Short Paper: The Last Challenge Attack on Fiat-Shamir in KZG-based SNARKs

Oana Ciobotaru*[1], Maxim Peter[2], and Vesselin Velichkov[2]

[1] Independent Researcher
[2] OpenZeppelin

Abstract. The Fiat-Shamir transform [12] is a fundamental technique for converting sound public-coin interactive protocols into sound non-interactive protocols. While the theoretical transformation is conceptually simple, implementation-level deviations—often motivated by performance optimisations—can introduce catastrophic security flaws. In this work, we present the *Last Challenge Attack* (LCA), a vulnerability arising from such a deviation in a real-world KZG-based SNARK verifier implementation. The vulnerability stems from an incorrect computation of the final KZG [18] protocol challenge, which is a batching challenge derived independently of the evaluation proofs. This flaw potentially affects any KZG implementation of batched proofs for multiple evaluation points. We demonstrate that a malicious prover can exploit this deviation to forge proofs of false statements. We provide a proof-of-concept implementation demonstrating the forgery of a proof for an arbitrary public input. This vulnerability was discovered during a security audit, responsibly disclosed, and fixed.

Keywords: Fiat-Shamir · SNARKs · KZG · PLONK · Vulnerability

1 Introduction

The Fiat-Shamir (FS) transform [12] is a well-known and widely employed technique for converting certain *sound public-coin interactive protocols* into functionally equivalent *sound non-interactive protocols*. The security of the FS transform is proven, in general, in the random oracle model [2,24] or, in special cases, relies on the *correlation intractability* of the hash function used [7,23].

In practice, the FS transform is instantiated with a hash function and the resulting algorithm is a *heuristic*: for the resulting non-interactive protocols, at the moment, there are no known general proofs of soundness based on the computational intractability of some well-studied mathematical problem (e.g., discrete logarithm problem). However, the protocols obtained using the FS transform are believed to be secure in practice.

Intuitively, in standardised implementations of the FS transform, a public-coin interactive protocol is transformed into a non-interactive protocol by replacing the

* Work conducted while the author was at OpenZeppelin.

random challenges at each interactive round with the hash of the concatenation of all the public parameters of the instance, the public input of the instance and all the messages exchanged between the interactive parties, up to the current round.

Even though the transformation itself is relatively clear and simple, there are various reasons why implementations of the FS transform deviate from standard specifications, ranging from misunderstandings to trying to optimise for efficiency of the implementation. In such cases, the honest parties are at increased risk of being amenable to various types of potentially disastrous attacks.

In the following, we describe a new type of attack targeting incorrect implementations of the FS transform, called the *Last Challenge Attack*. This attack arises in implementations of the KZG polynomial commitment scheme verifier [18] in which the FS transform is applied to batched KZG evaluation proofs for at least two distinct evaluation points, but the challenge used to batch the evaluation proofs is incorrectly computed. In particular, the *Last Challenge Attack* is likely to affect implementations in which the proof batching challenge is incorrectly computed prior to and, thus, independently of at least two individual KZG evaluation proofs. For concreteness, we show an example of a *Last Challenge Attack* found in a KZG-based PLONK [14] verifier implementation which a malicious PLONK prover could exploit in order to construct verifiable proofs of false statements.

The vulnerability presented in this work was discovered in September 2023 during an audit of the pre-release Linea[3] PLONK verifier implementation. It was responsibly disclosed to the `gnark` maintainers, who confirmed the flaw and released a critical security advisory (GHSA-7p92-x423-vwj6)[4] in October 2023. The fix enforces a canonical FS transform instantiation by ensuring the batching challenge is computed only after all evaluation proofs are included in the transcript.

1.1 Related Work

The Fiat-Shamir (FS) transform, a technique using a random oracle to obtain non-interactive protocols, was introduced in [12]. The first formal foundations for the random oracle model were published in [2]. The first security proof for an FS transform in the random oracle model is part of [24]. In particular, the authors show how to transform a constant-round interactive proof into a secure non-interactive signature scheme in the random oracle model. In practice, the random oracle is replaced by a hash function.

So far, no general proofs of security are known for the resulting non-interactive protocols without the random oracle model. Recently, the idea of proving the security of the FS transform based on standard cryptographic assumptions, even though applied to restricted types or sets of protocols, has gained more attention

[3] `https://linea.build/`

[4] `https://github.com/Consensys/gnark/security/advisories/GHSA-7p92-x423-vwj6`

from the research community [4,6,9,17,20]. This line of work is centred around the notion of *correlation intractability* which was first introduced in [23] and [7].

Finally, the closest related work describes attacks on implementations of the FS transform where the verifier deviates from the prescribed specifications [3,21,22]. FS transform security is traditionally analysed through the lens of *weak* instantiations (i.e., the transcript is missing some or all of the public inputs) versus *strong* instantiations (i.e., the transcript incorporates the full public statement); the latter is the standard for ensuring knowledge-soundness [11,16,25]. Recent theoretical work has pushed beyond this boundary; Khovratovich et al. [19] demonstrated that for non-contrived architectures like GKR [15], even the strong FS transform may be insufficient to ensure adaptive soundness.

Our work identifies a new type of failure: an implementation that satisfies statement-binding but fails to achieve a truly strong FS transform by omitting certain prover messages from the transcript. In turn, this partial transcript is used to derive the final challenge required only by the non-interactive verifier. The *Last Challenge Attack* vulnerability demonstrates that partial transcript binding is catastrophic, as it preserves algebraic linearities that a malicious prover can exploit to forge batched KZG evaluation proofs for false claims. Identifying this pitfall is critical for practitioners who may mistakenly assume that challenges used only for verifier-side batching require less rigorous binding than those used for prover responses.

The rest of the paper is organised as follows: in Section 2, we provide the necessary background. In Section 3, we state the knowledge-soundness property of the KZG polynomial commitment scheme [18] applied to batch evaluation proofs for at least two distinct evaluation points. In Section 4, we describe the LCA applied to the general KZG scheme and, also, to one of its concrete and widely spread use cases, namely the KZG-based PLONK protocol. Section 5 provides systemic mitigations for the identified vulnerabilities and offers concluding remarks.

2 Preliminaries

We denote by λ the security parameter; for simplicity, we sometimes may omit explicitly mentioning λ. All algorithms used in this work run in polynomial time in λ or, simply, $\mathsf{poly}(\lambda)$. For short, we call such algorithms *efficient* or *polynomially bounded*. We denote by $\mathbb{F}$ a *finite field*. We consider only finite fields such that $|\mathbb{F}| \in O(2^\lambda)$, where by $|S|$ we understand the cardinality of some set S. We denote by $\mathbb{F}_{<d}[X]$ the set of univariate polynomials of degree less than d over $\mathbb{F}$. By $[t]$, where t is a natural number, we denote the set $\{1, \ldots, t\}$. By $\mathsf{negl}(\lambda)$, we understand the set of all negligible functions with λ as parameter.

We denote by $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ groups of some prime size q such that e is a secure, efficiently computable non-degenerate pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We denote by g_1, g_2, g_T some generators of $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, respectively, chosen uniformly at random such that $e(g_1, g_2) = g_T$. We use additive notation for $\mathbb{G}_1$ and $\mathbb{G}_2$

and multiplicative notation for $\mathbb{G}_T$. We also use the notation $[x]_1 := x \cdot g_1$ and $[x]_2 := x \cdot g_2$.

Our security proof requires the *algebraic group model* (AGM) [13]. Given a cyclic group $\mathbb{G}$ of prime order p, by *algebraic adversary* with respect to $\mathbb{G}$ we understand an algorithm A such that whenever A outputs $h \in \mathbb{G}$, A also outputs a vector of field elements $\alpha = (\alpha_1, \ldots, \alpha_m) \in \mathbb{Z}_p^m$ such that $h = \sum_{i=1}^{m} \alpha_i \cdot L_i$, where $(L_1, \ldots, L_m)$ are all the group elements given to A during its execution so far.

The protocols we analyse use polynomial commitment schemes as defined, for example, in Section 3 from PLONK [14].

3 Security Considerations

In this section we detail the security property for a KZG commitment scheme [18] for multiple evaluation points (KZG MES). This is a generalisation of the scheme described in PLONK [14, page 13] to any number of distinct evaluation points and it allows us to best model our attack.

3.1 Security Analysis of KZG MES

Let $n \in \mathbb{N}$, $n \geq 2$. We denote the distinct evaluation points by $z_1, \ldots, z_n$ and the polynomials to be evaluated at these points by $t_1, \ldots, t_n$. We denote by $\{f_{1,i}(X)\}_{i\in[t_1]}, \ldots, \{f_{n,i}(X)\}_{i\in[t_n]}$ the sets of polynomials to be evaluated at $z_1, \ldots, z_n$, respectively.

Instantiation 1 (KZG Multipoint Evaluation Scheme). *The KZG MES assumes parties P_{KZG} and V_{KZG} and proceeds as follows*

- $\mathsf{gen}(d)$:
  Choose uniform $x \in \mathbb{F}$. *Output* $\mathsf{srs} = ([1]_1, [x]_1, \ldots, [x^{d-1}]_1, [1]_2, [x]_2)$.
- $\mathsf{com}(f, \mathsf{srs})$:
  Using the srs, *compute and output* $\mathsf{cm} = [f(x)]_1$.
- $\mathsf{open}(\{\mathsf{cm}_{1,j}\}_{j\in[t_1]}, \ldots, \{\mathsf{cm}_{n,j}\}_{j\in[t_n]}, \{z_1, \ldots, z_n\}, \{s_{1,j}\}_{j\in[t_1]}, \ldots, \{s_{n,j}\}_{j\in[t_n]})$:
  Round 1: V_{KZG} sends random $\gamma_1, \ldots, \gamma_n \in \mathbb{F}$ to P_{KZG}.

  Round 2: P_{KZG} computes the polynomials:

$$h_1(X) := \sum_{j=1}^{t_1} \gamma_1^{j-1} \cdot \frac{f_{1,j}(X) - f_{1,j}(z_1)}{X - z_1}$$

$$\ldots$$

$$h_n(X) := \sum_{j=1}^{t_n} \gamma_n^{j-1} \cdot \frac{f_{n,j}(X) - f_{n,j}(z_n)}{X - z_n}$$

and using srs *computes and sends to* V_{KZG}

$$W_1 := [h_1(x)]_1, \ldots, W_n := [h_n(x)]_1.$$

We denote $\pi_{KZG} = (W_i)_{i=1}^n$ *and we call it a proof for KZG MES.*

Round 3: V_{KZG} *chooses uniformly random* $u \in \mathbb{F}$.

V_{KZG} *computes curve points* $F_1, \ldots, F_n$ *where*

$$F_1 = \sum_{j \in [t_1]} {\gamma_1}^{j-1} \cdot \mathsf{cm_{1,j}} - \left[\sum_{j \in [t_1]} {\gamma_1}^{j-1} \cdot s_{1,j} \right]_1 ,$$

$$\ldots$$

$$F_n = \sum_{j \in [t_n]} {\gamma_n}^{j-1} \cdot \mathsf{cm_{n,j}} - \left[\sum_{j \in [t_n]} {\gamma_n}^{j-1} \cdot s_{n,j} \right]_1 .$$

V_{KZG} *outputs* acc *if and only if the following holds:*

$$e(F_1 + \ldots + u^{n-1} \cdot F_n + z_1 \cdot W_1 + \cdots + u^{n-1} \cdot z_n \cdot W_n, [1]_2) \cdot e(-W_1 - \ldots - u^{n-1} \cdot W_n, [x]_2) = 1 \quad (1)$$

In order to prove the security of Instantiation 1, we require:

Definition 1 (Q-DLOG Assumption). *For a fixed integer* Q*, the Q-DLOG assumption for* $(\mathbb{G}_1, \mathbb{G}_2)$ *states that given* $[1]_1, [x]_1, \ldots, [x^Q]_1, [1]_2, [x]_2, \ldots, [x^Q]_2$*, for uniformly chosen* $x \in \mathbb{F}$*, the probability of an efficient* A *outputting* x *is negligible in the security parameter.*

Lemma 1 (Schwartz-Zippel Lemma). *Let* $P(X_1, \ldots, X_k) \in \mathbb{F}[X_1, \ldots, X_k]$ *be a non-zero polynomial of total degree* d *over the finite field* $\mathbb{F}$*. Let* S *be a finite subset of* $\mathbb{F}$ *and let* $r_1, \ldots, r_k \in S$ *be selected uniformly at random. Then the probability that* $P(r_1, r_2, \ldots, r_k) = 0$ *is at most* $\frac{d}{|S|}$.

The main statement of this section is:

Lemma 2 (Security of KZG MES). *Assume the Q-DLOG for* $(\mathbb{G}_1, \mathbb{G}_2)$*. Let* $n \geq 2$ *be an integer such that* $n \in \mathsf{poly}(\lambda)$*. Then the KZG polynomial commitment scheme for multipoint evaluation as described in Instantiation 1 has completeness and knowledge-soundness in the AGM as per Definition 3.1 from PLONK [14].*

Proof. This is a proof sketch. The detailed proof is available in the full version [10].

Completeness follows from the linearity of the KZG opening procedure.

To prove knowledge-soundness, assume the final pairing check (1) holds. By the pairing bilinearity and the Schwartz-Zippel Lemma applied to the batching challenge u, this aggregate check implies that n independent KZG pairing checks—one for each distinct evaluation point $z_i, i \in [n]$—hold with overwhelming probability.

Following the security analysis in PLONK [14, Sect. 3.1], under the Q-DLOG assumption and the AGM, the validity of these n individual checks ensures the existence of an extractor capable of recovering the sets of low-degree polynomials $\{f_{i,j}(X)\}_{i\in[n],j\in[t_i]}$ such that $f_{i,j}(z_i) = s_{i,j}, \forall i \in [n], j \in [t_i]$. This satisfies the knowledge-soundness requirement for KZG MES in the AGM.

3.2 Discussion

The security proof for Lemma 2 relies fundamentally on the preconditions of the Schwartz-Zippel Lemma. The proof holds only if:

- The total degree of the polynomial is negligible relative to the field size $|\mathbb{F}|$, and
- The challenge u is chosen uniformly at random from $\mathbb{F}$.

In our construction, the relevant polynomial degree is at most $n - 1$. Given $n \in \mathsf{poly}(\lambda)$ and $|\mathbb{F}| \in \Omega(2^\lambda)$, the first condition is satisfied.

To satisfy the second condition and ensure resistance against a malicious prover, the verifier V_{KZG} has two primary implementation options:

1. Interactive: V_{KZG} follows Instantiation 1, sampling u uniformly at random from $\mathbb{F}$. On-chain, this is often prohibitive as it requires both expensive multi-round communication and access to a secure, unbiasable entropy source.
2. Non-interactive: The non-interactive verifier derives u via the Fiat-Shamir transform. Here, u must be computed as a hash of the complete protocol transcript, including all public parameters, public inputs, and prior prover messages (specifically $W_1, \ldots, W_n$).

Prior results (e.g., [1,5]) establish the soundness of batching linear relations, and Marlin [8] proves knowledge-soundness for multi-query KZG in the AGM. However, these are often presented in generalised, formalism-heavy contexts. Lemma 2 in Section 3.1 provides a self-contained, end-to-end analysis for the concrete batch KZG case, improving accessibility for auditors and implementers by removing superfluous complexity.

4 The Last Challenge Attack

We have shown that the interactive KZG multipoint evaluation scheme (KZG MES) is secure if the challenge used for batching KZG evaluations proofs is chosen uniformly at random after all the other transcript values are fixed. In this section we show that the above condition is not only sufficient, but a variation of it is also necessary in order for the knowledge-soundness property of non-interactive KZG MES to hold in the context of the FS transform.

To illustrate this, we first describe a general hypothetical attack strategy in Section 4.1, followed by a concrete instantiation observed during an audit of a production-ready PLONK verifier in Section 4.2. By starting with the general case, we show that this flaw is not a mere implementation quirk, but an inherent risk for any KZG verifier that deviates from correct Fiat-Shamir specifications.

4.1 A Hypothetical Attack on a Variation of KZG MES

This hypothetical attack targets a non-interactive (hence FS tranform derived) variation of the KZG MES with $n \geq 2$ evaluation points. In an FS-compliant implementation, challenges (γ_i) and u are derived by hashing the complete communication transcript up to the respective round. Our attack assumes a deviating non-interactive verifier incorrectly computes the final challenge u by hashing a transcript that excludes the evaluation proofs W_1 and W_2. A malicious prover can exploit the premature computation of u to forge a proof that satisfies the verifier's pairing check.

The Setting To formalise the attack, we introduce P_{KZGN} and V_{KZGN} as the non-interactive, FS-transform derived versions of P_{KZG} and V_{KZG}, alongside a malicious prover P'_{KZGN} and a vulnerable verifier V'_{KZGN}. The verifier V'_{KZGN} operates identically to V_{KZGN}, except for the specific exclusion of W_1 and W_2 from the transcript used to derive the batching challenge u.

The Attack The attack proceeds as follows:

Step 1. The malicious prover P'_{KZGN} computes some group elements $A, B \in \mathbb{G}_1$ satisfying

$$e(A, [x]_2) \cdot e(-B, [1]_2) = 1 \tag{2}$$

Concretely, P'_{KZGN} simulates KZG for a single polynomial and a single evaluation point for an arbitrary $f(X)$ evaluated at an arbitrary z. P'_{KZGN} takes both the roles of prover and verifier in the simulation. Hence P'_{KZGN} obtains group elements $A, B \in \mathbb{G}_1$ such that relation (2) holds.

Step 2. P'_{KZGN} sets the inputs of the non-interactive KZG open procedure (including $\{s_{i,j}\}_{i\in[n],j\in[t_i]}$) to some arbitrary values[5]. In preparation to create a proof π'_{KZG} for a false statement, P'_{KZGN} sets all KZG MES proof components $W_3, \ldots, W_n$ (so except for W_1 and W_2) to arbitrary values in $\mathbb{G}_1^{n-2}$.

Step 3. Using the KZG open procedure inputs, P'_{KZGN} computes group elements $F_1, \ldots, F_n$ as done by V_{KZG} described in Round 3 of Instantiation 1.

Step 4. V'_{KZGN} deviates from the FS transform by computing the final challenge u as the hash of the transcript excluding W_1, W_2. P'_{KZGN} exploits the fact that u is fixed before W_1, W_2 are. Concretely, P'_{KZGN} solves the following linear system with W_1, W_2 as unknowns. The system arises from the KZG MES pairing

[5] Note that this includes the special case when both the prover P'_{KZGN} and the verifier V'_{KZGN} need to agree on the inputs of the open procedure

check (1). It is defined by A, B computed in Step 1, by $W_3, \ldots, W_n$ chosen in Step 2, by $F_1, \ldots, F_n$ computed in Step 3 and by the value u:

$$\begin{cases} W_1 + u \cdot W_2 + u^2 \cdot W_3 + \ldots + u^{n-1} \cdot W_n = A \\ F_1 + \ldots + u^{n-1} \cdot F_n + z_1 \cdot W_1 + u \cdot z_2 \cdot W_2 + \\ \qquad\qquad + u^2 \cdot z_3 \cdot W_3 + \cdots + u^{n-1} \cdot z_n \cdot W_n = B \end{cases} \tag{3}$$

The linear system (3) has a solution for W_1, W_2 if and only if $u \neq 0$ and $z_1 \neq z_2$.

Step 5. P'_{KZGN} appropriately fills in the proof π'_{KZG} slots with the above W_1, W_2.

Step 6. If this attack step has been reached, the vulnerable V'_{KZGN} accepts forged π'_{KZG} with probability 1.

We call this exploit the *Last Challenge Attack* (LCA), as it targets an incorrect derivation of the final challenge u within the Fiat-Shamir transform of the KZG MES for the verifier. The attack leverages a truncated transcript hash that excludes critical prover messages, specifically two of the KZG evaluation proofs. This omission ensures those KZG proofs remain independent of the batching challenge u, effectively invalidating the security requirements established in Section 3.2. Consequently, a malicious prover P'_{KZGN} can compute proofs for arbitrary false statements, which a vulnerable verifier V'_{KZGN} will accept with probability 1.

4.2 The Last Challenge Attack on a PLONK Implementation

In this section we describe an instance of the attack presented in Section 4.1 observed in an actual implementation of a KZG-based PLONK verifier.

The PLONK Prover and Verifier Simplified We summarise the PLONK prover P_{PLONK} and verifier V_{PLONK}, as per [14, Sect. 8.1], highlighting only the necessary details to understand our attack.

The PLONK argument system uses the non-interactive version (i.e., the FS transform) of KZG MES for two evaluation points $z_1 = \mathfrak{z} \in \mathbb{F}$ and $z_2 = \mathfrak{z}\omega \in \mathbb{F}$ respectively, and two sets of polynomials of size $t_1 = 6$ and $t_2 = 1$, respectively: $\{f_{1,i}(X)\}_{i \in [t_1=6]} = \{\mathsf{r}(X), \mathsf{a}(X), \mathsf{b}(X), \mathsf{c}(X), \mathsf{S}_{\sigma_1}(X), \mathsf{S}_{\sigma_2}(X)\}$ and $\{f_{2,i}(X)\}_{i \in [t_2=1]} = \{\mathsf{z}(X)\}$. The evaluations of the polynomials $\{f_{1,i}\}_i$ and $\{f_{2,1}\}$ at $z_1 = \mathfrak{z}$ and $z_2 = \mathfrak{z}\omega$ are, respectively:

$$\{s_{1,i}\}_{i \in [t_1=6]} = \{\bar{r}, \bar{a}, \bar{b}, \bar{c}, \bar{s}_{\sigma_1}, \bar{s}_{\sigma_2}\}, \ \{s_{2,i}\}_{i \in [t_2=1]} = \{\bar{z}_\omega\}.$$

Prover: The PLONK prover P_{PLONK} produces a proof of the form:

$$\pi_{PLONK} = \left([a]_1, [b]_1, [c]_1, [z]_1, [t]_1, W_1, W_2, \bar{a}, \bar{b}, \bar{c}, \bar{\mathbf{s}}_{\sigma 1}, \bar{\mathbf{s}}_{\sigma 2}, \bar{z}_\omega\right) \tag{4}$$

where $[t]_1$ is the commitment to the quotient polynomial $\mathsf{t}(X)$ obtained by dividing all batched constraints by the vanishing polynomial $Z_H(X)$ with its roots represented by the elements of the multiplicative subgroup H. For slots 6 and 7 of π_{PLONK} we use the notation established in Section 3, i.e., W_1 and W_2.

Verifier: The PLONK verifier V_{PLONK} takes as input the public parameters, the public inputs and a proof π_{PLONK} and outputs `accept` or `reject`. It works in 12 steps, the last of which consists of verifying a pairing check which is the special case of the check (1) for $n = 2$.

Implementation-based Attack Description In the audited KZG-based PLONK verifier V'_{PLONK}, the challenge u is computed without hashing the group elements W_1 and W_2. This omission makes the hypothetical attack in Section 4.1 directly applicable. We detail the attack on this vulnerable implementation below, following the previously defined steps and noting specific deviations.

Step 1. From a well-formed PLONK proof computed correctly using any circuit of its choice, a malicious prover P'_{PLONK} produces A, B such that (2) holds. Alternatively, P'_{PLONK} can produce appropriate values A and B following Step 1. described in Section 4.1.

Step 2. P'_{PLONK} sets the public input PI and all PLONK proof components except for W_1 and W_2 to any arbitrary values of its choice in preparation to produce a proof π'_{PLONK} of a false statement. π'_{PLONK} will have the same number and type of components as π_{PLONK} described by relation (4).

Step 3. From the components of π'_{PLONK}, except for W_1 and W_2, and using steps 9, 10 and 11 of the PLONK verifier V_{PLONK} [14, Sect. 8.1]), P'_{PLONK} computes F_1, F_2. Note that W_1 and W_2 are indeed not needed in order for P'_{PLONK} to complete this computation.

Step 4. P'_{PLONK} exploits the fact that u can be computed, and hence, fixed before W_1 and W_2 are. This is analogous to Step 4. in Section 4.1 and is a direct implication of the fact that the non-interactive deviating verifier V'_{PLONK} does not include W_1 and W_2 as part of the full transcript to be hashed when deriving u. Thus, whenever a solution exists, P'_{PLONK} solves the following linear system with W_1, W_2 as unknowns.

$$\begin{cases} W_1 + u \cdot W_2 = A \\ F_1 + u \cdot F_2 + \mathfrak{z} \cdot W_1 + u \cdot \mathfrak{z} \cdot \omega \cdot W_2 = B \end{cases} \tag{5}$$

The system (5) has a solution if and only if $\mathfrak{z} \cdot u \neq 0$.

Step 5. P'_{PLONK} appropriately fills in π'_{PLONK} slots with the above W_1, W_2.

Step 6. If this attack step has been reached, the vulnerable verifier V'_{PLONK} accepts forged π'_{PLONK} with probability 1.

Proof of Concept A proof of concept (PoC) of this vulnerability is available online [6]. This Solidity-based PoC targets a production-ready PLONK verifier and follows the attack steps in Section 4.2 to forge a proof for an arbitrary, prover-selected public input.

Implications In the context of ZK-rollups on Ethereum, where prover circuits are tasked with simulating the Ethereum Virtual Machine to validate state transitions, the stakes of the *Last Challenge Attack* are particularly high. A verifier vulnerable to the *Last Challenge Attack* permits a malicious prover to forge proofs for invalid transitions with probability 1. Such an exploit would allow a prover to fraudulently claim ownership of rollup assets, demonstrating that even subtle implementation-level deviations in SNARK verifiers can fundamentally compromise the underlying blockchain's security model.

5 Mitigations and Conclusions

We introduced the *Last Challenge Attack* (LCA), a critical vulnerability in KZG-based SNARKs where failing to bind prover messages to the Fiat-Shamir transcript breaks knowledge-soundness with probability 1. Our discovery of this flaw in a production-ready verifier underscores that even minor optimisations can jeopardise the entire ZK-rollup ecosystem. To secure Fiat-Shamir implementations, we recommend that developers and auditors prioritise: (1) Transcript integrity, by adhering strictly to the protocol's chronological flow where challenges never precede the messages they bind; (2) Uniform security prioritisation, by ensuring all challenges, including verifier-only ones, are computed with the same rigour; and (3) Strong Fiat-Shamir, by hashing the full public statement to protect against wider attack classes.

The LCA serves as a reminder that cryptographic engineering requires full precision, particularly regarding transcript integrity. Mind your Fiat-Shamirs!

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. We also thank the `gnark` and Linea teams for their cooperation during the responsible disclosure of the vulnerability discussed in this work.

References

1. Bellare, M., Garay, J.A., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures. Cryptology ePrint Archive, Paper 1998/007 (1998)
2. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: CCS 1993. pp. 62–73 (1993)

[6] `https://github.com/OpenZeppelin/publications/tree/main/papers/the-last-challenge-attack/proof-of-concept`

3. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. Cryptology ePrint Archive, Paper 2016/771 (2016)
4. Brakerski, Z., Koppula, V., Mour, T.: NIZK from LPN and trapdoor hash via correlation intractability for approximable relations. In: CRYPTO 2020. pp. 738–767 (2020)
5. Camenisch, J., Hohenberger, S., Østergaard Pedersen, M.: Batch verification of short signatures. Cryptology ePrint Archive, Paper 2007/172 (2007)
6. Canetti, R., Chen, Y., Reyzin, L., Rothblum, R.D.: Fiat-Shamir and correlation intractability from strong KDM-secure encryption. In: EUROCRYPT 2018. pp. 91–122 (2018)
7. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. Cryptology ePrint Archive, Paper 1998/011 (1998)
8. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: Preprocessing zkSNARKs with universal and updatable srs. In: EUROCRYPT 2020. pp. 738–768 (2020)
9. Ciampi, M., Parisella, R., Venturi, D.: On adaptive security of delayed-input sigma protocols and Fiat-Shamir NIZKs. In: SCN 2020. pp. 670–690 (2020)
10. Ciobotaru, O., Peter, M., Velichkov, V.: The last challenge attack on fiat-shamir in KZG-based SNARKs. Cryptology ePrint Archive, Paper 2024/398 (2024), `https://eprint.iacr.org/2024/398`
11. Dao, Q., Miller, J., Melnikov, O., Oppenheim, A.: Weak Fiat-Shamir attacks on modern proof systems. In: IEEE Symposium on Security and Privacy (SP). pp. 199–216 (2023)
12. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO 1986. pp. 186–194 (1987)
13. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: CRYPTO 2018. pp. 33–62 (2018)
14. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Paper 2019/953 (2019)
15. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In: STOC 2008. p. 113–122 (2008)
16. Haines, T., Lewis, S.J., Pereira, O., Teague, V.: How not to prove your election outcome. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 644–660 (2020)
17. Kalai, Y.T., Rothblum, G.N., Rothblum, R.D.: From obfuscation to the security of Fiat-Shamir for proofs. In: CRYPTO 2017. pp. 224–251 (2017)
18. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: ASIACRYPT 2010. pp. 177–194 (2010)
19. Khovratovich, D., Rothblum, R.D., Soukhanov, L.: How to prove false statements: Practical attacks on fiat-shamir. Cryptology ePrint Archive, Paper 2025/118 (2025)
20. Lombardi, A., Vaikuntanathan, V.: Fiat-Shamir for repeated squaring with applications to PPAD-hardness and VDFs. In: CRYPTO 2020. pp. 632–651 (2020)
21. McMurtry, E., Pereira, O., Teague, V.: When is a test not a proof? Cryptology ePrint Archive, Paper 2020/909 (2020)
22. Miller, J.: Coordinated disclosure of vulnerabilities affecting Girault, Bulletproofs, and Plonk. Trail of Bits Blog (2022)
23. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: CRYPTO 1992 (1992)

24. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: EUROCRYPT 1996. pp. 387–398 (1996)
25. Thaler, J.: Proofs, Arguments, and Zero-Knowledge (Foundations and Trends in Privacy and Security). Now Publishers (2022)