How to Prove False Statements: Practical Attacks on Fiat-Shamir

Dmitry Khovratovich* Ron D. Rothblum† Lev Soukhanov‡

December 11, 2025

Abstract

The Fiat-Shamir (FS) transform is a prolific and powerful technique for compiling public-coin interactive protocols into non-interactive ones. Roughly speaking, the idea is to replace the random coins of the verifier with the evaluations of a complex hash function.

The FS transform is known to be sound in the random oracle model (i.e., when the hash function is modeled as a totally random function). However, when instantiating the random oracle using a concrete hash function, there are examples of protocols in which the transformation is not sound. So far, all of these examples have been contrived protocols that were specifically designed to fail.

In this work, we show such an attack for a standard and popular interactive succinct argument, based on the GKR protocol, for verifying the correctness of a non-deterministic bounded-depth computation. For every choice of the FS hash function, we show that a corresponding instantiation of this protocol, which has been widely studied in the literature and also used in practice, is *not (adaptively) sound* when compiled with the FS transform. Specifically, we construct an explicit circuit for which we can generate an accepting proof for a false statement.

We further extend our attack and show that for *every* circuit C and desired output y, we can construct a functionally equivalent circuit C^*, for which we can produce an accepting proof that C^* outputs y (regardless of whether or not this statement is true). This demonstrates that any security guarantee (if such exists) would have to depend on the specific implementation of the circuit C, rather than just its functionality.

Lastly, we also demonstrate versions of the attack that violate non-adaptive soundness of the protocol. That is, we generate an attacking circuit that is independent of the underlying cryptographic objects. However, these versions are either less practical (as the attacking circuit has very large depth) or make some additional (reasonable) assumptions on the underlying cryptographic primitives.

1 Introduction

The Fiat-Shamir (FS) transform [FS86] is an incredibly influential paradigm in cryptography. Originally, the transformation was suggested as a method for converting an identification scheme into a digital signature. Nowadays however, the FS transform is used much more broadly to convert general interactive (public-coin) protocols into non-interactive ones. Given the power of interaction, this transformation has become extremely important both in theory and practice, in particular for

*Ethereum Foundation. Email: khovratovich@gmail.com.

†Succinct. Email: rothblum@gmail.com.

‡Ethereum Foundation. Email: 0xdeadfae@gmail.com.

the construction of succinct non-interactive arguments aka SNARKs (see, e.g., [Tha22, Section 5] and [CY24, Section 14] for details).

In a nutshell, the simple but extremely powerful idea underlying the transformation is to replace the verifier's random coin tosses with the evaluation of a complex cryptographic hash function (applied to everything the verifier has seen thus far in the interaction). Given that the verifier's messages become deterministic, there is no need for back-and-forth interaction and the prover can generate the entire interaction transcript as a single message, by just evaluating the hash function.

It is clear that the FS transformation preserves completeness and intuitively it seems to also preserve soundness — assuming the hash function is sufficiently complex, the verifier's messages are unpredictable, and so it is unclear how an attacker can leverage the fact that they are chosen deterministically. Still, the question of formalizing this intuition is far from trivial and has intrigued researchers for decades. Given the prevalence of FS, this question has also become extremely important in practice.

A key argument for the security of the transform was given by Pointcheval and Stern [PS00], who showed that the FS transform is secure in the *random oracle model* [BR93].[1] In this model, the FS hash function is modeled as an entirely random function. Within the random oracle model the paradigm is secure, but since a random function has an exponential description size it is clear that this model is purely an idealization. Indeed, the common practice is to replace the random oracle with a concrete hash function, and argue that the random oracle security proof indicates the heuristic security of the construction and that any attack can only be due to a weakness in the hash function.

The influential work of Canetti, Goldreich and Halevi [CGH04] fundamentally challenged this assertion by exhibiting constructions of cryptographic primitives that are secure in the random oracle model but become totally insecure when the random oracle is instantiated, *no matter which concrete hash function is used*. Furthermore, Barak [Bar01] and Goldwasser and Kalai [GK03] specifically gave examples of secure interactive protocols for which the FS transform results in an insecure non-interactive protocol, again, no matter what hash function is used.

The above counterexamples were based on extremely contrived protocols that were specifically designed to fail. Thus, it was commonly believed that such attacks are not applicable in practice. The closest attempt at a more practical attack was given by Bartusek *et al.* [BBH+19] who showed that the standard approach for constructing hash-based proofs (à la [Kil92,Mic00,BCS16]) becomes insecure when applying the FS transform. Still, even their result is only applicable when at least one of the underlying components in the system (i.e., either the interactive oracle proof (IOP), or polynomial commitment scheme (PCS)) is contrived.

Overall, while it has been understood that there exist protocols for which applying the FS transform is insecure, all these attacks were based on some unnatural components and so far no attack is known against "real-world protocols", in particular, protocols that were not intentionally designed to be susceptible to such an attack.

1.1 Our Contributions

In this work we exhibit an attack that breaks the Fiat Shamir security of a standard, natural and practical proof-system. Similarly to the prior works, this attack can be implemented no matter

[1] Early works such as [PS00] focused on applying FS to constant-round protocols. Nowadays the paradigm is also applied to protocols with a super constant number of rounds that satisfy the stronger notion of *round-by-round soundness* [CCH+19].

what FS hash function is used in the following (roughly stated) sense: for every choice of FS hash function, there is a (natural) instantiation of the proof-system for which we can prove a false statement.

Thus, the attack does not point out a weakness in a particular hash function but rather shows that some *natural* schemes are insecure in a strong sense, and further raises serious concerns about the security of the general paradigm.

1.1.1 The Protocol to be Attacked

The interactive protocol for which we mount our attack is, by now, a standard succinct argument for proving the correctness of a computation expressed by a non-deterministic bounded-depth arithmetic circuit. The argument-system is based on a combination of a *multilinear polynomial commitment scheme*[2] (MLPCS) with the popular GKR protocol due to Goldwasser, Kalai and Rothblum [GKR15]. In the protocol, the verifier is given as input a depth d arithmetic circuit C, a (public) instance x and a claimed output y, and its goal is to verify that there exists a witness w such that $C(x, w) = y$.

This proof-system, or close variants[3], has been considered in a line of works [ZGK+17, WTS+18, XZZ+19, ZLW+21], see for example [XZZ+19, Construction 2], and lies at the heart of practical deployed systems such as Expander [Pol24].[4] As pointed out explicitly in these works, when compiled with Fiat-Shamir, the resulting protocol is secure in the random oracle model. Our attack shows that it is *insecure* when the random oracle is instantiated as long as the FS hash function and PCS are computable in (roughly) the same depth d supported by the proof-system.

In more detail, the interactive argument, on which our attack applies, combines an MLPCS comm with the GKR protocol. For a depth bound d, the protocol, which we denote by $\Pi_{\mathsf{comm},d}$, proceeds as follows:

Preprocessing:

1. The parametrization of the polynomial commitment scheme (i.e., a key/salt) is chosen by the verifier.

2. The prover and verifier agree on a depth d arithmetic circuit C. The verifier keeps some short digest of the circuit which we denote by $\langle C \rangle$.[5]

Online:

[2] An MLPCS allows a user to give a short commitment to a large multilinear polynomial P, and later give succinct proofs for evaluation queries of the form "$P(x) = y$". See, e.g., [Tha22, Chapters 14 – 16] for further details and constructions.

[3] These variants differ at some implementation details and optimizations, but seem equally susceptible to our attack.

[4] We informed the designers of [Pol24] of our attack prior to its publication and they have introduced mitigations, along the lines of those discussed in Section 5, see `https://github.com/PolyhedraZK/Expander/pull/184`.

[5] Usually this short description is a hash of (a suitable description of) the circuit, often called a "domain separator", or simply the code of an algorithm that prints the circuit description. The reason that the verifier only keeps a digest of C is that we would like for it to run in time sublinear in the size of C in the online phase. Our attack works for any fixed choice of digest.

3. The prover chooses an input x, witness w and output y. It sends x and y to the verifier in the clear, and uses the MLPCS comm to commit to the multilinear extension[6] (MLE) of the witness w. We denote the commitment by $\mathsf{comm}(w)$.

4. The verifier chooses a random point r and computes the MLE of the claimed output y at the point r. The verifier sends r to the prover.

5. The GKR protocol is used to reduce the claim about y to a claim about the MLE of the input x and the witness w. The verifier can check the claim about x by itself and uses the MLPCS to verify the claim about w.

The input x should be thought of as a parametrization of the circuit and in practice is sometimes not required (indeed, that will be the case in our basic attack below). See Section 2.1 for a more detailed description of the protocol.

Assuming the security of the underlying MLPCS comm, the interactive protocol $\Pi_{\mathsf{comm},d}$ can indeed be shown to be sound. The non-interactive version is obtained via Fiat-Shamir by replacing the verifier's random coin tosses by evaluations of a hash function h. For example, the random value r selected by the verifier at random in Step 4 is replaced with the hash of the protocol transcript: $r = h\big(\langle C\rangle, \mathsf{comm}(w), x, y\big)$. We denote the resulting protocol by $\mathsf{FS}_h(\Pi_{\mathsf{comm},d})$.

1.1.2 Our Basic Attack: a Flawed Circuit

We show that the $\mathsf{FS}_h(\Pi_{\mathsf{comm},d})$ protocol is *not* sound in the following strong sense: For every hash function h and MLPCS comm, there exists $d \in \mathbb{N}$ such that $\mathsf{FS}_h(\Pi_{\mathsf{comm},d})$ is not sound.

In our attack, we construct a circuit C^* (which does not need an additional input x) and an output y^* such that for every witness w it holds that $C^*(w) \neq y^*$, together with an accepting proof π for the false claim that there actually does exist w such that $C^*(w) = y^*$. The circuit C^* that we construct in our attack depends on the choice of comm and h. In particular, if these involve some parametrization (e.g., a salt or equivalently if they are families of functions that need to be instantiated) then these must be known to the cheating prover in order to mount its attack. As such, the attack breaks the *adaptive* soundness of the scheme.

Theorem 1 (Basic Attack)**.** *Let $\mathbb{F}$ be a finite field, comm an MLPCS over $\mathbb{F}$ computable by a depth d_{comm} arithmetic circuit and h a hash function computable by a depth d_h arithmetic circuit.*[7] *Then, for every $d \geq d_{\mathsf{comm}} + d_h + O(1)$ the $\mathsf{FS}_h(\Pi_{\mathsf{comm},d})$ protocol is not adaptively sound (see Definition 6 below).*

The circuit C^* which our adversary constructs to break soundness may at first glance seem odd. In particular, as alluded to above, it internally invokes the FS hash function h and the MLPCS comm. Indeed, this follows (and is inspired by) the diagonalization based approach in the prior works [CGH04, Bar01, GK03, BBH+19]. At second glance however, we remark that circuits that

[6] The multilinear extension is a particular way of encoding data. For the purposes of this work the specifics of the encoding are not so important, but briefly, for a finite field $\mathbb{F}$, the multilinear extension of a function $f : \{0,1\}^m \to \mathbb{F}$ is the (unique) multilinear polynomial $\hat{f} : \mathbb{F}^m \to \mathbb{F}$ that agrees f on $\{0,1\}^m$ (see [Tha22, Chapter 3] for additional details).

[7] In case the hash function is natively expressed via a Boolean circuit, we can emulate the Boolean operations using arithmetic ones with constant overhead.

invoke these underlying primitives are actually *quite common in practice*. In particular, in the context of recursive proof composition [Val08, BCCT13, BCTV14].

Regardless of whether the attacking circuit C^* that we construct is natural or not, the attack breaks the basic "contract": the proof-system is supposed to guarantee soundness wrt to every circuit that is used, whereas we exhibit a counterexample that demonstrates that this is not the case. Things become even worse when considering the following generalization of the attack.

1.1.3 Extended Attack: Arbitrary Circuit

One might hope to mitigate the basic attack by considering only functionalities that are unrelated to the underlying cryptographic primitives.

Unfortunately, we can extend the attack to arbitrary circuits in the following sense: given a circuit C, which computes some arbitrary functionality, we can insert a "backdoor" into it which allows us to prove any desired statement. In more detail, for any desired output y^*, we can modify C into a new *functionally equivalent* circuit C^* (i.e., C^* computes the exact same function as C), and yet for any input x, we are able to produce a proof of the statement "exists w such that $C^*(x, w) = y^*$", whether or not this statement is actually true. The only minor restriction that we impose is that the witness size for the circuit be at least as large as the digest $\langle C \rangle$ of C – we refer to such circuits as *admissible*.

Theorem 2 (Extended Attack)**.** *Let $\mathbb{F}$ be a finite field, comm an MLPCS over $\mathbb{F}$ computable by a depth d_{comm} arithmetic circuit and h a hash function computable by a depth d_h arithmetic circuit.*

There exist a polynomial-time algorithm A that takes as input

- *instantiations of comm and h,*
- *a depth-d admissible arithmetic circuit $C : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^\ell$,*
- *arbitrary input $x^* \in \mathbb{F}^n$,*
- *desired output $y^* \in \mathbb{F}^\ell$,*

and outputs

- *a circuit $C^* : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^\ell$ of depth $d' = d + d_{\mathsf{comm}} + O(d_h \cdot \log(\ell))$*
- *a proof-string π*

such that

- *C computes the same function as C^*:*

$$\forall\, (x, w) \in \mathbb{F}^n \times \mathbb{F}^m : \quad C(x, w) = C^*(x, w).$$

- *The $\mathsf{FS}_h(\Pi_{\mathsf{comm},d'})$ verifier accepts given (C^*, x^*, y^*, π).*

Theorem 2 demonstrates a crucial point: the soundness of the FS transform for $\Pi_{\mathsf{comm},d}$ (and potentially other protocols) fundamentally depends on the *specific implementation* of the circuit C being proved, rather than merely its *functionality*. Given the difficulty of "reverse engineering" the implementation of the circuit (especially given tools such as obfuscation) one should take extreme care when using this proof-system for *any* complex circuit C.

Remark 3. *The GKR protocol is sometimes used to prove correctness of very specific and simple circuits, especially in the context of lookup arguments (see e.g., [PH23,STW24]), in particular grand product arguments [Tha13]. Since these functionalities have a canonical representation, which is extremely simple, our attack does not seem to be applicable in this setting.*

Remark 4. *Theorem 1 can actually be derived as a simple corollary of Theorem 2 (by considering a circuit C that never outputs some fixed value y^*). Still, since the proof of Theorem 1 is significantly simpler, we prefer to keep their presentations separate.*

1.1.4 Universal Circuits

As noted above, the attacks described in Theorems 1 and 2 violate the *adaptive* soundness of the protocol in the following sense: the cheating prover needs to know the full specification (i.e., key/salt) of the hash function and polynomial commitment in order to construct and specify the attacking circuit.

In Section 4 we give extensions of the attack that utilize fixed circuits that are independent of the underlying cryptographic primitives. In a nutshell, these are constructed using universal circuits and quines.

1.2 Additional Related Works

Fiat-Shamir: The Theory Perspective. Given the known counterexamples discussed above for the security of Fiat-Shamir, a major line of work in the theory literature has attempted to establish security of the FS transform based on so-called "standard" cryptographic assumptions, in the plain model. In order to so these works focused on compiling interactive *proofs* or limited forms of *arguments*. Kalai, Rothblum and Rothblum [KRR17] gave the first proof-of-concept based on strong obfuscation assumptions (see also [CCR16]) but since then a line of work [CCRR18, CCH+19, BKM20, HLR21, JKKZ21, CJJ21, CGJ+23] has constructed schemes that are secure assuming standard assumptions such as LWE or (sub-exponential) DDH, and in the uniform setting from derandomization-style hardness assumptions [CT23, CRT24].

So far this line of work has focused primarily on deterministic computations (or some limited non-determinism), in particular due to the Gentry-Wichs barrier [GW11]. Our attack further motivates this line of work and raises the question of whether it can be made practical.

"Weak" Fiat-Shamir. While the Fiat-Shamir heuristic is easy to describe, it is somewhat notorious for suffering from implementation bugs. Most notably, some implementations accidentally used the so-called "weak" variant, in which the computational statement being proved is not included as an input to the hash. This has led to actual attacks [BPW12, HLPT20, DMWG23, Tha23].

In this work we consider the strong variant (which can be seen by the fact that the circuit description is included in the hash) and show that it is insecure when applied to the GKR-based argument-system.

Random Oracle Uninstantiability. The work of Branco, Döttling and Dujmovic [BDD22] shows a natural incompressible encryption scheme in the random oracle model, which becomes insecure when the random oracle is instantiated.

2 Background: The GKR Protocol

The GKR protocol [GKR15] is a doubly-efficient interactive proof for verifying the correctness of bounded-depth computations. We first describe the "vanilla" GKR protocol for *deterministic* computations (with statistical soundness) and then, in Section 2.1 explain how it is used to derive a succinct argument (i.e., with computational soundness) for *non-deterministic* computations. Since it suffices for our attack, we only give a fairly high-level description.

In the vanilla GKR protocol the goal is to verify a claim of the form $C(x) = y$, where C is a low-depth arithmetic circuit over a finite field $\mathbb{F}$, x is an input and y is a claimed output. Both x and y are known to to the verifier. Loosely speaking, the protocols is based on the following steps:

- (Output Layer:) First, the verifier evaluates the multilinear extension of the output y at a random point.[8]
- (Processing Circuit Layers:) The core of the protocol is an interactive reduction (based on the sumcheck protocol [LFKN92]) that reduces a claim about the multilinear extension of a layer i, to a claim about the multilinear extension of layer $i+1$ (where the output is layer 0).

  This step is repeated until eventually we get a claim about the input layer.
- (Input Layer:) After processing all of the layers of the circuit, the verifier is left with a claim about the multilinear extension of the input, which it can either compute by itself, or is sometimes passed on to some other protocol (see more below).

Remark 5. *Note that the structure of the GKR protocol means that for a composed circuit of the form $C(x) = C_2(C_1(x))$ the protocol can be thought of as a concatenation of two GKR protocols, one reducing a claim about the MLE of the output of C_2 to a claim about an input z for that circuit, and then viewing that string as the output of C_1 and applying another GKR protocol to reduce the latter claim to a claim about the input x.*

Additionally, we note that location of the claims in the MLEs generated throughout the interaction, depend only on the verifier's randomness.

For a detailed overview of the GKR protocol, see either [Tha22, Section 4] for an applied perspective or [Gol18, Section 3] for a theory oriented one.

2.1 GKR as a Succinct Argument

In practice, GKR is often used as a *computationally sound proof* (aka argument) for non-deterministic computations by combining it with a cryptographic multilinear polynomial commitment scheme (MLPCS) comm.

In this context, the goal is, for a given depth d arithmetic circuit $C : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^\ell$, input $x \in \{0,1\}^n$ and claimed output $y \in \mathbb{F}^\ell$, to verify that there exists a witness $w \in \mathbb{F}^m$ such that $y = C(x, w)$. The protocol, denoted $\Pi_{\mathsf{comm},d}$, proceeds as follows:

1. (Preprocessing:) The verifier chooses a parametrization of comm and the prover chooses a circuit $C : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^\ell$ and sends it to the verifier. The verifier maintains a short digest of

[8] Sometimes the output is just a single bit in which case this step can be skipped (indeed, that is the original description in [GKR15]). However, it will be convenient in our attack to consider circuits with a longer output.

C, which we denote by $\langle C \rangle$ (in practice this might be some hash of the circuit, aka a domain separator).

$$\mathbb{P} \xleftarrow{\mathsf{comm}} \mathbb{V}$$
$$\mathbb{P} \xrightarrow{C} \mathbb{V}$$

2. (Commitment:) The prover specifies an input $x \in \{0,1\}^n$, output $y \in \{0,1\}^\ell$ and witness $w \in \mathbb{F}^m$. The input and output are sent to the verifier in the clear, whereas only a commitment $\alpha = \mathsf{comm}(w)$ to w is sent.

$$\mathbb{P} \xrightarrow{x,y,\alpha=\mathsf{comm}(w)} \mathbb{V}$$

3. (Output Claim:) The verifier chooses a random $r \in \mathbb{F}^{\log(\ell)}$ and deduces a claim on $\hat{y}(r)$ (indeed, observe that since y has length ℓ, its multilinear extension is a function on $\log(\ell)$ variables).

$$\mathbb{P} \xleftarrow{r} \mathbb{V}$$

4. (GKR:) The prover and verifier run the vanilla GKR protocol (described above) to reduce the claim about $\hat{y}(r)$ to claims about $\hat{x}$ and $\hat{w}$.

$$\mathbb{P} \xleftrightarrow{\text{GKR}-\text{Transcript}(\langle C \rangle, x, y, \alpha, r)} \mathbb{V}$$
$$\mathbb{V}: \quad \hat{x}(r_x) \stackrel{?}{=} v_x$$
$$\mathbb{V}: \quad \hat{w}(r_w) \stackrel{?}{=} v_w$$

5. (PCS Evaluation:) The verifier checks the claim about $\hat{x}$ directly and checks the claim about $\hat{w}$ by running the evaluation phase of the MLPCS. If all tests pass then the verifier accepts, otherwise it rejects.

$$\mathbb{P} \xleftrightarrow{\text{MLPCS}-\text{Eval}(\langle C \rangle, \alpha, r_w, v_w \ldots,)} \mathbb{V}$$

The soundness of this interactive protocol follows immediately from the (computational) binding of the MLPCS and the (statistical) soundness of the GKR protocol.

2.2 Applying Fiat-Shamir to GKR

The non-interactive variant of the protocol operates relative to a Fiat-Shamir hash function h (in addition to the MLPCS comm). It proceeds similarly to the above interactive protocol except that the verifier's randomness is replaced by evaluations of the FS hash function applied to the transcript.

In particular, in Step 3 above, both parties set $r = h\big(\langle C \rangle, x, y, \alpha\big)$ and the GKR protocol is run to check that the multilinear extension of $C(x)$ at the specific point r is equal to $\hat{y}(r)$, where y is the claimed output.

The resulting protocol is denoted $\mathsf{FS}_h(\Pi_{\mathsf{comm},d})$.

Definition 6 (Adaptive Soundness of $\mathsf{FS}_h(\Pi_{\mathsf{comm},d})$)**.** *Let h be a hash function,* comm *an MLPCS and $d \in \mathbb{N}$. We say that $\mathsf{FS}_h(\Pi_{\mathsf{comm},d})$ is* adaptively sound *if for every polynomial-time algorithm A, that is given as input a specification of h and* comm *and outputs a circuit C, input x, output y and proof π the event that both*

- *for all w it holds that $C(x,w) \neq y$, and*
- *the verifier of $\mathsf{FS}_h(\Pi_{\mathsf{comm},d})$ accepts given (C,x,y,π),*

happens with negligible probability.

Recall that Theorem 1 establishes that $\mathsf{FS}_h(\Pi_{\mathsf{comm},d})$ is *not* adaptively sound. Moreover, as we show in Section 3, the proof of Theorem 1 actually shows a highly efficient attack that makes the verifier accept a false statement with probability 1.

3 Breaking $\mathsf{FS}_h(\Pi_{\mathsf{comm},d})$

Let h be a hash function computable in depth d_h and comm an MLPCS computable in depth d_{comm}. Let $d = d_h + d_{\mathsf{comm}} + O(1)$, where the $O(1)$ is a fixed constant that will be determined below. For simplicity we assume that h and comm are computed by arithmetic circuits over the relevant field (needless to say, in case they are Boolean circuits they can still be emulated by arithmetic circuits).

Section Organization. In Section 3.1 we prove Theorem 1 by demonstrating that the protocol $\mathsf{FS}_h(\Pi_{\mathsf{comm},d})$ is not adaptively sound (according to Definition 6). Then, in Section 3.2 we prove Theorem 2 by showing how to manipulate a given circuit in order to prove a false claim about it.

3.1 Proving a False Statement: Proof of Theorem 1

To demonstrate the attack, we will construct an explicit depth d circuit C^* and an accepting proof for a false claim about the circuit. In a gist, the idea underlying the attack, following [CGH04], is not to explicitly predict the verifier's randomness but rather craft a prover message (in this case a choice of circuit) for which the resulting verifier challenge will be favorable. We proceed to describe the attack.

The Attacking Circuit. In the attack we will be considering a circuit C^* that only gets as input a witness $w \in \mathbb{F}^m$ and does not get an input x beyond the witness. We set m to be sufficiently large so that the circuit digest $\langle C \rangle$ can fit in $\mathbb{F}^m$ (for example, if the digest is 256-bits long and $\mathbb{F}$ is a 128-bit field, then $m \geq 3$ suffices).

For intuition on the circuit construction, note that once we decide on a circuit C^* and desired output y^*, this determines "randomness" r that will be used by the verifier in Step 3 in the $\mathsf{FS}_h(\Pi_{\mathsf{comm},d})$. Namely, r will be fixed to $r = h(\langle C^* \rangle, y^*, \alpha)$, where α is a commitment to some witness. We would like for our circuit to be able internally to predict r in order to mount an attack. The challenge (which is rooted in the notion of *correlation intractability* [CGH04]) is that there appears to be a circular dependency – it seems that whatever change we make in C^* in order to predict r, changes C^* and therefore changes r, and we are back to square one. To get around the circular dependency, we simply provide C^* with its own digest $\langle C^* \rangle$ as a *witness*.

We proceed to the construction. Given $w \in \mathbb{F}^m$, the circuit $C^*(w)$ outputs two field elements as described in Construction 1.

Construction 1. *The circuit $C^*(w)$ is defined as follows:*

1. *Interpret w as a circuit digest ψ.*
2. *Compute $\alpha = \mathsf{comm}(w)$.*
3. *Compute $\gamma = h(\psi, y^*, \alpha)$, where $y^* = (0, 0)$.*
4. *Output $(\gamma, \gamma - 1)$.*

Note that C^* is polynomial-size and has depth $d_{\mathsf{comm}} + d_h + O(1)$ as required.

Proof of Theorem 1. We construct an accepting proof string π for the claim that there exists $w \in \mathbb{F}^m$ such that $C^*(w) = y^*$, where $y^* = (0, 0)$. Since by construction the two elements in the output of C^* are always distinct, this claim is clearly false.

The cheating proof string is constructed essentially honestly, relative to the witness $w = \langle C^* \rangle$, where $\langle C^* \rangle$ denotes the digest of the above circuit. In other words, the proof is constructed as follows:

- Set $w = \langle C^* \rangle$, and $\alpha = \mathsf{comm}(w)$.
- The remaining part of the proof is the GKR-Transcript and MLPCS-Eval execution (Steps 4 and 5 of $\Pi_{\mathsf{comm},d}$, Section 2.1), which can be constructed honestly as we explain below. Denote this part by τ.
- The proof-string is $\pi = (\alpha, \tau)$.

Now we prove that (α, τ) is accepted by the verifier. Recall that C^* denotes the circuit described in Construction 1, and we set $w = \langle C^* \rangle$ and $\alpha = \mathsf{comm}(w)$, as defined above.

At **Step 3** of $\Pi_{\mathsf{comm},d}$ the verifier constructs a multilinear extension of the output y using the circuit description C. By construction, the circuit C^* on input w outputs $y = (\gamma, \gamma - 1)$, where $\gamma = h(w, y^*, \alpha)$. The (multi-)linear extension $\widehat{y}$ of y is then the function $\widehat{y}(X) = \gamma - X$. By the construction of C^* we have

$$\gamma = h(w, y^*, \alpha) = h(\langle C^* \rangle, y^*, \alpha)$$

and, by definition of $\mathsf{FS}_h(\Pi_{\mathsf{comm},d})$, we have that

$$r = h(\langle C^* \rangle, y^*, \alpha).$$

Thus, the evaluation of $\widehat{y}$ at the specific point r is equal to $\widehat{y}(r) = \gamma - r = 0$. On the other hand, since $y^* = (0, 0)$ its multilinear evaluation at any point is equal to 0. We conclude that the multilinear extensions of y and y^* agree on the point r.

Steps 4 and 5. Since the multilinear extensions of y and y^* agree on the challenge point r, from here on the claim being proved by the GKR system is correct (in the language of round-by-round soundness, we have moved from a "doomed" state into an accepting state) and from here on the prover can simply run the honest prover strategy relative to the correct claim that $\hat{y}(r) = 0$, which is accepted by the verifier.

This concludes the proof of Theorem 1.

3.2 Attacking the Circuit Implementation: Proof of Theorem 2

Let $C : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^\ell$ be an admissible depth d arithmetic circuit. For any $x^* \in \mathbb{F}^n$ and $y^* \in \mathbb{F}^\ell$ we show how to construct a circuit $C^* : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^\ell$ that is functionally equivalent to C, but for which we can prove the (potentially false) claim that there exists w such that $C^*(x^*, w) = y^*$.

For the construction it will be convenient for us to use a fixed arithmetic circuit $\mathsf{IF\text{-}THEN\text{-}ELSE} : \mathbb{F}^{1+1+\ell+\ell} \to \mathbb{F}^\ell$ such that for $a, b \in \mathbb{F}$ and $c, d \in \mathbb{F}^\ell$ we define $\mathsf{IF\text{-}THEN\text{-}ELSE}(a, b, c, d) = (a - b) \cdot (c - d) + d$. The name of this function is due to the fact that if $a = b$ then $\mathsf{IF\text{-}THEN\text{-}ELSE}(a, b, c, d)$ outputs d and if $a = b + 1$ it outputs c (and we will only be using this function for inputs such that $a \in \{b, b + 1\}$).

We now describe the circuit C^*, as well as a function g on which it relies. The construction proceeds as follows.

Construction 2. *The circuit $C^*(x, w)$:*

1. *Compute $y = C(x, w)$.*
2. *Compute $\gamma = g(x, w, y) \in \mathbb{F}$, using the procedure described below.*
3. *Compute $u_{\mathsf{real}} = (\gamma, \gamma - 1, y, y^*)$.*
4. *Out $\mathsf{IF\text{-}THEN\text{-}ELSE}(u_{\mathsf{real}})$.*

The function $g(x, w, y)$:

1. *Interpret w as a circuit digest ψ.*
2. *Compute $\alpha = \mathsf{comm}(\psi)$.*
3. *Compute $r = h(\psi, x, y^*, \alpha)$. This will be the challenge generated in the interactive protocol when y^* is submitted as the claimged output (cf. Section 2.1)*
4. *Set $u_{\mathsf{fake}} = (0, 0, y, y^*)$. Compute the GKR transcript (Step 4 of Section 2.1) with respect to output y^*, circuit digest ψ, actual circuit $\mathsf{IF\text{-}THEN\text{-}ELSE}$, input u_{fake}. This results in a point r' to verify the claim about the MLE of u_{fake}. (Note that $\mathsf{IF\text{-}THEN\text{-}ELSE}(u_{\mathsf{fake}}) = y^*$ and so we can generate the transcript using the honest GKR prover.)*
5. *Output the first coordinate of r'.*

We proceed to prove Theorem 2 by showing that C^* is functionally equivalent to C, and yet its soundness can be broken.

Functional Equivalence. Let $x \in \mathbb{F}^n$ be an input and $w \in \mathbb{F}^m$. Let $y = C(x, w)$ be the real output of the circuit and recall that we fixed y^* as the desired output.

By construction, it holds that:

$$C^*(x, w) = \mathsf{IF\text{-}THEN\text{-}ELSE}(\gamma, \gamma - 1, y, y^*) = y,$$

where we note that the last equality holds regardless of how γ is generated. Thus, C^* is functionally equivalent to C.

Attacking C^*. In the attack, the prover claims that for a given x^*, there exists w s.t. $C^*(x^*, w) = y^*$.

The prover starts the attack following the steps of the GKR-based succinct argument (see Section 2.1):

1. The prover is given the commitment comm and hash function h.
2. The prover specifies the circuit C^*, as described in Construction 2, and the verifier stores the digest $\langle C^* \rangle$.
3. The prover sends x^* as the input, y^* as the output and $\alpha = \mathsf{comm}(w)$, where $w = \langle C^* \rangle$, as the witness commitment (where we pad w to be m bit).
4. Now, the "random" point r in the multilinear extension of y^* is determined by an application of Fiat-Shamir:
$$r = h(\langle C^* \rangle, x^*, y^*, \alpha).$$

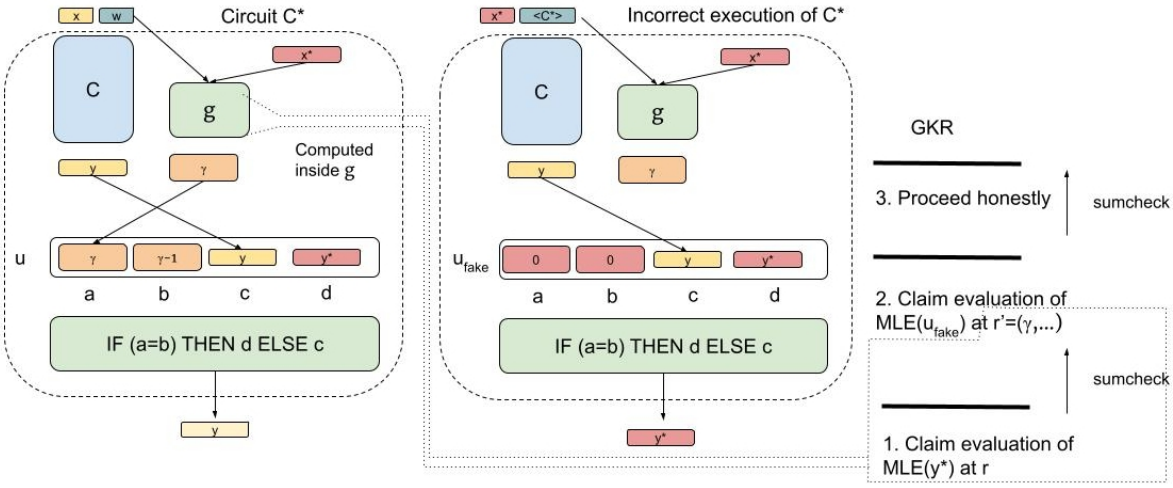

Figure 1: Attack on an arbitrary circuit C via an equivalent circuit C^*.

At this point in the attack (which is also illustrated in Fig. 1), the prover and (FS-)verifier start engaging in the actual GKR protocol to reduce the claim about $\widehat{y^*}(r)$ to a claim about the input. It will be convenient for us to think of the circuit C^* as operating in two phases: the first phase concludes with Step 3 in the description of C^* (this phase computes the vector $u_{\mathsf{real}} = (\gamma, \gamma - 1, y, y^*)$. In the second phase we apply the IF-THEN-ELSE circuit to u_{real}.

The GKR protocol corresponds to these two phases (see Remark 5): first the claim about $\widehat{y^*}(r)$ is reduced to a claim about $\widehat{u_{real}}$. Then, the latter is reduced to a claim about the MLE of the main input x^* and witness w.

The prover starts emulating the protocol corresponding to the second phase of the circuit (i.e., the IF-THEN-ELSE computation). The prover (which recall, is a cheating prover) runs the honest prover strategy for this subcircuit but rather than doing so wrt to u_{real}, it uses $u_{\mathsf{fake}} = (0, 0, y, y^*)$. The verifier challenges, as usual, are computed using Fiat-Shamir.

Since the claim that $\mathsf{IF\text{-}THEN\text{-}ELSE}(u_{\mathsf{fake}}) = y^*$ is correct, the result of this step is a correct claim about $\widehat{u_{\mathsf{fake}}}$. Namely, a claim of the form $\widehat{u_{\mathsf{fake}}}(r') = v$.

The crucial point is that, by construction of C^* (and specifically the definition of g), it holds that the first coordinate of r' is equal to $\gamma = g(x^*, w, y)$. Using this, we establish the following simple claim.

Proposition 7.

$$\widehat{u_{\mathsf{real}}}(r') = \widehat{u_{\mathsf{fake}}}(r')$$

Proof. For every $z \in \mathbb{F}^{1+\log(\ell+1)}$ it holds that,

$$\widehat{u_{\mathsf{real}}}(z) - \widehat{u_{\mathsf{fake}}}(z) = \left(\prod_{i=1}^{\log(\ell+1)} (1 - z_i)\right) \cdot \left(\gamma - z_{1+\log(\ell+1)}\right), \tag{1}$$

where z_i denotes the i-th coordinate of z. To see that Eq. (1) holds, observe that it holds for all Boolean values z (for $z = 0^{\log(\ell+1)+1}$ both sides evaluate to γ, for $z = 0^{\log(\ell+1)}1$ both evaluate to $\gamma - 1$ and everywhere else they evaluate to 0. Since both sides of the equation are multilinear polynomials, if they agree on Boolean values they must also agree on any $z \in \mathbb{F}^{1+\log(\ell+1)}$.

The claim now follows from the fact that the first coordinate of r' is γ, and so the RHS of Eq. (1) evaluates to 0. □

Thus, we see that the claim derived u_{fake} is true also for the *correct* evaluation u_{real}. Thus, from here on we can just run the honest prover for the correct GKR computation and obtain an accepting proof.

4 Universal Computation Attack

At this point, one might perilously assume that to deflect the attack, it is enough to ensure that the GKR circuit in question does not contain an implementation of the FS hash function or MLPCS. As we show in the next series of examples, this requirement is insufficient.

4.1 Quine Example

Recall that a quine is a program that outputs its own code. Any Turing-complete language admits a quine, according to the folklore adaptation of Kleene's second recursion theorem [Rog87].

We define a notion of efficiently reflexive admissible programming system (see Definition 13 below). See Appendix A for details, but basically it is a notion of programming language in which currying and composition of programs is expressed with constant overhead. Most realistic languages satisfy this notion.

For a program p we will denote by p the function that it computes, and by $[p]$ its code.

Lemma 8. *For any program f there is an f-*quine*: a program p that outputs (on any input) $f([p])$. Furthermore, there exists a universal constant c such that $|[p]| = |[f]| + c$ and if the runtime of f is polynomial in its input, then the runtime of p is polynomial in $|[f]|$.*

We defer the proof of Lemma 8 to Appendix A.

Consider a circuit simulating a universal Turing machine (see Fig. 2) which runs an interpreter of our language. We note that this construction is not efficient, but later, in Section 4.2, we will get efficiency back at the cost of some specialization. Most importantly, this attack is *independent* of the commitment scheme in question.

Universal Turing machine circuit $T(N, M)$:

1. Starts with an input w of size N. This encodes the initial state of the tape of the universal Turing machine, padded with 0-s to infinity outside of the initial $[0..N]$ segment.
2. Outputs the (fixed-size) output of the Turing machine if it terminated in $\leq M$ steps, and a special symbol $\perp$ otherwise. (normally, output tape of the Turing machine can be large, but we only output a fixed-size chunk of it for convenience).

Figure 2: Universal Turing machine circuit

Requirement. In the upcoming theorem, we will only consider commitment schemes that admit implementation with code size (at least linearly with factor < 1) smaller than the size of the input. Most meaningful commitment schemes fall into this natural category: including hash-based commitment schemes (FRI) and the seeded Pedersen commitment scheme. The schemes with structured reference string provided by trusted setup (such as KZG) do not belong to this category because the code of the implementing function needs to fully contain the reference string. In practice, however, even they typically do, because multiple polynomials are committed with the same reference string.

Theorem 9. *There exists a family of circuits $T'(N, M)$ such that for any (polynomial-time) commitment scheme satisfying our requirement and any hash function, there exists M with $M = \text{poly}(N)$, such that the FS-GKR protocol for the circuit $T'(N, M)$ is unsound. (We omit the dependence on a security parameter λ for clarity.)*

Proof. First, we define a small modification of a circuit $T(N, M)$. A circuit $T'(N, M)$ has the following functionality: it first computes output $\gamma = T(N, M)$ (at this stage treated as valid output of the machine), casts it to a field element and outputs $(\gamma, \gamma - 1)$.

Consider a function f which, given an input w of size N, computes the first challenge that the FS-GKR protocol for $T'(N, M)$ would give to the prover.

Modified circuit $T'(N, M)$:

1. Run $T(N, M)$ to obtain an output γ (of fixed bitsize size $\geq \log(|F|)$).
2. Cast it to a single field element and output a pair of field elements $(\gamma, \gamma - 1)$.

Figure 3: Circuit $T'(N, M)$

Denote by p an f-quine, which exists by Lemma 8. The size of $[p]$ is $|[f]|+c$, for a fixed constant c, and so for a large enough N it is smaller than N by our requirement.

Additionally, the evaluation time of p is polynomial in the evaluation time of f, which is itself independent of M and is polynomial in N. We can choose M to be a polynomial in N such that p halts in M steps.

We then pass the program $[p]$ as a witness to the circuit $T'(N, M)$. The program outputs a pair $(\gamma, \gamma - 1)$, as in Construction 1. We proceed with exactly the same attack: pretend that the output is $(0, 0)$, while in fact there is no witness that yields a $(0, 0)$ output. Therefore, the circuit is non-adaptively unsound (against any hash and commitment scheme satisfying our requirement). □

4.2 Universal GKR Example

In this section, we sketch a less universal but also interesting attack. Let $G(N_{in}, N_{out}, N, M)$ be a universal GKR circuit, with the following functionality:

Universal GKR circuit $G(N_{in}, N_{out}, N, M)$:

1. Starts with $w = (w_{in}|w_C)$, where w_C encodes a GKR circuit with N_{in} input size, M intermediate layers of size N, and output layer of size N_{out}, and w_{in} is a vector of size N_{in}.
2. Outputs $w_C(w_{in})$.

Figure 4: Universal GKR circuit

It might appear that it is impossible to mount an attack on such a circuit - after all, the w_C is incorporated in the commitment. This is, of course, not the case - for example, if our commitment has the form $f(\mathsf{comm}(w_C), \mathsf{comm}(w_{in}))$, the attack essentially reduces to the original C^*, with $\mathsf{comm}(w_C)$ playing the role of $\langle C \rangle$. Interestingly, for many commitment schemes it is exactly the case:

Theorem 10 (Informally Stated)**.** *For any additively-homomorphic commitment scheme (both truly additive such as elliptic-curve based commitments, or weakly additive such as lattices), the GKR protocol for a universal GKR circuit of large enough depth M is unsound.*

Proof sketch. Because the scheme is additively homomorphic, $\mathsf{comm}(w) = \mathsf{comm}(w_C) + \mathsf{comm}(w_{in})$. By passing $\mathsf{comm}(w_C)$ as a part of w_{in}, we reproduce the original attack method. □

While we are unable to directly mount this attack against FRI-based schemes, it should be noted that many modern FRI-based protocols use batching, which means that the data passed as a commitment $\mathsf{comm}(w)$ is in fact a disjoint union of multiple Merkle roots corresponding to different polynomials. Against such protocols, this attack can be mounted.

This means that at least for additively-homomorphic commitments, the concern is much larger. Specifically, to thwart the attack, the circuit designer must somehow ensure that there is no way of representing the commitment / hashing computation even on a *small, arbitrarily allocated part of the witness*. This likely restricts feasible GKR circuits as either circuits of bounded from above depth or circuits with extremely restricted data flow.

5 Conclusions and Mitigations

We find the violation of the Fiat-Shamir security of a standard and natural protocol to be very concerning. First and foremost it raises the question about the Fiat-Shamir security of other protocols. This calls for significant cryptographic effort by the community in studying the security, or potential insecurity, of Fiat-Shamir of other protocols used in practice.

Still, it is worthwhile to point out some specific properties of the GKR-based protocol that we considered in this work, which facilitated our attack. We emphasize that while we do not know attacks for protocols that do not satisfy these properties, this does not mean that such attacks do not exist.

First, it is useful to compare the modern applications of Fiat-Shamir to those originally envisioned (i.e., in the 80's and 90's), specifically in the context of constructions of digital signatures schemes. The original use was applied to very specific identification schemes (e.g., built around concrete number theoretic problems). In contrast, the modern usage is intentionally designed for protocols that are used to prove *general purpose computations*. Our attack leverages this in order to invoke the proof-system relative to a computation that involves the computation of the Fiat-Shamir hash function itself (as well as polynomial commitment).

Elaborating further on this point, the GKR-based protocol, in contrast to other protocols in the literature, has the key property that the prover does not commit to the full computation trace (indeed, this is one of the most *compelling* features of this protocol). Unfortunately, the fact that the computation is not committed to also enables our attack – we can consider explicitly invoking the Fiat-Shamir hash function, without needing to commit to the corresponding computation trace.

Thus, a natural countermeasure for the GKR-based protocol is to ensure that the circuit family considered is not powerful enough to compute the hash function. This can be due to any natural computational resource, but some natural ones are depth (as an arithmetic circuit) or potentially algebraic degree. This may be achieved by attempting to increase the depth of the hash function used (possibly via composition), or reducing the depth of the circuit in question (possibly by committing to some of the intermediate values).

In a followup work, Arnon and Yogev [AY25] propose a different mitigation in which the size of the Fiat-Shamir hash function is increased, while using a clever proof of work protocol so as not to pay in larger verifier time. The authors additionally propose an extension of the random oracle model geared at capturing diagonalization attacks. However, due to technical reasons their model fails to capture our attack (as well as attacks in [BBH+19]).

We emphasize that while we do not know how to attack protocols that place either our countermeasures or the one in [AY25], whether or not they are actually secure is an open question.

Acknowledgments

We thank Tohru Kohrita for useful comments.

References

[AY25] Gal Arnon and Eylon Yogev. Towards a white-box secure fiat-shamir transformation. *IACR Cryptol. ePrint Arch.*, page 329, 2025. 16

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 106–115. IEEE Computer Society, 2001. 2, 4

[BBH+19] James Bartusek, Liron Bronfman, Justin Holmgren, Fermi Ma, and Ron D. Rothblum. On the (in)security of Kilian-based SNARGs. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 522–551. Springer, 2019. 2, 4, 16

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 111–120. ACM, 2013. 5

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 31–60, 2016. 2

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 276–294. Springer, 2014. 5

[BDD22] Pedro Branco, Nico Döttling, and Jesko Dujmovic. Rate-1 incompressible encryption from standard assumptions. *IACR Cryptol. ePrint Arch.*, page 697, 2022. 6

[BKM20] Zvika Brakerski, Venkata Koppula, and Tamer Mour. NIZK from LPN and trapdoor hash via correlation intractability for approximable relations. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 738–767. Springer, 2020. 6

[BPW12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 626–643. Springer, 2012. 6

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st*

ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993, pages 62–73. ACM, 1993. 2

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1082–1090. ACM, 2019. 2, 6

[CCR16] Ran Canetti, Yilei Chen, and Leonid Reyzin. On the correlation intractability of obfuscated pseudorandom functions. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 389–415. Springer, 2016. 6

[CCRR18] Ran Canetti, Yilei Chen, Leonid Reyzin, and Ron D. Rothblum. Fiat-Shamir and correlation intractability from strong KDM-secure encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 91–122. Springer, 2018. 6

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004. 2, 4, 9

[CGJ+23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and SNARGs from sub-exponential DDH. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 635–668. Springer, 2023. 6

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for P from LWE. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 68–79. IEEE, 2021. 6

[CRT24] Lijie Chen, Ron D. Rothblum, and Roei Tell. Fiat-Shamir in the plain model from derandomization (or: Do efficient algorithms believe that NP = PSPACE?). *Electron. Colloquium Comput. Complex.*, TR24-116, 2024. 6

[CT23] Lijie Chen and Roei Tell. When Arthur has neither random coins nor time to spare: Superfast derandomization of proof systems. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 60–69. ACM, 2023. 6

[CY24] Alessandro Chiesa and Eylon Yogev. *Building Cryptographic Proofs from Hash Functions*. 2024. 2

[DMWG23] Quang Dao, Jim Miller, Opal Wright, and Paul Grubbs. Weak Fiat-Shamir attacks on modern proof systems. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 199–216. IEEE, 2023. 6

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986. 1

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 102–113. IEEE Computer Society, 2003. 2, 4

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for Muggles. *J. ACM*, 62(4):27:1–27:64, 2015. 3, 7

[Gol18] Oded Goldreich. On doubly-efficient interactive proof systems. *Found. Trends Theor. Comput. Sci.*, 13(3):158–246, 2018. 7

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 99–108. ACM, 2011. 6

[HLPT20] Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. How not to prove your election outcome. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 644–660. IEEE, 2020. 6

[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 Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 750–760. ACM, 2021. 6

[JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Yun Zhang. SNARGs for bounded depth computations and PPAD hardness from sub-exponential LWE. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 708–721. ACM, 2021. 6

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 723–732. ACM, 1992. 2

[KRR17] Yael Tauman Kalai, Guy N. Rothblum, and Ron D. Rothblum. From obfuscation to the security of Fiat-Shamir for proofs. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 224–251. Springer, 2017. 6

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992. 7

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000. 2

[PH23] Shahar Papini and Ulrich Haböck. Improving logarithmic derivative lookups using GKR. *IACR Cryptol. ePrint Arch.*, page 1284, 2023. 6

[Pol24] Polyhedra Network. Expander. https://github.com/PolyhedraZK/Expander, 2024. Accessed: 2025-1-15. 3

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptol.*, 13(3):361–396, 2000. 2

[Rog87] Hartley Rogers. *Theory of recursive functions and effective computability*. MIT Press, Cambridge, MA, USA, 1987. 13

[STW24] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Unlocking the lookup singularity with Lasso. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 180–209. Springer, 2024. 6

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2013. 6

[Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. *Found. Trends Priv. Secur.*, 4(2-4):117–660, 2022. 2, 3, 4, 7

[Tha23] Justin Thaler. 17 misconceptions about SNARKs (and why they hold us back). https://a16zcrypto.com/posts/article/17-misconceptions-about-snarks, 2023. Accessed: 2024-12-29. 6

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008. 5

[WTS+18] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 926–943. IEEE Computer Society, 2018. 3

[XZZ+19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara,*

CA, USA, August 18-22, 2019, Proceedings, Part III, volume 11694 of *Lecture Notes in Computer Science*, pages 733–764. Springer, 2019. 3

[ZGK+17] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 863–880. IEEE Computer Society, 2017. 3

[ZLW+21] 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 Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 159–177. ACM, 2021. 3

A Proof of Lemma 8

The result of this appendix seems to be largely parallel to Kleene's work. However, the specific form of the statements that we need (with explicit bounds on size and runtime) seems to be dependent on more subtle properties of the programming language: there are languages that only contain f-quines with code size much larger than code size of f.

In the following, we will discuss partial computable functions. These functions correspond to Turing machines, and return $\perp$ when the Turing machine halts. Functions of multiple arguments are interpreted as machines on a single tape, with arguments bounded by a special separator symbol.

Recall that for a program p we denote by p the partial computable function that corresponds to it and $[p]$ it's code (as defined below):

Definition 11. *The mapping from Turing machines to strings $p \mapsto [p]$ is called acceptable programming system if*

1. *The mapping $p \mapsto [p]$ is computable.*
2. *There is a family of universal partial computable functions φ^k satisfying*
$$\varphi^k([p], x_1, ...x_k) = p(x_1, ..., x_k)$$
3. *(Kleene's S-m-n theorem) There is a "computable currying" function S_n^m. Given $[p]$ code of a program with $m+n$ arguments it outputs a code of a program $[q] = S_n^m([p], x_1, ..., x_m)$ such that*
$$\forall y_{m+1}, ..., y_{m+n} : q(y_{m+1}, ..., y_{m+n}) = p(x_1, ..., x_m, y_{m+1}, ..., y_{m+n})$$

Under these conditions, we have Kleene's second recursion theorem:

Theorem 12 (Kleene's second recursion theorem)**.** *For any program Q taking two arguments there exists p such that*
$$\forall y : p(y) = Q([p], y).$$

Proof Sketch. Define the program u that does:

$$u(x, y) = Q(S_1^1(x, x), y).$$

Now, set

$$[p] = S_1^1([u], [u])$$

Then, $p(y) = u([u], y) = Q(S_1^1([u], [u]), y) = Q([p], y)$. □

However, these theorems don't tell us anything about the size or runtime of operations involved. Therefore, we strengthen the definitions a bit:

Definition 13. *An admissible programming system $\{\varphi_k, S_n^m\}$ is called efficiently reflexive if the following additional set of conditions holds:*

1. *Runtime of $p \mapsto [p]$ is polynomial in $|p|$.*
2. *The runtime of $\phi_k([p], x_1, ..., x_k)$ is polynomial in runtime of $p(x_1, ..., x_k)$.*
3. *The runtime of S_n^m is polynomial in size of it's arguments.*
4. *(Currying with constant overhead) Size of $[q] = S_n^m([p], x_1, ..., x_k)$ satisfies*

$$|[q]| = |[p]| + |x_1| + ... + |x_k|$$

5. *There is a composition operation, with code $[g \circ f]$ size $|[g]| + |[f]| + \mathsf{const}$, and polynomial runtime.*

Most realistic programming systems do satisfy these requirements. *Note: we think that the last requirement is not independent.*

We wish to prove the following "decompression" lemma, which states that short representations of programs lead to short implementations of programs.

Lemma 14. *Assume we have a family of strings $\{s\}$ (which represent compressed representation of programs) and family of program codes $\{[p]\}$ recovered by some computable decompression mapping:*

$$d(s) = [p]$$

We claim that p-s have an implementation with sizes $|s| + \mathsf{const}$, and at most polynomial slowdown in running time.

Proof. Idea is very simple: first, decompress the string and then run the interpreter. Specifically, the code of implementation for p is

$$S_1^1([\varphi_2], S_0^1([d], s))$$

Set of axioms that we have applied bounds this code size as $|[d]| + |s| + |[\varphi_2]| + \mathsf{const} = |s| + \mathsf{const}$. □

We are now ready to prove Lemma 8. Recall that the lemma asserts that any computable function f has an f-quine with size $|[f]| + \mathsf{const}$, and polynomial execution time in $|[f]|$ provided f execution time is polynomial in input size.

Lemma 15.

Proof of Lemma 8. Classical quines are obtained by setting $Q(x, y) = x$ in the setting of Theorem 12. Then this theorem finds p such that $p(y) = [p]$. To obtain an f-quine p we instead consider $Q(x, y) = f(x)$.

Then, Theorem 12 provides an f-quine. Moreover, given that the execution time of f was polynomial in input, p execution time is polynomial time in $|f|$. Indeed, explicitly:

$$u(x, y) = f(S_1^1(x, x))$$
$$[p] = S_1^1([u], [u])$$

Then, executing $p(\_)$ means running an interpreter $\varphi_2([u], [u], \_)$. That means (interpreted) execution of f on an argument $S_1^1([u], [u])$, which is polynomial size in $[u]$, which itself is a composition of f and S_1^1.

However, the size of f-quine constructed is too large. We use Lemma 14 to decrease it. Indeed, we can use $[f]$ as a description of the quine being constructed. By decompression lemma, we obtain an implementation of f-quine with size bounded by $|[f]| + \mathsf{const}$

□