Towards a White-Box Secure Fiat-Shamir Transformation

Gal Arnon
galarnon42@gmail.com
Weizmann Institute
and Bar-Ilan University

Eylon Yogev
eylon.yogev@biu.ac.il
Bar-Ilan University

June 30, 2025

Abstract

The Fiat–Shamir transformation is a fundamental cryptographic technique widely used to convert public-coin interactive protocols into non-interactive ones. This transformation is crucial in both theoretical and practical applications, particularly in the construction of succinct non-interactive arguments (SNARKs). While its security is well-established in the random oracle model, practical implementations replace the random oracle with a concrete hash function, where security is merely assumed to carry over.

A growing body of work has given theoretical examples of protocols that remain secure under the Fiat–Shamir transformation in the random oracle model but become insecure when instantiated with any white-box implementation of the hash function. Recent research has shown how these attacks can be applied to natural cryptographic schemes, including real-world systems. These attacks rely on a general diagonalization technique, where the protocol exploits its access to the white-box implementation of the hash function. These attacks cast serious doubt on the security of cryptographic systems deployed in practice today, leaving their soundness uncertain.

We propose a new Fiat–Shamir transformation (XFS) that aims to defend against a broad family of attacks. Our approach is designed to be practical, with minimal impact on the efficiency of the prover and verifier and on the proof length. At a high level, our transformation combines the standard Fiat–Shamir technique with a new type of proof-of-work that we construct.

We provide strong evidence for the security of our transformation by proving its security in a relativized random oracle model. Specifically, we show diagonalization attacks on the standard Fiat–Shamir transformation that can be mapped to analogous attacks within this model, meaning they do not rely on a concrete instantiation of the random oracle. In contrast, we prove unconditionally that our XFS variant of the Fiat–Shamir transformation remains secure within this model. Consequently, any successful attack on XFS must deviate from known techniques and exploit aspects not captured by our model.

We hope that our transformation will help preserve the security of systems relying on the Fiat–Shamir transformation.

Keywords: Fiat-Shamir transformation, proof-of-work, random oracle model, diagonalization

Contents

1 Introduction

The Fiat–Shamir (FS) transformation [FS86] is a powerful technique in cryptography. Initially introduced as a method for deriving digital signatures from identification schemes, today, the FS transformation is widely used to convert public-coin interactive protocols into non-interactive ones. This transformation plays a crucial role in both theory and practice, particularly in the construction of succinct non-interactive arguments (SNARKs).

The basic idea of the Fiat–Shamir transformation is to replace the verifier's randomness with a cryptographic hash function computed over everything the verifier has seen so far in the interaction. This allows the prover to generate the random coins by itself and thus removes the need to interact with the verifier (see [CY24] for details on this transformation). A rigorous soundness analysis of this transformation in the standard model remains an open challenge to this day.

Pointcheval and Stern [PS96] showed that the Fiat–Shamir transformation is sound in the random oracle model, where the cryptographic hash function is treated as a truly random function, yielding unconditional security. However, deploying the scheme in practice requires instantiating the random oracle with a concrete hash function (e.g., BLAKE2, SHA-256). There has been extensive research on proving the security of the Fiat–Shamir transformation for specific protocols using hash functions based on standard assumptions (e.g., [CCR16; KRR17; CCRR18; HL18; CCHLRRW19; PS19; BKM20; JJ21; HLR21; CJJ21; HJKS22; KLV23]). While these approaches offer theoretical security guarantees, their practical applicability is limited, and they often lack concrete efficiency. Hence, it is typically *assumed* that the soundness proven in the random oracle model carries over to the scheme when instantiated with a real-world concrete hash function. Nearly all SNARK-based systems today depend on this assumption for their security.

Attacks on instantiations of Fiat–Shamir. Several works highlight the risks of replacing the random oracle with a concrete hash function, regardless of how the hash function is implemented. Canetti, Goldreich, and Halevi [CGH04] showed the existence of cryptographic schemes that are secure in the random oracle model but are insecure in the standard model. The key issue is that an adversary with full (white-box) access to the hash function—rather than just oracle access—can exploit this additional knowledge to compromise security. Similar attacks have been shown for interactive protocols that, while secure in their original form, become insecure after applying the FS transformation, regardless of the specific hash function used [Bar01; GK03].

This line of work has been further extended to demonstrate the risks of the Fiat–Shamir transformation in more natural cryptographic schemes. Bartusek et al. [BBHMR19] showed that the standard approach for constructing hash-based proofs (in the style of [Kil92; Mic00; BCS16; CY21a]) becomes insecure when the FS transformation is applied. More recently, Khovratovich, Rothblum, and Soukhanov [KRS25] presented an attack that breaks the Fiat–Shamir security of a widely used, practical proof system. Specifically, their attack targets schemes based on the popular GKR protocol [GKR15]. This result is particularly alarming, as it threatens the security of various recent proof systems, including some deployed in practice to secure blockchain networks [ZGKPP17; WTSTW18; XZZPS19; ZLWZSXZ21; Pol24].

The attacks described above are sometimes referred to as *diagonalization* attacks. These attacks exploit the fact that the proof systems in question are designed to handle general-purpose computations. Specifically, they construct a proof instance where the circuit being proved evaluates the Fiat–Shamir hash function itself. As suggested in [KRS25], a natural countermeasure is to restrict the expressiveness of the circuits that the protocol supports, ensuring they cannot compute the Fiat–Shamir hash function (or, equivalently, using hard-to-compute hash functions for the FS transformation). While this approach is theoretically appealing, it is highly impractical. In real-world applications, hash functions are designed to be simple and efficient, whereas proof systems are built to verify large and complex circuits. Moreover, regardless of practical considerations, it remains unclear whether restricting the expressiveness of circuits provides any meaningful security benefits.

Defending against white-box attacks. We propose a new Fiat–Shamir transformation (denoted as XFS for *extended* FS) that aims to defend against a broad family of attacks, including most of the white-box attacks mentioned above. In particular, we demonstrate how it circumvents the recent practical attack on GKR [KRS25], and adapting this attack to break our construction appears nontrivial. Our approach is designed to be practical, with minimal impact on the efficiency of the prover and verifier and on the proof length. At a high level, our transformation combines the standard Fiat–Shamir technique with a new type of proof-of-work that has a strong security notion. We further present a practical construction of this proof-of-work.

While the overarching goal is to formally *prove* the security of our transformation in the standard model, doing so remains challenging without making assumptions about both the hash function and the interactive proof being compiled. Towards this end, we introduce an idealized model that abstracts the core principles of diagonalization attacks without assuming a white-box implementation. Within this model, we demonstrate that the standard Fiat–Shamir transformation is insecure, whereas our proposed XFS transformation achieves provable security. This result suggests that any successful attack would need to diverge significantly from existing techniques. However, it remains an open question whether our framework truly captures all attack strategies in the standard model.

In the following sections, we describe our transformation in detail and outline our security analysis.

1.1 A new Fiat–Shamir transformation

Our XFS transformation builds upon the original Fiat–Shamir paradigm by integrating a new variant of a "proof-of-work", to which we also give a concretely efficient instantiation. The key idea is that the proof-of-work increases the complexity of the Fiat–Shamir hash function, making it resistant to diagonalization attacks. However, since the verifier is provided with the solution upfront, the verification complexity remains unchanged. In this sense, the non-deterministic complexity of the hash remains low, while the deterministic complexity of the hash is high.

In more detail, given a protocol transcript tr up to round i, we first hash tr to get a puzzle z for the proof-of-work (using the Fiat–Shamir hash, or possibly a separate dedicated hash). Then, the prover solves the puzzle to get a solution s. Only after obtaining s do we apply the Fiat–Shamir hash function to both the transcript tr and the solution s to derive verifier randomness.

Informally, a proof-of-work ([DN92]) is a pair of algorithms $(\mathsf{Solve}, \mathsf{Check})$ such that one can sample puzzles of certain hardness ℓ, and then Solve provides a solution in time ℓ (a solution can be efficiently checked by Check). Our security notion requires that any algorithm that runs in time significantly less than ℓ can solve the puzzle only with negligible probability (even if it has a long preprocessing phase before the puzzle is sampled). Note that this is different from the proof-of-work that is used in Bitcoin.[1] See Section 6 for further details and a construction of our proof-of-work

[1] In the proof-of-work used in Bitcoin, the solution is some s that hashes to an output that begins with $\log \ell$ many 0's. Such a scheme can be solved in constant time with probability $1/\ell$ simply by guessing.

variant.

Our XFS transformation is formally defined in Section 3, and we give an informal description below.

Construction 1.1 (XFS)**.** Let $(\mathbf{P}, \mathbf{V})$ be a 3-message interactive argument (i.e., a Sigma protocol) for a language L, and let $(\mathsf{Solve}, \mathsf{Check})$ be a proof-of-work. Given a hash function h, we derive our non-interactive argument as follows:

- Prover: Given $\mathbb{x}$ and $\mathbb{w}$,
  1. Compute the prover's first message $m_1 = \mathbf{P}(\mathbb{x}, \mathbb{w})$.
  2. Compute the puzzle for the proof-of-work $z = h(\mathbb{x}, m_1)$.
  3. Solve the proof-of-work $s = \mathsf{Solve}(z)$.
  4. Set the pad $\tau = 0^{|\langle \mathbf{V} \rangle|}$, where $|\langle \mathbf{V} \rangle|$ is the circuit size of $\mathbf{V}$ (see remark about this below).
  5. Derive randomness for the verifier $\rho = h(\tau, \mathbb{x}, m_1, s)$.
  6. Compute the response of the prover $m_2 = \mathbf{P}(\mathbb{x}, \mathbb{w}, \rho)$.
  7. Output (m_1, s, m_2).
- Verifier: Given x and (m_1, s, m_2),
  1. Compute the puzzle for the proof-of-work $z = h(\mathbb{x}, m_1)$.
  2. Set the pad $\tau = 0^{|\langle \mathbf{V} \rangle|}$.
  3. Derive randomness for the verifier $\rho = h(\tau, \mathbb{x}, m_1, s)$.
  4. Accept if and only if the puzzle is correct $\mathsf{Check}(z, s) = 1$ and the underlying verifier accepts $\mathbf{V}(\mathbb{x}, m_1, \rho, m_2) = 1$.

We would like to highlight a few key points about our construction.

- (Preprocessing): In our formal construction, we allow the prover to first choose a circuit C that represents the language (i.e., $\mathbb{x}$ is in the language if and only if there exists a witness $\mathbb{w}$ such that $C(\mathbb{x}, \mathbb{w}) = 1$). In such a case, the verifier will get a short digest of the circuit computed by a trusted indexer. This generalization is used by some of the attacks (e.g., [KRS25]).

- (Proof-of-work hardness): In order for the above protocol to be secure, the hardness parameter ℓ for the proof -of-work must be larger than the circuit complexity of the circuit C representing the language (as explained in the preprocessing case). See Section 6 for a simple construction of our proof-of-work.

- (Multi-round): The construction above is defined for 3-message protocols. It is desirable to have a generalization of XFS to multi-round protocols, similar to the generalization of the standard Fiat–Shamir transformation to multi-round protocols (observe that for standard FS, the stronger notion of round-by-round soundness is required). However, for simplicity, we focus on the 3-message case, which is challenging on its own, and leave the multi-round case for future work.

- (Black-box with respect to circuits): We emphasize that our construction treats all circuits as *black-boxes*, including the prover and verifier circuits, as well as a circuit C potentially provided during the preprocessing phase. In particular, the construction does not rely on any ability to identify when a circuit is computing the hash function h, nor does it modify the given circuits in any way. Consequently, our transformation remains fully functional even if an adversary submits an obfuscated version of its circuit.

- (Padding): In Item 4 of the prover code (and similarly in the verifier code), the computation of the padding τ plays a crucial role. The padding is used in our main proof, as it disallows the verifier $\mathbf{V}$ to make queries to the Fiat–Shamir hash function, as this is longer than the code of the verifier. (In fact, if the verifier is an AND of many $\mathcal{V}_i$'s, then it suffices for the pad to be longer than each $\mathcal{V}_i$.)

  Typically, the verifier is a small program that is under our complete control, unlike the circuit C. Thus, it is reasonable to assume that in natural protocols, the verifier will not compute the Fiat–Shamir hash (or will not compute it on specific restricted prefixes). In this case, we can remove the padding or pad the Fiat–Shamir hash only with the restricted prefix. See Section 5.1, specifically Remark 5.4, for further discussion.

Applicability. Our transformation serves as a replacement for the FS transformation and is applicable in the same settings (namely, public-coin protocols). Other transformations (e.g., [Pas03; Fis05; RT24]) for removing interaction have more limited applicability. For example, Fischlin's transformation [Fis05] is secure only for protocols that satisfy special soundness, even when analyzed in the random oracle model. In contrast, diagonalization attacks apply to protocols over a rich and expressive circuit family, which typically do not satisfy special soundness. We remark that white-box attacks on Fischlin's protocol remain largely unexplored, making it an interesting open question to identify suitable attack strategies against this transformation.

1.2 On the white-box security of our transformation

We cannot hope to prove security of our transformation in the standard model without making assumptions about both the hash function and the interactive proof being compiled. Instead, our goal is to give a formal proof of security in a suitable model. Towards that end, we need to first understand why and where exactly security proofs in the random oracle model (ROM) break down.

In the ROM, the Fiat–Shamir transformation is shown secure using the following logic [CY24; PS96; Mic00; CY21a; CY21b]. First, you fix some interactive protocol (either a proof system or an argument system). Then, you introduce the random oracle and apply the Fiat–Shamir transformation. That is, the interactive protocol has no knowledge about the random oracle. Even if the interactive protocol was defined in the random oracle model, the FS transformation will introduce a *new* oracle independent of the one used for the protocol. In this case, the prover, the verifier of the protocol, and the circuit C (chosen by the malicious prover in a preprocessing phase) *do not have access to the random oracle for the Fiat–Shamir transformation.*

This completely breaks down when instantiating the random oracle with any white-box implementation. In particular, now the random oracle is replaced with a small circuit that can be computed by the verifier or the circuit C. Indeed, this is exactly what the white-box attacks exploit.

A relativized world. To address these attacks, we consider compiling protocols in a *relativized world* [CT10; CCS22; CCGOS23; CGSY24; BCG24]. In this model, all parties have oracle access to the random oracle used in the Fiat–Shamir transformation. Specifically, we assume a *single* random oracle f, which is accessible to the (potentially malicious) prover, the verifier, and the circuit C representing the underlying language. The circuit C includes f-gates, granting direct access to the random oracle f. Notably, this setup mirrors the structure exploited in the recent attack on the GKR scheme [KRS25], which we discuss further in Section 2.

If one applies the *standard* Fiat–Shamir transformation, diagonalization attacks can be carried out *within the relativized model* described above—meaning they do not require an explicit instantiation of the random oracle. In contrast, we prove unconditionally that our XFS variant of the Fiat–Shamir transformation remains secure within this model when applied to a Sigma protocol with an appropriate knowledge definition.

We consider preprocessing 3-message protocols (Sigma protocols) for circuit family $\mathcal{C}$ with oracle gates to f, that have round-by-round knowledge soundness (our proof requires a slightly stronger version of round-by-round knowledge, see Definition 4.1 for the precise definition). In this settings, we prove the following theorem.

Theorem 1.2 (Informal). *Consider the following ingredients in the relativized ROM:*

- *Let $\mathcal{C}$ be a family of circuits, where $\mathbf{q}_C$ is a bound on the number of oracle gates.*
- *Let $(\mathbf{I}_{\mathrm{SP}}, \mathbf{P}_{\mathrm{SP}}, \mathbf{V}_{\mathrm{SP}})$ be a preprocessing Sigma protocol for a family of circuits $\mathcal{C}$ in the random oracle model, with (straightline) round-by-round knowledge soundness error κ_{SP}, and indexer complexity $\mathbf{q}_{\mathrm{I}}$.*
- *Let $(\mathsf{Solve}, \mathsf{Check})$ be a strong proof-of-work with soundness error $\varepsilon_{\mathrm{POW}}$ where Check performs at most $\mathbf{q}_{\mathrm{POW}}$ queries.*

Then, for every security parameter $\lambda \in \mathbb{N}$, and $\ell \in \mathbb{N}$, the XFS transformation yields a SNARK for $\mathcal{C}$ with adaptive knowledge error κ_{ARG} such that

$$\kappa_{\mathrm{ARG}}(\lambda, t) = O\left(t \cdot \kappa_{\mathrm{SP}}(\lambda, t_1) + t \cdot \varepsilon_{\mathrm{POW}}(\lambda, \ell, t, t_2)\right) .$$

where $t_1 := t + \mathbf{q}_{\mathrm{I}}$, and $t_2 := \mathbf{q}_{\mathrm{POW}} \cdot (\mathbf{q}_C + \mathbf{q}_{\mathrm{I}})$.

A precise formulation of the relativized model appears in Section 4, and the formal statement and proof of the theorem can be found in Section 5. We leave a formal analysis of the multi-round case for future work.

Discussion. There are inherent limitations to constructing relativized SNARKs in the random oracle model, as discussed in [BCG24]. Our construction does not attempt to bypass these limitations. Instead, we *assume* the existence of a relativized interactive argument scheme and prove the security of the XFS transformation under this assumption. Our security proof demonstrates that any attack that can be mapped to the relativized setting will fail against XFS. Consequently, any successful attack on XFS must deviate from known techniques and exploit aspects not captured by our model.

Furthermore, while our transformation is provably secure in the relativized model, this does not rule out the possibility of attacks in the standard model when the adversary has a white-box implementation of the Fiat–Shamir hash. In such cases, an attacker could exploit structural weaknesses of the hash function or the underlying proof system in ways that our model does not capture. Therefore, while our transformation provides a strong layer of defense against known diagonalization techniques, it should be applied with the necessary caution.

2 Our Techniques

We give a high-level overview of the Fiat–Shamir transform, the known attacks, how our transformation mitigates these attacks, and how we formally model and prove security of our transformation.

- In Section 2.1, we give an illustrative example of known attacks.
- In Section 2.2, we describe how our transformation mitigates these attacks.
- In Section 2.3, we explain the security of our transformation.
- In Section 2.4, we describe how the recent on the GKR protocol fit into our model of security.

2.1 An illustrative example of known attacks

In this section, we describe a simple protocol, mainly inspired by Barak's protocol [Bar01], that essentially captures all known attacks on the Fiat–Shamir transformation.

A toy protocol. We describe a toy Sigma protocol which illustrates standard attacks on Fiat–Shamir. In this protocol, a prover provides a circuit $C\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^{r+1}$ which is preprocessed into a digest ψ and an input/output pair $(\mathbb{x}, \mathbb{y})$, and claims that he knows a witness $\mathbb{w}$ so that $C(\mathbb{x}, \mathbb{w}) = \mathbb{y}$. This will be a trivial protocol that ends in the prover sending C and $\mathbb{w}$ to the verifier. In real protocols for which Fiat–Shamir is unsound, this step is replaced by more sophisticated protocols for showing that the committed circuit and witness satisfy $C(\mathbb{x}, \mathbb{w}) = \mathbb{y}$. In more detail, the protocol works as follows:

1. (Preprocessing): The preprocessing algorithm receives a circuit C and outputs a short digest ψ of this circuit. The honest prover is given C, and the verifier is given ψ. (This digest may contain a hash of C, but can also contain other information about the circuit, such as its depth, etc.)

2. (Prover): The honest prover sends a succinct commitment of the witness $m_1 = \mathsf{Com}(\mathbb{w})$ to the verifier.

3. (Verifier): The verifier replies with a random challenge $\rho \leftarrow \{0,1\}^r$.

4. (Prover): The honest prover sends C and $\mathbb{w}$ to the verifier.

5. (Verifier's decision): The verifier accepts if ψ is the digest of C, it holds that $m_1 = \mathsf{Com}(\mathbb{w})$, and either $C(\mathbb{x}, \mathbb{w}) = \mathbb{y}$, or $C(\mathbb{x}, \mathbb{w}) = (0\|\rho)$.

If there exists a witness $\mathbb{w}$ so that $C(\mathbb{x}, \mathbb{w}) = \mathbb{y}$, then the honest prover will send this, and it will be accepted by the verifier. Furthermore, the protocol has small soundness error: a malicious prover that chooses C, $\mathbb{x}$, $\mathbb{y}$, and $\mathbb{w}$ cannot predict the verifier's challenge ρ. However, as we will see, this protocol is not secure after applying the Fiat–Shamir heuristic.

Attacking Fiat–Shamir. Applying the Fiat–Shamir transformation using a hash function h to the above sigma protocol yields the following non-interactive protocol:

1. (Preprocessing): The preprocessing algorithm receives a circuit C and outputs a digest of this circuit ψ. The honest prover is given C, and the verifier is given ψ.

2. (Prover): The honest prover generates a succinct commitment of the witness $m_1 = \mathsf{Com}(\mathbb{w})$ and sends $(m_1, C, \mathbb{w})$ to the verifier.

3. (Verifier): The verifier computes $\rho = h(\psi, \mathbb{x}, \mathbb{y}, m_1)$, and accepts if and only if ψ is the digest of C, it holds that $m_1 = \mathsf{Com}(\mathbb{w})$, and either $C(\mathbb{x}, \mathbb{w}) = \mathbb{y}$, or $C(\mathbb{x}, \mathbb{w}) = (0 \| \rho)$.

We now describe an attack where the prover chooses a circuit $\tilde{C}$ and input/output pair $(\mathbb{x}, \mathbb{y})$ so that $\tilde{C}(\mathbb{x}, \mathbb{w}) \neq \mathbb{y}$ for every $\mathbb{w}$, and yet the prover causes the verifier to accept with probability 1:

- The input will be $\mathbb{x} = 0^n$ and the output will be $\mathbb{y} = 1^{r+1}$.
- The circuit $\tilde{C}(\mathbb{x}, \mathbb{w})$ is as follows: On input $\mathbb{x}$ and $\mathbb{w}$, parse $(\psi, \mathbb{y}) = \mathbb{w}$, and compute $m_1 = \mathsf{Com}(\psi, \mathbb{y})$ and $\rho = h(\psi, \mathbb{x}, \mathbb{y}, m_1)$. Output $(0 \| \rho)$.
- The malicious prover computes $m_1 = \mathsf{Com}(\psi, \mathbb{y})$, and sends $(m_1, \tilde{C}, \mathbb{w} = (\psi, \mathbb{y}))$.

Since the output of $\tilde{C}$ always begins with 0, there will be no $\mathbb{w}$ so that $\tilde{C}(\mathbb{x}, \mathbb{w}) = \mathbb{y} = 1^{r+1}$, and so $(\tilde{C}, \mathbb{x}, \mathbb{y})$ aught to be rejected. However, observe that $\tilde{C}(\mathbb{x}, \mathbb{w}) = \tilde{C}(\mathbb{x}, (\psi, \mathbb{y})) = (0 \| \rho)$ where $\rho = h(\psi, \mathbb{x}, \mathbb{y}, m_1)$ is the challenge that the verifier will compute. Therefore, the verifier will always accept.

2.2 How our transformation mitigates these attacks

The main issue described in the previous section is that the circuit $\tilde{C}^f$ can compute the Fiat–Shamir *next verifier message* function, as this function is simply an evaluation of the hash function. A naive mitigation attempt, also suggested in [KRS25], is to make the Fiat–Shamir hash function so computationally hard that the circuit cannot evaluate it. However, this approach has two fundamental drawbacks:

- Practicality: The verifier evaluates the hash function, implying that its complexity exceeds that of the hash function, which in turn exceeds the complexity of the circuit. In practice, we aim for a succinct verifier that is significantly more efficient than the circuit itself. This becomes particularly challenging when considering recursive compositions of proofs.
- Soundness: Regardless of practical considerations, certain subtleties arise when attempting to argue soundness. Recall that the hash function in the Fiat–Shamir transformation receives inputs (e.g., $\psi, \mathbb{x}, \mathbb{y}, m_1$) that are influenced by the prover. Thus, ensuring the soundness of the transformation requires the hash function to be hard over *all possible distributions* of these inputs for any prover. This imposes a significantly stronger requirement on the hash function—one that is not met merely by guaranteeing that the hash function has high circuit complexity.

Our transformation addresses both challenges using a suitable variant of a proof-of-work scheme.

Proofs of work to the rescue. Our new transformation requires the prover to submit a solution to a proof-of-work in the Fiat–Shamir "next verifier message" function. A *proof-of-work* is a scheme with two algorithms, Solve and Check, where Solve receives a random puzzle z and outputs a solution s and Check checks that s is a valid solution for z.[2] Moreover, for a parameter ℓ, the puzzle should take $\ell > |\tilde{C}|$ time to solve (so that computing the solution should be hard for the circuit $\tilde{C}$) but can be verified in time $o(\ell)$ (so that the verifier can check it efficiently).

The security notion we use is as follows: for a security parameter λ, and any pair of circuits $(\mathbf{A}_1, \mathbf{A}_2)$ where $\mathbf{A}_1$ has size $\mathrm{poly}(\lambda, \ell)$ and $\mathbf{A}_2$ has size $o(\ell)$, the following experiment outputs 1 with probability at most $\mathsf{negl}(\lambda)$,

[2]Some proof-of-work schemes additionally require a setup function, which is omitted in this high-level explanation.

1. $\mathbf{A}_1$ outputs a state aux.
2. A random puzzle z is chosen.
3. $\mathbf{A}_2(\mathsf{aux}, z)$ outputs s.
4. Output 1 if and only if $\mathsf{Check}(\ell, z, s) = 1$.

See Section 3 for a formal definition of strong proof-of-work.

To see how adding a proof-of-work protects our transformation against attacks, we turn back to the toy protocol.

Our transformation applied to the toy protocol. We apply the XFS transformation, described in Section 1.1, to the toy protocol. We assume that, after C is chosen, both the prover and the verifier know $|C|$ so that we can choose a hardness parameter $\ell > |C|$ for the proof-of-work. Note that this is without loss of generality, as the preprocessing algorithm can add this to the digest ψ. For this intuitive overview, we omit the padding string τ.

1. (Preprocessing): The preprocessing algorithm receives a circuit C and outputs a digest ψ of this circuit. The honest prover is given C, and the verifier is given ψ.

2. The honest prover:
   (a) Generates $m_1 = \mathsf{Com}(\mathbb{w})$,
   (b) Computes $z = h(\psi, \mathbb{x}, \mathbb{y}, m_1)$
   (c) Solves $s = \mathsf{Solve}(\ell, z)$,
   (d) Outputs $(m_1, s, C, \mathbb{w})$.

3. The verifier:
   (a) Computes $z = h(\psi, \mathbb{x}, \mathbb{y}, m_1)$,
   (b) Computes $\rho = h(\psi, \mathbb{x}, \mathbb{y}, m_1, s)$,
   (c) Accepts if all of the following hold:
      i. ψ is the digest of C,
      ii. $m_1 = \mathsf{Com}(\mathbb{w})$,
      iii. $\mathsf{Check}(\ell, z, s) = 1$,
      iv. Either $C(\mathbb{x}, \mathbb{w}) = \mathbb{y}$, or $C(\mathbb{x}, \mathbb{w}) = (0||\rho)$.

In order to mount an attack like the one previously discussed, the circuit $\tilde{C}$ needs to internally compute $h(\psi, \mathbb{x}, \mathbb{y}, m_1, s)$, which requires knowing a solution s. While $\tilde{C}$ can compute the puzzle $z = h(\psi, \mathbb{x}, \mathbb{y}, m_1)$, it cannot solve it by itself, as the hardness parameter of Solve is set to be greater than the size of $\tilde{C}$.

The only option, then, is to provide the circuit with inputs $(\mathbb{x}, \mathbb{w})$ so that allow it to compute s. However, as $\tilde{C}$ has size less than ℓ, this is a challenging task for the prover. Observe that the prover must first choose $(\tilde{C}, \mathbb{x}, \mathbb{y}, \mathbb{w})$, and only then it learns the puzzle $z = h(\psi, \mathbb{x}, \mathbb{y}, \mathsf{Com}(\mathbb{w}))$ where ψ is derived from $\tilde{C}$. If the hash function is sufficiently unpredictable, then this generalizes to the following game: the prover searches for a value aux before knowing z, so that solving z takes time less than ℓ given aux. This is precisely the security notion that we require for the proof-of-work.

How to formally argue security? Up to this point, we have only an intuitive explanation of why our transformation protects against attacks such as the one described in the toy protocol. In order to meaningfully argue soundness of this heuristic, we turn back to the random oracle model (ROM). We discuss our modeling of such protocols in the next section.

2.3 On the security of our transformation

We prove that our version of the Fiat–Shamir transformation is secure within the model of relativized proof systems. This result implies that any successful white-box attack must rely on techniques that fall outside the scope of this model and thus significantly deviate from existing attack approaches.

We focus our attention on three-message protocols (Sigma protocols). In the random oracle model, the Fiat–Shamir transformation is typically defined as follows: Given a Sigma protocol (which may have either information-theoretic or computational soundness) for some language L, or more generally for a relation defined by a circuit $C(\mathbb{x}, \mathbb{w})$, where $\mathbb{x}$ represents the instance and $\mathbb{w}$ the witness, we introduce a random oracle f to compile the Sigma protocol into a non-interactive one.

This transformation is secure in the random oracle model. The key reason is that the initial Sigma protocol does *not have access* to the random oracle f. The diagonalization attack described in the toy protocol exploits precisely this gap, allowing the circuit C to access f and predict the verifier's next message. This explains why the Fiat–Shamir transformation remains secure in the random oracle model but loses its security in the standard model.

Modeling attacks in the ROM. Our goal is to capture these attacks in a relativized model. Thus, we begin with a relativized Sigma protocol in the random oracle model, with oracle f. That is, *all* parties have access to f. In particular, we allow the circuit C to include f-gates. In an f-gate, the input wires specify an input x to the gate, and the output wires specify y such that $f(x) = y$. Then, the Fiat–Shamir (FS) transformation is applied using the *same oracle* f as the FS hash function.

If one applies the standard Fiat–Shamir (FS) transformation, then the diagonalization attacks can be implemented *within the model* without requiring an explicit instantiation of the random oracle. Indeed, observe that the prover and the circuit $\tilde{C}$ attacking the toy problem rely solely on the ability to evaluate the hash function h. Using our XFS variant of the FS transformation, we (unconditionally) establish the security of the transformation in this model.

Sketching our proof. We show that adaptive (straightline) knowledge soundness of our scheme reduces to adaptive (straightline) round-by-round knowledge soundness of the original Sigma protocol, assuming the Sigma protocol meets natural security requirements.[3] Recall that straightline round-by-round knowledge soundness says that if the prover can convince the verifier with probability greater than κ, then an extractor can extract a witness from the prover's first message along with the trace of its queries to the random oracle.

As a first step, our proof follows a similar structure to existing proofs for the Fiat–Shamir transformation in the ROM (i.e., [CY24, Chapter 10]). We reduce a prover $\mathcal{P}$ attacking the argument after applying our Fiat–Shamir transformation to a prover $\mathbf{P}$ attacking the Sigma protocol with roughly the same probability. In order to choose $(C^f, \mathbb{x}, \mathbb{y}, m_1, s, m_2)$ as its message, the prover $\mathcal{P}$ must query $x = (\mathbb{x}, \mathbb{y}, m_1, s)$ to receive the challenge ρ, as otherwise it will not be able to convince the SP verifier on a random challenge that it does not know. We guess which one of its queries this is and plant the random verifier challenge $\rho \leftarrow \{0, 1\}^\lambda$ in the oracle response (recall that $\mathcal{P}$ expects this challenge to come from f). Let $f[(x, \rho)]$ be the "programmed oracle" which answers as f at all points except x, where it answers with ρ. Our goal is to show that if $\mathcal{P}$ is run with this oracle rather

[3] In more detail, in addition to adaptive straightline round-by-round knowledge soundness, we require that the circuit digest ψ to act as a commitment to C, and a natural notion saying that the extractors for the circuit and witness do not change their output when more queries are appended to the prover random-oracle trace. See Section 4 for precise definitions.

than the real function f, the verifier's behavior does not change. Once this is shown, it concludes our proof.

In the standard Fiat–Shamir proof, this step is trivial since neither the verifier nor the circuit has access to the oracle f. However, in our case, both may make f-queries. We handle separately the queries made by the verifier and those made by the circuit.

First, note that the verifier is an "honest" algorithm selected by the designer of the Sigma protocol. We ensure that it does not query the programmed point by incorporating a suitable padding mechanism. By adding sufficient padding to the query, i.e., $f[(0^{|\langle \mathbf{V}_{\mathrm{SP}} \rangle|} \| x, \rho)]$, we can ensure that the verifier cannot directly make such queries (as they are larger than its description). We formally define these pads as *prefix-avoiding* and prove their existence in the ROM. Moreover, we present more practical and efficient alternatives to the simple padding scheme described above (i.e., the long zero string). However, for the sake of clarity in this exposition, we continue to use this basic padding scheme. A more detailed discussion on alternative padding methods can be found in Section 5.1.

We are left with showing that C can not make queries to the oracle. In contrast to the verifier, the circuit is chosen by the malicious prover and can consist of any arbitrary, adversarially crafted code. Moreover, the circuit is run on the witness $\mathtt{w}$, which can be generated in time that is much longer than the hardness of the puzzle. To address this, we leverage the proof-of-work mechanism. Observe that the puzzle z is generated by evaluating $f(\psi, \mathtt{x}, \mathtt{y}, m_1)$. The digest ψ commits the prover to the circuit C and, by round-by-round knowledge of the Sigma protocol, m_1 is "committing" to $\mathtt{w}$ in the sense that it can be used to extract $\mathtt{w}$. Thus, the prover is essentially committed to $\mathtt{x}$, $\mathtt{y}$, C and $\mathtt{w}$ *before* knowing the puzzle. Since our claim is that $C^f(\mathtt{x}, \mathtt{w}) = \mathtt{y}$ (i.e., the circuit runs only on inputs $\mathtt{x}$ and $\mathtt{w}$), the prover cannot pass to the circuit any information about the solution s to the puzzle z (as this would violate security of the proof-of-work). In other words, C^f cannot make the query $x = (\mathtt{x}, \mathtt{y}, m_1, s)$ as it contains s.

The formal proof is very subtle and takes many careful steps. See Section 5 for the full proof details.

2.4 Protecting the GKR protocol

Khovratovich, Rothblum, and Soukhanov [KRS25] show attacks on the application of Fiat–Shamir to a natural protocol in common use, namely the GKR protocol [GKR15] (adapted to nondeterministic computations with large witnesses). These attacks are especially dangerous, as, unlike prior attacks on Fiat–Shamir, they work directly on a natural and widely used protocol. We briefly discuss the GKR protocol, the attack, and evidence towards it being mitigated by our transformation.

Overview of GKR. In the GKR protocol, the prover wants to prove that $C(\mathtt{x}, \mathtt{w}) = \mathtt{y}$. The verifier is provided with $\mathtt{x}$, $\mathtt{y}$, and a digest ψ of C which includes a succinct representation of C. The prover begins by sending a commitment of $\mathtt{w}$. In fact, this will be a special commitment known as a multilinear PCS, which allows the verifier (with the aid of the prover) to evaluate the low-degree extension of $\mathtt{w}$ at any point.[4] The verifier then chooses a random challenge ρ which defines a random claim about the low-degree extension of $\mathtt{y}$. Following this, an interactive protocol is run that uses C (through its digest ψ) to reduce the claim on $\mathtt{y}$ to claims about the low-degree extensions of $\mathtt{x}$ and $\mathtt{w}$. The verifier checks these by computing the low-degree extension of $\mathtt{x}$ itself

[4]Recall that the (multilinear) low-degree extension of a string $\mathtt{w} \in \{0,1\}^m$ is the unique m-variate multilinear polynomial that agrees with $\mathtt{w}$ on the Boolean hypercube.

and using the PCS to verify the claim about $\mathbb{w}$.

The attacks. [KRS25] provide a number of clever attacks that utilize properties of the GKR protocol to cause it to become unsound once it is transformed using Fiat–Shamir. The main attack, of which all other attacks are variants, utilizes the following observation about the GKR protocol: there exists $\mathbb{y}$ so that, given ρ, an alternate output $\mathbb{y}_\rho \neq \mathbb{y}$ can be found such that the claims on the low-degree extensions of $\mathbb{y}_\rho$ and $\mathbb{y}$ are identical. Thus, by choosing $\mathbb{y}$ as the output, a circuit $\tilde{C}$ that can predict ρ can compute and output $\mathbb{y}_\rho$. Then, the remainder of the GKR proof is run on the false output $\mathbb{y}$, but about a true claim, so that the GKR verifier ends up accepting.

All that remains is to enable $\tilde{C}$ to predict the challenge ρ. This is done in a similar manner to the toy protocol: the circuit $\tilde{C}$ receives as its input $\mathbb{w}$ (and $\mathbb{x}$), computes the PCS commitment of $\mathbb{w}$, and runs the Fiat–Shamir hash on the outputs.

Protection using our transformation. The GKR protocol cannot be described as a relativized protocol as this would require a succinct representation of $\tilde{C}$, and hence of the random oracle. As a result, we cannot directly prove the security of the GKR protocol after applying our XFS variant of Fiat–Shamir. However, our unconditional result yields evidence of its security in the standard model.

The main capability being exploited in the attack is that the circuit $\tilde{C}$ computes the Fiat–Shamir hash function. In fact, the succinct representation of the circuit $\tilde{C}$ plays no part in the attack. Thus, we can map the attack to a protocol where the circuit $\tilde{C}$ is simply run directly on its claimed inputs, as is directly captured by the toy protocol. Indeed, following our transformation, the circuit $\tilde{C}$ would, in order to compute the challenge ρ, need to solve a puzzle that depends on its own description, which it cannot do. Hence, in order to attack the XFS transformation applied to the GKR protocol, one must explicitly exploit the fact the structural properties of the GKR protocol (such as the succinct representation or the precise PCS implementation), which seems more challenging to mount, especially without resulting in a very contrived attack.

Attacks not captured by our model. While our construction successfully captures the recent attacks on GKR, contrived protocols still exist for which our transformation is unsound:

- Barak's [Bar01] non-blackbox zero-knowledge argument begins with the prover sending a commitment which is capable of holding the description of the verifier circuit. A bound t on the verifier running time is known prior to the interaction, which results in an argument that is zero-knowledge against malicious verfiers that run in time t. The simpler version of the protocol sets t to be a fixed polynomial in the input size (which results in zero-knowledge against t-time verifiers). We can securely apply XFS to this version of the protocol by setting the proof-of-work hardness ℓ to be greater than t. However, in the more advanced version of his protocol, Barak sets t to be super-polynomial in order to achieve zero-knowledge against all polynomially bounded verifiers. This would require us to set the proof-of-work to have ℓ to be super-polynomial, which is impractical.

- Bartusek et. al [BBHMR19] show an attack on the application of the Fiat–Shamir transformation to Kilian's succinct argument. Kilian's argument is based on committing to a PCP using Merkle commitments with a collision-resistant hash (CRH). [BBHMR19] give a CRH which allows for attacking the FS transformation by (among other things) adding to the CRH a SNARK proof that the Fiat–Shamir hash function has a specific input/output pair. This contrived attack is not covered by our model, as we do not allow for proving general statements about the Fiat-Shamir hash function (without the verifier performing the queries needed to verify the statement).

It remains an open problem to adapt the XFS transformation to protect against these additional attacks.

2.5 A simple construction of a strong proof-of-work

We have described the need in our construction for a strong proof-of-work. Informally, this is a pair of algorithms $(\mathsf{Solve}, \mathsf{Check})$ such that Solve solves a randomly chosen puzzle of certain hardness ℓ, and Check verify the validity of a solution quickly. The security notion that we need in our construction considers the following experiment: prior to the puzzle being chosen, a preprocessing algorithm is allowed to run for a long time and output an auxiliary state aux. Then, after the puzzle z is sampled, a second online algorithm is given z and aux and runs in time less than ℓ (say, in time $\ell/4$). The proof-of-work is secure if the adversary in this experiment can solve the puzzle only with *negligible probability*. See Definition 6.1 for the precise definition. It is instructive to notice that the proof-of-work that is used in Bitcoin (finding an input that hashes to $\log \ell$ many 0's) does not satisfy this requirement. Indeed, such a scheme can be solved in constant time with probability $1/\ell$ simply by guessing the solution.

Below we describe a simple strong proof-of-work with negligible soundness error, and outline its proof. In Section 6.2, we introduce a further optimized construction that is better suited for practical applications.

This shows a practical way to implement the proof-of-work required for the XFS transformation. Our construction is based on computing a Merkle tree. Roughly speaking, given a random puzzle z, the solver Solve computes a Merkle tree root rt of the message $[(z, 1), \ldots, (z, \ell)]$. Then, it hashes the root rt along with the puzzle z to get random subset of λ leafs to open. The solver provides a valid opening (i.e., an authentication path) for each sampled leaf. The root and the openings are the solution. Checking a solution is natural, one checks the validity of all the authentication paths and that each leaf contains the appropriate value (z, i).

In more detail, we use the notation of a Merkle tree commitment scheme (in the ROM), as defined in [CY24, Chapter 18]. Specifically, a Merkle commitment scheme has the standard three algorithms, $\mathsf{MT} = (\mathsf{MT.Commit}, \mathsf{MT.Open}, \mathsf{MT.Check})$, to commit, open and check an opening. Given the notation of the Merkle tree, our construction works as follows.

Construction 2.1 (Strong PoW)**.** Let $\mathsf{MT} = (\mathsf{MT.Commit}, \mathsf{MT.Open}, \mathsf{MT.Check})$ be a Merkle tree as defined in [CY24, Chapter 18]. We build a strong proof-of-work $(\mathsf{Solve}, \mathsf{Check})$ as follows.

$\mathsf{Solve}^f(\ell, z)$:

1. Compute the message $\mathbf{m}$ of length $\ell/2$, where $\mathbf{m}[i] = (z, i)$.
2. Compute the Merkle commitment: $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MT.Commit}^f(\mathbf{m})$.
3. Compute $\rho \leftarrow f(z, \mathsf{rt})$, and interpret ρ as a subset $I \subseteq [\ell/2]$ of size λ.
4. Compute $\mathsf{pf} \leftarrow \mathsf{MT.Open}^f(\mathsf{td}, I)$.
5. Output $s = (\mathsf{rt}, \mathsf{pf})$.

$\mathsf{Check}^f(\ell, z, s)$:

1. Parse s as $s = (\mathsf{rt}, \mathsf{pf})$ (if it cannot be parsed then reject).
2. Compute $\rho \leftarrow f(z, \mathsf{rt})$, and interpret ρ as a subset $I \subseteq [\ell/2]$ of size λ.
3. Compute the message $\mathbf{m}$ of length $\ell/2$, where $\mathbf{m}[i] = (z, i)$.
4. Set $\mathbf{m}[I]$ such that $\mathbf{m}[i] = (z, i)$ for all $i \in I$.

5. Verify that $\mathsf{MT.Check}^f(\mathsf{rt}, I, \mathbf{m}[I], \mathsf{pf}) = 1$.

Notice that the solver above runs in time ℓ (since the tree has $\ell/2$ leaves and thus ℓ nodes). We prove the security of this construction. In particular, we show that

Theorem 2.2. *Any algorithm with query complexity t_1 for the preprocessing phase, and query complexity $t_2 \leq \ell/4$ for the online phase, solves a random puzzle with probability at most $\frac{t_1+t_2+1}{2^\lambda}$.*

We give a quick sketch of the main idea behind the proof. The preprocessing adversary performs t_1 arbitrary queries. However, the random puzzle z will not be an answer to any of then, except with probability $\frac{t_1}{2^\lambda}$. The online algorithm is limited to making $t_2 \leq \ell/4$ queries and thus can compute at most half of the leaves of the tree (as the queries from the preprocessing phase are not useful in constructing the Merkle authentication paths). Thus, a random subset of size λ will be fully contained in the leaves that the algorithm computed only with probability $2^{-\lambda}$. The algorithm can resample the random subset at most t_2 times (actually much less). Finally, the algorithm, if failed, can output a random solution in hope for a final success. Thus, the overall success probability can be bounded by $\frac{t_1+t_2+1}{2^\lambda}$.

See Section 6 for further details and a construction of our proof-of-work variant.

3 A new Fiat–Shamir transformation

In this section we formally define strong proofs-of-work, and define our XFS transformation.

We define a variant of proof-of-work that are secure even if the adversary has a large preprocessing time prior to the puzzle being sampled. See Section 6 for discussion and construction of this primitive in the ROM.

Definition 3.1 (Strong proof-of-work)**.** *A tuple* $(\mathsf{Setup}, \mathsf{Samp}, \mathsf{Solve}, \mathsf{Check})$ *is a strong* proof-of-work *(PoW) if the following properties hold:*

- Completeness. *A PoW scheme has is perfectly complete if for every* $\ell, \lambda \in \mathbb{N}$*,*

$$\Pr\left[\mathsf{Check}(\mathsf{pp}, z, s) = 1 \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \ell) \\ z \leftarrow \mathsf{Samp}(\mathsf{pp}) \\ s \leftarrow \mathsf{Solve}(\mathsf{pp}, z) \end{array}\right] = 1\,.$$

- Soundness. *A strong PoW scheme has* $(\mathbf{t}_1, \mathbf{t}_2, \varepsilon_{\text{POW}})$*-soundness if for every* $\lambda, \ell \in \mathbb{N}$*, and adversary* $\mathbf{A} = (\mathbf{A}_1, \mathbf{A}_2)$ *such that* $\mathbf{A}_1$ *is a* $\mathbf{t}_1$*-time algorithm and* $\mathbf{A}_2$ *is a* $\mathbf{t}_2$*-time algorithm:*

$$\Pr\left[\mathsf{Check}^f(\mathsf{pp}, z, s) = 1 \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \ell) \\ \mathsf{aux} \leftarrow \mathbf{A}_1(\mathsf{pp}) \\ z \leftarrow \mathsf{Samp}(\mathsf{pp}) \\ s \leftarrow \mathbf{A}_2(\mathsf{pp}, \mathsf{aux}, z) \end{array}\right] \leq \varepsilon_{\text{POW}}(\lambda, \ell, \mathbf{t}_1, \mathbf{t}_2)\,.$$

- Bounded solving time. *For every* $\lambda, \ell \in \mathbb{N}$*,* pp *in the image of* $\mathsf{Setup}(1^\lambda, \ell)$*, and* z *in the image of* $\mathsf{Samp}(\mathsf{pp})$*,* $\mathsf{Solve}(\mathsf{pp}, z)$ *runs in time* ℓ*.*

We turn to formally define our transformation in the standard model. The transformation is the relativized model in given in Construction 5.5.

Construction 3.2 (XFS in the standard model)**.** Given a 3-round interactive protocol $(\mathbf{I}, \mathbf{P}, \mathbf{V})$, and a strong proof-of-work $(\mathsf{Setup}, \mathsf{Samp}, \mathsf{Solve}, \mathsf{Check})$ and a hash function h:

- $\mathcal{G}(1^\lambda, \ell)$: Output $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \ell)$.
- $\mathcal{I}(\mathbb{i})$: Output $\psi \leftarrow \mathbf{I}(\mathbb{i})$.
- $\mathcal{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$:
  1. Compute digest: $\psi \leftarrow \mathbf{I}(\mathbb{i})$
  2. Get prover first message: $m_1 = \mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$.
  3. Derive puzzle randomness: $r = h(\psi, \mathbb{x}, m_1)$.
  4. Compute puzzle: $z = \mathsf{Samp}(\mathsf{pp}; r)$.
  5. Solve puzzle: $s \leftarrow \mathsf{Solve}(\mathsf{pp}, z)$.
  6. Set the pad $\tau = 0^{|\langle \mathbf{V} \rangle|}$, where $\langle \mathbf{V} \rangle$ is the circuit representation of the verifier.
  7. Derive verifier message: $\rho = h(\tau, \psi, \mathbb{x}, m_1, s)$.
  8. Get second prover message: $m_2 = \mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w}, \rho)$.
  9. Output $\pi = (m_1, s, m_2)$.
- $\mathcal{V}(\psi, \mathbb{x}, \pi = (m_1, s, m_2)$:

1. Derive puzzle randomness: $r = h(\psi, \mathbb{x}, m_1)$.
2. Compute puzzle: $z = \mathsf{Samp}(\mathsf{pp}; r)$.
3. Set the pad $\tau = 0^{|\langle \mathbf{V} \rangle|}$.
4. Derive verifier message: $\rho = h(\tau, \psi, \mathbb{x}, m_1, s)$.
5. Accept if $\mathbf{V}(\psi, \mathbb{x}, m_1, \rho, m_2) = 1$ and $\mathsf{Check}(\mathsf{pp}, z, s) = 1$.

Remark 3.3. In the case that the proof-of-work does not require a setup (which is the case with hash-based solutions), such as our construction in Section 6 with the random-oracle replaced with a CRH, the protocol after transformation will not include a setup phase.

Definition 3.4 (Non-contrived verifier)**.** *We say that an SP verifier $\mathbf{V}_{\text{SP}}$ is non-contrived with respect to CRH family $\mathcal{H}$ and padder* Padder *if there exists a simulator $\mathbf{S}$ such that for any PPT adversary $\mathbf{A}$ the following two distributions are computationally indistinguishable:*

$$\left\{ \mathbf{V}_{\text{SP}}(h, \psi, \mathbb{x}, m_1, \rho, m_2) \;\middle|\; \begin{array}{l} h \leftarrow \mathcal{H} \\ \tau \leftarrow \mathsf{Padder}(h) \\ (\psi, \mathbb{x}, m_1, \mathsf{aux}) \leftarrow \mathbf{A}(h, \tau) \\ \rho \leftarrow \{0,1\}^{\lambda} \\ m_2 \leftarrow \mathbf{A}(\mathsf{aux}, \rho) \end{array} \right\} ,$$

and

$$\left\{ \mathbf{S}^{h|_{\tau}}(\tau, \psi, \mathbb{x}, m_1, \rho, m_2) \;\middle|\; \begin{array}{l} h \leftarrow \mathcal{H} \\ \tau \leftarrow \mathsf{Padder}(h) \\ (\psi, \mathbb{x}, m_1, \mathsf{aux}) \leftarrow \mathbf{A}(h, \tau) \\ \rho \leftarrow \{0,1\}^{\lambda} \\ m_2 \leftarrow \mathbf{A}(\mathsf{aux}, \rho) \end{array} \right\} ,$$

where $h|_{\tau}$ is identical to h except that it outputs $\perp$ when given as input a string beginning with τ.

4 Relativized proof systems

We describe how relativized Sigma protocols and SNARKs are modeled in the random oracle model (ROM). Every algorithm with oracle access to a function f in the image of $\mathcal{U}(\lambda)$ is implicitly assumed to receive 1^λ as an additional input.

Notation. For a list $\mu = \{(x, y)\}$ and a function f, we define the μ programmed derivative of f, $f[\mu]$ as follows:

$$f[\mu] := \begin{cases} y & \text{if } (x,y) \in \mu \\ f(x) & \text{otherwise} \end{cases} .$$

We begin by describing relativized Sigma-protocols in the ROM. For simplicity, we focus on Boolean circuits; however, it is straightforward to generalize all the results in this section to arbitrary circuits. We give a notion of round-by-round knowledge, which additionally includes the indexer. For technical reasons related to our main proof, our notion differs slightly from similar definitions found in the literature. However, most known constructions naturally satisfy this definition.

Definition 4.1 (Relativized SP in the ROM)**.** *A triplet of algorithms* $(\mathbf{I}_{\text{SP}}, \mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ *is a relativized SP in the ROM for a family of Boolean circuits* $\mathcal{C}$ *if the following properties hold:*

- Completeness. *For every* $C \in \mathcal{C}$, $\mathbb{x}$, $\mathbb{w}$, *and oracle* $f \in \mathcal{U}(\lambda)$, *for which* $C^f(\mathbb{x}, \mathbb{w}) = 1$, *it holds that*
$$\Pr\left[\mathbf{V}^f_{\text{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) = 1 \;\middle|\; \begin{array}{l} m_1 \leftarrow \mathbf{P}^f_{\text{SP}}(C, \mathbb{x}, \mathbb{w}) \\ \rho \leftarrow \{0,1\}^\lambda \\ m_2 \leftarrow \mathbf{P}^f_{\text{SP}}(\mathsf{aux}, \rho) \\ \psi \leftarrow \mathbf{I}^f_{\text{SP}}(C) \end{array} \right] = 1 .$$

- Adaptive (straightline) round-by-round knowledge. *There exists a polynomial-time extractor* $\mathbf{E}_{\text{SP}}$ *such that the following holds:*

  1. *For every malicious t-query SP prover* $\mathbf{P}$*,*
$$\Pr\left[\begin{array}{l} C \in \mathcal{C} \\ \wedge\, C^f(\mathbb{x}, \mathbb{w}) = 0 \\ \wedge\, \mathbf{V}^f_{\text{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) = 1 \end{array} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (C, \mathbb{x}, m_1, \mathsf{aux}) \xleftarrow{\mathsf{tr}_1} \mathbf{P}^f \\ \rho \leftarrow \{0,1\}^\lambda \\ m_2 \leftarrow \mathbf{P}^f(\mathsf{aux}, \rho) \\ \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_1) \\ \psi \leftarrow \mathbf{I}^f_{\text{SP}}(C) \end{array} \right] \le \kappa_{\text{SP}}(\lambda, t) .$$

  2. *For every malicious t-query SP prover* $\mathbf{P}$ *and every t-query algorithm* $\mathbf{A}$*,*
$$\Pr\left[\begin{array}{l} C \in \mathcal{C} \\ \wedge\, \mathbb{w} \neq \mathbb{w}' \end{array} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (C, \mathbb{x}, m_1, \mathsf{aux}) \xleftarrow{\mathsf{tr}_1} \mathbf{P}^f \\ \bot \xleftarrow{\mathsf{tr}_2} \mathbf{A}^f(\mathsf{aux}) \\ \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_1) \\ \mathbb{w}' \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_1 || \mathsf{tr}_2) \end{array} \right] \le \kappa_{\text{SP}}(\lambda, t) .$$

- Adaptive (straightline) indexer-knowledge. *There exists an polynomial-time extractor* $\mathbf{E}_{\mathrm{I}}$*, such that the following conditions holds:*

  1. *For every malicious* t*-query adversary* $\mathbf{A}$*,*

$$\Pr\left[\begin{array}{l} C \neq C' \\ \wedge\, \mathbf{I}^{f}_{\mathrm{SP}}(C) = \psi \end{array} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\psi, \mathsf{aux}) \xleftarrow{\mathsf{tr}} \mathbf{A}^{f} \\ C' \leftarrow \mathbf{E}_{\mathrm{I}}(\psi, \mathsf{tr}) \\ C \leftarrow \mathbf{A}^{f}(\mathsf{aux}) \end{array}\right] \leq \kappa_{\mathrm{I}}(\lambda, t)\,.$$

  2. *For every malicious* t*-query adversaries* $\mathbf{A}_1, \mathbf{A}_2$*,*

$$\Pr\left[\; C \neq C' \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\psi, \mathsf{aux}) \xleftarrow{\mathsf{tr}_1} \mathbf{A}_1^{f} \\ C \leftarrow \mathbf{E}_{\mathrm{I}}(\psi, \mathsf{tr}_1) \\ \bot \xleftarrow{\mathsf{tr}_2} \mathbf{A}_2^{f}(\mathsf{aux}) \\ C' \leftarrow \mathbf{E}_{\mathrm{I}}(\psi, \mathsf{tr}_1 || \mathsf{tr}_2) \end{array}\right] \leq \kappa_{\mathrm{I}}(\lambda, t)\,.$$

We now define relativized SNARKs in the ROM. In fact, as we do not have a succinctness requirement, we define the more general notion of non-interactive arguments of knowledge (NARKs), but use the name SNARK for simplicity.

Definition 4.2 (Relativized SNARK in the ROM)**.** *A triplet of algorithms* $(\mathcal{I}, \mathcal{P}, \mathcal{V})$ *in the random oracle model is a relativized SNARK in the ROM for a family of Boolean circuits* $\mathcal{C}$ *if the following properties hold:*

- Completeness. *For every* $C \in \mathcal{C}$*,* $\mathbb{x}$*,* $\mathbb{w}$*, and oracle* $f \in \mathcal{U}(\lambda)$*, for which* $C^f(\mathbb{x}, \mathbb{w}) = 1$*, it holds that*

$$\Pr\left[\; \mathcal{V}^{f}(\psi, \mathbb{x}, \pi) = 1 \;\middle|\; \begin{array}{l} \pi \leftarrow \mathcal{P}^{f}(C, \mathbb{x}, \mathbb{w}) \\ \psi \leftarrow \mathcal{I}^{f}(C) \end{array}\right] = 1\,.$$

- Adaptive (straightline) knowledge. *For every malicious* t*-query argument prover* $\tilde{\mathcal{P}}$*,*

$$\Pr\left[\begin{array}{l} C \in \mathcal{C} \\ \wedge\, C^{f}(\mathbb{x}, \mathbb{w}) = 0 \\ \wedge\, \mathcal{V}^{f}(\psi, \mathbb{x}, \pi) = 1 \end{array} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (C, \mathbb{x}, \pi) \xleftarrow{\mathsf{tr}_{\mathcal{P}}} \tilde{\mathcal{P}}^{f} \\ \psi \leftarrow \mathcal{I}^{f}(C) \\ \mathbb{w} \leftarrow \mathcal{E}(C, \mathbb{x}, \pi, \mathsf{tr}_{\mathcal{P}}) \end{array}\right] \leq \kappa_{\mathrm{ARG}}(\lambda, t)\,.$$

5 On the security of our transformation

In this section, we prove the security of our transformation for relativized proof systems (see Section 4) in which all parties and circuits have access to the random oracle which is used for the transformation.

- In Section 5.1, we describe a prefix-avoiding padder – a simple primitive that we use in our construction.
- In Section 5.2, we express our transformation in the context of relativized proof systems and state our main theorem.
- In Section 5.3, we give a detailed proof of our main theorem.

5.1 Prefix avoiding padders

We begin by describing a notion of *prefix avoidance*, representing the capability of the prover to make the verifier sample within a restricted prefix. We give a general definition with respect to an oracle machine $\mathbf{M}$. In our main proof, we use a padder where the machine $\mathbf{M}$ is the verifier of the Sigma protocol $\mathbf{V}_{\text{SP}}$.

Definition 5.1. *A oracle algorithm* $\mathbf{M}$ *is* ε_{PAD}*-prefix-avoiding with respect to padder* Padder *if for every* $\lambda \in \mathbb{N}$ *and* t*-query adversary* $\mathbf{A}$*,*

$$\Pr\left[(\tau, \gamma) \in \mathsf{tr}_{\text{PAD}} \cup \mathsf{tr}_{\mathbf{M}} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\gamma, m) \leftarrow \mathbf{A}^f \\ \tau \xleftarrow{\mathsf{tr}_{\text{PAD}}} \mathsf{Padder}^f(\gamma) \\ y \xleftarrow{\mathsf{tr}_{\mathbf{M}}} \mathbf{M}^f(m) \end{array} \right] < \varepsilon_{\text{PAD}}(\lambda, t)\,.$$

We observe that any algorithm $\mathbf{M}$ has a padder, and there are several different alternative. Let $\langle \mathbf{M} \rangle$ be any succinct description on the machine $\mathbf{M}$ (e.g., a Rust code representation).

- Any $\mathbf{M}$ is 0-prefix-avoiding with respect to the padder $\mathsf{Padder}^f(\gamma)$ that outputs $\tau = 0^k$ for k that the size of the circuit describing $\mathbf{M}$. This is since $\mathbf{M}$ simply cannot make a query that is larger than its circuit description.

- If $\mathbf{M}$ be be written as an AND of several smaller $\mathbf{M}_i$'s (which is usually the case with verifiers) then the above holds for k that is size of the largest $\mathbf{M}_i$.

- For the same reasons, any $\mathbf{M}$ that makes queries of length at most ℓ is 0-prefix-avoiding with respect to the padder $\mathsf{Padder}^f(\gamma)$ that outputs $\tau = 0^\ell$. Note that ℓ is always bounded by the circuit size of $\mathbf{M}$, but in practical schemes, it may be significantly smaller.

- One can also design a padder that ignores the input γ. This has a big advantage, since this padder only depends on the oracle f as thus can be computed one and for all in a preprocessing phase, and the value τ can be reused in all executions of the protocol. To design such a padder, one needs to take a long enough pad that will avoid the queries of $\mathbf{M}$, for any possible input.

The observations above are captured in the following simple lemma.

Lemma 5.2. *Let* $\mathbf{M}$ *be oracle algorithm that makes at most* q *queries and code description* $\langle \mathbf{M} \rangle$*. Then the following holds:*

1. $\mathbf{M}$ *is* 0*-prefix-avoiding with respect to the padder* $\mathsf{Padder}^f(\gamma) = 0^\ell$*, where* ℓ *is a bound on the length of queries made by* $\mathbf{M}$*. Note that it always holds that* $\ell \leq |\langle \mathbf{M} \rangle|$*.*

2. *Let* $\mathbf{M}$ *be an algorithm that gets* ℓ *bits as input. Then,* $\mathbf{M}$ *is* $O(q/2^{2\lambda})$*-prefix-avoiding with respect to the padder* $\mathsf{Padder}^f(\gamma) = (f(\langle \mathbf{M} \rangle, \gamma, 1), \ldots, f(\langle \mathbf{M} \rangle, \gamma, k))$*, where* k *satisfies* $k \cdot \lambda = \ell + 2\lambda$*.*

Remark 5.3. The second padder in Lemma 5.2 satisfies a stronger security notion (with the same error). Here, we allow the adversary to choose the machine $\mathbf{M}$ adaptively after performing queries for the random oracle. More formally, the stronger notion that these padders satisfy is: Padder is ε_{PAD}-avoiding if for every $\lambda, q \in \mathbb{N}$ and t-query adversary $\mathbf{A}$ that outputs circuits C that make at most q queries,

$$\Pr\left[(\tau, \gamma) \in \mathsf{tr}_{\text{PAD}} \cup \mathsf{tr}_{\mathbf{M}} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\mathbf{M}, \gamma, m) \leftarrow \mathbf{A}^f \\ \tau \xleftarrow{\mathsf{tr}_{\text{PAD}}} \mathsf{Padder}^f(\mathbf{M}, \gamma) \\ y \xleftarrow{\mathsf{tr}_{\mathbf{M}}} \mathbf{M}^f(m) \end{array} \right] < \varepsilon_{\text{PAD}}(\lambda, t, q)\,.$$

Intuitively, this notion says that no one can even *design* a machine $\mathbf{M}$ (or more concretely, a verifier $\mathbf{V}_{\text{SP}}$ for a Sigma protocol) and a set of inputs to it so that it makes queries to the restricted area of the random oracle generated by Padder. This gives us strong security guarantees against a malicious prover which adaptivly designed a Sigma protocol as a function of the oracle f. For simplicity, our transformation has the Sigma protocol fixed in advanced, but the definition and proof can all be easily augmented to this setting.

Remark 5.4 (Prefix avoiding verifiers)**.** In our proof, we replace the machine $\mathbf{M}$ in the above prefix-avoiding definition with the verifier $\mathbf{V}_{\text{SP}}$ in order to claim that the verifier does not query in the zone reserved for the Fiat–Shamir hash function.

Typically, verifiers are small programs designed to perform specific, prescribed actions, in contrast to the circuit C, which is large and capable of executing arbitrarily chosen malicious computations. Therefore, it is reasonable to assume that there are known regions of the random oracle that the verifier will not query, regardless of the messages it receives from the prover.

In this case, τ can be chosen in advance by the padder as a *preprocessing step*, without requiring any additional inputs (which is practically much more efficient). Moreover, in real-world protocols, the verifier may have access to arbitrary information about the circuit C and other information. To account for this, we provide a simulator-based definition, which asserts that the verifier can be simulated by an unbounded simulator with access to all the information in the game except for the restricted region of the random oracle.

The definition for a specific verifier $\mathbf{V}_{\text{SP}}$ and padder Padder follows: for every $\lambda, t \in \mathbb{N}$ and SP prover $\mathbf{P}$ that makes at most t queries there exists an unbounded simulator $\mathbf{S}$ such that the two distributions below have statistical distance at most $\varepsilon(\lambda, t)$:

$$\left\{ \mathbf{V}_{\text{SP}}^f(\psi, \mathbb{x}, m_1, \rho, m_2) \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\psi, \mathbb{x}, m_1, \mathsf{aux}) \xleftarrow{\mathsf{tr}} \mathbf{P}^f \\ \rho \leftarrow \{0,1\}^\lambda \\ m_2 \leftarrow \mathbf{P}^f(\mathsf{aux}, \rho) \end{array} \right\},$$

and

$$\left\{ \mathbf{S}(\tau, f|_\tau, \psi, \psi', \mathbb{x}, m_1, \rho, m_2, C, \mathbb{w}, \mathsf{tr}_{\mathbf{I}}, \mathsf{tr}_C, \mathsf{tr}_{\text{PAD}}) \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \tau \xleftarrow{\mathsf{tr}_{\text{PAD}}} \mathsf{Padder}^f \\ (\psi, \mathbb{x}, m_1, \mathsf{aux}) \xleftarrow{\mathsf{tr}} \mathbf{P}^f \\ \rho \leftarrow \{0,1\}^\lambda \\ m_2 \leftarrow \mathbf{P}^f(\mathsf{aux}, \rho) \\ (C, \mathbb{w}) \leftarrow \mathbf{E}_{\text{SP}}(\psi, \mathbb{x}, m_1, \mathsf{tr}) \\ \psi' \xleftarrow{\mathsf{tr}_{\mathbf{I}}} \mathbf{I}^f_{\text{SP}}(C) \\ b \xleftarrow{\mathsf{tr}_C} C^f(\mathbb{x}, \mathbb{w}) \end{array} \right\} ,$$

where $f|_\tau$ is identical to f except that it outputs $\perp$ if given as input a string beginning in τ.

Our proof of security works (with slight adaptations) assuming the verifier and padder have this property.

5.2 Our transformation in the ROM

We can now describe our Fiat–Shamir transformation for relativized SPs in the ROM. For simplicity, we assume that the verifier randomness complexity and the size of the random puzzles are both λ bits.

Construction 5.5 (XFS in the ROM)**.**

- $\mathcal{I}^f(C)$: Output $\psi \leftarrow \mathbf{I}^f_{\text{SP}}(C)$.
- $\mathcal{P}^f(C, \mathbb{x}, \mathbb{w})$:
  1. Compute digest: $\psi \leftarrow \mathbf{I}^f_{\text{SP}}(C)$.
  2. Run the SP prover: $(m_1, \mathsf{aux}) \leftarrow \mathbf{P}^f_{\text{SP}}(C, \mathbb{x}, \mathbb{w})$.
  3. Compute the puzzle: $z = f(\psi, \mathbb{x}, m_1)$.
  4. Solve $s \leftarrow \mathsf{Solve}^f(\ell, z)$.
  5. Set $\tau \leftarrow \mathsf{Padder}^f(\psi, \mathbb{x}, m_1, s)$.
  6. Derive SP challenge: $\rho := f(\tau, \psi, \mathbb{x}, m_1, s)$.
  7. Run the SP prover: $m_2 \leftarrow \mathbf{P}^f_{\text{SP}}(\mathsf{aux}, \rho)$.
  8. Output the argument string $\pi := (m_1, s, m_2)$.
- $\mathcal{V}^f(\psi, \mathbb{x}, \pi)$:
  1. Parse the argument string π as a tuple (m_1, s, m_2).
  2. Compute the puzzle: $z = f(\psi, \mathbb{x}, m_1)$.
  3. Set $\tau \leftarrow \mathsf{Padder}^f(\psi, \mathbb{x}, m_1, s)$.
  4. Derive SP challenge: $\rho := f(\tau, \psi, \mathbb{x}, m_1, s)$.
  5. Check: $\mathbf{V}^f_{\text{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) = 1$.
  6. Check: $\mathsf{Check}^f(\ell, z, s) = 1$.

Theorem 5.6. *Assume we have the following ingredients:*

- *Let $\mathcal{C}$ be a family of circuits, where $\mathbf{q}_C$ is a bound on the number of oracle gates.*
- *Let $(\mathbf{I}_{\text{SP}}, \mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$ be a relativized SP in the ROM for a family of circuits $\mathcal{C}$, with round-by-round knowledge soundness error κ_{SP}, index-knowledge κ_{I}, and indexer query complexity $\mathbf{q}_{\text{I}}$ (see Definition 4.1).*

- *Let* $(\mathsf{Solve}, \mathsf{Check})$ *be a strong PoW in the ROM with soundness error* ε_{POW} *and checking query complexity* $\mathbf{q}_{\text{POW}}$ *(see Definition 6.1).*
- *Let* Padder *be prefix avoiding for* $\mathbf{V}_{\text{SP}}$ *with error* ε_{PAD} *and query complexity* $\mathbf{q}_{\text{PAD}}$ *(see Definition 5.1).*

For every security parameter $\lambda \in \mathbb{N}$*, and* $\ell \in \mathbb{N}$*, the system compiled in Construction 5.5 is relativized SNARK in the ROM for* $\mathcal{C}$ *with adaptive knowledge error* κ_{ARG} *(see Definition 4.2) such that*

$$\kappa_{\text{ARG}}(\lambda, t) \leq (t+1) \cdot \kappa_{\text{SP}}(\lambda, t) + (t+1) \cdot \varepsilon_{\text{POW}}(\lambda, \ell, t, t_2) + 2\kappa_{\text{I}}(\lambda, t_1) + \varepsilon_{\text{PAD}}(\lambda, t_1) .$$

where $t_1 := t + \mathbf{q}_{\text{I}} + \mathbf{q}_{\text{PAD}} + 1$*, and* $t_2 := \mathbf{q}_{\text{POW}} \cdot (\mathbf{q}_C + \mathbf{q}_{\text{I}})$*.*

The construction running times are:

- *Indexer:* $T_{\mathcal{I}} = T_{\mathbf{I}}$*.*
- *Prover:* $T_{\mathcal{P}} = T_{\mathbf{P}} + T_{\mathbf{I}} + T_{\text{POW.slv}} + T_{\text{PAD}} + O(1)$*.*
- *Verifier:* $T_{\mathcal{V}} = T_{\mathbf{V}} + T_{\text{POW.chk}} + T_{\text{PAD}} + O(1)$*.*
- *Extractor:* $T_{\mathcal{E}} = T_{\mathbf{E}}$*.*

where $(T_{\mathbf{I}}, T_{\mathbf{P}}, T_{\mathbf{V}}, T_{\mathbf{E}})$ *are the indexer, prover, verifier and extractor times of the SP,* $(T_{\text{POW.slv}}, T_{\text{POW.chk}})$ *are the solving and checking times of the proof-of-work, and* T_{PAD} *is the padder running time.*

5.3 Security proof of our construction

We prove the theorem by a reduction from a malicious SNARK to a malicious SP prover. The majority of our proof is in proving Lemma 5.7, where we show a reduction for *admissible* adversaries. An argument adversary $\tilde{\mathcal{P}}$ is admissible if whenever $\tilde{\mathcal{P}}$ outputs $(C, \mathbb{x}, \pi = (m_1, s, m_2))$:

Admissible:

1. The following queries are contained in its trace: $(\psi, \mathbb{x}, m_1)$, and $(\tau, \psi, \mathbb{x}, m_1, s)$ where $\psi \leftarrow \mathbf{I}_{\text{SP}}^f(C)$ and $\tau \leftarrow \mathsf{Padder}^f(\psi, \mathbb{x}, m_1, s)$,
2. Letting tr be the trace up to the query $(\psi, \mathbb{x}, m_1)$, and $\mathsf{tr}_{\mathcal{P}}$ be the full trace of $\tilde{\mathcal{P}}$, it holds that $C = \mathbf{E}_{\text{I}}(\psi, \mathsf{tr})$, and $\mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}) = \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_{\mathcal{P}})$, and
3. Letting tr be the trace up to the query $(\tau, \psi, \mathbb{x}, m_1, s)$, it holds that $C = \mathbf{E}_{\text{I}}(\psi, \mathsf{tr})$.

Following the proof of the lemma, we conclude the proof of Theorem 5.6 by showing how to handle adversaries which are not admissible.

Lemma 5.7. *Let* $\mathbf{E}_{\text{SP}}$ *be the SP extractor for* $(\mathbf{I}_{\text{SP}}, \mathbf{P}_{\text{SP}}, \mathbf{V}_{\text{SP}})$*. There exists an argument extractor* $\mathcal{E}$ *and an SP prover* $\mathbf{P}$ *such that for every query bound* $t \in \mathbb{N}$*, malicious* t*-query admissible argument* *prover* $\tilde{\mathcal{P}}$,

$$\Pr\left[\begin{array}{l|l} C \in \mathcal{C} & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, y = 0 & (C, \mathbb{x}, (m_1, s, m_2)) \xleftarrow{\mathsf{tr}_{\mathcal{P}}} \tilde{\mathcal{P}}^f \\ \wedge\, b = 1 & \psi \leftarrow \mathcal{I}^f(C) \\ & \mathbb{w} \leftarrow \mathcal{E}(C, \mathbb{x}, m_1, \mathsf{tr}_{\mathcal{P}}) \\ & b \leftarrow \mathcal{V}^f(\psi, \mathbb{x}, \pi) \\ & y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array}\right]$$

$$\leq t \cdot \Pr\left[\begin{array}{l|l} C \in \mathcal{C} & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, y = 0 & (C, \mathbb{x}, m_1, \mathsf{aux}) \xleftarrow{\mathsf{tr}} \mathbf{P}^f(\tilde{\mathcal{P}}) \\ \wedge\, b = 1 & \rho \leftarrow \{0,1\}^\lambda \\ & m_2 \leftarrow \mathbf{P}^f(\mathsf{aux}, \rho) \\ & \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}) \\ & \psi \leftarrow \mathbf{I}^f_{\text{SP}}(C) \\ & b \leftarrow \mathbf{V}^f_{\text{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) \\ & y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array}\right] + t \cdot \varepsilon_{\text{POW}}(\lambda, \ell, t, t_2) + \varepsilon_{\text{PAD}}(\lambda, t_1)\,.$$

Above, $t_1 := t + \mathbf{q}_{\text{I}} + \mathbf{q}_{\text{PAD}} + 1$ *and* $t_2 := \mathbf{q}_{\text{POW}} \cdot (\mathbf{q}_C + \mathbf{q}_{\text{I}})$.

Proof. Throughout this proof, we define several probability experiments. In these experiments, will use a special notation to denote the case when two algorithms are forced have the same output by the probability statement (e.g., $x_1 \leftarrow \mathbf{A}_1$ and $x_2 \leftarrow \mathbf{A}_2$ where the probability experiment posits that $x_1 = x_2$). If the distinction in the algorithm that generates the output is unimportant, we replace one of the outputs with $\cdot$ (e.g., $x \leftarrow \mathbf{A}_1$ and $(\cdot) \leftarrow \mathbf{A}_2$).

For simplicity of the proof, we furthermore assume that for $z \in \{0,1\}^\lambda$ and any s, the proof-of-work check function $\mathsf{Check}^f(z, s)$ does not make queries of length $|(\psi, \mathbb{x}, m_1)|$. This can be enforced by sufficient domain separation of the random oracle for the proof-of-work or, alternatively, by another padder. We note that for the proofs-of-work that we construct, at most a constant number of padding bits are necessary.

The argument extractor $\mathcal{E}$ of our compiled scheme is defined as follows.

Construction 5.8. The argument extractor $\mathcal{E}$ receives as input an instance $\mathbb{x}$, argument string π, query-answer trace tr, and works as follows.

$\mathcal{E}(C, \mathbb{x}, \pi, \mathsf{tr}_{\mathcal{P}})$:
1. Parse $\pi = (m_1, s, m_2)$.
2. Compute the witness: $\mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_{\mathcal{P}})$.
3. Output $\mathbb{w}$.

We define algorithms describing running the malicious prover for i queries. For an index $i \in \mathbb{N}$, we define $\tilde{\mathcal{P}}_{0\to i}$ and $\tilde{\mathcal{P}}_{i\to t}$ as follows:

- $\tilde{\mathcal{P}}^f_{0\to i}$:
  1. Simulate $\tilde{\mathcal{P}}$ up until the i-th query. If $\tilde{\mathcal{P}}$ has already finished running, then let $x = \bot$.
  2. Outputs the i-th query x and its state $\mathsf{aux}_{\tilde{\mathcal{P}}}$.
- $\tilde{\mathcal{P}}^f_{i\to t}(\mathsf{aux}_{\tilde{\mathcal{P}}}, \rho)$:

1. Simulate $(C, \mathbb{x}, (m_1, s, m_2)) \leftarrow \tilde{\mathcal{P}}^f$ using $\mathsf{aux}_{\tilde{\mathcal{P}}}$ to continue the execution from the i-th query given oracle response ρ.
2. Output $(C, \mathbb{x}, (m_1, s, m_2))$.

The construction of the SP prover $\mathbf{P}$ works as follows.

Construction 5.9. The SP prover $\mathbf{P}$ is parameterized by the query bound t and receives an argument prover $\tilde{\mathcal{P}}$ as a black box, and works as follows.

- First SP prover message: $\mathbf{P}^f(\tilde{\mathcal{P}})$.
  1. Sample an index $i \leftarrow [t]$ at random.
  2. Run $(x, \mathsf{aux}_{\tilde{\mathcal{P}}}) \xleftarrow{\mathsf{tr}} \tilde{\mathcal{P}}^f_{0\to i}$.
  3. Parse $x = (\tau, \psi, \mathbb{x}, m_1, s)$ (if it cannot be parsed in this way, then output $(C, \mathbb{x}, m_1, \mathsf{aux}) = (\bot, \bot, \bot, \bot)$).
  4. Compute $C \leftarrow \mathbf{E}_{\mathrm{I}}(\psi, \mathsf{tr})$.
  5. Output $(C, \mathbb{x}, m_1, \mathsf{aux} = (i, (\tau, \psi, \mathbb{x}, m_1, s), \mathsf{aux}_{\tilde{\mathcal{P}}}))$.
- Second SP prover message: $\mathbf{P}^f(\mathsf{aux} = (i, (\tau, \psi, \mathbb{x}, m_1, s), \mathsf{aux}_{\tilde{\mathcal{P}}}), \rho)$.
  1. Run $(C', \mathbb{x}', (m_1', s', m_2')) \leftarrow \tilde{\mathcal{P}}^f_{i\to t}(\mathsf{aux}_{\tilde{\mathcal{P}}}, \rho)$.
  2. Output m_2'.

We first write the argument probability in terms of the Sigma protocol and the proof-of-work:

$$
\Pr\left[\begin{array}{l|l} & f \leftarrow \mathcal{U}(\lambda) \\ C \in \mathcal{C} & (C, \mathbb{x}, (m_1, s, m_2)) \xleftarrow{\mathsf{tr}_{\mathcal{P}}} \tilde{\mathcal{P}}^f \\ \wedge\, y = 0 & \psi \leftarrow \mathcal{I}^f(C) \\ \wedge\, b = 1 & \mathbb{w} \leftarrow \mathcal{E}(C, \mathbb{x}, m_1, \mathsf{tr}_{\mathcal{P}}) \\ & b \leftarrow \mathcal{V}^f(\psi, \mathbb{x}, \pi) \\ & y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array}\right]
$$

$$
= \Pr\left[\begin{array}{l|l} & f \leftarrow \mathcal{U}(\lambda) \\ & (C, \mathbb{x}, (m_1, s, m_2)) \xleftarrow{\mathsf{tr}_{\mathcal{P}}} \tilde{\mathcal{P}}^f \\ & \psi \leftarrow \mathbf{I}^f_{\mathrm{SP}}(C) \\ C \in \mathcal{C} & z \leftarrow f(\psi, \mathbb{x}, m_1) \\ \wedge\, y = 0 & \tau \leftarrow \mathsf{Padder}^f(\psi, \mathbb{x}, m_1, s) \\ \wedge\, b = 1 & \rho \leftarrow f(\tau, \psi, \mathbb{x}, m_1, s) \\ \wedge\, b_{\mathrm{POW}} = 1 & \mathbb{w} \leftarrow \mathbf{E}_{\mathrm{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_{\mathcal{P}}) \\ & b \leftarrow \mathbf{V}^f_{\mathrm{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) \\ & b_{\mathrm{POW}} \leftarrow \mathsf{Check}^f(z, s) \\ & y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array}\right]
$$

Let $\mathsf{E}_{\mathrm{POW}}$ be the event that $C \in \mathcal{C}$ and one of $\mathbf{I}^f_{\mathrm{SP}}(C)$ or $C^f(\mathbb{x}, \mathbb{w})$ makes the query $(\tau, \psi, \mathbb{x}, m_1, s)$. Similarly, let $\mathsf{E}_{\mathrm{PAD}}$ be the event that one of $\mathbf{V}^f_{\mathrm{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2)$ and $\mathsf{Padder}^f(\psi, \mathbb{x}, m_1, s)$ makes the query $(\tau, \psi, \mathbb{x}, m_1, s)$. Then the above probability is bounded from above by:

$$\Pr\left[\begin{array}{l|l} C \in \mathcal{C} & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, y = 0 & (C, \mathbb{x}, (m_1, s, m_2)) \xleftarrow{\mathsf{tr}_{\mathcal{P}}} \tilde{\mathcal{P}}^f \\ \wedge\, b = 1 & \psi \leftarrow \mathbf{I}^f_{\text{SP}}(C) \\ \wedge\, b_{\text{POW}} = 1 & z \leftarrow f(\psi, \mathbb{x}, m_1) \\ \wedge\, \overline{\mathsf{E}_{\text{POW}}} & \tau \leftarrow \mathsf{Padder}^f(\psi, \mathbb{x}, m_1, s) \\ \wedge\, \overline{\mathsf{E}_{\text{PAD}}} & \rho \leftarrow f(\tau, \psi, \mathbb{x}, m_1, s) \\ & \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_{\mathcal{P}}) \\ & b \leftarrow \mathbf{V}^f_{\text{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) \\ & b_{\text{POW}} \leftarrow \mathsf{Check}^f(z, s) \\ & y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array}\right] + \Pr[\mathsf{E}_{\text{POW}}] + \Pr[\mathsf{E}_{\text{PAD}}]\,.$$

We argue that the first probability is bounded by t times the probability of $\mathbf{P}$ succeeding in the knowledge experiment. Following this, we will show that $\Pr[\mathsf{E}_{\text{POW}}] \leq t \cdot \varepsilon_{\text{POW}}(\lambda, \ell, t, t_2)$ and $\Pr[\mathsf{E}_{\text{PAD}}] \leq \varepsilon_{\text{PAD}}(\lambda, t_1)$, which proves the lemma.

Claim 5.10.

$$\Pr\left[\begin{array}{l|l} C \in \mathcal{C} & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, y = 0 & (C, \mathbb{x}, (m_1, s, m_2)) \xleftarrow{\mathsf{tr}_{\mathcal{P}}} \tilde{\mathcal{P}}^f \\ \wedge\, b = 1 & \psi \leftarrow \mathbf{I}^f_{\text{SP}}(C) \\ \wedge\, b_{\text{POW}} = 1 & z \leftarrow f(\psi, \mathbb{x}, m_1) \\ \wedge\, \overline{\mathsf{E}_{\text{POW}}} & \tau \leftarrow \mathsf{Padder}^f(\psi, \mathbb{x}, m_1, s) \\ \wedge\, \overline{\mathsf{E}_{\text{PAD}}} & \rho \leftarrow f(\tau, \psi, \mathbb{x}, m_1, s) \\ & \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_{\mathcal{P}}) \\ & b \leftarrow \mathbf{V}^f_{\text{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) \\ & b_{\text{POW}} \leftarrow \mathsf{Check}^f(z, s) \\ & y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array}\right] \leq t \cdot \Pr\left[\begin{array}{l|l} C \in \mathcal{C} & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, y = 0 & ((C, \mathbb{x}, m_1), \mathsf{aux}) \xleftarrow{\mathsf{tr}_1} \mathbf{P}^f(\tilde{\mathcal{P}}) \\ \wedge\, b = 1 & \rho \leftarrow \{0,1\}^\lambda \\ & m_2 \xleftarrow{\mathsf{tr}_2} \mathbf{P}^f(\tilde{\mathcal{P}}, \mathsf{aux}, \rho) \\ & \psi \leftarrow \mathbf{I}^f_{\text{SP}}(C) \\ & \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_1 \cup \mathsf{tr}_2) \\ & b \leftarrow \mathbf{V}^f_{\text{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) \\ & y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array}\right].$$

Proof. Our goal is to, step by step, reduce the left-hand-side of the claim to a format which will, eventually, (up to a factor of t) be equivalent to the Sigma protocol experiment with prover $\mathbf{P}(\tilde{\mathcal{P}})$.

The Sigma protocol is a two-part algorithm, and so our first step is to consider $\tilde{\mathcal{P}}$ structurally as having two parts. Recall that, since $\tilde{\mathcal{P}}$ is admissible (Item 1), it always makes the query $(\tau, \psi, \mathbb{x}, m_1, z)$ where $\psi = \mathbf{I}^f_{\text{SP}}(C)$ and $\tau = \mathsf{Padder}^f(\psi, \mathbb{x}, m_1, s)$. Thus we can consider first part of $\tilde{\mathcal{P}}$ its computation before making the query $(\tau, \psi, \mathbb{x}, m_1, z)$, and its second par as being everything after this query has been made.

Formally, let $i \in [t]$ be the random variable representing the index of the query $(\tau, \psi, \mathbb{x}, m_1, z)$ in the query-trace $\mathsf{tr}_{\mathcal{P}}$ of $\tilde{\mathcal{P}}$. Then we can rewrite the left-hand-side of the probability in the claim statement as:

$$\Pr\left[\begin{array}{l|l} C \in \mathcal{C} & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, y = 0 & ((\tau, \psi, \mathbb{x}, m_1, s), \mathsf{aux}_{\tilde{\mathcal{P}}}) \leftarrow \tilde{\mathcal{P}}^f_{0\to i} \\ \wedge\, b = 1 & \rho \leftarrow f(\tau, \psi, \mathbb{x}, m_1, s) \\ \wedge\, b_{\text{POW}} = 1 & (C, \cdot, (\cdot, \cdot, m_2)) \leftarrow \tilde{\mathcal{P}}^f_{i\to t}(\mathsf{aux}_{\tilde{\mathcal{P}}}, \rho) \\ \wedge\, \overline{\mathsf{E}_{\text{POW}}} & z \leftarrow f(\psi, \mathbb{x}, m_1) \\ \wedge\, \overline{\mathsf{E}_{\text{PAD}}} & (\cdot) \leftarrow \mathsf{Padder}^f(\psi, \mathbb{x}, m_1, s) \\ \psi = \mathbf{I}^f_{\text{SP}}(C) & \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_{\mathcal{P}}) \\ & b \leftarrow \mathbf{V}^f_{\text{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) \\ & b_{\text{POW}} \leftarrow \mathsf{Check}^f(z, s) \\ & y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array}\right]$$

$$\leq \Pr\left[\begin{array}{l|l} C \in \mathcal{C} & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, y = 0 & ((\tau, \psi, \mathbb{x}, m_1, s), \mathsf{aux}_{\tilde{\mathcal{P}}}) \leftarrow \tilde{\mathcal{P}}^f_{0\to i} \\ \wedge\, b = 1 & \rho \leftarrow f(\tau, \psi, \mathbb{x}, m_1, s) \\ \wedge\, \overline{\mathsf{E}_{\text{POW}}} & (C, \cdot, (\cdot, \cdot, m_2)) \leftarrow \tilde{\mathcal{P}}^f_{i\to t}(\mathsf{aux}_{\tilde{\mathcal{P}}}, \rho) \\ \wedge\, \overline{\mathsf{E}_{\text{PAD}}} & \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_{\mathcal{P}}) \\ \psi = \mathbf{I}^f_{\text{SP}}(C) & b \leftarrow \mathbf{V}^f_{\text{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) \\ & y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array}\right],$$

Where the second inequality holds by removing the padder and the restrictions on the proof-of-work. As the SP prover $\mathbf{P}(\tilde{\mathcal{P}})$ uses $\mathbf{E}_{\text{I}}$, we need to add it into the experiment. Since we have no requirements on $\mathbf{E}_{\text{I}}$, this does not change the probabilities. However, note that since $\tilde{\mathcal{P}}$ is admissible, (Item 3), for $C' \leftarrow \mathbf{E}_{\text{I}}(\psi, \mathsf{tr}_1)$, it holds that $C' = C$. Thus, the above probability is equal:

$$\Pr\left[\begin{array}{l|l} C \in \mathcal{C} & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, y = 0 & ((\tau, \psi, \mathbb{x}, m_1, s), \mathsf{aux}_{\tilde{\mathcal{P}}}) \xleftarrow{\mathsf{tr}_1} \tilde{\mathcal{P}}^f_{0\to i} \\ \wedge\, b = 1 & C \leftarrow \mathbf{E}_{\text{I}}(\psi, \mathsf{tr}_1) \\ \wedge\, \overline{\mathsf{E}_{\text{POW}}} & \rho \leftarrow f(\tau, \psi, \mathbb{x}, m_1, s) \\ \wedge\, \overline{\mathsf{E}_{\text{PAD}}} & (\cdot, \cdot, (\cdot, \cdot, m_2)) \xleftarrow{\mathsf{tr}_2} \tilde{\mathcal{P}}^f_{i\to t}(\mathsf{aux}_{\tilde{\mathcal{P}}}, \rho) \\ \wedge\, \psi = \mathbf{I}^f_{\text{SP}}(C) & \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_1 \cup \mathsf{tr}_2) \\ & b \leftarrow \mathbf{V}^f_{\text{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) \\ & y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array}\right].$$

Our next goal is to change ρ from being sampled as the output of the random oracle, to being uniformly random. We begin by differentiating the use of the random oracle by $\mathbf{I}_{\text{SP}}$, C, and $\mathbf{V}_{\text{SP}}$ from that of $\tilde{\mathcal{P}}$. Since $\overline{\mathsf{E}_{\text{POW}}}$ and $\overline{\mathsf{E}_{\text{PAD}}}$ hold, we have that $\mathbf{I}_{\text{SP}}$, C, and $\mathbf{V}_{\text{SP}}$ do not make the query $(\tau, \psi, \mathbb{x}, m_1, s)$. Thus we can resample it uniformly and give them a programmed oracle without changing the probability that the verifier accepts, or that the circuit outputs 0. Thus, the above is equal to:

$$\Pr\left[\begin{array}{l|l} & f \leftarrow \mathcal{U}(\lambda) \\ & ((\tau,\psi,\mathbb{x},m_1,s),\mathsf{aux}_{\tilde{\mathcal{P}}}) \xleftarrow{\mathsf{tr}_1} \tilde{\mathcal{P}}^f_{0\to i} \\ C \in \mathcal{C} & C \leftarrow \mathbf{E}_{\text{I}}(\psi,\mathsf{tr}_1) \\ \wedge\, y = 0 & \rho \leftarrow f(\tau,\psi,\mathbb{x},m_1,s) \\ \wedge\, b = 1 & (\cdot,\cdot,(\cdot,\cdot,m_2)) \xleftarrow{\mathsf{tr}_2} \tilde{\mathcal{P}}^f_{i\to t}(\mathsf{aux}_{\tilde{\mathcal{P}}},\rho) \\ \wedge\, \overline{\mathsf{E}_{\text{POW}}} & \rho^* \leftarrow \{0,1\}^\lambda \\ \wedge\, \overline{\mathsf{E}_{\text{PAD}}} & \mu = \{((\tau,\psi,\mathbb{x},m_1,s),\rho^*)\} \\ \wedge\, \psi = \mathbf{I}^{f[\mu]}_{\text{SP}}(C) & \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C,\mathbb{x},m_1,\mathsf{tr}_1\cup\mathsf{tr}_2) \\ & b \leftarrow \mathbf{V}^{f[\mu]}_{\text{SP}}(\psi,\mathbb{x},m_1,\rho,m_2) \\ & y \leftarrow C^{f[\mu]}(\mathbb{x},\mathbb{w}) \end{array}\right].$$

Since f is uniformly random, this is equivalent to $\mathbf{I}_{\text{SP}}$, C, and $\mathbf{V}_{\text{SP}}$ running with f, and $\tilde{\mathcal{P}}_{i\to t}$ receiving an independent random sample. Additionally, as we have no more use for them, we can remove the requirement that $\overline{\mathsf{E}_{\text{POW}}}$ and $\overline{\mathsf{E}_{\text{PAD}}}$ hold. This, the above is bounded by:

$$\Pr\left[\begin{array}{l|l} & f \leftarrow \mathcal{U}(\lambda) \\ & ((\tau,\psi,\mathbb{x},m_1,s),\mathsf{aux}_{\tilde{\mathcal{P}}}) \xleftarrow{\mathsf{tr}_1} \tilde{\mathcal{P}}^f_{0\to i} \\ C \in \mathcal{C} & C \leftarrow \mathbf{E}_{\text{I}}(\psi,\mathsf{tr}_1) \\ \wedge\, y = 0 & \rho \leftarrow \{0,1\}^\lambda \\ \wedge\, b = 1 & (\cdot,\cdot,(\cdot,\cdot,m_2)) \xleftarrow{\mathsf{tr}_2} \tilde{\mathcal{P}}^f_{i\to t}(\mathsf{aux}_{\tilde{\mathcal{P}}},\rho) \\ \wedge\, \psi = \mathbf{I}^{f}_{\text{SP}}(C) & \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C,\mathbb{x},m_1,\mathsf{tr}_1\cup\mathsf{tr}_2) \\ & b \leftarrow \mathbf{V}^{f}_{\text{SP}}(\psi,\mathbb{x},m_1,\rho,m_2) \\ & y \leftarrow C^{f}(\mathbb{x},\mathbb{w}) \end{array}\right].$$

Finally, the only thing left is that we cannot compute the random variable i. However, by sampling $j \leftarrow [t]$, we choose $j = i$ with probability t. This is exactly what $\mathbf{P}(\tilde{\mathcal{P}})$ does. Thus, the above probability is bounded from above by:

$$t\cdot\Pr\left[\begin{array}{l|l} & f \leftarrow \mathcal{U}(\lambda) \\ & j \leftarrow [t] \\ & ((\tau,\psi,\mathbb{x},m_1,s),\mathsf{aux}_{\tilde{\mathcal{P}}}) \xleftarrow{\mathsf{tr}_1} \tilde{\mathcal{P}}^f_{0\to j} \\ C \in \mathcal{C} & C \leftarrow \mathbf{E}_{\text{I}}(\psi,\mathsf{tr}_1) \\ \wedge\, y = 0 & \rho \leftarrow \{0,1\}^\lambda \\ \wedge\, b = 1 & (\cdot,\cdot,(\cdot,\cdot,m_2)) \xleftarrow{\mathsf{tr}_2} \tilde{\mathcal{P}}^f_{j\to t}(\mathsf{aux}_{\tilde{\mathcal{P}}},\rho) \\ \wedge\, \psi = \mathbf{I}^{f}_{\text{SP}}(C) & \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C,\mathbb{x},m_1,\mathsf{tr}_1\cup\mathsf{tr}_2) \\ & b \leftarrow \mathbf{V}^{f}_{\text{SP}}(\psi,\mathbb{x},m_1,\rho,m_2) \\ & y \leftarrow C^{f}(\mathbb{x},\mathbb{w}) \end{array}\right].$$

Finally, we complete the proof by observing that, by definition of $\mathbf{P}(\tilde{\mathcal{P}})$, the above probability is equal:

$$t \cdot \Pr\left[\begin{array}{l} C \in \mathcal{C} \\ \wedge\ y = 0 \\ \wedge\ b = 1 \end{array} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ ((C, \mathbb{x}, m_1), \mathsf{aux}) \xleftarrow{\mathsf{tr}_1} \mathbf{P}^f(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0,1\}^\lambda \\ m_2 \xleftarrow{\mathsf{tr}_2} \mathbf{P}^f(\tilde{\mathcal{P}}, \mathsf{aux}, \rho) \\ \psi \leftarrow \mathbf{I}^f_{\text{SP}}(C) \\ \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_1 \cup \mathsf{tr}_2) \\ b \leftarrow \mathbf{V}^f_{\text{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) \\ y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array}\right].$$

□

We now prove that E_{POW} occurs with small probability.

Claim 5.11. $\Pr[\mathsf{E}_{\text{POW}}] \leq t \cdot \varepsilon_{\text{POW}}(\lambda, \ell, t, t_2)$.

Proof. Out goal in this proof is to reduce the probability that E_{POW} occurs to an attacker for the proof-of-work experiment (up to a t factor). We begin by opening up the expression E_{POW}, and removing from the probability statement and experiment any parts that will not be crucial. Note that by removing requirements from the probability statement we can only increase the probability. Thus,

$$\Pr[\mathsf{E}_{\text{POW}}] \leq \Pr\left[\begin{array}{l} C \in \mathcal{C} \\ \wedge\ b_{\text{POW}} = 1 \\ (\tau, \psi, \mathbb{x}, m_1, s) \in \mathsf{tr}_{\mathbf{I}} \cup \mathsf{tr}_C \end{array} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (C, \mathbb{x}, (m_1, s, m_2)) \xleftarrow{\mathsf{tr}_{\mathcal{P}}} \tilde{\mathcal{P}}^f \\ \psi \xleftarrow{\mathsf{tr}_{\mathbf{I}}} \mathbf{I}^f_{\text{SP}}(C) \\ z \leftarrow f(\psi, \mathbb{x}, m_1) \\ \tau \leftarrow \mathsf{Padder}^f(\psi, \mathbb{x}, m_1, s) \\ \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_{\mathcal{P}}) \\ b_{\text{POW}} \leftarrow \mathsf{Check}^f(z, s) \\ \bot \xleftarrow{\mathsf{tr}_C} C^f(\mathbb{x}, \mathbb{w}) \end{array}\right]$$

Recall that in the proof-of-work security game, the adversary has two stages, where the puzzle is sampled in between them. Looking forward, the preprocessing stage of the adversary will run $\tilde{\mathcal{P}}$ up to the query $(\psi, \mathbb{x}, \rho)$, and the solution-finding stage will be to run ψ and C (since if they make a query that includes a valid solution, it will braek the proof-of-work z). Recall that since $\tilde{\mathcal{P}}$ is admissible, specifically by Item 1, such a query always exists in the trace of $\tilde{\mathcal{P}}$. Towards this end, we split the prover into two.

Formally, let $i \in [t]$ be the random variable representing the index of the query $(\psi, \mathbb{x}, m_1)$. Then the above probability can be rewritten as:

$$\Pr\left[\begin{array}{l|l} C \in \mathcal{C} & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, y = 0 & ((\psi, \mathbb{x}, m_1), \mathsf{aux}_{\tilde{\mathcal{P}}}) \xleftarrow{\mathsf{tr}_1} \tilde{\mathcal{P}}^f_{0\to i} \\ \wedge\, b_{\text{POW}} = 1 & z \leftarrow f(\psi, \mathbb{x}, m_1) \\ (\tau, \psi, \mathbb{x}, m_1, s) \in \mathsf{tr}_{\mathbf{I}} \cup \mathsf{tr}_C & (C, \cdot, (\cdot, s, m_2)) \xleftarrow{\mathsf{tr}_2} \tilde{\mathcal{P}}^f_{i\to t}(\mathsf{aux}_{\tilde{\mathcal{P}}}, z) \\ & (\cdot) \xleftarrow{\mathsf{tr}_{\mathbf{I}}} \mathbf{I}^f_{\text{SP}}(C) \\ & \tau \leftarrow \mathsf{Padder}^f(\psi, \mathbb{x}, m_1, s) \\ & \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_1 \cup \mathsf{tr}_2) \\ & b_{\text{POW}} \leftarrow \mathsf{Check}^f(z, s) \\ & y \xleftarrow{\mathsf{tr}_C} C^f(\mathbb{x}, \mathbb{w}) \end{array}\right]$$

In order to launch our attack on the proof-of-work, we will need to run the circuit C on $\mathbb{w}$, but since we will only want to run $\tilde{\mathcal{P}}$ up to query i (and not until the end), we only have ψ. Thus, we will add the index extractor to convert ψ into C and the witness extractor to generate $\mathbb{w}$. By Item 2, these extractors must succeed. Thus, the above probability is equal:

$$\Pr\left[\begin{array}{l|l} C \in \mathcal{C} & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, y = 0 & ((\psi, \mathbb{x}, m_1), \mathsf{aux}_{\tilde{\mathcal{P}}}) \xleftarrow{\mathsf{tr}_1} \tilde{\mathcal{P}}^f_{0\to i} \\ \wedge\, b_{\text{POW}} = 1 & C \leftarrow \mathbf{E}_{\text{I}}(\psi, \mathsf{tr}_1) \\ (\tau, \psi, \mathbb{x}, m_1, s) \in \mathsf{tr}_{\mathbf{I}} \cup \mathsf{tr}_C & \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_1) \\ & z \leftarrow f(\psi, \mathbb{x}, m_1) \\ & (\cdot, \cdot, (\cdot, s, m_2)) \xleftarrow{\mathsf{tr}_2} \tilde{\mathcal{P}}^f_{i\to t}(\mathsf{aux}_{\tilde{\mathcal{P}}}, z) \\ & (\cdot) \xleftarrow{\mathsf{tr}_{\mathbf{I}}} \mathbf{I}^f_{\text{SP}}(C) \\ & \tau \leftarrow \mathsf{Padder}^f(\psi, \mathbb{x}, m_1, s) \\ & b_{\text{POW}} \leftarrow \mathsf{Check}^f(z, s) \\ & y \xleftarrow{\mathsf{tr}_C} C^f(\mathbb{x}, \mathbb{w}) \end{array}\right]$$

We now switch to sampling z from the uniform distribution rather than as an output of f. By assumption, $\mathsf{Check}^f(z, s)$ does not query f at $(\psi, \mathbb{x}, m_1)$ (see the beginning of proof of the Lemma), and so, rather than giving it access to f programmed at $(\psi, \mathbb{x}, m_1)$ it remains with access to f.

$$\Pr\left[\begin{array}{l|l} C \in \mathcal{C} & f \leftarrow \mathcal{U}(\lambda) \\ \wedge\, y = 0 & ((\psi, \mathbb{x}, m_1), \mathsf{aux}_{\tilde{\mathcal{P}}}) \xleftarrow{\mathsf{tr}_1} \tilde{\mathcal{P}}^f_{0\to i} \\ \wedge\, b_{\text{POW}} = 1 & C \leftarrow \mathbf{E}_{\text{I}}(\psi, \mathsf{tr}_1) \\ (\tau, \psi, \mathbb{x}, m_1, s) \in \mathsf{tr}_{\mathbf{I}} \cup \mathsf{tr}_C & \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_1) \\ & z \leftarrow \{0,1\}^\lambda \\ & \mu = \{((\psi, \mathbb{x}, m_1), z)\} \\ & (\cdot, \cdot, (\cdot, s, m_2)) \xleftarrow{\mathsf{tr}_2} \tilde{\mathcal{P}}^f_{i\to t}(\mathsf{aux}_{\tilde{\mathcal{P}}}, z) \\ & (\cdot) \xleftarrow{\mathsf{tr}_{\mathbf{I}}} \mathbf{I}^{f[\mu]}_{\text{SP}}(C) \\ & \tau \leftarrow \mathsf{Padder}^{f[\mu]}(\psi, \mathbb{x}, m_1, s) \\ & b_{\text{POW}} \leftarrow \mathsf{Check}^f(z, s) \\ & y \xleftarrow{\mathsf{tr}_C} C^{f[\mu]}(\mathbb{x}, \mathbb{w}) \end{array}\right]$$

The above is bounded by the probability that there is some $(\tau', \psi', \mathbb{x}', m_1', s') \in \mathsf{tr}_{\mathbf{I}} \cup \mathsf{tr}_C$ where s' is a valid solution to z. In other words, the above is bounded by the probability:

$$\Pr\left[\begin{array}{l|l} & f \leftarrow \mathcal{U}(\lambda) \\ & ((\psi, \mathbb{x}, m_1), \mathsf{aux}_{\tilde{\mathcal{P}}}) \xleftarrow{\mathsf{tr}_1} \tilde{\mathcal{P}}^f_{0\to i} \\ & C \leftarrow \mathbf{E}_{\mathrm{I}}(\psi, \mathsf{tr}_1) \\ C \in \mathcal{C} & \mathbb{w} \leftarrow \mathbf{E}_{\mathrm{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_1) \\ \wedge\, y = 0 & z \leftarrow \{0,1\}^\lambda \\ \wedge\, b_{\mathrm{POW}} = 1 & \mu = \{((\psi, \mathbb{x}, m_1), z)\} \\ \exists(\tau', \psi', \mathbb{x}', m_1', s') \in \mathsf{tr}_{\mathbf{I}} \cup \mathsf{tr}_C & (\cdot, \cdot, (\cdot, s, m_2)) \xleftarrow{\mathsf{tr}_2} \tilde{\mathcal{P}}^f_{i\to t}(\mathsf{aux}_{\tilde{\mathcal{P}}}, z) \\ \wedge\, \mathsf{Check}(z, s') = 1 & (\cdot) \xleftarrow{\mathsf{tr}_{\mathbf{I}}} \mathbf{I}^{f[\mu]}_{\mathrm{SP}}(C) \\ & \tau \leftarrow \mathsf{Padder}^{f[\mu]}(\psi, \mathbb{x}, m_1, s) \\ & b_{\mathrm{POW}} \leftarrow \mathsf{Check}^f(z, s) \\ & y \xleftarrow{\mathsf{tr}_C} C^{f[\mu]}(\mathbb{x}, \mathbb{w}) \end{array}\right]$$

$$\leq \Pr\left[\begin{array}{l|l} & f \leftarrow \mathcal{U}(\lambda) \\ & ((\psi, \mathbb{x}, m_1), \mathsf{aux}_{\tilde{\mathcal{P}}}) \xleftarrow{\mathsf{tr}_1} \tilde{\mathcal{P}}^f_{0\to i} \\ C \in \mathcal{C} & C \leftarrow \mathbf{E}_{\mathrm{I}}(\psi, \mathsf{tr}_1) \\ \wedge\, \exists(\tau', \psi', \mathbb{x}', m_1', s') \in \mathsf{tr}_{\mathbf{I}} \cup \mathsf{tr}_C & \mathbb{w} \leftarrow \mathbf{E}_{\mathrm{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_1) \\ \mathsf{Check}^f(z, s') = 1 & z \leftarrow \{0,1\}^\lambda \\ & \mu = \{((\psi, \mathbb{x}, m_1), z)\} \\ & (\cdot) \xleftarrow{\mathsf{tr}_{\mathbf{I}}} \mathbf{I}^{f[\mu]}_{\mathrm{SP}}(C) \\ & y \xleftarrow{\mathsf{tr}_C} C^{f[\mu]}(\mathbb{x}, \mathbb{w}) \end{array}\right].$$

Now, rather than using the random variable i, we guess it at random by choosing $j \leftarrow [t]$ uniformly at random. Since $i = j$ with probability $1/t$, the above is bounded by:

$$t \cdot \Pr\left[\begin{array}{l|l} & f \leftarrow \mathcal{U}(\lambda) \\ & j \leftarrow [t] \\ & ((\psi, \mathbb{x}, m_1), \mathsf{aux}_{\tilde{\mathcal{P}}}) \xleftarrow{\mathsf{tr}_1} \tilde{\mathcal{P}}^f_{0\to j} \\ C \in \mathcal{C} & C \leftarrow \mathbf{E}_{\mathrm{I}}(\psi, \mathsf{tr}_1) \\ \wedge\, \exists(\tau', \psi', \mathbb{x}', m_1', s') \in \mathsf{tr}_{\mathbf{I}} \cup \mathsf{tr}_C & \mathbb{w} \leftarrow \mathbf{E}_{\mathrm{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_1) \\ \mathsf{Check}^f(z, s') = 1 & z \leftarrow \{0,1\}^\lambda \\ & \mu = \{((\psi, \mathbb{x}, m_1), z)\} \\ & (\cdot) \xleftarrow{\mathsf{tr}_{\mathbf{I}}} \mathbf{I}^{f[\mu]}_{\mathrm{SP}}(C) \\ & y \xleftarrow{\mathsf{tr}_C} C^{f[\mu]}(\mathbb{x}, \mathbb{w}) \end{array}\right]$$

We now use the soundness of the proof-of-work to bound the above probability by defining an algorithm $\mathbf{A} = (\mathbf{A}_1, \mathbf{A}_2)$ with oracle access to f for solving puzzles:

$\mathbf{A}^f_1$:

1. Sample $j \leftarrow [t]$.
2. Execute $((\psi, \mathbb{x}, m_1), \mathsf{aux}_{\tilde{\mathcal{P}}}) \xleftarrow{\mathsf{tr}_1} \tilde{\mathcal{P}}^f_{0\to j}$ (if the j-th query cannot be parsed in this way then abort).
3. Compute $C \leftarrow \mathbf{E}_{\mathrm{I}}(\psi, \mathsf{tr}_1)$.
4. Compute $\mathbb{w} \leftarrow \mathbf{E}_{\mathrm{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_1)$.
5. Output $\mathsf{aux} = (C, \psi, \mathbb{x}, m_1, \mathbb{w})$.

Then, given a randomly sampled puzzle $z \leftarrow \{0,1\}^\lambda$, the algorithm $\mathbf{A}_2$ solves the puzzle z as follows:

$\mathbf{A}_2^f(\mathsf{aux} = (C, \psi, \mathbb{x}, m_1, \mathbb{w}), z)$:

1. Define $\mu = \{((\psi, \mathbb{x}, m_1), z)\}$.
2. Execute $(\cdot) \xleftarrow{\mathsf{tr}_\mathbf{I}} \mathbf{I}_{\text{SP}}^{f[\mu]}(C)$ and $(\cdot) \xleftarrow{\mathsf{tr}_C} C^{f[\mu]}(\mathbb{x}, \mathbb{w})$ (if C makes more than $\mathbf{q}_C$ queries, then stop it before it makes the next query, and output $\bot$).
3. For each query x in $\mathsf{tr}_\mathbf{I} \cup \mathsf{tr}_C$ do:
   (a) Parse x as $(\tau', \psi', \mathbb{x}', m_1', s')$ (if it cannot be parsed then move to next query).
   (b) If $\mathsf{Check}^f(\ell, z, s') = 1$ then output s' (and halt).
4. Output $\bot$.

Observe that the query complexity of $\mathbf{A}_1$ is at most t, and the query complexity of $\mathbf{A}_2$ is at most $t_2 := \mathbf{q}_{\text{POW}} \cdot (\mathbf{q}_C + \mathbf{q}_\text{I})$. Finally, by definition of $\mathbf{A}$, and by soundness of the proof-of-work we have that the probability statement prior to the algorithm definition is bounded by:

$$t \cdot \Pr\left[\mathsf{Check}^f(\ell, z, s) = 1 \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ \mathsf{aux} \leftarrow \mathbf{A}_1^f \\ z \leftarrow \{0,1\}^\lambda \\ s \leftarrow \mathbf{A}_2^f(\mathsf{aux}, z) \end{array} \right] \leq t \cdot \varepsilon_{\text{POW}}(\lambda, \ell, t, t_2)\,.$$

□

Finally, we bound the probability that $\mathbf{V}_{\text{SP}}^f(\psi, \mathbb{x}, m_1, \rho, m_2)$ makes the query $(\tau, \psi, \mathbb{x}, m_1, s)$:

Claim 5.12. $\Pr[\mathsf{E}_{\text{PAD}}] \leq \varepsilon_{\text{PAD}}(\lambda, t_1)$.

Proof. Consider the following adversary $\mathbf{A}$ for the prefix-avoidance experiment,

1. Run $(C, \mathbb{x}, (m_1, s, m_2)) \xleftarrow{\mathsf{tr}_\mathcal{P}} \tilde{\mathcal{P}}^f$.
2. $\psi \leftarrow \mathbf{I}_{\text{SP}}^f(C)$.
3. $\tau \leftarrow \mathsf{Padder}^f(\psi, \mathbb{x}, m_1, s)$.
4. $\rho \leftarrow f(\tau, \psi, \mathbb{x}, m_1, s)$.
5. Output $\gamma = (\psi, \mathbb{x}, m_1, s)$ and $m = (\psi, \mathbb{x}, m_1, \rho, m_2)$.

And the following machine $\mathbf{M}^f$,

1. Given $m = (\psi, \mathbb{x}, m_1, \rho, m_2)$.
2. Run $\mathbf{V}_{\text{SP}}^f(\psi, \mathbb{x}, m_1, \rho, m_2)$.

Observe that $\mathbf{A}$ runs in time $t_1 = t + \mathbf{q}_\text{I} + \mathbf{q}_{\text{PAD}} + 1$. Moreover, whenever $(\tau, \psi, \mathbb{x}, m_1, s) \in \mathsf{tr}_\mathbf{V} \cup \mathsf{tr}_{\text{PAD}}$, it holds that (τ, γ) is in the combined trace of Padder and $\mathbf{M}$. Thus, by prefix-avoidance,

$$\Pr[\mathsf{E}_{\text{PAD}}] \leq \Pr\left[(\tau, \psi, \mathbb{x}, m_1, s) \in \mathsf{tr}_{\mathbf{V}} \cup \mathsf{tr}_{\text{PAD}} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (C, \mathbb{x}, (m_1, s, m_2)) \leftarrow \tilde{\mathcal{P}}^f \\ \psi \leftarrow \mathbf{I}^f_{\text{SP}}(C) \\ \tau \xleftarrow{\mathsf{tr}_{\text{PAD}}} \mathsf{Padder}^f(\psi, \mathbb{x}, m_1, s) \\ \rho \leftarrow f(\tau, \psi, \mathbb{x}, m_1, s) \\ \bot \xleftarrow{\mathsf{tr}_{\mathbf{V}}} \mathbf{V}^f_{\text{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) \end{array} \right]$$

$$= \Pr\left[(\tau, \gamma) \in \mathsf{tr}_{\mathbf{V}} \cup \mathsf{tr}_{\text{PAD}} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (\gamma, m) \leftarrow \mathbf{A}^f \\ \tau \xleftarrow{\mathsf{tr}_{\text{PAD}}} \mathsf{Padder}^f(\gamma) \\ b \xleftarrow{\mathsf{tr}_{\mathbf{M}}} \mathbf{M}^f(m) \end{array} \right] < \varepsilon_{\text{PAD}}(\lambda, t_1)\,.$$

□

□

We can now conclude the theorem for all adversaries.

Proof of Theorem 5.6. For every query bound $t \in \mathbb{N}$, malicious t-query argument prover $\tilde{\mathcal{P}}$, we show that

$$\Pr\left[\begin{array}{l} C \in \mathcal{C} \\ \wedge\, y = 0 \\ \wedge\, b = 1 \end{array} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (C, \mathbb{x}, \pi) \xleftarrow{\mathsf{tr}_{\mathcal{P}}} \tilde{\mathcal{P}}^f \\ \psi \leftarrow \mathcal{I}^f(C) \\ \mathbb{w} \leftarrow \mathcal{E}(C, \mathbb{x}, \mathsf{tr}_{\mathcal{P}}) \\ b \leftarrow \mathcal{V}^f(\psi, \mathbb{x}, \pi) \\ y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array} \right]$$

$$= \Pr\left[\begin{array}{l} C \in \mathcal{C} \\ \wedge\, y = 0 \\ \wedge\, b = 1 \end{array} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (C, \mathbb{x}, \pi = (m_1, s, m_2)) \xleftarrow{\mathsf{tr}_{\mathcal{P}}} \tilde{\mathcal{P}}^f \\ \psi \leftarrow \mathcal{I}^f(C) \\ \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_{\mathcal{P}}) \\ b \leftarrow \mathcal{V}^f(\psi, \mathbb{x}, \pi) \\ y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array} \right]$$

$$\leq t \cdot \Pr\left[\begin{array}{l} C \in \mathcal{C} \\ \wedge\, y = 0 \\ \wedge\, b = 1 \end{array} \;\middle|\; \begin{array}{l} f \leftarrow \mathcal{U}(\lambda) \\ (C, \mathbb{x}, m_1, \mathsf{aux}) \xleftarrow{\mathsf{tr}} \mathbf{P}^f(\tilde{\mathcal{P}}) \\ \rho \leftarrow \{0,1\}^\lambda \\ m_2 \leftarrow \mathbf{P}^f(\mathsf{aux}, \rho) \\ \mathbb{w} \leftarrow \mathbf{E}_{\text{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}) \\ \psi \leftarrow \mathbf{I}^f_{\text{SP}}(C) \\ b \leftarrow \mathbf{V}^f_{\text{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) \\ y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array} \right]$$

$$+ \varepsilon_{\text{PAD}}(\lambda, t_1) + t \cdot \varepsilon_{\text{POW}}(\lambda, \ell, t, t_2) + 2\kappa_{\text{I}}(\lambda, t_1) + \kappa_{\text{SP}}(\lambda, t) + \varepsilon_{\text{POW}}(\lambda, \ell, t_1, 0)\,.$$

Above, $t_1 := t + \mathbf{q}_{\text{I}} + \mathbf{q}_{\text{PAD}} + 1$ and $t_2 := \mathbf{q}_{\text{POW}} \cdot (\mathbf{q}_C + \mathbf{q}_{\text{I}})$. Define the event E that $\tilde{\mathcal{P}}$ acts in an admissible manner. I.e., whenever $\tilde{\mathcal{P}}$ outputs $(C, \mathbb{x}, \pi = (m_1, s, m_2))$:

- The following queries are contained in its trace: $(\psi, \mathbb{x}, m_1)$, and $(\tau, \psi, \mathbb{x}, m_1, s)$ where $\psi \leftarrow \mathbf{I}^f_{\mathrm{SP}}(C)$ and $\tau \leftarrow \mathsf{Padder}^f(\psi, \mathbb{x}, m_1, s)$,
- Letting tr be the trace up to the query $(\psi, \mathbb{x}, m_1)$, and $\mathsf{tr}_{\mathcal{P}}$ be the full trace of $\tilde{\mathcal{P}}$, it holds that $C = \mathbf{E}_{\mathrm{I}}(\psi, \mathsf{tr})$, and $\mathbf{E}_{\mathrm{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}) = \mathbf{E}_{\mathrm{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_{\mathcal{P}})$, and
- Letting tr be the trace up to the query $(\tau, \psi, \mathbb{x}, m_1, s)$, it holds that $C = \mathbf{E}_{\mathrm{I}}(\psi, \mathsf{tr})$.

We begin with the left-hand side. By the total probability theorem, the left-hand probability equals

$$\Pr\left[\begin{array}{l|l} & f \leftarrow \mathcal{U}(\lambda) \\ C \in \mathcal{C} & (C, \mathbb{x}, \pi = (m_1, s, m_2)) \xleftarrow{\mathsf{tr}_{\mathcal{P}}} \tilde{\mathcal{P}}^f \\ \wedge\, y = 0 & \psi \leftarrow \mathcal{I}^f(C) \\ \wedge\, b = 1 & \mathbb{w} \leftarrow \mathbf{E}_{\mathrm{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_{\mathcal{P}}) \\ \wedge\, \mathsf{E} & b \leftarrow \mathcal{V}^f(\psi, \mathbb{x}, \pi) \\ & y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array}\right] + \Pr\left[\begin{array}{l|l} & f \leftarrow \mathcal{U}(\lambda) \\ C \in \mathcal{C} & (C, \mathbb{x}, \pi = (m_1, s, m_2)) \xleftarrow{\mathsf{tr}_{\mathcal{P}}} \tilde{\mathcal{P}}^f \\ \wedge\, y = 0 & \psi \leftarrow \mathcal{I}^f(C) \\ \wedge\, b = 1 & \mathbb{w} \leftarrow \mathbf{E}_{\mathrm{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_{\mathcal{P}}) \\ \wedge\, \overline{\mathsf{E}} & b \leftarrow \mathcal{V}^f(\psi, \mathbb{x}, \pi) \\ & y \leftarrow C^f(\mathbb{x}, \mathbb{w}) \end{array}\right]$$

The fist term is bounded by Lemma 5.7, since when the event E occurs then the adversary behaves in an admissible manner. We focus on bounding the second term. There are two cases. Let the output of the adversary be denoted by $(C, \mathbb{x}, \pi = (m_1, s, m_2))$.

Case 1. Assume that the query $(\psi, \mathbb{x}, m_1)$ is not contained the trace $\mathsf{tr}_{\mathcal{P}}$. For the argument verifier to accept, it must be that s is a valid solution to the puzzle $z = f(\psi, \mathbb{x}, m_1)$. However, the puzzle z has not been sampled yet, while the solution in already fixed in the preprocessing phase. Thus, from the soundness of the puzzle (with a preprocessing adversary that simulates the above and output s as the auxiliary state, and a second adversary that simply outputs s while making 0 queries), we get that

$$\Pr[\mathsf{Check}^f(\ell, z, s) = 1] \leq \varepsilon_{\mathrm{POW}}(\lambda, \ell, t_1, 0)\,.$$

Case 2. Assume that the query $(\tau, \psi, \mathbb{x}, m_1, s)$ is not contained in the trace $\mathsf{tr}_{\mathcal{P}}$. In this case, the SP response message is fixed, while the SP challenge $\rho = f(\tau, \psi, \mathbb{x}, m_1, s)$ has not been sampled yet. The argument verifier checks that $\mathbf{V}^f_{\mathrm{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) = 1$. Thus a malicious prover can be easily converted to an SP malicious prover, simply by simulating to to the end before the challenge phase. Therefore, the term is bounded by

$$\Pr\left[\begin{array}{l} C \in \mathcal{C} \\ \wedge\, C^f(\mathbb{x}, \mathbb{w}) = 0 \\ \wedge\, \mathbf{V}^f_{\mathrm{SP}}(\psi, \mathbb{x}, m_1, \rho, m_2) = 1 \end{array}\right] \leq \kappa_{\mathrm{SP}}(\lambda, t)\,.$$

Case 3. Both $(\psi, \mathbb{x}, m_1)$ and $(\tau, \psi, \mathbb{x}, m_1, s)$ are contained in the trace, but one of the following holds, where tr is the trace up to the query $(\psi, \mathbb{x}, m_1)$, tr' is the trace up to $(\tau, \psi, \mathbb{x}, m_1, s)$, and $\mathsf{tr}_{\mathcal{P}}$ be the full trace of $\tilde{\mathcal{P}}$:
- $C \neq \mathbf{E}_{\mathrm{I}}(\psi, \mathsf{tr})$, or
- $C \neq \mathbf{E}_{\mathrm{I}}(\psi, \mathsf{tr}')$, or
- $\mathbf{E}_{\mathrm{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}) \neq \mathbf{E}_{\mathrm{SP}}(C, \mathbb{x}, m_1, \mathsf{tr}_{\mathcal{P}})$.

The first two items occur with probability bounded by $\kappa_{\mathrm{I}}(\lambda, t)$ by indexer knowledge of the Sigma protocol (for a total of $2\kappa_{\mathrm{I}}(\lambda, t)$), and the final item is bounded by $\kappa_{\mathrm{SP}}(\lambda, t)$ by knowledge soundness of the Sigma protocol.

Overall, we get that

$$\begin{aligned}\kappa_{\text{ARG}}(\lambda,t) &\leq (t\cdot\kappa_{\text{SP}}(\lambda,t)+t\cdot\varepsilon_{\text{POW}}(\lambda,\ell,t,t_2)+\varepsilon_{\text{PAD}}(\lambda,t_1))+(2\kappa_{\text{I}}(\lambda,t_1)+\kappa_{\text{SP}}(\lambda,t)+\varepsilon_{\text{POW}}(\lambda,\ell,t_1,0))\\ &\leq (t+1)\cdot\kappa_{\text{SP}}(\lambda,t)+(t+1)\cdot\varepsilon_{\text{POW}}(\lambda,\ell,t,t_2)+2\kappa_{\text{I}}(\lambda,t_1)+\varepsilon_{\text{PAD}}(\lambda,t_1)\,.\end{aligned}$$

□

6 Strong proofs of work in the ROM

We define strong proofs of work in the random oracle model.

Definition 6.1 (Strong PoW in the ROM)**.** *A pair of algorithms* (Solve, Check) *is strong* proof-of-work *in the random oracle model if the following properties hold:*

- Completeness. *A PoW scheme is perfectly complete if for every* $\ell, \lambda \in \mathbb{N}$*,*

$$\Pr\left[\mathsf{Check}^f(\ell, z, s) = 1 \;\middle|\; \begin{array}{r} f \leftarrow \mathcal{U}(\lambda) \\ z \leftarrow \{0,1\}^\lambda \\ s \leftarrow \mathsf{Solve}^f(\ell, z) \end{array}\right] = 1\,.$$

- Soundness. *A PoW scheme has soundness error* ε_{POW} *if for every* $\lambda, \ell, t_1, t_2 \in \mathbb{N}$*, and adversary* $\mathbf{A} = (\mathbf{A}_1, \mathbf{A}_2)$ *such that* $\mathbf{A}_1$ *is a* t_1*-query algorithm that outputs* t_2 *bits of auxiliary state and* $\mathbf{A}_2$ *is a* t_2*-query algorithm:*

$$\Pr\left[\mathsf{Check}^f(\ell, z, s) = 1 \;\middle|\; \begin{array}{r} f \leftarrow \mathcal{U}(\lambda) \\ \mathsf{aux} \leftarrow \mathbf{A}_1^f \\ z \leftarrow \{0,1\}^\lambda \\ s \leftarrow \mathbf{A}_2^f(\mathsf{aux}, z) \end{array}\right] \leq \varepsilon_{\text{POW}}(\lambda, \ell, t_1, t_2)\,.$$

- Query bound. *For every* $\lambda, \ell \in \mathbb{N}$ *and* f *in the image of* $\mathcal{U}(\lambda)$*,* $\mathsf{Solve}^f(\ell, z)$ *makes at most* ℓ *queries to* f*.*

One can observe, that constructions of our notion of proof-of-work follow from various works including work on *sequential* proofs of work (e.g., [MMV13; CP18; DLM19]). However, our notion is weaker. In the next section we give a simple and practically efficient construction that satisfies the above definition.

6.1 A simple construction in the ROM

In this section, we give a simple construction of a strong proof-of-work with small soundness error. This shows a practical way to implement the proof-of-work required for the XFS transformation. See Section 2.5 for an overview of the construction and its proof.

For the construction, we use the notation of a Merkle tree commitment scheme (in the ROM), as defined in [CY24, Chapter 18]. Specifically, a Merkle commitment scheme has three algorithms, MT = (MT.Commit, MT.Open, MT.Check), described as follows. The algorithms receive query access to a random oracle $f \in \mathcal{U}(\lambda)$ and have the following syntax.

- *Commit.* The algorithm MT.Commit receives as input a message vector $\mathbf{m} \in \Sigma^\ell$, and computes a Merkle commitment $\mathsf{rt} \in \{0,1\}^\lambda$ and corresponding opening trapdoor $\mathsf{td} \in \{0,1\}^{O(\lambda \cdot \ell)}$. This algorithm is deterministic.

- *Open.* The algorithm MT.Open receives as input opening trapdoor td and a subset $I \subseteq [\ell]$, and computes an opening proof pf that authenticates the values at the locations in I.

- *Check.* The algorithm MT.Check receives as input a Merkle commitment rt, subset $I \subseteq [\ell]$, claimed values $\mathbf{a} \in \Sigma^I$, and opening proof pf, and computes a bit indicating whether the opening proof pf authenticates $\mathbf{a}$ as values for the locations in I with respect to rt.

Construction 6.2 (Strong PoW)**.** Let $\mathsf{MT} = (\mathsf{MT.Commit}, \mathsf{MT.Open}, \mathsf{MT.Check})$ be a Merkle tree as defined in [CY24, Chapter 18]. Let $n = \ell/2$. We build a strong proof-of-work $(\mathsf{Solve}, \mathsf{Check})$ as follows.

$\mathsf{Solve}^f(\ell, z)$:

1. Compute the message $\mathbf{m}$ of length n, where $\mathbf{m}[i] = (z, i)$.
2. Compute the Merkle commitment: $(\mathsf{rt}, \mathsf{td}) \leftarrow \mathsf{MT.Commit}^f(\mathbf{m})$.
3. Compute $\rho \leftarrow f(z, \mathsf{rt})$, and interpret ρ as a subset $I \subseteq [n]$ of size λ.
4. Compute $\mathsf{pf} \leftarrow \mathsf{MT.Open}^f(\mathsf{td}, I)$.
5. Output $s = (\mathsf{rt}, \mathsf{pf})$.

$\mathsf{Check}^f(\ell, z, s)$:

1. Parse $s = (\mathsf{rt}, \mathsf{pf})$ (if it cannot be parsed then reject).
2. Compute $\rho \leftarrow f(z, \mathsf{rt})$, and interpret ρ as a subset $I \subseteq [n]$ of size λ.
3. Set $\mathbf{m}[I]$ such that $\mathbf{m}[i] = (z, i)$ for all $i \in I$.
4. Verify that $\mathsf{MT.Check}^f(\mathsf{rt}, I, \mathbf{m}[I], \mathsf{pf}) = 1$.

Observe that a solution to a puzzle in this construction has length $n \cdot \log m$ bits.

Theorem 6.3. *For any $t_2 \leq \ell/4$, the proof-of-work above has soundness error*

$$\varepsilon_{\text{POW}}(\lambda, \ell, t_1, t_2) \leq \frac{t_1 + t_2 + 1}{2^\lambda} .$$

Proof. Consider the trace of queries tr_1 (of length t_1) in the first phase. Since the puzzle z is sampled uniformly at random, the probability that is a (z, i) is in any of queries in tr_1 is at most $\frac{t_1}{2^\lambda}$. We continue the proof condition on this not having any such query.

Consider the leaf queries, that have the form (z, i), for some $i \in [n]$. For every $j \in [t_2]$, let E_j be the event that the j-th query is of the form $I = f(z, \mathsf{rt})$, for which the trace at that time (denoted by tr_j) contains all queries of the form (z, i) for every $i \in I$. The trace can contain at most t_2 such queries. Thus, we get that tr_j contains at most α fraction of the leaves where

$$\alpha := \frac{t_2}{n} \leq \frac{\ell}{4n} \leq \frac{2n}{4n} = \frac{1}{2} .$$

Thus, we get that $\Pr[\mathsf{E}_j] \leq 2^{-\lambda}$. Let $\mathsf{E} := \vee_{j \in [t_2]} \mathsf{E}_j$, then $\Pr[\mathsf{E}] \leq \frac{t_2}{2^\lambda}$.

Finally, if E did not occur, then the algorithm does not contain a valid opening for all locations in I. Thus, the probability of outputting a valid opening (without performing any other queries), is at most 2^λ. Overall, the total probability is bounded by:

$$\varepsilon_{\text{POW}}(\lambda, \ell, t_1, t_2) \leq \frac{t_1}{2^\lambda} + \frac{t_2}{2^{2\lambda}} + \frac{1}{2^\lambda} = \frac{t_1 + t_2 + 1}{2^\lambda} .$$

□

6.2 An optimized construction

An optimization to the above construction was suggested in [Nao25]. We give a formal description of this alternative construction that is practically more efficient, but where the solver algorithm performs ℓ queries in expectation (rather than in worst-case), and the solution length is also measured in expectation. Set n and m be such that $n \cdot m = \ell$ (or $O(\ell)$ if it does not divide). A typical setting would be $n = 2\lambda$.

Construction 6.4 (Practical strong PoW)**.** Let $n, m \in \mathbb{N}$ be parameters. We build a strong proof-of-work $(\mathsf{Solve}, \mathsf{Check})$ as follows.

$\mathsf{Solve}^f(\ell, z)$:

1. Find the shortest set of strings $(s_1, \ldots, s_n)$ such that for every $i \in [n]$, $y_i = f(z, s_i)$ begins with $\log m$ zeros.
2. Output $s = (s_1, \ldots, s_n)$.

$\mathsf{Check}^f(\ell, z, s)$:

1. Parse $s = (s_1, \ldots, s_n)$ (if it cannot be parsed then reject).
2. Verify that $s_1, \ldots, s_n$ are all distinct.
3. Compute $y_i = f(z, i, s_i)$.
4. Verify that for all $i \in [m]$, y_i begins with $\log m$ zeros (and reject otherwise).

Theorem 6.5. *For any $t_2 \le \ell/4 - n$, the proof-of-work above has soundness error*

$$\varepsilon_{\text{POW}}(\lambda, \ell, t_1, t_2) \le \frac{t_1}{2^\lambda} + \frac{1}{2^{0.64n}} \, .$$

The solver has expected query complexity ℓ. Moreover, for any constant $c > 1$, the solver provides a correct solution using $c \cdot \ell$ queries with probability at least $1 - e^{-n \cdot \frac{(c-1)^2}{2c}}$.

A typical setting of the above theorem is to use $n = 1.55\lambda$. In this case, we get that the error is bounded by $\frac{t_1+1}{2^\lambda}$. Moreover, the expected encoding size of a solution is $n \cdot \log m = 1.55\lambda \cdot \log(\ell/1.55\lambda)$. For $\lambda = 256$ and large values of hardness up to 2^{25}, the solution has has expected length $1.55 \cdot 256 \cdot \log(2^{25}/1.55 \cdot 256) < 6500$ bits, which less than 0.8 KiB. With these parameters, the solver will find a solution in time 1.3ℓ, with error approximately 2^{-20}.

Proof of Theorem 6.5. Consider any adversary $\mathbf{A} = (\mathbf{A}_1, \mathbf{A}_2)$. We begin by describing an adversary $\mathbf{A}'_2$ such that $(\mathbf{A}_1, \mathbf{A}'_2)$ success with the same probability. We construct $\mathbf{A}'_2$ to behave like $\mathbf{A}_2$ with the following assumptions:

- $\mathbf{A}'_2$ never performs a duplicate query.
- $\mathbf{A}_2$ verifies its solutions at the end.
- $\mathbf{A}'_2$ performs exactly $t'_2 := \ell/4$ queries.
- $\mathbf{A}'_2$ performs only queries that begin with the given puzzle z.

It is straightforward to construct $\mathbf{A}'_2$. We prevent duplicate queries by storing previous answers. We explicit run the Check at the end of the execution (this takes n queries), and then if needed run arbitrary distinct queries (that begin with z) to get to $\ell/4$ queries. Finally, any query that does not begin with z can be answered randomly (using internal randomness), without affecting the success probability (since Check never uses these queries).

We continue the proof with $(\mathbf{A}_1, \mathbf{A}'_2)$. Consider the trace of queries tr_1 (of length t_1) in the first phase (queries performed by $\mathbf{A}_1$). Let E_1 be the event that there exist $i \in [n]$ and $\gamma \in \{0,1\}^\lambda$ so that $(z, i, \gamma) \in \mathsf{tr}_1$. Since the puzzle z is sampled uniformly at random from $\{0,1\}^\lambda$ after $\mathbf{A}_1$ has run, $\Pr[\mathsf{E}_1] \le \frac{t_1}{2^\lambda}$. We continue the proof conditioned on the event $\overline{\mathsf{E}_1}$.

Consider queries in the second phase (performed by $\mathbf{A}'_2$). For every $j \in [t'_2]$, let X_j be the indicator random variable that the answer to the j-th query begins with $\log m$ zeros. As this query was not issued before (it must begin with z, thus it is not in tr_1 and there are no duplicate queries), we get that $\Pr[X_j = 1] = 1/m$. Let $X = \sum_{j \in [t_2']} X_j$. Notice that these random variables are independent, and that $\mathbb{E}[X] = t_2'/m = \ell/4m = n/4$.

Let E_2 be the event that at the end of the second phase, for every $i \in [n]$, the trace contains a query of form (z, s_i) such that $f(z, s_i)$ begins with $\log m$ zeros. Then, by a Chernoff bound (Lemma 6.6) for $\delta = 3$, we have that

$$\begin{aligned}
\Pr[\mathsf{E}_2] &\leq \Pr[X \geq n] \\
&= \Pr[X \geq (1+\delta)\mathbb{E}[X]] \\
&\leq e^{-\frac{\delta^2}{2+\delta} \cdot \frac{n}{4}} \\
&= e^{-\frac{9n}{20}} \\
&\leq 2^{-0.649n}
\end{aligned}$$

Overall, the total probability is bounded by:

$$\varepsilon_{\text{POW}}(\lambda, \ell, t_1, t_2) \leq \Pr[\mathsf{E}_1] + \Pr[\mathsf{E}_2 | \overline{\mathsf{E}_1}] \leq \frac{t_1}{2^\lambda} + \frac{1}{2^{0.649n}} .$$

The solver algorithm Solve simply queries candidate solutions s in an arbitrary order. It is easy to see that the solver will have a expected runtime of ℓ. We show that the moreover part. Let $t := c \cdot \ell$ be the query complexity Solve. For any subset $I \subseteq [t]$ of size n let E_I be the event that $X_i = 1$ for all $i \in I$. We have that $\mathbb{E}[X] = 1/m \cdot t = c\ell/m = cn$. Then, for $\delta = 1 - 1/c$, we have that $n = (1-\delta)\mathbb{E}[X]$. Using a Chernoff bound (Lemma 6.6), the error of the algorithm is bounded by:

$$\begin{aligned}
\Pr[X < n] &= \Pr[X < (1-\delta)\mathbb{E}[X]] \\
&\leq e^{-\frac{cn\delta^2}{2}} \\
&= e^{-\frac{cn(1-1/c)^2}{2}} \\
&= e^{-n \cdot \frac{(c-1)^2}{2c}} .
\end{aligned}$$

□

Lemma 6.6 (Chernoff bound)**.** *Let $X_1, \ldots, X_n$ be independent binary random variables such that $\Pr[X_i = 1] = p_i$ for every $i \in [n]$. Let $X := \sum_{i \in [n]} X_i$ and $\mu := \mathbb{E}[X] = \sum_{i \in [n]} p_i$. Then,*

- $\Pr[X \geq (1+\delta)\mu] \leq e^{-\frac{\delta^2}{2+\delta} \cdot \mu}$, *for all* $\delta > 0$.
- $\Pr[X \leq (1-\delta) \cdot \mu] \leq e^{-\frac{\delta^2}{2} \cdot \mu}$, *for all* $0 < \delta < 1$.

Acknowledgments

We are grateful to Alessandro Chiesa and Ron Rothblum for their invaluable feedback and insightful discussions, which played a crucial role in refining our model and results. Their perspectives helped us better articulate the assumptions, and scope of our framework, which strengthens the clarity of our work. We thank Moni Naor for suggesting the optimized proof-of-work construction.

Gal Arnon is supported by the European Research Union (ERC, CRYPTOPROOF, 101164375). Eylon Yogev is supported by the Israel Science Foundation (Grant No. 2302/22), European Research Union (ERC, CRYPTOPROOF, 101164375), and by an Alon Young Faculty Fellowship. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[BBHMR19] James Bartusek, Liron Bronfman, Justin Holmgren, Fermi Ma, and Ron D. Rothblum. "On the (In)security of Kilian-Based SNARGs". In: *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*. Ed. by Dennis Hofheinz and Alon Rosen. Vol. 11892. Lecture Notes in Computer Science. Springer, 2019, pp. 522–551.

[BCG24] Annalisa Barbara, Alessandro Chiesa, and Ziyi Guan. "Relativized Succinct Arguments in the ROM Do Not Exist". In: *IACR Cryptol. ePrint Arch.* (2024), p. 728.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive Oracle Proofs". In: *Proceedings of the 14th Theory of Cryptography Conference*. TCC '16-B. 2016, pp. 31–60.

[BKM20] Zvika Brakerski, Venkata Koppula, and Tamer Mour. "NIZK from LPN and Trapdoor Hash via Correlation Intractability for Approximable Relations". In: *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. Lecture Notes in Computer Science. Springer, 2020, pp. 738–767.

[Bar01] Boaz Barak. "How to Go Beyond the Black-Box Simulation Barrier". In: *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*. FOCS '11. 2001, pp. 106–115.

[CCGOS23] Megan Chen, Alessandro Chiesa, Tom Gur, Jack O'Connor, and Nicholas Spooner. "Proof-Carrying Data from Arithmetized Random Oracles". In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14005. Lecture Notes in Computer Science. Springer, 2023, pp. 379–404.

[CCHLRRW19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. "Fiat-Shamir: from practice to theory". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*. Ed. by Moses Charikar and Edith Cohen. ACM, 2019, pp. 1082–1090.

[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*. Ed. by Eyal Kushilevitz and Tal Malkin. Vol. 9562. Lecture Notes in Computer Science. Springer, 2016, pp. 389–415.

[CCRR18] Ran Canetti, Yilei Chen, Leonid Reyzin, and Ron D. Rothblum. "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*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10820. Lecture Notes in Computer Science. Springer, 2018, pp. 91–122.

[CCS22] Megan Chen, Alessandro Chiesa, and Nicholas Spooner. "On Succinct Non-interactive Arguments in Relativized Worlds". In: *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II*. Ed. by Orr Dunkelman and Stefan Dziembowski. Vol. 13276. Lecture Notes in Computer Science. Springer, 2022, pp. 336–366.

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

[CGSY24] Alessandro Chiesa, Ziyi Guan, Shahar Samocha, and Eylon Yogev. "Security Bounds for Proof-Carrying Data from Straightline Extractors". In: *Theory of Cryptography - 22nd International Conference, TCC 2024, Milan, Italy, December 2-6, 2024, Proceedings, Part II*. Ed. by Elette Boyle and Mohammad Mahmoody. Vol. 15365. Lecture Notes in Computer Science. Springer, 2024, pp. 464–496.

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. "SNARGs for $\mathcal{P}$ from LWE". In: *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*. IEEE, 2021, pp. 68–79.

[CP18] Bram Cohen and Krzysztof Pietrzak. "Simple Proofs of Sequential Work". 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 II*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10821. Lecture Notes in Computer Science. Springer, 2018, pp. 451–467.

[CT10] Alessandro Chiesa and Eran Tromer. "Proof-Carrying Data and Hearsay Arguments from Signature Cards". In: *Proceedings of the 1st Symposium on Innovations in Computer Science*. ICS '10. 2010, pp. 310–331.

[CY21a] Alessandro Chiesa and Eylon Yogev. "Subquadratic SNARGs in the Random Oracle Model". In: *Proceedings of the 41st Annual International Cryptology Conference*. CRYPTO '21. 2021, pp. 711–741.

[CY21b] Alessandro Chiesa and Eylon Yogev. "Tight Security Bounds for Micali's SNARGs". In: *Proceedings of the 19th Theory of Cryptography Conference*. TCC '21. 2021, pp. 401–434.

[CY24] Alessandro Chiesa and Eylon Yogev. "Building Cryptographic Proofs from Hash Functions". In: *URL: https://github. com/hash-based-snargs-book* (2024).

[DLM19] Nico Döttling, Russell W. F. Lai, and Giulio Malavolta. "Incremental Proofs of Sequential Work". In: *IACR Cryptol. ePrint Arch.* (2019), p. 650.

[DN92] Cynthia Dwork and Moni Naor. "Pricing via Processing or Combatting Junk Mail". In: *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*. Ed. by Ernest F. Brickell. Vol. 740. Lecture Notes in Computer Science. Springer, 1992, pp. 139–147.

[FS86] Amos Fiat and Adi Shamir. "How to prove yourself: practical solutions to identification and signature problems". In: *Proceedings of the 6th Annual International Cryptology Conference*. CRYPTO '86. 1986, pp. 186–194.

[Fis05] Marc Fischlin. "Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors". In: *Proceedings of the 25th Annual International Cryptology Conference*. CRYPTO '05. 2005, pp. 152–168.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. "On the (In)security of the Fiat-Shamir Paradigm". In: *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*. FOCS '03. 2003, pp. 102–113.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating Computation: Interactive Proofs for Muggles". In: *Journal of the ACM* 62.4 (2015), 27:1–27:64.

[HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. “SNARGs for P from Sub-exponential DDH and QR”. In: *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II*. Ed. by Orr Dunkelman and Stefan Dziembowski. Vol. 13276. Lecture Notes in Computer Science. Springer, 2022, pp. 520–549.

[HL18] Justin Holmgren and Alex Lombardi. “Cryptographic Hashing from Strong One-Way Functions (Or: One-Way Product Functions and Their Applications)”. In: *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*. Ed. by Mikkel Thorup. IEEE Computer Society, 2018, pp. 850–858.

[HLR21] Justin Holmgren, Alex Lombardi, and Ron D. Rothblum. “Fiat-Shamir via list-recoverable codes (or: parallel repetition of GMW is not zero-knowledge)”. In: *STOC ’21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*. Ed. by Samir Khuller and Virginia Vassilevska Williams. ACM, 2021, pp. 750–760.

[JJ21] Abhishek Jain and Zhengzhong Jin. “Non-interactive Zero Knowledge from Sub-exponential DDH”. In: *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. Lecture Notes in Computer Science. Springer, 2021, pp. 3–32.

[KLV23] Yael Tauman Kalai, Alex Lombardi, and Vinod Vaikuntanathan. “SNARGs and PPAD Hardness from the Decisional Diffie-Hellman Assumption”. In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14005. Lecture Notes in Computer Science. Springer, 2023, pp. 470–498.

[KRR17] Yael Tauman Kalai, Guy N. Rothblum, and Ron D. Rothblum. “From Obfuscation to the Security of Fiat-Shamir for Proofs”. In: *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10402. Lecture Notes in Computer Science. Springer, 2017, pp. 224–251.

[KRS25] Dmitry Khovratovich, Ron D Rothblum, and Lev Soukhanov. “How to Prove False Statements: Practical Attacks on Fiat-Shamir”. In: *Cryptology ePrint Archive* (2025).

[Kil92] Joe Kilian. “A note on efficient zero-knowledge proofs and arguments”. In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*. STOC ’92. 1992, pp. 723–732.

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. “Publicly verifiable proofs of sequential work”. In: *Innovations in Theoretical Computer Science, ITCS ’13, Berkeley, CA, USA, January 9-12, 2013*. Ed. by Robert D. Kleinberg. ACM, 2013, pp. 373–388.

[Mic00] Silvio Micali. “Computationally Sound Proofs”. In: *SIAM Journal on Computing* 30.4 (2000). Preliminary version appeared in FOCS ’94., pp. 1253–1298.

[Nao25] Moni Naor. *Private communication*. Private communication. 2025.

[PS19] Chris Peikert and Sina Shiehian. “Noninteractive Zero Knowledge for NP from (Plain) Learning with Errors”. In: *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11692. Lecture Notes in Computer Science. Springer, 2019, pp. 89–114.

[PS96] David Pointcheval and Jacques Stern. “Security Proofs for Signature Schemes”. In: *Proceedings of the 14th Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT ’96. 1996, pp. 387–398.

[Pas03] Rafael Pass. “On Deniability in the Common Reference String and Random Oracle Model”. In: *Proceedings of the 23rd Annual International Cryptology Conference*. CRYPTO ’03. 2003, pp. 316–337.

[Pol24] Polyhedra Network. *Expander*. `https://github.com/PolyhedraZK/Expander`. Accessed: 2025-01-15. 2024.

[RT24] Lior Rotem and Stefano Tessaro. “Straight-Line Knowledge Extraction for Multi-Round Protocols”. In: *IACR Cryptol. ePrint Arch.* (2024), p. 1724.

[WTSTW18] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. “Doubly-efficient zkSNARKs without trusted setup”. In: *Proceedings of the 39th IEEE Symposium on Security and Privacy*. S&P ’18. 2018, pp. 926–943.

[XZZPS19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. “Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation”. In: *Proceedings of the 39th Annual International Cryptology Conference*. CRYPTO ’19. 2019, pp. 733–764.

[ZGKPP17] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. “vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases”. In: *Proceedings of the 38th IEEE Symposium on Security and Privacy*. S&P ’17. 2017, pp. 863–880.

[ZLWZSXZ21] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. “Doubly Efficient Interactive Proofs for General Arithmetic Circuits with Linear Prover Time”. In: *CCS ’21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*. Ed. by Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi. ACM, 2021, pp. 159–177.