively, it can ask the signer for a signature on this message msg, and so obtain the secret signing-key. □

4.2.1 Third Step

We now use CS-proofs to construct a new signature scheme that operates within the Random Oracle Model. This approach is similar to the one detailed in step 2, but rather than verifying that $(\text{msg}, \mathcal{O}(\text{msg})) \in R^U$, the signer/verifier obtains a CS-proof for this claim, requiring only the validation of the CS-proof. Since validating a CS-proof is significantly more efficient than verifying the claim from scratch, the signing and verification algorithms in the new scheme can run in polynomial time. Conversely, when the scheme is implemented using the function ensemble F^i, providing the appropriate CS-proof (i.e., for $(\text{msg}, f_s^i(\text{msg})) \in R^U$) only necessitates polynomial-time operations (specifically, time polynomial in the evaluation of f_s^i). This leads to the following result:

Theorem 4.4. *There exists a signature scheme that is existentially unforgeable under a chosen message attack in the Random Oracle Model, but when implemented with any function ensemble, the resulting scheme is existentially forgeable using a key-only attack and totally breakable under a chosen message attack.*

It is worth noting that, unlike the signature scheme discussed in the second step, the signature scheme presented here operates in polynomial time.

Proof. We outline the construction of such a signature scheme using the following components:

- Let $S = (G, S, V)$ be a signature scheme operating in the Random Oracle Model, which is existentially unforgeable under a chosen message attack.
- A straightforward and easily computable parsing rule that interprets messages as triples of strings, i.e., $\mathrm{msg} = \langle i, s, \pi \rangle$.
- The algorithms Prv and Ver of a CS-proof system.
- Access to three independent random oracles, which is feasible with a single oracle $\mathcal{O}$. Specifically, define $\mathcal{O}'(x) = \mathcal{O}(01 \parallel x)$, $\mathcal{O}''(x) = \mathcal{O}(10 \parallel x)$, and $\mathcal{O}'''(x) = \mathcal{O}(11 \parallel x)$. We will use $\mathcal{O}'''$ for the basic scheme S, $\mathcal{O}''$ for the CS-proofs, and $\mathcal{O}'$ for our evasive relation. Note that if $\mathcal{O}$ is an ℓ_{out}-oracle, then so are $\mathcal{O}'$, $\mathcal{O}''$, and $\mathcal{O}'''$.
- The universal function ensemble $\mathcal{U}$ from step 2, with proper complexity bound $t(n) = n^{\log n}$. We denote by $M_{\mathcal{U}}$ the universal machine that decides the relation $R^d_{\mathcal{U}}$. On input $(\langle i, s \rangle, y)$, $M_{\mathcal{U}}$ invokes the i-th evaluation algorithm and accepts if $f^i_s(\langle i, s \rangle) = y$. $M_{\mathcal{U}}$ works in time t in the worst case. More importantly, if F^i is a function ensemble that can be computed in time $p_i(\cdot)$ (where p_i is some polynomial), then for any strings s, y on input $(\langle i, s \rangle, y)$, machine $M_{\mathcal{U}}$ works for only $\mathrm{poly}(|i|) \cdot p_i(|s|)$ steps.

Using all the above, we describe an ideal signature scheme $S'_{\mathcal{U}} = (G, S'_{\mathcal{U}}, V'_{\mathcal{U}})$. As usual, the key generation algorithm, G, remains unchanged. The signature and verification algorithms proceed as follows:

$S'^{\mathcal{O}}_{\mathcal{U}}(sk, msg) =$

1. Parse msg as $(\langle i, s, \pi \rangle)$, and set $x = \langle i, s \rangle$ and $y = \mathcal{O}'(x)$. Let $n = |(x, y)|$.
2. Apply $\mathrm{Ver}^{\mathcal{O}''}$ to verify whether π is a valid CS-proof, with respect to the oracle $\mathcal{O}''$ and security parameter $1^{n+\lambda}$, for the claim that the machine $M_{\mathcal{U}}$ accepts the input (x, y) within time $t(n)$.
   - *(The key point is that we do not directly check whether the machine $M_{\mathcal{U}}$ accepts the input (x, y) within time $t(n)$, but rather only if π is a valid CS-proof of this claim. Although $t(n) = n^{\log n}$, this CS-proof can be verified in polynomial time.)*
3. If π is a valid proof, then output $(\mathrm{sk}, \mathrm{msg})$.
4. Otherwise, output $S^{\mathcal{O}'''}(\mathrm{sk}, \mathrm{msg})$.

$V'^{\mathcal{O}}_{\mathcal{U}}(pk, msg, \sigma) =$

1. Step 1 and 2 same as above.
2. If π is a valid proof, then accept
3. Otherwise, output $V^{\mathcal{O}'''}(vk, msg, \sigma)$.

The computation required in step 2 above of the signature and verification algorithms can be executed in polynomial time. This is because, by definition, verifying a CS-proof can be done in polynomial time, provided the statement can be decided in at most exponential time (which is the case here since we have $t(n) = O(n^{\log n})$). It is also easy to see that for every pair $(\mathrm{sk}, \mathrm{vk})$ output by G, and for every msg and every $\mathcal{O}$, the string $S'^{\mathcal{O}}_{\mathcal{U}}(\mathrm{sk}, \mathrm{msg})$ constitutes a valid signature of msg relative to vk and the oracle $\mathcal{O}$.

To show that the scheme is secure in the Random Oracle Model, we first observe that on security parameter 1^{λ} it is infeasible to find a string x such that $(x, \mathcal{O}'(x)) \in R^d_{\mathcal{U}}$, since $R^d_{\mathcal{U}}$ is evasive. By construction, it is also infeasible to find (x, π) such that $(x, \mathcal{O}'(x)) \notin R_{\mathcal{U}}$ and yet π is a valid CS-proof of the contrary relative to $\mathcal{O}''$ (with security parameter $1^{|x|+\ell_{\text{out}}(\lambda)+\lambda}$). Thus, it is infeasible for a polynomial-time adversary to find a message that would pass the test in Item 2 of the signature/verification algorithms above, and so we infer that the modified signature is secure in the Random Oracle Model.

We now show that for every candidate implementation $\mathcal{F}$, there exists a polynomial-time adversary effecting total break via a chosen message attack (or, analogously, an existential forgery via a "key-only" attack). First, for each function $f_s \in \mathcal{F}$, denote $f_s(x) = f_s(01x)$, $f'_s(x) = f_s(10x)$, and $f''_s(x) = f_s(11x)$. Then denote by $\mathcal{F}'$ the ensemble of the f'_s functions.

Suppose that $\mathcal{F}'$ is the i-th function ensemble in the enumeration mentioned above, namely $\mathcal{F}' = F^i$. Given a randomly chosen λ-bit seed s, the adversary generates a message $\text{msg} = \langle i, s, \pi \rangle$ so that π is a CS-proof (with respect to the adequate security parameter) for the true statement that $M_{\mathcal{U}}$ accepts the input (x, y) within $t(|x| + |y|)$ steps, where $x = \langle i, s \rangle$ and $y = f'_s(x)$. Recall that the above statement is indeed true (since $f'_s = f^i_s$), and hence the adversary can generate a proof for it in time which is polynomial in the time that it takes to compute f^i_s. (By the perfect completeness property of the CS-proof system, the ability to prove correct statements holds for any choice of the random oracle, and in particular when it is equal to f'_s.) Since this adversary is specifically designed to break the scheme in which the random oracle is implemented by $\mathcal{F}$, then the index i - which depends only on the choice of $\mathcal{F}$ - can be incorporated into the program of this adversary.

By the efficiency condition of CS-proofs, it is possible to find (π). $\square$

In this chapter, we explored the inherent limitations of the Random Oracle Model (ROM) in guaranteeing the security of cryptographic schemes when instantiated with real-world hash functions. We demonstrated that a scheme secure in the ROM does not necessarily remain secure when the random oracle is replaced with any function ensemble. This discrepancy is highlighted through the construction of signature and encryption schemes that, while secure in the ROM, become vulnerable when implemented with any practical hash function.

We presented a step-by-step construction of a signature scheme to illustrate this point. First, we showed that for any function ensemble, a signature scheme secure in the ROM can become insecure when the random oracle is replaced by that ensemble. Then, using diagonalization techniques, we constructed a signature scheme that is secure in the ROM but insecure with any function ensemble. Finally, we utilized CS-proofs to eliminate super-polynomial running times, resulting in a polynomial-time signature scheme that is secure in the ROM but has no secure implementation.

Some focus points include:

- **Sparse Relations:** Using sparse relations, we illustrated how a scheme secure in the ROM can be easily broken when instantiated with any function ensemble.

- **Universal Ensemble and universal relation:** We defined a universal function ensemble and its associated universal relation, demonstrating the infeasibility of maintaining security across all polynomial-time ensembles.

- **CS-Proofs:** By incorporating CS-proofs, we constructed a scheme that remains efficient while highlighting the gap between theoretical security in the ROM and practical security.

Through these constructions, we emphasized the caution required when translating cryptographic security from the idealized ROM to real-world implementations.

5 Correlation Intractable Hash Functions and Fiat-Shamir

In this section, we explore the concept of correlation intractable hash functions and how they underpin the security of the Fiat-Shamir transformation. In the previous section, we demonstrated the security of the Fiat-Shamir transform within the random oracle model and we also explained its limitations. Now, we turn our attention to the main objective of this paper and we try to provide CI constructions to showcase some uses of CI hash functions.

To begin, let's first define what we mean by a hash family.

Definition 5.1 (Hash Family). [LV20] . A family of functions $\mathcal{H} = \{h_k : \mathcal{K}(\lambda) \times \mathcal{D}(\lambda) \to \mathcal{C}(\lambda)\}$ of keyed hash functions along with a pair of PPT algorithms:

- $H.Gen(1^\lambda)$ outputs a hash key $k \in \{0,1\}^{\mathcal{K}(\lambda)}$ describing a hash function h
- $H.Hash(k, x)$ computes the function $h_k(x)$. We may just use the notation $h(x)$ instead of $h_k(x)$ to denote the hash evaluation when the hash family is clear from the context.

Now, while random oracles provide an idealised model for analysing cryptographic protocols, real-world implementations need to withstand practical attacks. To bridge this gap, we introduce the concept of correlation intractability. This property ensures that the hash functions remain robust against specific attacks (Chosen Input Attack, Collision Attack, Forgery Attack, etc), even when we move beyond the theoretical nature of random oracles. Correlation intractability is a crucial property that ensures hash functions remain robust against specific attacks, even beyond the idealized random oracle model.

5.1 Correlation Intractability

We begin by defining single-input correlation intractability [LV20].

Definition 5.2 (Correlation Intractability). Let $\ell_{out} : N \to N$ be a length function and let $\mathcal{F}$ be an ℓ_{out} function ensemble. For a binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$, we say that $\mathcal{F}$ is correlation intractable with respect to R if for every PPT machine M, we have that:

$$\Pr_{s \in \{0,1\}^\lambda} [x \leftarrow M(s),\ (x, f_s(x)) \in R] = \text{negl}(\lambda)$$

where the probability is taken over the choice of s and the coins of M. We say that $\mathcal{F}$ is correlation intractable, if it is correlation intractable with respect to every sparse relation. If we only consider the relations R such that there exists a PPT algorithm which on input (x, y) decides whether or not $(x, y) \in R$, we call it *Weak Correlation Intractability.*

We can extend the definition above (single input) to multi-input correlation intractability.

Definition 5.3 (**Multi-Input Correlation Intractability**)**.** For a given relation ensemble $R = \{R_\lambda \subseteq (\mathcal{D}(\lambda) \times \mathcal{C}(\lambda)$, a hash family $\mathcal{H} = \{h_\lambda : \{0,1\}^{\kappa(\lambda)} \times \{0,1\}^{\nu(\lambda)} \to \{0,1\}^{\mu(\lambda)}\}$ is said to be R-correlation intractable with security (s, δ) if for every s-size adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}$, we have:

$$\Pr_{\substack{k \leftarrow \mathcal{H}.\mathrm{Gen}(1^\lambda) \\ x=(x_1,\ldots,x_t) \leftarrow \mathcal{A}(k)}} [(x, y = (h(x_1), \ldots, h(x_t))) \in R] = O(\delta(\lambda)).$$

We say that $\mathcal{H}$ is R-correlation intractable with security δ if it is (λ^c, δ)-correlation intractable for all $c > 1$. Finally, we say that $\mathcal{H}$ is R-correlation intractable if it is $(\lambda^c, \frac{1}{\lambda^c})$-correlation intractable for all $c > 1$.

Claim 5.4. *There exits no correlation intractable ensembles, not even in the weak sense.*

Proof. For $\mathcal{F} = \{f_s\}$ an ℓ_{out}-ensemble recall the definition of the binary relation:

$$R^{\mathcal{F}} = \bigcup_\lambda \left\{ (s, f_s(s)) : s \in \{0,1\}^\lambda \right\}$$

Clearly, this relation is polynomial-time recognizable, since f_s can be computed in polynomial time. Also, the relation is sparse since for every $x \in \mathcal{D}(\lambda)$ there is at most one $y \in \mathcal{C}(\lambda)$ satisfying $(x, y) \in R^F$, and so

$$\Pr_y[(x, y) \in R^{\mathcal{F}}] \leq 2^{-\ell_{\text{out}}(\lambda)} = 2^{-\omega(\log \lambda)} = \text{negl}(\lambda).$$

On the other hand, consider the machine I that computes the identity function, $I(x) = x$ for all x. It violates the correlation intractability requirement, since for all λ,

$$\Pr_{s \in \{0,1\}^\lambda}[(I(s), f_s(I(s))) \in R^{\mathcal{F}}] = \Pr_{s \in \{0,1\}^\lambda}[(s, f_s(s)) \in R^{\mathcal{F}}] = 1.$$

In fact, since R^F is polynomial-time recognizable, even the weak correlation intractability of $\mathcal{F}$ is violated.

□

Definition 5.5. For a given relation $R = \{R_\lambda \subseteq \mathcal{D}(\lambda) \times \mathcal{C}(\lambda)\}$ and a hash family $\mathcal{H} = \{h_k : \mathcal{K}(\lambda) \times \mathcal{D}(\lambda) \to \mathcal{C}(\lambda)\}$, we say $\mathcal{H}$ is R-correlation intractable with security (σ, δ) if for every σ-size adversary $\mathcal{A} = (\mathcal{A}_\lambda)$, we have:

$$\Pr\left[(x, h_k(x)) \in \mathcal{R}_\lambda : k \overset{\$}{\leftarrow} \mathcal{H}.Gen(1^\lambda); x \overset{\$}{\leftarrow} \mathcal{A}(k)\right] = O(\delta(\lambda)).$$

In simpler terms, this means that even if an adversary $\mathcal{A}$ has computational resources that are upper bounded by σ (for example $\sigma = \text{poly}(\lambda)$ for a PPT adversary), the probability of $\mathcal{A}$ finding a pair $(x, h_k(x)) \in R_\lambda$ is *extremely low.* This property is crucial to ensure the security of cryptographic schemes.

In order to better quantify *extremely low*, we need to consider the notion of sparsity of relation ensembles, which tells us how likely random elements are to belong to a certain relation R_λ.

Definition 5.6 (Sparsity)**.** For any relation ensemble $R = \{R_\lambda \subseteq \mathcal{D}(\lambda) \times \mathcal{C}(\lambda)\}$, we say that R is $p(\cdot)$-sparse if for all $\lambda \in \mathbb{N}$ and for any $x \in \mathcal{D}(\lambda)$, it holds that $(x, y) \in R_\lambda$ with probability at most $p(\lambda)$ over the choice of $y \in \mathcal{C}(\lambda)$. If p is a negligible function, then we say that R is sparse.

Definition 5.7 (Unique output relation)**.** We say that a relation R is a unique output relation if, for every $x \in \mathcal{D}(\lambda)$, there exists at most one output $y \in \mathcal{C}(\lambda)$ such that $(x, y) \in R$.

Definition 5.8 (1-universality)**.** We say that a family $\mathcal{H}$ is *1-universal* if for any $\lambda \in \mathbb{N}$, input $x \in \mathcal{D}(\lambda)$ and output $y \in \mathcal{C}(\lambda)$, we have:

$$\Pr\left[h_k(x) = y \mid k \overset{\$}{\leftarrow} \mathcal{H}.Gen(1^\lambda)\right] = 2^{-\mathcal{C}(\lambda)}.$$

We say that $\mathcal{H}$ is programmable if it is 1-universal and if there exists an efficient sampling algorithm $\text{Samp}(1^\lambda, x, y)$ that samples from the conditional distribution

$$k \overset{\$}{\leftarrow} \mathcal{H}.Gen(1^\lambda) \mid h_k(x) = y$$

.

We

5.2 Instance-Dependent Trapdoor Σ-Protocols

Before proving the security of Fiat-Shamir transform using a CI hash function, we need to give a different definition of the Σ-Protocols.

Definition 5.9 (CRS Σ-Protocol)**.** We say that a three-message honest-verifier zero-knowledge proof system $\Sigma = (Gen, P, V)$ in the Common Reference String model is a Σ-protocol if for every common reference string CRS, every instance $x \notin L$, and every first message a, there is at most one challenge $e := f(CRS, x, a)$ such that (CRS, x, a, e, z) is an accepting transcript for any choice of third message z.

We informally call f the *"bad-challenge function"* associated to Σ, and note that f may not be efficiently computable.

We now define a trapdoor Σ-protocol to be a Σ-protocol that has a trapdoor making the bad-challenge function f efficiently computable.

Definition 5.10 (Trapdoor Σ-Protocol)**.** We say that a Σ-protocol $\Sigma = (Gen, P, V)$ with bad-challenge function f is a trapdoor Σ-protocol if there are PPT algorithms **TrapGen**, **BadChallenge** with the following syntax:

- **TrapGen**(1^λ) takes as input the security parameter. It outputs a common reference string CRS along with a trapdoor τ.

- **BadChallenge**(τ, CRS, x, a) takes as input a trapdoor τ, common reference string CRS, instance x, and first message a. It outputs a challenge e.

We additionally require the following properties.

- **CRS Indistinguishability:** An honestly generated common reference string CRS is computationally indistinguishable from a common reference string output by $TrapGen(1^\lambda)$.

- **Correctness:** For every instance $x \notin L$ and for all $(CRS, \tau) \leftarrow TrapGen(1^\lambda)$, we have that $BadChallenge(\tau, CRS, x, a) = f(CRS, x, a)$.

While this definition is sufficient to describe our modification to the Σ-protocol, it is inherently restricted to using a common reference string. To encompass the original protocol [Dam10], we extend our definition so that the trapdoor τ can depend on the instance x.

Definition 5.11 (Instance-Dependent Trapdoor Σ-Protocol)**.** We say that a Σ-protocol $\Sigma = (Gen, P, V)$ with bad-challenge function f is an instance-dependent trapdoor Σ-protocol if there are PPT algorithms *TrapGen*, *BadChallenge* with the following syntax:

- **TrapGen**$(1^\lambda, x, \text{aux})$ takes as input the security parameter, an instance x, and an auxiliary input aux. It outputs a common reference string CRS along with a trapdoor τ.
- **BadChallenge**(τ, CRS, x, a) takes as input a trapdoor τ, common reference string CRS, instance x, and first message a. It outputs a challenge e.

We additionally require the following properties.

- **CRS Indistinguishability:** For any (x, aux), an honestly generated common reference string CRS is computationally indistinguishable from a common reference string output by $\text{TrapGen}(1^\lambda, x, \text{aux})$.
- **Correctness:** For every instance $x \notin L$, there exists an auxiliary input aux such that for all $(CRS, \tau) \leftarrow \text{TrapGen}(1^\lambda, x, \text{aux})$, we have that $\text{BadChallenge}(\tau, CRS, x, a) = f(CRS, x, a)$.

Given this definition, we can now state our result on Fiat-Shamir for instance-dependent trapdoor Σ-protocols.

5.3 FS transform security using CI hash functions

Theorem 5.12. *[Can+19] Suppose that $\mathcal{H}$ is a hash family that is correlation intractable for all sub-exponentially sparse relations that are searchable in time T. Let*

$$\Sigma = (Gen, P, V, TrapGen, BadChallenge)$$

be an instance-dependent trapdoor sigma protocol with $2^{-\lambda^\epsilon}$ soundness for some $\epsilon > 0$, such that $BadChallenge(\tau, CRS, x, a)$ is computable in time T. We then have that $\mathcal{H}$ soundly instantiates the Fiat-Shamir heuristic for Σ.

Proof. Let Σ_{FS} denote the one-message protocol resulting from applying the Fiat-Shamir transform to a 3-round protocol Σ using a hash function from $\mathcal{H}$. We define Σ_{FS} as follows:

- The common reference string CRS_Σ associated to Σ along with the hash key $k \overset{\$}{\leftarrow} \mathcal{H}.Gen(1^\lambda)$.
- Prover's message: (a, e, z) where $a = \Sigma.P(CRS_\Sigma, x, w)$, $e = h_k(a)$ and $z = \Sigma.P(CRS_\Sigma, x, w, a, e)$.
- The verifier accepts a transcript $(CRS_\Sigma, k, x, a, e, z)$ if $\Sigma.V(CRS_\Sigma, a, e, z) = 1$ and $e = h_k(a)$

By definition of Σ-protocols, we know that for every $x \notin L$ and every CRS_Σ, an accepting $(CRS_\Sigma, k, x, a, e, z)$ transcript must satisfy the condition that $h_k(a) = e = f(CRS_\Sigma, x, a)$.

Suppose that some efficient prover P^*, given $x \notin L$ and a random CRS $= (CRS_\Sigma, k)$, could find (a, e, z) making the transcript $(CRS_\Sigma, k, x, a, e, z)$ accepting with non-negligible probability. Then, by CRS indistinguishability, the same would be true for CRS_Σ sampled by the algorithm $\text{TrapGen}(1^\lambda, x, \text{aux})$ for an auxiliary input *aux* satisfying the correctness property of instance-independent $\Sigma - protocols$. In other words, for

$$(CRS_\Sigma, \tau_\Sigma) \leftarrow \text{TrapGen}(1^\lambda, x, \text{aux}) \text{ and } CRS = (CRS_\Sigma, k),$$

$P^*(x, CRS)$ would output (with non-negligible probability) some a such that $h_k(a) = f(CRS_\Sigma, x, a) = \text{BadChallenge}(\tau_\Sigma, CRS_\Sigma, x, a)$.

This directly contradicts the correlation intractability of $\mathcal{H}$ for the relation

$$R_{\tau_\Sigma, CRS_\Sigma, x} = \{(a, e) : e = \text{BadChallenge}(\tau_\Sigma, CRS_\Sigma, x, a)\} ..$$

In more detail, a correlation-intractability adversary $\mathcal{A}$ could break the correlation intractability of $\mathcal{H}$ by sampling $(CRS_\Sigma, \tau_\Sigma)$ itself, declaring the relation $R_{\tau_\Sigma, CRS_\Sigma, x}$ to be broken, and then running $P^*(x, CRS)$ after being given $k \leftarrow \mathcal{H}.\text{Gen}(1^\lambda)$. Since Π originally had $2^{-\lambda^c}$ soundness, the relation $R_{\tau_\Sigma, CRS_\Sigma, x}$ indeed has sub-exponential sparsity, so this contradicts our assumption on $\mathcal{H}$. Thus, we conclude that $\mathcal{H}$ soundly instantiates the Fiat-Shamir heuristic for Π, as desired.

□

The above theorem and its proof establish a critical result: the security of the Fiat-Shamir transformation can be maintained using a correlation intractable hash family. This bridges the gap between theoretical constructs and practical implementations, ensuring robust security even in real-world scenarios.

In the next section, we will explore specific constructions of correlation intractable hash families that will further illustrate the practical applicability of the theoretical findings.

6 Correlation Intractability Constructions

6.1 CI Hash Functions via Shift-Hiding

We begin by first giving a definition of Shift-Hiding Shiftable Functions from [LV20]. We don't describe everything in details in this paper as it would deviate from our objectives.

Definition 6.1 (**Shift-Hiding Shiftable Functions**). . Let $p = p(\lambda)$ be an efficiently computable function of λ. We define a family of shift-hiding shiftable functions with input space $\mathcal{D}(\lambda)$ and output space $\mathbb{Z}_{p(\lambda)}^{\mu(\lambda)} = \{0,1\}^{\mu(\lambda)\log p(\lambda)}$ for arbitrary polynomial functions $(\nu(\lambda), \mu(\lambda))$.

For a given class $\mathcal{C}$ of function ensembles $\mathcal{F} = \{f_\lambda : \mathcal{D}(\lambda) \to \mathbb{Z}_{p(\lambda)}^{\mu(\lambda)}\}$, a shift-hiding shiftable function family SHSF = (Gen, Shift, Eval, SEval) consists of four PPT algorithms:

- $\text{Gen}(1^\lambda)$ outputs a master secret key msk and public parameters pp.
- $\text{Shift}(\text{msk}, f)$ takes as input a secret key msk and a function $f \in \mathcal{F}$. It outputs a shifted key sk_f.

- $\mathrm{Eval}(\mathrm{pp}, \mathrm{msk}, x)$, given a secret key msk and input $x \in \{0,1\}^{\nu(\lambda)}$, outputs an evaluation $y \in \mathbb{Z}_{p(\lambda)}^{\mu(\lambda)}$.
- $\mathrm{SEval}(\mathrm{pp}, \mathrm{sk}_f, x)$, given a shifted key sk_f and input $x \in \{0,1\}^{\nu(\lambda)}$, outputs an evaluation $y \in \mathbb{Z}_{p(\lambda)}^{\mu(\lambda)}$.

We will sometimes use the notation $F_{\mathrm{sk}}(x)$ to mean either $\mathrm{Eval}(\mathrm{sk}, x)$ or $\mathrm{SEval}(\mathrm{sk}, x)$ when the context is clear.

We require that SHSF satisfies the following two properties:

1. **Computational Correctness**: For any function $f \in \mathcal{C}$, given public parameters pp and a shifted key $\mathrm{sk}_f \leftarrow \mathrm{Shift}(\mathrm{msk}, f)$ (for $(\mathrm{pp}, \mathrm{msk}) \leftarrow \mathrm{Gen}(1^\lambda)$), it is computationally hard to find an input $x \in \{0,1\}^{\nu(\lambda)}$ such that $\mathrm{Eval}(\mathrm{sk}_f, x) \neq \mathrm{Eval}(\mathrm{msk}, x) + f(x) \mod p$. In other words, the equation
$$F_{\mathrm{sk}_f}(x) = F_{\mathrm{msk}}(x) + f(x)$$
holds computationally (mod p).
2. **Shift Hiding**: For any pair of functions $f, g \in \mathcal{C}$,
$$\mathrm{sk}_f \approx_c \mathrm{sk}_g,$$
where $\mathrm{sk}_f \leftarrow \mathrm{Shift}(\mathrm{msk}, f)$, $\mathrm{sk}_g \leftarrow \mathrm{Shift}(\mathrm{msk}, g)$, and $\mathrm{msk} \leftarrow \mathrm{Gen}(1^\lambda)$.

6.2 Towards a CI construction using SHSFs

Formally, let SHSF = (Gen, Shift, Eval) be a SHSF family that represents functions of the form $F_{\mathrm{sk}} : \mathcal{D}(\lambda) \to \mathbb{Z}_{p(\lambda)}^{\mu(\lambda)}$ and supports shifts for functions $f \in \mathcal{C}$, where $\mathcal{C}$ is some class that contains the all-zero function ensemble. We then consider two hash functions $\mathcal{H}_{\mathrm{plain}}$ and $\mathcal{H}_{\mathrm{shift}}$:

- $\mathcal{H}_{\mathrm{plain}}$ uses msk as a hash key and computes the function $h(\mathrm{msk}, x) = F_{\mathrm{msk}}(x)$.
- $\mathcal{H}_{\mathrm{shift}}$ uses sk_Z as a hash key, where $Z : \{0,1\}^\nu \to \mathbb{Z}_p$ is an identically zero function. It computes the function $h(\mathrm{sk}_Z, x) = F_{\mathrm{sk}_Z}(x)$.

Theorem 6.2. *Let R_{out} be an efficiently decidable output relation. If SHSF is a shift-hiding shiftable function family for $\mathcal{C}$ and $\mathcal{H}_{plain}$ is R_{out}-output intractable, then $\mathcal{H}_{shift}$ is (R, f)-correlation intractable for any $f \in \mathcal{C}$.*

Moreover, if $\mathcal{H}_{plain}$ is NAE-R_{out}-output intractable, then $\mathcal{H}_{shift}$ is NAE-(R, f)-CI for any $f \in \mathcal{C}$.

Proof. Suppose that a PPT adversary $\mathcal{A}$ breaks the (R, f)-correlation intractability of $\mathcal{H}_{\mathrm{shift}}$, which means that $\mathcal{A}$ wins the following challenger-based security game with non-negligible probability:

1. The challenger samples $\mathrm{msk} \leftarrow \mathrm{Gen}(1^\lambda)$.
2. The challenger samples $\mathrm{sk}_Z \leftarrow \mathrm{Shift}(\mathrm{msk}, Z)$ and sends sk to $\mathcal{A}$.
3. $\mathcal{A}(\mathrm{sk})$ outputs $x = (x_1, \ldots, x_t)$.

4. $\mathcal{A}$ wins if :
    (i) The inputs x_i are distinct and
    (ii) for $y_i = F_{\text{sk}}(x_i) - f(x_i)$, the relation $R_{\text{out}}(y)$ holds.

Then, $\mathcal{A}$ also wins each of the following modified security games with non-negligible probability:

- Hybrid Hyb_1: Same as the honest security game, except that in step (2), we sample $\text{sk}_f \leftarrow \text{Shift}(\text{msk}, f)$.

  This is indistinguishable from the original security game by the shift-hiding of SHSF.
- Hybrid Hyb_2: Same as Hyb_1, except that in step (4), we change the win condition (ii) so that $\mathcal{A}$ wins if $y_i = F_{\text{msk}}(x_i)$, the relation $R_{\text{out}}(y)$ holds.

  This is indistinguishable from Hyb_1 by the computational correctness of SHSF.

Finally, we show that $\mathcal{A}$'s success in Hyb_2 leads to an attack $\mathcal{A}'$ on the R_{out}-output intractability of $\mathcal{H}_{\text{plain}}$. The attack works as follows:

1. The challenger samples msk $\leftarrow \text{Gen}(1^\lambda)$ and sends msk to $\mathcal{A}'$.
2. $\mathcal{A}'(\text{msk})$ samples $\text{sk}_f \leftarrow \text{Shift}(\text{msk}, f)$.
3. $\mathcal{A}'$ then calls $\mathcal{A}(\text{sk}_f)$ and outputs $x = (x_1, \ldots, x_t)$.
4. By definition, $\mathcal{A}'$ wins if (i) the x_i are distinct, and (ii) for $y_i = F_{\text{msk}}(x_i)$, the relation $R_{\text{out}}(y)$ holds.

By construction, $\mathcal{A}'$ above wins with the same probability that $\mathcal{A}$ wins in Hyb_2, contradicting the R_{out}-output intractability of $\mathcal{H}_{\text{plain}}$.

The same argument as above applies to NAE-CI, with the condition (i) replaced by "the inputs x_i are not all equal." This completes the proof. □

This was only one concrete construction of correlation intractable hash functions but it isn't the only one. Describing each one of them would deviate from the goal of this paper which aims to provide a complete study of correlation intractable hash functions and the utility of the correlation intractability in security proofs. However, we can mention some other important constructions for those who are interested to take a look at different approaches.

6.3 Other CI constructions

- *CI of obfuscated pseudo random functions* described in [CCR16]. This paper presents the construction of correlation intractable function ensembles capable of withstanding all relations with a priori bounded polynomial complexity. The authors leverage sub-exponentially secure indistinguishability obfuscators, puncturable pseudo-random functions, and input-hiding obfuscators for evasive circuits. The existence of input-hiding obfuscation is critical, particularly for evasive circuit families, implying that it is infeasible for an adversary to find preimages of non-zero outputs given an obfuscated function. The study significantly advances the understanding of bounded correlation intractability, highlighting its theoretical implications and potential practical applications in areas like blockchain technology.

- *Fiat-Shamir and correlation intractability from strong KDM-secure encryption* [Can+18b]. This paper presents a novel construction of hash functions that are correlation intractable for all sparse relations using symmetric encryption schemes with key-dependent message (KDM) security. The authors propose a simple, yet effective, method to derive these hash functions from any symmetric encryption satisfying certain structural properties alongside robust KDM security. They demonstrate that such constructions can inherently support the Fiat-Shamir transform to convert any constant round, statistically sound interactive proof to a non-interactive argument. This work substantially extends the understanding of correlation intractability and provides a practical framework for achieving it through well-understood cryptographic primitives. Additionally, the implications for proof-of-work systems such as Bitcoin are discussed, emphasizing the relevance of the proposed methods in contemporary cryptographic applications.

7 Applications of correlation intractable hash functions

In this section we take a look at the potential applications of correlation intractable hash functions, be it single or multi input. Often when we mention correlation intractable hash functions, we refer to single-input CI; that is, we consider CI for relations with a single input x and a single corresponding output y. However, there is a natural generalization of CI to relations with multiple input-output pairs: a hash family $\mathcal{H}$ is defined to be CI for a relation $R(x_1, \dots, x_t, y_1, \dots, y_t)$ if it is computationally hard (given a hash function $h \leftarrow \mathcal{H}$) to find inputs $x_1, \dots, x_t$ such that $(x_1, \dots, x_t, h(x_1), \dots, h(x_t)) \in R$. In contrast to the single-input case, multi-input correlation intractability (for any $t \geq 2$) is a far less well-understood primitive. Perhaps the simplest nontrivial example of multi-input CI is for the relation R where $R(x_1, x_2, y_1, y_2) = 1$ if and only if $y_1 = y_2$ but $x_1 \neq x_2$. A CI hash family for R is precisely a collision-resistant hash family. However, most multi-input relations do not correspond to security notions that are simple to understand or previously studied. CI for more general multi-input relations also has interesting applications, including:

1. As a useful tool for the untrusted setup of public parameters: Multi-input CI hash functions allow n parties $P_1, \dots, P_n$ with inputs $x_1, \dots, x_n$ to compute public outputs $y_i = H(x_i)$ that can be used to generate public parameters for a multi-party protocol. Correlation intractability of $\mathcal{H}$ is necessary to ensure that a "bad CRS" is not accidentally (or maliciously) agreed upon.

2. As a hash function in proof-of-work protocols: In the context of blockchain technology, for example, in the Bitcoin protocol, a miner succeeds in adding a block to the blockchain when they find an x such that $y = H(x \| B_i)$ starts with a specified number of zeros (here, B_i is the i-th block and once found, y is placed in the next block B_{i+1}). A very desirable property in this setting is that a single miner (or a collection of colluding miners) cannot find multiple consecutive blocks with significantly less effort than finding them sequentially. This property can be formalized as a quantitatively precise variant of multi-input CI. For example, in the case of two consecutive blocks, simplifying the setting a little, we require a 2-input CI for the relation R where $R(x_1, x_2, y_1, y_2) = 1$ if y_1 and y_2 start with a pre-specified number ℓ of zeros, and y_1 is a suffix of x_2.

8 Conclusion

This report aims to provide a clear definition of correlation intractable (CI) hash functions and explains their importance in cryptographic protocols. We discussed the limitations of the Random Oracle Model (ROM). Through different examples, we showed that a scheme secure in the ROM might not stay secure when the random oracle is replaced with any function ensemble. This shows potential weaknesses in real-world implementations of schemes thought to be secure in the ROM. We provided examples to illustrate how a signature scheme, secure in the ROM, can become insecure with practical hash functions. By using sparse relations and universal function ensembles, we highlighted the difficulty in keeping security across all polynomial-time ensembles.

We also looked at the role of CI hash functions in securing the Fiat-Shamir transformation. By giving formal definitions and properties of CI hash functions, and extending these to multi-input correlation intractability, our results showed that CI hash functions are crucial for maintaining the security of the Fiat-Shamir transformation in practical situations.

8.1 Future Work and Implications

The study of correlation intractable hash functions opens several potentially interesting future research:

- **Enhanced CI Hash Function Constructions**: Developing more efficient and robust CI hash functions that can be easily implemented in practical systems remains a significant area of focus. This includes exploring new mathematical structures and cryptographic primitives that can enhance the intractability properties of these functions.
- **Broader Applications**: While this report focused on the Fiat-Shamir transformation, CI hash functions have potential applications in various other cryptographic protocols, including secure multiparty computations, blockchain technologies, and privacy-preserving systems. Exploring these applications can provide further insights into the versatility and importance of CI hash functions.
- **Real-World Implementations**: Testing and validating the theoretical constructs in real-world scenarios is crucial. This involves implementing CI hash functions in actual cryptographic systems and evaluating their performance and security under practical attack models.

This report underscores the critical role of correlation intractable hash functions in modern cryptography, particularly in ensuring the security of the Fiat-Shamir transformation. By bridging the gap between theoretical security models and practical implementations, CI hash functions provide a robust framework for developing secure cryptographic protocols. As the field of cryptography continues to evolve, the principles and findings discussed in this report will serve as a foundational reference for future research and development.

References

[FS86] Amos Fiat and Adi Shamir. "How to prove yourself: Practical solutions to identification and signature problems". In: *Conference on the theory and application of cryptographic techniques*. Springer. 1986, pp. 186–194.

[BR93] 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.

[PS96] David Pointcheval and Jacques Stern. "Security proofs for signature schemes". In: *International conference on the theory and applications of cryptographic techniques*. Springer. 1996, pp. 387–398.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. "The random oracle methodology, revisited". In: *Journal of the ACM (JACM)* 51.4 (2004), pp. 557–594.

[Dam10] Ivan Damgard. "On Sigma Protocols". In: *CPT* (2010).

[CCR16] Ran Canetti, Yilei Chen, and Leonid Reyzin. "On the correlation intractability of obfuscated pseudorandom functions". In: *Theory of Cryptography: 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I 13*. Springer. 2016, pp. 389–415.

[Can+18a] Ran Canetti et al. "Fiat-Shamir and correlation intractability from strong KDM-secure encryption". In: *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part I 37*. Springer. 2018, pp. 91–122.

[Can+18b] Ran Canetti et al. "Fiat-Shamir From Simpler Assumptions". In: *IACR Cryptol. ePrint Arch.* 2018 (2018), p. 1004. URL: `https://api.semanticscholar.org/CorpusID:53239322`.

[Can+19] Ran Canetti et al. "Fiat-Shamir: from practice to theory". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. 2019, pp. 1082–1090.

[GMR19] Shafi Goldwasser, Silvio Micali, and Chales Rackoff. "The knowledge complexity of interactive proof-systems". In: *Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali*. 2019, pp. 203–225.

[LV20] Alex Lombardi and Vinod Vaikuntanathan. "Correlation-intractable hash functions via shift-hiding". In: *Cryptology ePrint Archive* (2020).

[CX23] Michele Ciampi and Yu Xia. "Multi-Theorem Fiat-Shamir Transform from Correlation-Intractable Hash Functions". In: *International Conference on Applied Cryptography and Network Security*. Springer. 2023, pp. 555–581.

[Vau24] Serge Vaudenay. "Advanced Cryptography Lecture Notes". In: 2024, pp. 37–44.