A Report On (In)security of Fiat-Shamir Transformation in Practice

Course: Topics in Cryptology
Instructor: Prof. Shahram Khazaei

Prepared by:
Arman Keshazar
Department of Mathematical Sciences
Sharif University of Technology

September 13, 2025

Abstract
The Fiat-Shamir transformation is a cornerstone technique which enables the conversion of public-coin interactive protocols into efficient non-interactive protocols. Its widespread use in modern systems, particularly cryptocurrencies and SNARKs, has motivated extensive study of its theoretical foundations and practical security. Although its security is established in the random oracle model, there are several sources of vulnerabilities when used in practice. This report surveys four recent papers [DMWG23, NHB24, KRS25, AY25] that address vulnerabilities arising in the gap between theory and practice, ranging from implementation and design errors to issues caused by instantiating the random oracle with a white-box hash function, and provides solutions and mitigation strategies.

1 Introduction

The FS transform is an incredibly influential paradigm in cryptography. Originally suggested as a method of converting an ID scheme into a digital signature, nowadays it is broadly used to convert general public-coin protocols into non-interactive ones.
Its core idea revolves around replacing the verifier's random coin tosses with the evaluation of a complex hash function. This way verifier's messages become deterministic. Hence there is no need for interaction.
The transformation's ease of use and its wide applicability are part of the reason why there is not enough discussion on how to implement FS in different protocols. This lack of discussion has caused surprisingly many imprecise and wrong deployments of FS in practice.
Since in probabilistic proof systems, knowing the verifier's challenges beforehand usually breaks the soundness and assists the malicious prover in forging, FS implementation flaws can result in total breach of security.
Another notable thing is that FS transformation allows rewinding and repeating. A malicious prover, in contrast with the interactive setting, is not punished for a failed attempt at forging a proof. Therefore the concrete analysis of schemes is utterly crucial as brute-force attacks are not costly.
FS in the wild [NHB24] tries to classify some of the common pitfalls. We will explain these risks and provide examples of how they can affect us.
The most common implementation flaw must be the use of weak-FS transformation. This is when the public information such as statement is not contained in the hashing context. Intuitively, hashing public information ensures that generated challenges depend on the public information, preventing the malicious prover from adaptively choosing it during proof generation.
Bernhard et al. [BPW12] highlighted this issue and devised attacks on two classic proof systems, Schnorr and Chaum-Pedersen. However, despite this and similar prior works, little is known about dangers of weak-FS in modern proof systems.
[DMWG23] studies the risks of weak-FS in several modern proof systems. They survey over 75 open-source proof system implementations that use FS, uncovering 36 weak-FS implementations across 12 different proof systems. We will discuss two of these proof systems.

Until now we have only considered implementation and design flaws. So provided we avoid these pitfalls, will our protocols remain secure? Pointcheval and Stern [PS96] showed that

the FS transform is sound in the random oracle model, where hash function is treated as a truly random function. Yet in practice we instantiate RO with a concrete hash function, since implementing a true RO requires exponential space.
Barak [Bar01] and Goldwasser and Kalai [GK03] gave examples of secure interactive protocols for which FS transform results in an insecure non-interactive protocol, *no matter what concrete hash function is used.* Still, all these counter-examples were contrived protocols designed to fail. Khovratovich et al. [KRS25] show 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.
The mentioned attack and many more fall into the cluster of *diagonalization* attacks. Arnon and Yogev [AY25] design and suggest XFS(extended FS) as an alternative for FS transformation with little computation overhead and prove it secure against this family of attacks.

2 Design and Implementation Pitfalls

Figure 1 is the classification given in [NHB24]. We discuss them in order;

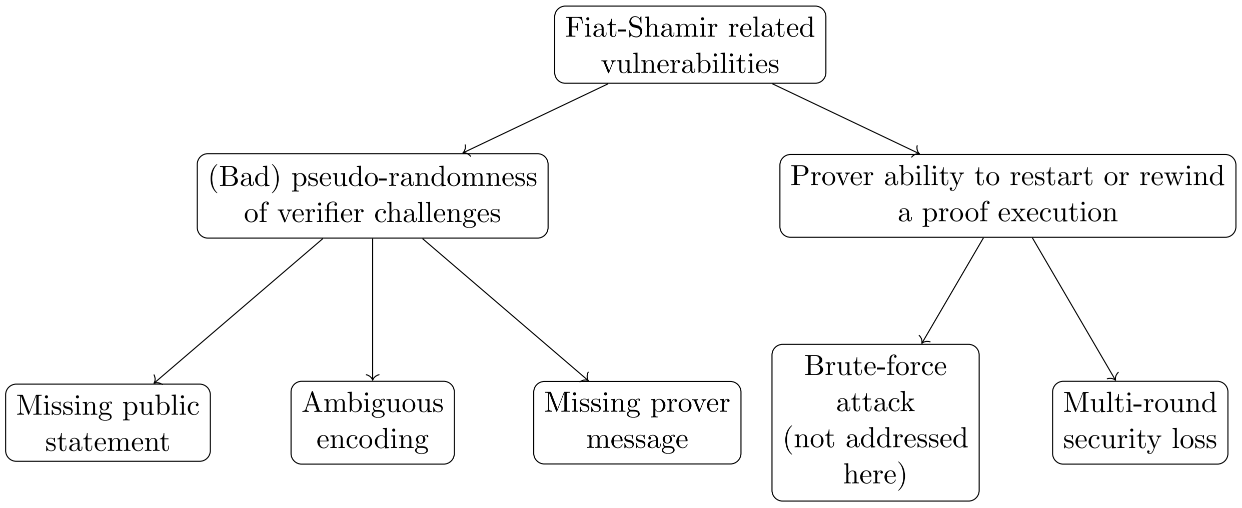
Fiat-Shamir related vulnerabilities
(Bad) pseudo-randomness of verifier challenges
Prover ability to restart or rewind a proof execution
Missing public statement
Ambiguous encoding
Missing prover message
Brute-force attack (not addressed here)
Multi-round security loss

Figure 1: A classification of Fiat-Shamir-transformation-related vulnerabilities.

2.1 Missing public statement

This type of vulnerability, known as the weak variant of Fiat-Shamir transformation, has been extensively studied in [DMWG23]. In scenarios where prover is to pick the statement, she could exploit this vulnerability by generating a transcript and then finding a statement which would make the transcript accepted(as the verifier challenges remain intact).
If proving some random false statement is dangerous in a protocol, then it must be carefully examined for this weakness.

2.1.1 Case study: PLONK

PLONK [GWC19] (Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge) is a general-purpose SNARK introduced in 2019 that has quickly become a foundation for real-world applications. At a high level, PLONK encodes an arbitrary computation as an arithmetic circuit and reduces the task of proving correct execution to a set of

low-degree polynomial identities. The protocol is structured as a polynomial IOP (Interactive Oracle Proof), where the prover commits to certain polynomials and the verifier issues random challenges to enforce consistency checks, including a critical permutation argument that enforces correct wiring of the circuit.
We will not dwell on the details and content ourselves with the technicalities required. See Figure 2 for a simplified version of the FS-transformed PLONK prover.

1) Compute 3 wire polynomials a(X), b(X), c(X) and output their polynomial commitments. During an honest execution these polynomials are constructed from the values associated with the wires of the circuit.
2) Derive permutation challenges β, γ compute the permutation polynomial z(X) and output its polynomial commitment.
3) Derive a quotient challenge α, compute the quotient polynomial t(X)(as described in Equation 1 below), and output its polynomial commitment.
4) Derive an evaluation challenge ξ, evaluate the committed polynomials at the desired points (ξ and others), and output the evaluations together with their proofs of correctness.

Figure 2: simplified FS-transformed PLONK prover overview

Finally, PLONK verifier checks that:

- Polynomial evaluations are correct.

- Below polynomial constraint holds at ξ:

$$\begin{aligned} t(X)Z_H(X) = a(X)b(X)q_M(X) + a(X)q_L(X) + b(X)q_R(X) \\ + c(X)q_O(X) + q_C(X) + PI(X) \\ + \alpha\big(a(X) + \beta X + \gamma\big)\big(b(X) + \beta k_1 X + \gamma\big)\big(c(X) + \beta k_2 X + \gamma\big)z(X) \\ - \alpha\big(a(X) + \beta S_{\sigma 1}(X) + \gamma\big)\big(b(X) + \beta S_{\sigma 2}(X) + \gamma\big) \\ \cdot \big(c(X) + \beta S_{\sigma 3}(X) + \gamma\big)z(Xw) \\ + \alpha^2(z(X) - 1)L_1(X) \end{aligned} \tag{1}$$

where PI(X) is the polynomial constructed from public wire values of the circuit. H is the $< w >$ subgroup and $Z_H(X) = \prod_{h \in H}(X - h)$. Other unmentioned terms are either public or circuit-related and known to both the verifier and the prover. For example, q_M, q_L, q_R, q_O, q_C are defined by the circuit.

The issue is fairly obvious: PI(X) is not part of the hashing context. So changing PI(X) does not change any other part of the Equation 1(the challenges remain the same). The malicious prover can directly exploit this vulnerability by choosing a(X), b(X), c(X), z(X) and t(X) totally random and picking PI(X) in a way that the equation holds at ξ.
Of the 12 implementations surveyed in [DMWG23], they found 5 to be vulnerable. One them

being Dusk Network, a privacy preserving distributed ledger protocol, was exposed to unlimited money issuance (theft from the system).

2.1.2 Case study: Wesolowski's VDF

A VDF(Verifiable Delay Function) is a function whose output should be known only after a certain time delay, and also contains a proof of correct evaluation. We can define it as triple of algorithms:

- $setup(1^\lambda) \to pp$ which sets the public parameters
- $eval(pp, T, x) \to (\pi, y)$ which evaluates the VDF with time delay T, returning proof and output
- $verify(pp, T, x, \pi, y)$ which verifies the proof

We briefly describe the Wesolowski's non-interactive protocol [Wes20]. It contains two hash functions. One is H_G which takes bit string as input and outputs an element of the group G. And the other one is H_{prime} which is used for FS instantiation and outputs prime numbers of length 2λ bits where λ is the security parameter. Figure 3 for an overview of VDF prover algorithm.

1) Compute y. During an honest execution $y = g^{2^T}$ where $g = H_G(x)$.
2) Derive prime challenge $p = H_{prime}(x, y)$. Compute $\pi = g^{\lfloor 2^T/p \rfloor}$ and output (π, y).

Figure 3: weak FS-transformed VDF prover

Finally, the verifier only computes the residue $r = 2^T \bmod p$ where p is the output of $H_{prime}(x, y)$ and accepts if $\pi^p g^r = y$.
Note that time delay T had no contribution in deriving the challenge, p. Hence, the malicious attacker could easily compute an honest proof (x, t, y, π) where t is small. This proof is accepted for any larger T with the same residue, $2^T = 2^t \bmod p$; for example $T := t + p - 1$. With high probability $y \neq g^{2^T}$, otherwise, we know that $g^{2^T - 2^t} = 1$, which breaks low order assumption of group G contradicting results of [BBF18].
In conclusion we could exploit weak-FS vulnerability and break the soundness of Wesolowski's VDF.

2.2 Missing prover message

Missing prover message could be exploited the same way weak-FS was. The core vulnerability is the malicious prover changing or rewinding her messages without affecting the derived challenges.
A vulnerable implementation of PLONK protocol was reported in [NHB24]. We will not repeat the basics of PLONK (Figure 2 and Equation 1). Nguyen et al. [NHB24] have provided the summary of challenge derivation of the insecure code snippet [1]

[1]There are two more challenges in [NHB24]'s summary. However we trim them as they seem unnecessary.

- $\beta = H\big(\text{public inputs, commitments of } a(X), b(X), c(X)\big)$
- $\gamma = H(\beta)$
- $\alpha = H\big(\text{commitment of } z(X)\big)$
- $\xi = H\big(\text{commitment of } t(X)\big)$

The malicious attacker's goal is to satisfy Equation 1 when $X = \xi$ for any PI(X). Due to the flawed FS, she can:

1. Pick arbitrary t(X) and compute ξ, depending only on t(X).
2. Choose z(X) such that it equals 0 at ξ and ξw. Now β and γ are cut off from 1 and α is computable from z(X).
3. Finally, pick appropriate a(X), b(X), c(X) such that Equation 1 is satisfied. Since changing these polynomials does not change any of the computed challenges.

As you can see, the flaw enabled us to avoid computing sequentially and we were given access to challenges up ahead. In some sense, instead of solving an equation in a high dimension space with variables highly depending on each other, we separated the main task into solving several easier problems each of them in a much smaller space size.
This example reminds us the importance of including previous rounds' challenges in the hashing context.

2.3 Ambiguous encoding and Multi-round security loss

We group two classes of [NHB24] as we believe this is adequate attention.

Ambiguous encoding. Deriving the challenge using Fiat-Shamir sometimes involves the unsaid step of encoding, applying a method to the *object* so it is digestible for the hash function.

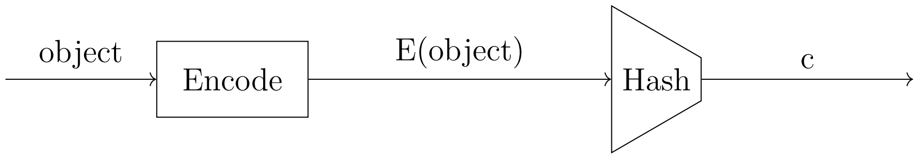

Figure 4: Applying the Fiat-Shamir transformation involves an encoding scheme.[NHB24]

Whenever the coding scheme is ambiguous, meaning that different objects have the same encoding, the whole FS transformation is not collision resistant. The naivest attempt at exploiting this vulnerability could be the prover swapping her messages with their collision, not affecting any derived challenge.

Multi-round security loss. The root cause of certain Fiat–Shamir vulnerabilities in multi-round protocols lies in the prover's ability to rewind to any previous round while keeping earlier challenges fixed. This "state restoration" capability contrasts with the interactive setting, where all challenges must be regenerated from the start, and can degrade soundness from negligible to non-negligible.

The most famous examples are the sequential repetition of a round with constant soundness in interactive settings. It is known that applying Fiat-Shamir allows a malicious prover to brute-force challenges round-by-round, requiring only polynomial effort to forge a proof.
However, there are more refined analyses in the literature:

- *Round-by-round soundness* introduces the concept of doomed state, a partial transcript where a malicious prover has a negligible chance of winning. And a partial transcript is not doomed if the probability of transitioning into a doomed state is negligible.
- *State restoration soundness* notices the ability of rewinding in some contexts. Hence, in interactive setting enables the prover to repeatedly asking for new challenge in each round, addressing the vulnerabilities introduced in other settings ie. non-interactive.
- Round-by-round soundness implies state-restoration soundness.

For a thorough and more explanatory discussion, see Chiesa and Yogev's book [CY24, Chapter 31].

3 White-box instantiation of random oracle

Many protocols are proven secure in the random oracle model (ROM). In practice, however, random oracles must be instantiated with concrete hash functions. For the majority of the protocols, security cannot be formally guaranteed in this standard setting, and one merely assumes that the properties established in the ROM carry over to the white-box instantiation setting.
Khovratovich, Rothblum, Sukhanov[KRS25] present an attack on a widely-used, standard, succinct argument-system for proving the correctness of a computation expressed by a non-deterministic bounded depth arithmetic circuit. This protocol is a combination of a multilinear polynomial commitment scheme (MLPCS) with Goldwasser, Kalai and Rothblum GKR [GKR08] scheme.

3.1 Preliminaries

We briefly discuss required concepts and tools: [2]

3.1.1 Multilinear extension

For a finite field $\mathbb{F}$, the multilinear extension of a function $f\colon \{0,1\}^m \to \mathbb{F}$ is the (unique) multilinear polynomial, multivariate polynomial that is linear in each of its variables separately, $\hat{f}\colon \mathbb{F}^m \to \mathbb{F}$ that agrees on $\{0,1\}^m$ with f.
Note that for a vector $y = (y_0, y_1, \ldots, y_{m-1}) \in \mathbb{F}^m$, we denote by $\hat{y}$ the *multilinear extension* of the function

$$f : \{0,1\}^{\log m} \to \mathbb{F}, \qquad f(i) = y_i \text{ for all } 0 \le i \le m-1.$$

3.1.2 MLPCS

It is a commitment scheme allowing the user to give short commitments to large polynomials, and provide succinct proofs for evaluation queries of the form $P(x) = y$.

[2]Majority of these explanations are borrowed from[KRS25]

3.1.3 Vanilla GKR

This protocol is a doubly-efficient interactive proof for verifying the correctness of bounded-depth computations. Here we consider the simple GKR for deterministic computations and content ourselves with a high-level discussion.
The claim is of the form C(x)=y, where C is a low-depth arithmetic circuit over a finite field $\mathbb{F}$. All of the C, x, y are known to the verifier. We can break the protocol into the following steps:

1. The verifier evaluates the multilinear extension of the output y at a random point.
2. (Processing circuit layers) The core idea of the protocol is instead of verifying the whole circuit at once, it peels the circuit one layer at a time. It reduces a claim about MLE of layer i to a claim about MLE of layer i+1(where output is layer 0), using the sumcheck protocol[LFKN92].
3. After processing all of the layers, the verifier is left with a claim about MLE of the input layer, which she can compute herself.

Remark. Noting the structure of GKR, it is evident that for a composed circuit $C(x) = C_2(C_1(x))$ the protocol can be thought of concatenation of two GKR protocol. First one reducing a claim about MLE of the output of $C_2(x)$ to a claim about an input z for the circuit, and then viewing z as a claim about the output of $C_1(x)$ and reducing it to a claim about MLE of x.

3.1.4 GKR for non-deterministic computation

Using vanilla GKR and an MLPCS, a computationally sound argument for non-deterministic computation can be made.
In this setting, we aim to verify that, for a given depth d arithmetic circuit $C\colon \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^l$, input $x \in \{0,1\}^n$ and output $y \in \mathbb{F}^l$, there exists a witness $w \in \mathbb{F}^m$ such that $y = C(x, w)$. We call this protocol $\prod_{d,comm}$, where comm is the used MLPCS.

1. (Preprocessing) The verifier chooses the public parameters eg. comm. The prover chooses a circuit $C\colon \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^l$ and gives it to the verifier. The verifier keeps a short digest of the circuit which we denote by $\langle C \rangle$.
2. (Commitment) The prover chooses $x \in \{0,1\}^n$, $y \in \mathbb{F}^l$ and $w \in \mathbb{F}^m$. All three of x, y and $\alpha = comm(w)$ are sent to the verifier.
3. (Output claim) The verifier specifies a random $r \in \mathbb{F}^{\log l}$ and computes a claim on $\hat{y}(r)$(Observe that since y has length l, then its MLE is function with $\log l$ variables.)
4. (Vanilla GKR) The prover and verifier take part in a run of vanilla GKR protocol to reduce the claim of $\hat{y}(r)$ to claims about $\hat{x}$ and $\hat{w}$.
5. (PCS evaluation) The verifier checks the claim about $\hat{x}$ herself and checks the claim about $\hat{w}$ by using the evaluation phase of the MLPCS.

3.2 Attacking the protocol

Deriving a non-interactive argument from $\prod_{d,comm}$ using FS transformation is straightforward. Let's call it $FS_h(\prod_{d,comm})$, where h is the hash function used in the FS transformation. [KRS25] give attacks to this scheme proving it adaptively unsound.

3.2.1 Proving a false statement

Let's assume in $FS_h(\prod_{d,comm})$, comm and h are computable with circuits of depth, d_{comm} and d_h. Khovratovich et al. [KRS25] construct a circuit C^* of depth $d_{comm} + d_h + O(1)$, which only gets $w \in \mathbb{F}^m$ as a witness and nothing else as input. Intuitively, they enable C^* to predict the randomness used by the verifier in step 3 of the GKR protocol, $r = h(\langle C^* \rangle, y^*, \alpha)$ where α is a commitment to some witness and y^* is the claimed output.
The obvious obstacle here is the circular nature of the problem, if you change C^* in anyway to predict r, then r changes as well. They overcome this issue by providing C^* with its own digest $\langle C^* \rangle$ as the witness.

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

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

Note that C^* has depth $d_{comm} + d_h + O(1)$.
Now they construct an accepting proof string π for the claim that there is a $w \in \mathbb{F}^m$ such that $C^*(w) = y^*$, where $y^* = (0, 0)$. Since by construction the two output elements of C^* are distinct, this claim is clearly false. The forged proof π is constructed essentially honestly, relative to witness $w = \langle C^* \rangle$:

1. Let $w = \langle C^* \rangle$, and $\alpha = comm(w)$.
2. Engage in an honest run of $FS_h(\prod_{d,comm})$ and denote the transcript with τ.
3. Output the proof string $\pi = (\alpha, \tau)$.

Now consider the verification process. Let y be the real output of circuit C^* on input $w = \langle C^* \rangle$. By construction the circuit outputs $y = (\gamma - 1, \gamma)$, where $\gamma = h(w, y^*, \alpha)$. The multilinear extension of y is simply $\hat{y}(X) = \gamma - X$. By construction

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

Thus evaluation of $\hat{y}$ at point r equals to $\hat{y}(r) = \gamma - r = 0$. Additionally, since $y^* = (0, 0)$ its MLE is constantly 0.
Due to the equality of $\hat{y}$ and $\hat{y^*}$ on the challenge point, from here on the claim being proved is *true* (in the round-by-round literature we can say prover moved from a doomed state to an accepting state). So she can simply run the honest prover strategy to the correct claim of $\hat{y}(r) = 0$.
This shows that $FS_h(\prod_{d,comm})$ is not adaptively sound.

3.2.2 Beyond certain functionalities

One might hope to mitigate the given attack by allowing only certain functionalities. However, in [KRS25] they extend the attack to arbitrary functions. Given any circuit C, they insert a "backdoor" in it so the functionality does not change but they can prove any desired statement. Let $C(x, w) = y \colon \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^l$ be a depth d circuit with a large enough m. For any $x^* \in \mathbb{F}^n$ and $y^* \in \mathbb{F}^l$ they show how to construct a circuit C^* that is functionally equivalent to C, but for which they can prove that there is a w such that $C^*(x^*, w) = y^*$.

They use a fixed *IF-THEN-ELSE*: $\mathbb{F}^{1+1+l+l} \to \mathbb{F}^l$ circuit, such that for arbitrary a,b,c,d *IF-THEN-ELSE*$(a,b,c,d) = (a-b).(c-d)+d$. They only feed them inputs that satisfy $a \in \{b, b+1\}$.

Construction 2. The circuit $C^*(x,w)$ is defined as follows:

$$\left.\begin{array}{l} \text{1. Compute } y = C(x,w). \\ \text{2. Compute } \gamma = g(w,y) \in \mathbb{F}. \\ \text{3. Compute } u_{real} = (\gamma, \gamma-1, y, y^*). \end{array}\right\} C_1(x,w)$$

$$\left.\begin{array}{l} \text{4. Output } \textit{IF-THEN-ELSE}(u_{real}). \end{array}\right\} C_2(u_{real})$$

where the function $g(w,y)$ is defined as follows:

1. Interpret w as a circuit digest ψ.
2. Compute $\alpha = comm(\psi)$.
3. Generate the claim about MLE of y^* as in the GKR prtocol(using randomness $r = h(\psi, x^*, y^*, \alpha)$.
4. Let $u_{fake} = (0,0,y,y^*)$ and observe that *IF-TEHN-ELSE*$(u_{fake}) = y^*$. Simulate the GKR protocol on the *IF-THEN-ELSE* circuit to reduce the claim about the MLE of y^* at point r to a claim about the MLE of u_{fake} at a point r', with the only exception of using ψ as the circuit digest when deriving verifer challenges.
5. Output the first coordinate of r'.

Note that circuit $C^*(x,w)$ is illustrated as the combination of $C_2(C_1(x,w))$.

Remark. It is fairly easy to check that by construction for every x and w the output of C^* equals output of C. So the two circuits are **functionally equivalent**.

Let's see how the malicious prover forges a proof π for $C^*(x^*,w) = y^*$):

1. The prover sends the constructed $C^*, x*, y^*, \alpha = comm(\langle C^* \rangle)$ to the verifier.
2. She engages in an honest prover strategy of GKR for the correct *IF-THEN-ELSE*$(u_{fake}) = y^*$ computation. Hence, reduces the claim about $\hat{y}(r)$, where $r = h(\langle C^* \rangle, x^*, y^*, \alpha)$, to a claim about the $\widehat{u_{fake}}(r') = v$.

   We claim that
   $$\widehat{u_{fake}}(r') = \widehat{u_{real}}(r') \tag{2}$$
3. So the prover runs the GKR protocol for the correct computation $C_1(x^*,w) = u_{real}$.

Proof of Equation (2)**.** The crucial point is that, by the definition of g, the first coordinate of r' equals $\gamma = g(w,y)$.
Now we show that for every $z \in \mathbb{F}^{1+\log(l+1)}$ it holds that:

$$\widehat{u_{real}}(z) - \widehat{u_{fake}} = (\gamma - z_0) \cdot \prod_{i=2}^{\log(l+1)} (1 - z_i).$$

We only need to check for $z \in \{0,1\}^{1+\log(l+1)}$ as both sides of the equation are multilinear polynomials.

For $z = 0^{1+\log(l+1)}$ the equation evaluates to $\gamma - 0 = \gamma \cdot \prod_{i=2}^{\log(l+1)} 1$. For $z = 10^{\log(l+1)}$ the equation evaluates to $(\gamma - 1) - 0 = (\gamma - 1) \cdot \prod_{i=2}^{\log(l+1)} 1$. For any other z both side equal to zero ($\widehat{u_{real}}(z) = \widehat{u_{fake}}$ since these sub-vectors $u_{real}[2, 1 + \log(1+1)] = u_{fake}[2, 1 + \log(1+l)]$).
Equation (2) now follows from the fact that the first coordinate of r' is γ, and so the difference of the two MLE is 0.

3.2.3 Universal computation attack

One might hope that to deflect the given attacks, it suffices to ensure that the GKR circuit in question does not contain an implementation of the FS hahs function or MLPCS.
However, in [KRS25] they show that by feeding the code of some f-quine, a program p that outputs *f([p])* where [p] is program's code, to the universal turing machine circuit they can mount similar attacks.

4 Extended Fiat-Shamir(XFS)

As stated earlier, in contrast to previous white-box instantiation attacks, Khovratovich et al.[KRS25]'s attacks require immediate attention since they target a widely-used protocol. To mitigate the attacks, they suggested using a hard-to-compute hash functions for the FS transformation so that the circuits supported by the protocol cannot compute it. Though this approach theoretically works, the whole concept of computation delegation is forgotten since verifier has more power than any circuit under computation by the prover.
To this end, Arnon and Yogev [AY25] proposed XFS, an alternative to the FS, and provide reasons why it holds secure against [KRS25] or similar attacks.

4.1 Core vulnerability

First, let's study a simple sigma protocol which admits an attack similar to the GKR one. This will help us identify the core vulnerability and provide a classification for these attacks.

4.1.1 Toy protocol

In [AY25] this toy sigma protocol is illustrated which is mainly inspired by Barak's protocol [Bar01]:

1. (Preprocessing): The prover provides a circuit $C\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^{r+1}$ which is preprocessed and the digest ψ is sent to the verifier.
2. Prover sends x, y, and a commitment of the witness $m_1 = comm(w)$.
3. Verifier replies with a challenge $\rho \leftarrow \{0,1\}^r$. ($\rho \leftarrow h(\psi, x, y, m_1)$ if it is compiled with FS)
4. Prover sends C and w.
5. (verifier's decision): The verifier accepts if ψ is the digest of C, $w = comm(w)$, and either $C(x, w) = y$ or $C(x, w) = (0||\rho)$.

It's obvious that the toy protocol serves no practical purpose since the verifier computes the circuit C herself. The only divergence from the trivial protocol is that the verifier accepts if the circuit can *guess the random challenge* and this does not affect soundness in the interactive setting. However, that is exactly the weakness our attack exploits when toy protocol is compiled with FS transformation.

4.1.2 Attacking the non-interactive toy protocol

Suppose using hash function h for the FS, the above protocol is transformed to a non-interactive protocol.
Fix a commitment scheme *comm* and hash function h. For $\mathbb{y} = 1^{r+1}$, consider the circuit $\tilde{C}(x, w) = \big(0||h(w, x, \mathbb{y}, comm(w)\big)$. The malicious attacker runs the honest prover strategy with $\tilde{C}$, arbitrary x, $\mathbb{y} = 1^{r+1}$, and $\mathbb{w} = \psi$ where ψ is the digest of $\tilde{C}$.
Since all outputs of $\tilde{C}$ start with 0, then obviously $\tilde{C}(x, w) \neq \mathbb{y}$ and the statement is false.
However $\tilde{C}(x, \mathbb{w}) = \big(0||h(\psi, x, \mathbb{y}, comm(\psi))\big) = (0||\rho)$, hence the verifier accepts the false statement.

4.1.3 Classification of the attack

Arnon and yogev [AY25] suggest that the toy protocol mimics the weakness of GKR protocol, which is:

- Both protocols handle general-purpose computations.
- Circuits have access to the hash function(random oracle).
- There exists circuits that can predict the *next-verifier-message*.

As illustrated in these two cases and in almost any interactive proof, prover's access to the verifier upcoming *random* challenges would break the soundness property.

4.2 PoW to the rescue

Authors' brilliant extension to the FS transformation, makes it infeasible for the circuit to compute the random challenge. They require the prover to submit a solution to a proof-of-work puzzle in order to derive the challenge. A puzzle too hard for the circuit to solve and easily verifiable so the verifier remains efficient.

4.2.1 Formalizing PoW

A proof-of-work is a tuple (Setup, Samp, Solve, Check) that satisfies:

- Completeness. For every $\lambda, \ell \in \mathbb{N}$,

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

- Soundness. For every $\lambda, \ell \in \mathbb{N}$, and adversary $A = (A_1, A_2)$ such that A_1 is t_1-time bounded and A_2 is t_2-time bounded,

$$\Pr\left[\text{Check}(pp, z, s) = 1 \;\middle|\; \begin{array}{l} pp \leftarrow \text{Setup}(1^\lambda, \ell) \\ aux \leftarrow A_1(pp) \\ z \leftarrow \text{Samp}(pp) \\ s \leftarrow A_2(pp, aux, z) \end{array}\right] \leq \epsilon_{Pow}(\lambda, \ell, t_1, t_2).$$

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

The constructions of PoW of our concern satisfy $t_2 < \ell << t_1$, efficient Check, and negligible ϵ_{PoW} for $t_1 = poly(\lambda)$ since in our schemes, prover plays the role of A_1, verifier needs to run Check, and the circuit is A_2.

4.2.2 XFS for sigma protocols

Given a 3-round interactive protocol (I, P, V), and a proof-of-work (Setup, Samp, Solve, Check) and a hash function h, XFS transformed protocol in the standard model is:

- $pp \leftarrow Setup(1^\lambda, \ell)$.
- $\psi \leftarrow I(\mathbb{i})$.
- $\mathcal{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$:
  1. Compute digest: $\psi \leftarrow I(\mathbb{i})$.
  2. Get prover first message: $m_1 = P(\mathbb{i}, \mathbb{x}, \mathbb{w})$.
  3. Derive puzzle randomness: $r = h(\psi, \mathbb{x}, m_1)$.
  4. Compute puzzle: $z = Samp(pp, r)$.
  5. Solve puzzle: $s \leftarrow Solve(pp, z)$.
  6. Set padding for example $\tau = 0^{|V|}$, so that next query is separated from the verifier query domain.
  7. Derive verifier message: $\rho = h(\tau, \psi, \mathbb{x}, m_1, s)$.
  8. Get the second prover message: $m_2 = P(\mathbb{i}, \mathbb{x}, \mathbb{w}, \rho)$.
  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: $z = Samp(pp, r)$.
  3. Set the padding $\tau = 0^{|V|}$.
  4. Derive verifier message: $\rho = h(\tau, \psi, \mathbb{x}, m_1, s)$.
  5. Accept if $V(\psi, \mathbb{x}, m_1, \rho, m_2) = 1$ and $Check(pp, z, s) = 1$.

After the circuit $\mathbb{i}$ has been chosen by the prover, preprocessing algorithm selects $\ell > |\mathbb{i}|$, where $|\mathbb{i}|$ is the size of the circuit. Hence, characteristics of PoW guarantee that the circuit $\mathbb{i}$ cannot compute the challenge since puzzle is harder that its computation power.

4.2.3 PoW construction in the ROM

Arnon and Yogev [AY25] present two constructions of PoW schemes, both in the random oracle model. We will discuss the second construction which is more optimized as well.

Construction (Practical PoW). Let $n, m \in \mathbb{N}$ be parameters such that $m \cdot n = \Omega(\ell)$ and f is the random oracle.
In the ROM, Setup would be just choosing appropriate n, and m and Samp would be just picking a random λ bit string.

- $Solve^f(\ell, z)$:
    1. By checking random strings, find a set $(s_1, ..., s_n)$ such that for every $i \in [n]$, $y_i = f(z, i, s_i)$ begins with logm zeros.
    2. Output $(s_1, ..., s_n)$.
- $Check^f(\ell, z, s)$:
    1. Parse $s = (s_1, ..., s_n)$ (if not possible reject).
    2. Compute $y_i = f(z, i, s_i)$.
    3. Verify that for all $i \in [n]$, y_i begins with logm zeros (and reject otherwise).

It is fairly obvious that the naive solver who plainly checks new ss till the list is complete, has expected complexity ℓ.

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

$$\epsilon_{PoW}(\lambda, \ell, t_1, t_2) \leq \frac{t_1}{2^\lambda} + \frac{1}{2^{0.64n}}.$$

Proof. Consider any adversary $A = (A_1, A_2)$. We first construct a modified adversary A_2' such that (A_1, A_2') succeeds with the same probability as (A_1, A_2). The adversary A_2' is defined to behave like A_2, with the following restrictions:

- A_2' never performs duplicate queries.
- A_2' executes the Check procedure at the end of its execution.
- A_2' makes exactly $t_2' := \ell/4$ queries.
- All queries of A_2' begin with the given puzzle z.

It is straightforward to realize such an A_2'. We prevent duplicate queries by storing all previous answers. We explicitly invoke Check at the end of the execution (this requires n queries). If needed, we then perform arbitrary distinct queries (all beginning with z) to reach $\ell/4$ total queries. Finally, any query not beginning with z can be answered internally at random, since Check never relies on such queries.

Now consider the combined adversary (A_1, A_2'). Let tr_1 denote the trace of queries (of length t_1) made during the first phase by A_1. Define event E_1 as the existence of some $i \in [n]$ and

$\gamma \in \{0,1\}^\lambda$ such that $(z, i, \gamma) \in tr_1$. Since z is sampled uniformly at random from $\{0,1\}^\lambda$ *after* A_1 has executed, we have

$$\Pr[E_1] \leq \frac{t_1}{2^\lambda}.$$

We henceforth condition on $\bar{E_1}$.

In the second phase, consider the queries of A_2'. For each $j \in [t_2']$, let X_j be the indicator random variable that the j-th query returns an answer beginning with $\log m$ zeros. Since all queries must begin with z, none of them appear in tr_1 and there are no duplicates. Therefore,

$$\Pr[X_j = 1] = \frac{1}{m}.$$

Let $X = \sum_{j \in [t_2']} X_j$. These random variables are independent, and hence

$$E[X] = \frac{t_2'}{m} = \frac{\ell}{4m} = \frac{n}{4}.$$

Finally, let event E_2 denote that, at the end of the second phase, for every $i \in [n]$ the trace contains a query of the form (z, s_i) such that $f(z, s_i)$ begins with $\log m$ zeros.

Applying a Chernoff bound for $\delta = 3$, we obtain

$$\begin{aligned}
\Pr[E_2] &\leq \Pr[X \geq n] \\
&= \Pr[X \geq (1+\delta)\,\mathbb{E}[X]] \\
&\leq \exp\left(-\frac{\delta^2}{2+\delta} \cdot \frac{n}{4}\right) \\
&= \exp\left(-\tfrac{9n}{20}\right) \\
&\leq 2^{-0.649n}.
\end{aligned}$$

Overall, the probability is bounded by:

$$\epsilon_{PoW} \leq \Pr[E_1] + \Pr[E_2 | \bar{E_1}] \leq \frac{t_1}{2^\lambda} + \frac{1}{2^{0.649n}}.$$

□

A typical setting suggested 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 $2^2 5$, the solution has expected length $1.55 \cdot 256 \cdot \log(2^{25}/1.55 \cdot 256) < 6500$ bits, which is less than 0.8 KiB. With these parameters, the solver will find a solution in time 1.3ℓ, with error approximately 2^{20}.

4.3 Relativized setting

Until now, we have only an intuitive explanation of why our transformation protects against attacks such as the one described in the toy protocol.
Proving security in the standard model has many complications and seems infeasible. So to argue soundness, we turn back to the random oracle model. Despite the given attack on the toy protocol, we know that it is proven secure in the ROM. The key reason is that in the random oracle model, for compiling the FS transform, we introduce a fresh random oracle, which the initial sigma protocol did not have access to. The diagonalization attacks discussed exploit precisely this gap.
To capture these attacks, yet still having enough assumptions to carry out a rigorous proof, they[AY25] consider the relativized model. Thus they begin with a relativized sigma protocol in the random oracle model, with oracle f. Meaning that all parties have access to oracle f. In particular, the circuits can have f-gates. Then the FS transform is applied using the same oracle f.
As it is fairly obvious, the attack on the toy protocol can be easily translated into this setting. You just need to have a circuit with only one f-gate whose input wires are x and w(inputs of the circuit).

Remark. 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, [AY25] could not directly prove the security of the GKR protocol after applying the XFS. However, the result yields *evidence* of its security in the standard model.

4.3.1 Preliminary security definitions

To argue soundness for the XFS transformed protocol, any NARK adversary is reduced to an adversary for the underlying SP, with roughly the same success rate. However, to do that the sigma protocol is expected to have several properties, which we explain informally:

- **Adaptive (straighline) round-by-round knowledge.** There exists an extractor E_{SP} such that given $(C, \mathbb{x}, m_1, tr_1)$, belonging to a successful adversary, can extract a witness for $\mathbb{x}$ except with negligible probability, where C is the adversary's chosen circuit(Language), m_1 is adversary's first message, and tr_1 is the trace of adversary's oracle queries before sending m_1.
- Additionally it is expected that the extractor E_{SP} behaves robustly, meaning that if the trace of oracle queries t_1 gets appended, the extracted witness does not change except with negligible probability.
- **Adaptive (straightline) indexer-knowledge.** The indexer algorithm I of the sigma protocol acts also as a commitment scheme for the circuits. And there exist an extractor E_I such that given the digest(commitment) ψ and the trace of oracle queries tr before adversary committing to $\psi = I(C)$, can extract the circuit C except with negligible probability.
- Additionally, it is expected that the extractor E_I behaves robustly, meaning that if the trace of oracle queries t gets appended, the extracted circuit does not change except with negligible probability.

4.3.2 Proof sketch

As a first step, their[AY25] proof follows a similar structure to existing proofs for the Fiat–Shamir transformation in the ROM(i.e. [CY24, Chapter 10]). They reduce a prover $\mathcal{P}$ attacking the argument after applying the XFS to a prover P attacking the sigma protocol. If $\mathcal{P}$ outputs $(C^f, \mathbb{x}, \mathbb{y}, m_1, s, m_2)$ then it must have queried $(\tau, \psi, \mathbb{x}, \mathbb{y}, m_1, s)$ to receive the challenge ρ, as it is really unlikely to convince the verifier on a random challenge it does not know. They guess which query it is and plant the random sigma verifier challenge $\rho \leftarrow \{0,1\}^\lambda$ in the oracle response. The goal is to show that $\mathcal{P}$ running with this programmed oracle does not change the verifier's behavior.
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 this case, both may make f-queries. They handle separately the queries made by the verifier and those made by the circuit.
First, note that the verifier is an honest algorithm and so we can predict its trace. By using a suitable padding, τ, we ensure that the verifier does not directly make such queries, for example $\tau = 0^{|V|}$.
The only thing left is showing that C cannot 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 $\mathbb{w}$, 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 $f(\psi, \mathbb{x}, \mathbb{y}, m_1)$. The digest ψ commits the prover to C, and by round-by-round knowledge of the sigma protocol, m_1 is "committing" to $\mathbb{w}$ in the sense that alongside the trace of queries, it can be used to extract the witness. Thus, the prover is essentially committed to $\mathbb{x}, \mathbb{y}, C,$ and $\mathbb{w}$ before knowing the puzzle. Since the claim is that $C^f(\mathbb{x}, \mathbb{w}) = \mathbb{y}$ (the circuit is not provided anything other than $\mathbb{x}$ and $\mathbb{w}$ which are both computed before knowing the puzzle), security of proof-of-work guarantees that C does not compute solution s. Hence the circuit C cannot query $(\tau, \psi, \mathbb{x}, \mathbb{y}, m_1, s)$.
This shows that verifier's behavior roughly remains the same and completes the proof.

References

[AY25] Gal Arnon and Eylon Yogev. Towards a white-box secure fiat-shamir transformation. Cryptology ePrint Archive, Paper 2025/329, 2025.

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

[BBF18] Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Paper 2018/712, 2018.

[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*, pages 626–643, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

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

[DMWG23] Quang Dao, Jim Miller, Opal Wright, and Paul Grubbs. Weak fiat-shamir attacks on modern proof systems. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 199–216. IEEE, 2023.

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

[GKR08] Shafi Goldwasser, Yael Kalai, and Guy Rothblum. Delegating computation: Interactive proofs for muggles. volume 62, pages 113–122, 05 2008.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019.

[KRS25] Dmitry Khovratovich, Ron D. Rothblum, and Lev Soukhanov. How to prove false statements: Practical attacks on fiat-shamir. Cryptology ePrint Archive, Paper 2025/118, 2025.

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

[NHB24] Hieu Nguyen, Uyen Ho, and Alex Biryukov. Fiat-shamir in the wild. In *Cyber Security, Cryptology, and Machine Learning: 8th International Symposium, CSCML 2024, Be'er Sheva, Israel, December 19–20, 2024, Proceedings*, page 135–150, Berlin, Heidelberg, 2024. Springer-Verlag.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *Advances in Cryptology — EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer, 1996.

[Wes20] Benjamin Wesolowski. Efficient verifiable delay functions. *Journal of Cryptology*, 33(4):2113–2147, 2020.