Exploiting Vulnerable Implementations of ZK-based Cryptographic Schemes Used in the Ethereum Ecosystem

Oana Ciobotaru [1] Maxim Peter [2] Vesselin Velichkov [2]
Nikesh Nazareth [2] Sam Wong [2]

[1] Pi Squared

[2] OpenZeppelin

Real World Crypto Symposium
Sofia, March 26-28, 2025

(Incomplete) History of Vulnerabilities in ZK Schemes

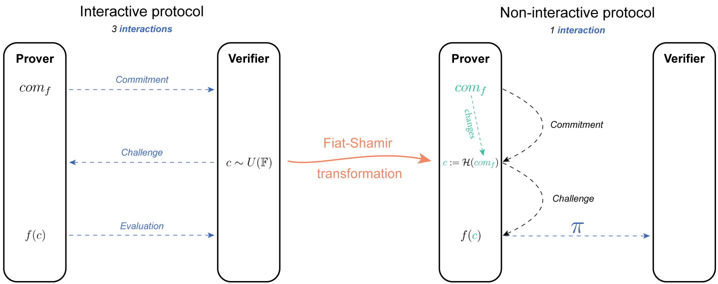
Theoretical Attacks with Implications in Practice

- Zcash counterfeiting vulnerability [G19].
- The lack of security for the Fiat-Shamir transform applied to the GKR protocol and hash function circuits [KRS25].

Vulnerabilities Encountered in Practice

- Attacks on insecure implementation of the Fiat-Shamir transform (e.g., [BPW12], [HLPT20], [DMWG23]).
- Attack on a Nova folding scheme implementation [NBS23].

Interactive vs. Non-interactive Arguments



The Fiat-Shamir (FS) Transform

- By default, computing proof/argument $\boldsymbol{\pi}$ is an interactive process between the prover $\boldsymbol{\mathcal{P}}$ and the verifier $\boldsymbol{\mathcal{V}}$.
- The FS transform turns that into a non-interactive process $(\boldsymbol{\mathcal{P}_n}, \boldsymbol{\mathcal{V}_n})$ via an idealised random oracle model (ROM).
- In practice, $\boldsymbol{\mathcal{P}_n}$ and $\boldsymbol{\mathcal{V}_n}$ independently compute challenges as the hash of the computation transcript up to that point.

History of Attacks on the Fiat-Shamir Transform

Theoretical Attacks

- (Contrived) attacks on cryptographic primitives secure in the ROM but insecure when ROM is instantiated ([Bar01], [CK03], [CGH04], [BBH+19]).

Theoretical Attacks with Implications in Practice

- Proven lack of *adaptive soundness* for the FS transform applied to the GKR15 protocol and certain circuits arithmetising hash functions [KRS25].

Vulnerabilities Encountered in Practice

- Lack of adaptive soundness for FS transform implementations if the public input is omitted from the transcript (e.g.,[BPW12], [HLPT20], [DMWG23]).

History of Attacks on the Fiat-Shamir Transform

Theoretical Attacks

- (Contrived) attacks on cryptographic primitives secure in the ROM but insecure when ROM is instantiated ([Bar01], [CK03], [CGH04], [BBH+19]).

Theoretical Attacks with Implications in Practice

- Proven lack of *adaptive soundness* for the FS transform applied to the GKR15 protocol and certain circuits arithmetising hash functions [KRS25].

Vulnerabilities Encountered in Practice

- Lack of adaptive soundness for FS transform implementations if the public input is omitted from the transcript (e.g.,[BPW12], [HLPT20], [DMWG23]).

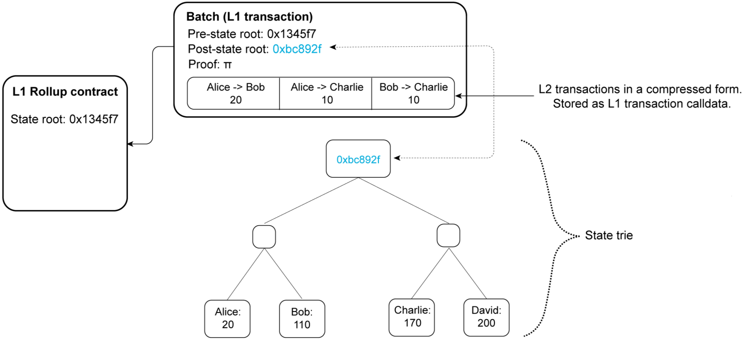
Assume a verifier omits from the transcript components different from the public input. Are attacks still possible?

Talk Outline

1. Setting: Scaling Ethereum
2. A New Type of Adaptive Soundness Attack on Vulnerable FS Transform Implementations
    - First Attack:
      The Last Challenge Attack (LCA)
    - Second Attack:
      Fiat-Shamir Array Inputs Not Transcribed (FAINT)
    - Implications
3. Subtle Attack on Statistical Zero-Knowledge
4. Conclusions

Setting: Scaling Ethereum

- L2 ZK-Rollups execute transactions off-chain.
- (SNARK) prover $\mathcal{P}_n$ provides a succinct ZK argument π on L1.
- π testifies that transactions were executed correctly.
- (SNARK) verifier $\mathcal{V}_n$ verifies on L1 the correctness of π.
- The state of L2 on L1 (and the state of L1) are updated accordingly.



Based on: https://vitalik.eth.limo/general/2021/01/05/rollup.html

New Soundness Attack on FS Implementations

Assume deviating $\boldsymbol{\mathcal{V}'_n}$ omits hashing a transcript component other than the public input. Malicious $\boldsymbol{\mathcal{P}'_n}$ chooses public input $\boldsymbol{x'}$ (*adaptive soundness attack*), then:

New Soundness Attack on FS Implementations

Assume deviating $\mathcal{V}'_n$ omits hashing a transcript component other than the public input. Malicious $\mathcal{P}'_n$ chooses public input x' (*adaptive soundness attack*), then:

$\mathcal{P}'_n$ Mounts a 6-Steps Attack Against $\mathcal{V}'_n$:

1. $\mathcal{P}'_n$ simulates a version of the pairing-based PCS underlying the SNARK. All inputs to the PCS are chosen by $\mathcal{P}'_n$.

New Soundness Attack on FS Implementations

Assume deviating $\mathcal{V}'_n$ omits hashing a transcript component other than the public input. Malicious $\mathcal{P}'_n$ chooses public input x' (*adaptive soundness attack*), then:

$\mathcal{P}'_n$ Mounts a 6-Steps Attack Against $\mathcal{V}'_n$:

1. $\mathcal{P}'_n$ simulates a version of the pairing-based PCS underlying the SNARK. All inputs to the PCS are chosen by $\mathcal{P}'_n$.
2. $\mathcal{P}'_n$ chooses freely all components of proof π' apart from those omitted by $\mathcal{V}'_n$ in the transcript (i.e., $\mathcal{V}'_n$'s degrees of freedom).

New Soundness Attack on FS Implementations

Assume deviating $\mathcal{V}_n'$ omits hashing a transcript component other than the public input. Malicious $\mathcal{P}_n'$ chooses public input x' (*adaptive soundness attack*), then:

$\mathcal{P}_n'$ Mounts a 6-Steps Attack Against $\mathcal{V}_n'$:

1. $\mathcal{P}_n'$ simulates a version of the pairing-based PCS underlying the SNARK. All inputs to the PCS are chosen by $\mathcal{P}_n'$.
2. $\mathcal{P}_n'$ chooses freely all components of proof π' apart from those omitted by $\mathcal{V}_n'$ in the transcript (i.e., $\mathcal{V}_n'$'s degrees of freedom).
3. Using x' and the above values, $\mathcal{P}_n'$ simulates* $\mathcal{P}_n$ and instantiates the system of constraints with $\mathcal{V}_n'$ degrees of freedom as the only unknowns.

New Soundness Attack on FS Implementations

Assume deviating $\mathcal{V}'_n$ omits hashing a transcript component other than the public input. Malicious $\mathcal{P}'_n$ chooses public input x' (*adaptive soundness attack*), then:

$\mathcal{P}'_n$ Mounts a 6-Steps Attack Against $\mathcal{V}'_n$:

1. $\mathcal{P}'_n$ simulates a version of the pairing-based PCS underlying the SNARK. All inputs to the PCS are chosen by $\mathcal{P}'_n$.
2. $\mathcal{P}'_n$ chooses freely all components of proof π' apart from those omitted by $\mathcal{V}'_n$ in the transcript (i.e., $\mathcal{V}'_n$'s degrees of freedom).
3. Using x' and the above values, $\mathcal{P}'_n$ simulates* $\mathcal{P}_n$ and instantiates the system of constraints with $\mathcal{V}'_n$ degrees of freedom as the only unknowns.
4. $\mathcal{P}'_n$ solves the above system, if feasible.

New Soundness Attack on FS Implementations

Assume deviating $\mathcal{V}'_n$ omits hashing a transcript component other than the public input. Malicious $\mathcal{P}'_n$ chooses public input x' (*adaptive soundness attack*), then:

$\mathcal{P}'_n$ Mounts a 6-Steps Attack Against $\mathcal{V}'_n$:

1. $\mathcal{P}'_n$ simulates a version of the pairing-based PCS underlying the SNARK. All inputs to the PCS are chosen by $\mathcal{P}'_n$.
2. $\mathcal{P}'_n$ chooses freely all components of proof π' apart from those omitted by $\mathcal{V}'_n$ in the transcript (i.e., $\mathcal{V}'_n$'s degrees of freedom).
3. Using x' and the above values, $\mathcal{P}'_n$ simulates* $\mathcal{P}_n$ and instantiates the system of constraints with $\mathcal{V}'_n$ degrees of freedom as the only unknowns.
4. $\mathcal{P}'_n$ solves the above system, if feasible.
5. $\mathcal{P}'_n$ fills in the remaining components of π' using a solution to the system.

New Soundness Attack on FS Implementations

Assume deviating $\mathcal{V}'_n$ omits hashing a transcript component other than the public input. Malicious $\mathcal{P}'_n$ chooses public input x' (*adaptive soundness attack*), then:

$\mathcal{P}'_n$ Mounts a 6-Steps Attack Against $\mathcal{V}'_n$:

1. $\mathcal{P}'_n$ simulates a version of the pairing-based PCS underlying the SNARK. All inputs to the PCS are chosen by $\mathcal{P}'_n$.
2. $\mathcal{P}'_n$ chooses freely all components of proof π' apart from those omitted by $\mathcal{V}'_n$ in the transcript (i.e., $\mathcal{V}'_n$'s degrees of freedom).
3. Using x' and the above values, $\mathcal{P}'_n$ simulates* $\mathcal{P}_n$ and instantiates the system of constraints with $\mathcal{V}'_n$ degrees of freedom as the only unknowns.
4. $\mathcal{P}'_n$ solves the above system, if feasible.
5. $\mathcal{P}'_n$ fills in the remaining components of π' using a solution to the system.
6. $\mathcal{V}'_n$ accepts π' as valid with probability 1.

The Last Challenge Attack - Context

Short Background

Secure pairing function ($\boldsymbol{e}$): bilinear, non-degenerate. First argument (e.g., $[\boldsymbol{a}]_{\mathbf{1}}$): EC point in $\mathbb{F}_{\boldsymbol{p}} \times \mathbb{F}_{\boldsymbol{p}}$. Second argument (e.g., $[\boldsymbol{b}]_{\mathbf{2}}$) in $\mathbb{F}_{\boldsymbol{p}^k} \times \mathbb{F}_{\boldsymbol{p}^k}$. $\boldsymbol{a}, \boldsymbol{b}$ scalars in $\mathbb{F}_{\boldsymbol{r}}$.

The Last Challenge Attack - Context

Short Background

Secure pairing function (e): bilinear, non-degenerate. First argument (e.g., $[a]_1$): EC point in $\mathbb{F}_p \times \mathbb{F}_p$. Second argument (e.g., $[b]_2$) in $\mathbb{F}_{p^k} \times \mathbb{F}_{p^k}$. a, b scalars in $\mathbb{F}_r$.

The KZG-based $\mathcal{P}_{PLONK}$ Prover Simplified

R.	In FS Chall.	Transcript	Out FS Chall.

The Last Challenge Attack - Context

Short Background

Secure pairing function (e): bilinear, non-degenerate. First argument (e.g., $[a]_1$): EC point in $\mathbb{F}_p \times \mathbb{F}_p$. Second argument (e.g., $[b]_2$) in $\mathbb{F}_{p^k} \times \mathbb{F}_{p^k}$. a, b scalars in $\mathbb{F}_r$.

The KZG-based $\mathcal{P}_{PLONK}$ Prover Simplified

R.	In FS Chall.	Transcript	Out FS Chall.
1	$\emptyset$	$Tr_1 = (pp, [a]_1, [b]_1, [c]_1)$	$\beta = \text{hash}(Tr_1, 0)$ $\gamma = \text{hash}(Tr_1, 1)$

The Last Challenge Attack - Context

Short Background

Secure pairing function (e): bilinear, non-degenerate. First argument (e.g., $[a]_1$): EC point in $\mathbb{F}_p \times \mathbb{F}_p$. Second argument (e.g., $[b]_2$) in $\mathbb{F}_{p^k} \times \mathbb{F}_{p^k}$. a, b scalars in $\mathbb{F}_r$.

The KZG-based $\mathcal{P}_{PLONK}$ Prover Simplified

R.	In FS Chall.	Transcript	Out FS Chall.
1	$\emptyset$	$Tr_1 = (pp, [a]_1, [b]_1, [c]_1)$	$\beta = \text{hash}(Tr_1, 0)$ $\gamma = \text{hash}(Tr_1, 1)$
2	β, γ	$Tr_2 = [z]_1$	$\alpha = \text{hash}(Tr_1, Tr_2)$

The Last Challenge Attack - Context

Short Background

Secure pairing function ($\boldsymbol{e}$): bilinear, non-degenerate. First argument (e.g., $[\boldsymbol{a}]_1$): EC point in $\mathbb{F}_p \times \mathbb{F}_p$. Second argument (e.g., $[\boldsymbol{b}]_2$) in $\mathbb{F}_{p^k} \times \mathbb{F}_{p^k}$. $\boldsymbol{a}, \boldsymbol{b}$ scalars in $\mathbb{F}_r$.

The KZG-based $\mathcal{P}_{PLONK}$ Prover Simplified

R.	In FS Chall.	Transcript	Out FS Chall.
1	$\emptyset$	$Tr_1 = (pp, [a]_1, [b]_1, [c]_1)$	$\beta = \text{hash}(Tr_1, 0)$ $\gamma = \text{hash}(Tr_1, 1)$
2	β, γ	$Tr_2 = [z]_1$	$\alpha = \text{hash}(Tr_1, Tr_2)$
3	β, γ, α	$Tr_3 = ([t_{lo}]_1, [t_{mi}]_1, [t_{hi}]_1)$	$\mathfrak{z} = \text{hash}(Tr_1, Tr_2, Tr_3)$

The Last Challenge Attack - Context

Short Background

Secure pairing function (e): bilinear, non-degenerate. First argument (e.g., $[a]_1$): EC point in $\mathbb{F}_p \times \mathbb{F}_p$. Second argument (e.g., $[b]_2$) in $\mathbb{F}_{p^k} \times \mathbb{F}_{p^k}$. a, b scalars in $\mathbb{F}_r$.

The KZG-based $\mathcal{P}_{PLONK}$ Prover Simplified

R.	In FS Chall.	Transcript	Out FS Chall.
1	$\emptyset$	$Tr_1 = (pp, [a]_1, [b]_1, [c]_1)$	$\beta = \text{hash}(Tr_1, 0)$ $\gamma = \text{hash}(Tr_1, 1)$
2	β, γ	$Tr_2 = [z]_1$	$\alpha = \text{hash}(Tr_1, Tr_2)$
3	β, γ, α	$Tr_3 = ([t_{lo}]_1, [t_{mi}]_1, [t_{hi}]_1)$	$\mathfrak{z} = \text{hash}(Tr_1, Tr_2, Tr_3)$
4	$\beta, \gamma, \alpha, \mathfrak{z}$	$Tr_4 = (\bar{a}, \bar{b}, \bar{c}, \bar{S}_{\sigma 1}, \bar{S}_{\sigma 2}, \bar{z}_\omega)$	$v = \text{hash}(Tr_1, Tr_2, Tr_3, Tr_4)$

The Last Challenge Attack - Context

Short Background

Secure pairing function (e): bilinear, non-degenerate. First argument (e.g., $[a]_1$): EC point in $\mathbb{F}_p \times \mathbb{F}_p$. Second argument (e.g., $[b]_2$) in $\mathbb{F}_{p^k} \times \mathbb{F}_{p^k}$. a, b scalars in $\mathbb{F}_r$.

The KZG-based $\mathcal{P}_{PLONK}$ Prover Simplified

R.	In FS Chall.	Transcript	Out FS Chall.
1	$\emptyset$	$Tr_1 = (pp, [a]_1, [b]_1, [c]_1)$	$\beta = \text{hash}(Tr_1, 0)$ $\gamma = \text{hash}(Tr_1, 1)$
2	β, γ	$Tr_2 = [z]_1$	$\alpha = \text{hash}(Tr_1, Tr_2)$
3	β, γ, α	$Tr_3 = ([t_{lo}]_1, [t_{mi}]_1, [t_{hi}]_1)$	$\mathfrak{z} = \text{hash}(Tr_1, Tr_2, Tr_3)$
4	$\beta, \gamma, \alpha, \mathfrak{z}$	$Tr_4 = (\bar{a}, \bar{b}, \bar{c}, \bar{S}_{\sigma 1}, \bar{S}_{\sigma 2}, \bar{z}_\omega)$	$v = \text{hash}(Tr_1, Tr_2, Tr_3, Tr_4)$
5	$\beta, \gamma, \alpha, \mathfrak{z}, v$	$Tr_5 = ([W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1,)$	$u = \text{hash}(Tr_1, Tr_2, Tr_3, Tr_4, Tr_5)$

The Last Challenge Attack - Context

Short Background

Secure pairing function (e): bilinear, non-degenerate. First argument (e.g., $[a]_1$): EC point in $\mathbb{F}_p \times \mathbb{F}_p$. Second argument (e.g., $[b]_2$) in $\mathbb{F}_{p^k} \times \mathbb{F}_{p^k}$. a, b scalars in $\mathbb{F}_r$.

The KZG-based $\mathcal{P}_{PLONK}$ Prover Simplified

R.	In FS Chall.	Transcript	Out FS Chall.
1	$\emptyset$	$Tr_1 = (pp, [a]_1, [b]_1, [c]_1)$	$\beta = \text{hash}(Tr_1, 0)$ $\gamma = \text{hash}(Tr_1, 1)$
2	β, γ	$Tr_2 = [z]_1$	$\alpha = \text{hash}(Tr_1, Tr_2)$
3	β, γ, α	$Tr_3 = ([t_{lo}]_1, [t_{mi}]_1, [t_{hi}]_1)$	$\mathfrak{z} = \text{hash}(Tr_1, Tr_2, Tr_3)$
4	$\beta, \gamma, \alpha, \mathfrak{z}$	$Tr_4 = (\bar{a}, \bar{b}, \bar{c}, \bar{S}_{\sigma 1}, \bar{S}_{\sigma 2}, \bar{z}_\omega)$	$v = \text{hash}(Tr_1, Tr_2, Tr_3, Tr_4)$
5	$\beta, \gamma, \alpha, \mathfrak{z}, v$	$Tr_5 = ([W_\mathfrak{z}]_1, [W_{\mathfrak{z}\omega}]_1,)$	$u = \text{hash}(Tr_1, Tr_2, Tr_3, Tr_4, Tr_5)$

$$\textbf{Proof } \pi_{PLONK} = \begin{pmatrix} [a]_1, [b]_1, [c]_1, [z]_1, [t_{lo}]_1, [t_{mi}]_1, [t_{hi}]_1, [W_\mathfrak{z}]_1, [W_{\mathfrak{z}\omega}]_1, \\ \bar{a}, \bar{b}, \bar{c}, \bar{S}_{\sigma 1}, \bar{S}_{\sigma 2}, \bar{z}_\omega \end{pmatrix}$$

The Last Challenge Attack - Context

Short Background

Secure pairing function ($\boldsymbol{e}$): bilinear, non-degenerate. First argument (e.g., $[\boldsymbol{a}]_1$): EC point in $\mathbb{F}_p \times \mathbb{F}_p$. Second argument (e.g., $[\boldsymbol{b}]_2$) in $\mathbb{F}_{p^k} \times \mathbb{F}_{p^k}$. $\boldsymbol{a}, \boldsymbol{b}$ scalars in $\mathbb{F}_r$.

The KZG-based $\mathcal{P}_{PLONK}$ Prover Simplified

R.	In FS Chall.	Transcript	Out FS Chall.
1	$\emptyset$	$Tr_1 = (pp, [a]_1, [b]_1, [c]_1)$	$\beta = \text{hash}(Tr_1, 0)$ $\gamma = \text{hash}(Tr_1, 1)$
2	β, γ	$Tr_2 = [z]_1$	$\alpha = \text{hash}(Tr_1, Tr_2)$
3	β, γ, α	$Tr_3 = ([t_{lo}]_1, [t_{mi}]_1, [t_{hi}]_1)$	$\mathfrak{z} = \text{hash}(Tr_1, Tr_2, Tr_3)$
4	$\beta, \gamma, \alpha, \mathfrak{z}$	$Tr_4 = (\bar{a}, \bar{b}, \bar{c}, \bar{S}_{\sigma 1}, \bar{S}_{\sigma 2}, \bar{z}_\omega)$	$v = \text{hash}(Tr_1, Tr_2, Tr_3, Tr_4)$
5	$\beta, \gamma, \alpha, \mathfrak{z}, v$	$Tr_5 = ([W_\mathfrak{z}]_1, [W_{\mathfrak{z}\omega}]_1,)$	$u = \text{hash}(Tr_1, Tr_2, Tr_3, Tr_4, Tr_5)$

$$\textbf{Proof } \pi_{PLONK} = \begin{pmatrix} [a]_1, [b]_1, [c]_1, [z]_1, [t_{lo}]_1, [t_{mi}]_1, [t_{hi}]_1, [W_\mathfrak{z}]_1, [W_{\mathfrak{z}\omega}]_1, \\ \bar{a}, \bar{b}, \bar{c}, \bar{S}_{\sigma 1}, \bar{S}_{\sigma 2}, \bar{z}_\omega \end{pmatrix}$$

Warning! The last challenge $\boldsymbol{u}$ is not used at all by $\mathcal{P}_{PLONK}$.

The Last Challenge Attack - Context

Short Background

Secure pairing function ($\boldsymbol{e}$): bilinear, non-degenerate. First argument (e.g., $[\boldsymbol{a}]_1$): EC point in $\mathbb{F}_p \times \mathbb{F}_p$. Second argument (e.g., $[\boldsymbol{b}]_2$) in $\mathbb{F}_{p^k} \times \mathbb{F}_{p^k}$. $\boldsymbol{a}, \boldsymbol{b}$ scalars in $\mathbb{F}_r$.

The KZG-based $\mathcal{P}_{PLONK}$ Prover Simplified

R.	In FS Chall.	Transcript	Out FS Chall.
1	$\emptyset$	$Tr_1 = (pp, [a]_1, [b]_1, [c]_1)$	$\beta = \text{hash}(Tr_1, 0)$ $\gamma = \text{hash}(Tr_1, 1)$
2	β, γ	$Tr_2 = [z]_1$	$\alpha = \text{hash}(Tr_1, Tr_2)$
3	β, γ, α	$Tr_3 = ([t_{lo}]_1, [t_{mi}]_1, [t_{hi}]_1)$	$\mathfrak{z} = \text{hash}(Tr_1, Tr_2, Tr_3)$
4	$\beta, \gamma, \alpha, \mathfrak{z}$	$Tr_4 = (\bar{a}, \bar{b}, \bar{c}, \bar{S}_{\sigma 1}, \bar{S}_{\sigma 2}, \bar{z}_\omega)$	$v = \text{hash}(Tr_1, Tr_2, Tr_3, Tr_4)$
5	$\beta, \gamma, \alpha, \mathfrak{z}, v$	$Tr_5 = ([W_\mathfrak{z}]_1, [W_{\mathfrak{z}\omega}]_1,)$	$u = \text{hash}(Tr_1, Tr_2, Tr_3, Tr_4, Tr_5)$

$$\textbf{Proof } \pi_{PLONK} = \begin{pmatrix} [a]_1, [b]_1, [c]_1, [z]_1, [t_{lo}]_1, [t_{mi}]_1, [t_{hi}]_1, [W_\mathfrak{z}]_1, [W_{\mathfrak{z}\omega}]_1, \\ \bar{a}, \bar{b}, \bar{c}, \bar{S}_{\sigma 1}, \bar{S}_{\sigma 2}, \bar{z}_\omega \end{pmatrix}$$

Warning! The last challenge $\boldsymbol{u}$ is not used at all by $\mathcal{P}_{PLONK}$.

The KZG-based $\mathcal{V}_{PLONK}$ Verifier Simplified

(Mainly) re-computes the FS challenges, $[\boldsymbol{E}]_1$, $[\boldsymbol{F}]_1$; verifies the pairing equation:

$$e([W_\mathfrak{z}]_1 + u \cdot [W_{\mathfrak{z}\omega}]_1, [x]_2) \stackrel{?}{=} e(\mathfrak{z} \cdot [W_\mathfrak{z}]_1 + u\mathfrak{z}\omega \cdot [W_{\mathfrak{z}\omega}]_1 + [F]_1 - [E]_1, [1]_2).$$

The Last Challenge Attack - Steps 1–3

$$e([W_{\mathfrak{z}}]_1 + u \cdot [W_{\mathfrak{z}\omega}]_1, [x]_2) \stackrel{?}{=} e(\mathfrak{z} \cdot [W_{\mathfrak{z}}]_1 + u\mathfrak{z}\omega \cdot [W_{\mathfrak{z}\omega}]_1 + [F]_1 - [E]_1, [1]_2)$$

The Last Challenge Attack - Steps 1–3

$$e([W_{\mathfrak{z}}]_1 + u \cdot [W_{\mathfrak{z}\omega}]_1, [x]_2) \stackrel{?}{=} e(\mathfrak{z} \cdot [W_{\mathfrak{z}}]_1 + u\mathfrak{z}\omega \cdot [W_{\mathfrak{z}\omega}]_1 + [F]_1 - [E]_1, [1]_2)$$

Can $\mathcal{V}_{PLONK}$ be Made More Efficient?

$\mathcal{V}_{PLONK}$ computes $u = \text{hash}(Tr_1, Tr_2, Tr_3, Tr_4, Tr_5)$, $Tr_5 = ([W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1)$.
If $\mathcal{V}_{PLONK}$ omits part of the full transcript for u, is soundness still preserved?
Assume deviating $\mathcal{V}'_{PLONK}$ does not include Tr_5 in the transcript for u.

The Last Challenge Attack - Steps 1–3

$$e([W_{\mathfrak{z}}]_1 + u \cdot [W_{\mathfrak{z}\omega}]_1, [x]_2) \stackrel{?}{=} e(\mathfrak{z} \cdot [W_{\mathfrak{z}}]_1 + u\mathfrak{z}\omega \cdot [W_{\mathfrak{z}\omega}]_1 + [F]_1 - [E]_1, [1]_2)$$

Can $\mathcal{V}_{PLONK}$ be Made More Efficient?

$\mathcal{V}_{PLONK}$ computes $u = \text{hash}(Tr_1, Tr_2, Tr_3, Tr_4, Tr_5)$, $Tr_5 = ([W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1)$.
If $\mathcal{V}_{PLONK}$ omits part of the full transcript for u, is soundness still preserved?
Assume deviating $\mathcal{V}'_{PLONK}$ does not include Tr_5 in the transcript for u.

LCA: Malicious $\mathcal{P}'_{PLONK}$ Steps 1–3

1. Simulating a KZG commitment to a polynomial of its choice, $\mathcal{P}'_{PLONK}$ produces A, B s.t.: $e(A, [x]_2) = e(B, [1]_2)$.

The Last Challenge Attack - Steps 1–3

$$e([W_{\mathfrak{z}}]_1 + u \cdot [W_{\mathfrak{z}\omega}]_1, [x]_2) \stackrel{?}{=} e(\mathfrak{z} \cdot [W_{\mathfrak{z}}]_1 + u\mathfrak{z}\omega \cdot [W_{\mathfrak{z}\omega}]_1 + [F]_1 - [E]_1, [1]_2)$$

Can $\mathcal{V}_{PLONK}$ be Made More Efficient?

$\mathcal{V}_{PLONK}$ computes $u = \text{hash}(Tr_1, Tr_2, Tr_3, Tr_4, Tr_5)$, $Tr_5 = ([W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1)$.
If $\mathcal{V}_{PLONK}$ omits part of the full transcript for u, is soundness still preserved?
Assume deviating $\mathcal{V}'_{PLONK}$ does not include Tr_5 in the transcript for u.

LCA: Malicious $\mathcal{P}'_{PLONK}$ Steps 1–3

1. Simulating a KZG commitment to a polynomial of its choice, $\mathcal{P}'_{PLONK}$ produces A, B s.t.: $e(A, [x]_2) = e(B, [1]_2)$.
2. $\mathcal{P}'_{PLONK}$ chooses freely *all public inputs* and proof components except for $[W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1$, preparing to produce a false proof:

$$\pi'_{PLONK} = \left(\begin{array}{l} [a]_1, [b]_1, [c]_1, [z]_1, [t_{lo}]_1, [t_{mi}]_1, [t_{hi}]_1, \\ [W_{\mathfrak{z}}]_1, [W_{\mathfrak{z}\omega}]_1, \bar{a}, \bar{b}, \bar{c}, \bar{S}_{\sigma 1}, \bar{S}_{\sigma 2}, \bar{z}_{\omega} \end{array} \right)$$

The Last Challenge Attack - Steps 1–3

$$e([W_\mathfrak{z}]_1 + u \cdot [W_{\mathfrak{z}\omega}]_1, [x]_2) \stackrel{?}{=} e(\mathfrak{z} \cdot [W_\mathfrak{z}]_1 + u\mathfrak{z}\omega \cdot [W_{\mathfrak{z}\omega}]_1 + [F]_1 - [E]_1, [1]_2)$$

Can $\mathcal{V}_{PLONK}$ be Made More Efficient?

$\mathcal{V}_{PLONK}$ computes $u = \text{hash}(Tr_1, Tr_2, Tr_3, Tr_4, Tr_5)$, $Tr_5 = ([W_\mathfrak{z}]_1, [W_{\mathfrak{z}\omega}]_1)$.
If $\mathcal{V}_{PLONK}$ omits part of the full transcript for u, is soundness still preserved?
Assume deviating $\mathcal{V}'_{PLONK}$ does not include Tr_5 in the transcript for u.

LCA: Malicious $\mathcal{P}'_{PLONK}$ Steps 1–3

1. Simulating a KZG commitment to a polynomial of its choice, $\mathcal{P}'_{PLONK}$ produces A, B s.t.: $e(A, [x]_2) = e(B, [1]_2)$.
2. $\mathcal{P}'_{PLONK}$ chooses freely *all public inputs* and proof components except for $[W_\mathfrak{z}]_1, [W_{\mathfrak{z}\omega}]_1$, preparing to produce a false proof:

$$\pi'_{PLONK} = \begin{pmatrix} [a]_1, [b]_1, [c]_1, [z]_1, [t_{lo}]_1, [t_{mi}]_1, [t_{hi}]_1, \\ [W_\mathfrak{z}]_1, [W_{\mathfrak{z}\omega}]_1, \bar{a}, \bar{b}, \bar{c}, \bar{S}_{\sigma 1}, \bar{S}_{\sigma 2}, \bar{z}_\omega \end{pmatrix}$$

3. Following honest $\mathcal{V}_{PLONK}$'s computation and using the already chosen components in π'_{PLONK}, $\mathcal{P}'_{PLONK}$ computes $[F]_1$, $[E]_1$.

LCA: Malicious $\mathcal{P}'_{PLONK}$ Steps 4–6

4. $\mathcal{P}'_{PLONK}$ uses A, B from Step 1 and exploits the independence between u and $[W_{\mathfrak{z}}]_1$ and $[W_{\mathfrak{z}\omega}]_1$ to solve a system of **2** linear equations with **2** unknowns

$$\begin{cases} X + uY = A \\ \mathfrak{z}X + u\mathfrak{z}\omega Y + C = B \end{cases}$$

with $X = [W_{\mathfrak{z}}]_1,\ Y = [W_{\mathfrak{z}\omega}]_1$ as unknowns and $A, B, C, u, \mathfrak{z}, \omega$ as constants

$$e(\underbrace{\underbrace{[W_{\mathfrak{z}}]_1}_{X} + u \cdot \underbrace{[W_{\mathfrak{z}\omega}]_1}_{Y}}_{A}, [x]_2) \stackrel{?}{=} e(\underbrace{\mathfrak{z} \cdot \underbrace{[W_{\mathfrak{z}}]_1}_{X} + u\mathfrak{z}\omega \cdot \underbrace{[W_{\mathfrak{z}\omega}]_1}_{Y} + \underbrace{[F]_1 - [E]_1}_{C}}_{B}, [1]_2).$$

The Last Challenge Attack - Steps 4–6

LCA: Malicious $\mathcal{P}'_{PLONK}$ Steps 4–6

4. $\mathcal{P}'_{PLONK}$ uses $\boldsymbol{A}, \boldsymbol{B}$ from Step 1 and exploits the independence between $\boldsymbol{u}$ and $[\boldsymbol{W}_{\mathfrak{z}}]_1$ and $[\boldsymbol{W}_{\mathfrak{z}\omega}]_1$ to solve a system of **2** linear equations with **2** unknowns

$$\begin{cases} X + uY = A \\ \mathfrak{z}X + u\mathfrak{z}\omega Y + C = B \end{cases}$$

with $\boldsymbol{X} = [\boldsymbol{W}_{\mathfrak{z}}]_1$, $\boldsymbol{Y} = [\boldsymbol{W}_{\mathfrak{z}\omega}]_1$ as unknowns and $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{u}, \mathfrak{z}, \boldsymbol{\omega}$ as constants

$$e(\underbrace{\underbrace{[W_{\mathfrak{z}}]_1}_{X} + u \cdot \underbrace{[W_{\mathfrak{z}\omega}]_1}_{Y}}_{A}, [x]_2) \stackrel{?}{=} e(\underbrace{\mathfrak{z} \cdot \underbrace{[W_{\mathfrak{z}}]_1}_{X} + u\mathfrak{z}\omega \cdot \underbrace{[W_{\mathfrak{z}\omega}]_1}_{Y} + \underbrace{[F]_1 - [E]_1}_{C}}_{B}, [1]_2).$$

5. $\mathcal{P}'_{PLONK}$ fills in missing $\boldsymbol{X} = [\boldsymbol{W}_{\mathfrak{z}}]_1$, $\boldsymbol{Y} = [\boldsymbol{W}_{\mathfrak{z}\omega}]_1$ and completes $\boldsymbol{\pi}'_{PLONK}$.

The Last Challenge Attack - Steps 4–6

LCA: Malicious $\mathcal{P}'_{PLONK}$ Steps 4–6

4. $\mathcal{P}'_{PLONK}$ uses A, B from Step 1 and exploits the independence between u and $[W_{\mathfrak{z}}]_1$ and $[W_{\mathfrak{z}\omega}]_1$ to solve a system of 2 linear equations with 2 unknowns

$$\begin{cases} X + uY = A \\ \mathfrak{z}X + u\mathfrak{z}\omega Y + C = B \end{cases}$$

with $X = [W_{\mathfrak{z}}]_1,\ Y = [W_{\mathfrak{z}\omega}]_1$ as unknowns and $A, B, C, u, \mathfrak{z}, \omega$ as constants

$$e(\underbrace{\underbrace{[W_{\mathfrak{z}}]_1}_{X} + u \cdot \underbrace{[W_{\mathfrak{z}\omega}]_1}_{Y}}_{A}, [x]_2) \stackrel{?}{=} e(\underbrace{\mathfrak{z} \cdot \underbrace{[W_{\mathfrak{z}}]_1}_{X} + u\mathfrak{z}\omega \cdot \underbrace{[W_{\mathfrak{z}\omega}]_1}_{Y} + \underbrace{[F]_1 - [E]_1}_{C}}_{B}, [1]_2).$$

5. $\mathcal{P}'_{PLONK}$ fills in missing $X =$ $[W_{\mathfrak{z}}]_1$, $Y =$ $[W_{\mathfrak{z}\omega}]_1$ and completes π'_{PLONK}.
6. Deviating $\mathcal{V}'_{PLONK}$ accepts false proof π'_{PLONK} as valid with probability 1!

Fiat-Shamir Array Inputs Not Transcribed - Context

FFLONK: a variant of PLONK SNARK having the most efficient verifier at the expense of increased proof length. Underlying PCS: KZG-based SHPLONK.

FFLONK: a variant of PLONK SNARK having the most efficient verifier at the expense of increased proof length. Underlying PCS: KZG-based SHPLONK.

The $\mathcal{P}_{FFLONK}$ Prover Simplified

R.	In FS Chall.	Transcript	Out FS Chall.
1	$\emptyset$	$Tr_1 = (pp, \mathrm{PI}, [c_0]_1, [c_1]_1)$	$\beta = \mathrm{hash}(Tr_1, 0)$ $\gamma = \mathrm{hash}(Tr_1, 1)$
2	β, γ	$Tr_2 = [c_2]_1$	$\mathfrak{z} = (\mathrm{hash}(Tr_1, Tr_2))^{24}$
3	$\beta, \gamma, \mathfrak{z}$	$Tr_3 = (\bar{q_L}, \ldots, \bar{t_0})$	$\alpha = \mathrm{hash}(Tr_1, Tr_2, Tr_3)$
4	$\beta, \gamma, \mathfrak{z}, \alpha$	$Tr_4 = ([W]_1)$	$y = \mathrm{hash}(Tr_1, Tr_2, Tr_3, Tr_4)$
5	$\beta, \gamma, \mathfrak{z}, \alpha, y$	$Tr_5 = ([W']_1)$	

$$\pi_{FFLONK} = \left(\begin{array}{c} [c_1]_1, [c_2]_1, [W]_1, [W']_1, \\ \bar{q_L}, \bar{q_R}, \bar{q_O}, \bar{q_M}, \bar{q_{const}}, \bar{S}_{\sigma_1}, \bar{S}_{\sigma_2}, \bar{S}_{\sigma_3}, \bar{a}, \bar{b}, \bar{c}, \bar{z}, \bar{z}_\omega, \bar{t_1}, \bar{t_2}, \bar{t_1}_\omega, \bar{t_2}_\omega, \bar{t_0} \end{array} \right)$$

Fiat-Shamir Array Inputs Not Transcribed - Context

FFLONK: a variant of PLONK SNARK having the most efficient verifier at the expense of increased proof length. Underlying PCS: KZG-based SHPLONK.

The $\mathcal{P}_{FFLONK}$ Prover Simplified

R.	In FS Chall.	Transcript	Out FS Chall.
1	$\emptyset$	$Tr_1 = (pp, \mathrm{PI}, [c_0]_1, [c_1]_1)$	$\beta = \mathrm{hash}(Tr_1, 0)$ $\gamma = \mathrm{hash}(Tr_1, 1)$
2	β, γ	$Tr_2 = [c_2]_1$	$\mathfrak{z} = (\mathrm{hash}(Tr_1, Tr_2))^{24}$
3	$\beta, \gamma, \mathfrak{z}$	$Tr_3 = (\bar{q_L}, \ldots, \bar{t_0})$	$\alpha = \mathrm{hash}(Tr_1, Tr_2, Tr_3)$
4	$\beta, \gamma, \mathfrak{z}, \alpha$	$Tr_4 = ([W]_1)$	$y = \mathrm{hash}(Tr_1, Tr_2, Tr_3, Tr_4)$
5	$\beta, \gamma, \mathfrak{z}, \alpha, y$	$Tr_5 = ([W']_1)$	

$$\pi_{FFLONK} = \left(\begin{array}{c} \underline{[c_1]_1, [c_2]_1}, [W]_1, [W']_1, \\ \bar{q_L}, \bar{q_R}, \bar{q_O}, \bar{q_M}, \bar{q_{const}}, \bar{S_{\sigma_1}}, \bar{S_{\sigma_2}}, \bar{S_{\sigma_3}}, \bar{a}, \bar{b}, \bar{c}, \bar{z}, \bar{z_\omega}, \bar{t_1}, \bar{t_2}, \bar{t_{1\omega}}, \bar{t_{2\omega}}, \bar{t_0} \end{array} \right)$$

The $\mathcal{V}_{FFLONK}$ Verifier Simplified

Re-computes the FS challenges, $[D]_1$ and the meaningful scalar ms and verifies the pairing equation: $e([D]_1 - ms \cdot [1]_1, [1]_2) \stackrel{?}{=} e([W']_1, [x]_2)$.

Fiat-Shamir Array Inputs Not Transcribed - Context

FFLONK: a variant of PLONK SNARK having the most efficient verifier at the expense of increased proof length. Underlying PCS: KZG-based SHPLONK.

The $\mathcal{P}_{FFLONK}$ Prover Simplified

R.	In FS Chall.	Transcript	Out FS Chall.
1	$\emptyset$	$Tr_1 = (pp, \mathrm{PI}, [c_0]_1, [c_1]_1)$	$\beta = \mathrm{hash}(Tr_1, 0)$ $\gamma = \mathrm{hash}(Tr_1, 1)$
2	β, γ	$Tr_2 = [c_2]_1$	$\mathfrak{z} = (\mathrm{hash}(Tr_1, Tr_2))^{24}$
3	$\beta, \gamma, \mathfrak{z}$	$Tr_3 = (\bar{q_L}, \ldots, \bar{t_0})$	$\alpha = \mathrm{hash}(Tr_1, Tr_2, Tr_3)$
4	$\beta, \gamma, \mathfrak{z}, \alpha$	$Tr_4 = ([W]_1)$	$y = \mathrm{hash}(Tr_1, Tr_2, Tr_3, Tr_4)$
5	$\beta, \gamma, \mathfrak{z}, \alpha, y$	$Tr_5 = ([W']_1)$	

$$\pi_{FFLONK} = \left(\begin{array}{c} [c_1]_1, [c_2]_1, [W]_1, [W']_1, \\ \bar{q_L}, \bar{q_R}, \bar{q_O}, \bar{q_M}, \bar{q_{const}}, \bar{S_{\sigma_1}}, \bar{S_{\sigma_2}}, \bar{S_{\sigma_3}}, \bar{a}, \bar{b}, \bar{c}, \bar{z}, \bar{z_\omega}, \bar{t_1}, \bar{t_2}, \bar{t_{1\omega}}, \bar{t_{2\omega}}, \bar{t_0} \end{array} \right)$$

The $\mathcal{V}_{FFLONK}$ Verifier Simplified

Re-computes the FS challenges, $[D]_1$ and the meaningful scalar ms and verifies the pairing equation: $e([D]_1 - ms \cdot [1]_1, [1]_2) \stackrel{?}{=} e([W']_1, [x]_2)$.

What if a deviating $\mathcal{V}'_{FFLONK}$ omits checking the length of evaluations array Tr_3?

Fiat-Shamir Array Inputs Not Transcribed - Steps 1–3

Can $\mathcal{V}_{FFLONK}$ Safely Omit Checking Length of Evaluations Array?

Assume $\mathcal{V}'_{FFLONK}$ is identical to $\mathcal{V}_{FFLONK}$ but it does not check the length of the evaluations array. Can $\mathcal{P}'_{FFLONK}$ set the evaluations array to an empty one?

Fiat-Shamir Array Inputs Not Transcribed - Steps 1–3

Can $\mathcal{V}_{FFLONK}$ Safely Omit Checking Length of Evaluations Array?

Assume $\mathcal{V}'_{FFLONK}$ is identical to $\mathcal{V}_{FFLONK}$ but it does not check the length of the evaluations array. Can $\mathcal{P}'_{FFLONK}$ set the evaluations array to an empty one?

FAINT: Malicious $\mathcal{P}'_{FFLONK}$ Steps 1–3

1. Simulating a version of SHPLONK* to a public polynomial* and two freely chosen polynomials plus an empty array for evaluations, $\mathcal{P}'_{FFLONK}$ produces $[c_1]_1$, $[c_2]_1$, $[W]_1$, $[W']_1$, β, γ, $\mathfrak{z}$, α, y verifying the respective pairing check.

Fiat-Shamir Array Inputs Not Transcribed - Steps 1–3

Can $\mathcal{V}_{FFLONK}$ Safely Omit Checking Length of Evaluations Array?

Assume $\mathcal{V}'_{FFLONK}$ is identical to $\mathcal{V}_{FFLONK}$ but it does not check the length of the evaluations array. Can $\mathcal{P}'_{FFLONK}$ set the evaluations array to an empty one?

FAINT: Malicious $\mathcal{P}'_{FFLONK}$ Steps 1–3

1. Simulating a version of SHPLONK* to a public polynomial* and two freely chosen polynomials plus an empty array for evaluations, $\mathcal{P}'_{FFLONK}$ produces $[c_1]_1$, $[c_2]_1$, $[W]_1$, $[W']_1$, β, γ, $\mathfrak{z}$, α, y verifying the respective pairing check.
2. $\mathcal{P}'_{FFLONK}$ chooses freely *all public inputs* and proof components except for the scalars representing polynomial evaluations, preparing to produce false proof:

$$\pi'_{FFLONK} = \begin{pmatrix} [c_1]_1, [c_2]_1, [W]_1, [W']_1, \\ \bar{q_L}, \bar{q_R}, \bar{q_O}, \bar{q_M}, \bar{q_{const}}, \bar{S}_{\sigma_1}, \bar{S}_{\sigma_2}, \bar{S}_{\sigma_3}, \\ \bar{a}, \bar{b}, \bar{c}, \bar{z}, \bar{z}_\omega, \bar{t_1}, \bar{t_2}, \bar{t_{1\omega}}, \bar{t_{2\omega}}, \bar{t_0} \end{pmatrix}$$

Fiat-Shamir Array Inputs Not Transcribed - Steps 1–3

Can $\mathcal{V}_{FFLONK}$ Safely Omit Checking Length of Evaluations Array?

Assume $\mathcal{V}'_{FFLONK}$ is identical to $\mathcal{V}_{FFLONK}$ but it does not check the length of the evaluations array. Can $\mathcal{P}'_{FFLONK}$ set the evaluations array to an empty one?

FAINT: Malicious $\mathcal{P}'_{FFLONK}$ Steps 1–3

1. Simulating a version of SHPLONK* to a public polynomial* and two freely chosen polynomials plus an empty array for evaluations, $\mathcal{P}'_{FFLONK}$ produces $[c_1]_1$, $[c_2]_1$, $[W]_1$, $[W']_1$, β, γ, $\mathfrak{z}$, α, y verifying the respective pairing check.
2. $\mathcal{P}'_{FFLONK}$ chooses freely *all public inputs* and proof components except for the scalars representing polynomial evaluations, preparing to produce false proof:

$$\pi'_{FFLONK} = \begin{pmatrix} [c_1]_1, [c_2]_1, [W]_1, [W']_1, \\ \bar{q_L}, \bar{q_R}, \bar{q_O}, \bar{q_M}, \bar{q_{const}}, \bar{S}_{\sigma_1}, \bar{S}_{\sigma_2}, \bar{S}_{\sigma_3}, \\ \bar{a}, \bar{b}, \bar{c}, \bar{z}, \bar{z_\omega}, \bar{t_1}, \bar{t_2}, \bar{t_{1\omega}}, \bar{t_{2\omega}}, \bar{t_0} \end{pmatrix}$$

3. Following honest $\mathcal{V}_{FFLONK}$'s computation and using the values obtained in Step 1 above, $\mathcal{P}'_{FFLONK}$ computes a scalar ms involved in the final check of $\mathcal{V}'_{FFLONK}$.

Fiat-Shamir Array Inputs Not Transcribed - Steps 4–6

FAINT: Malicious $\mathcal{P}'_{PLONK}$ Steps 4–6

4. $\mathcal{P}'_{FFLONK}$ solves for the vector of scalars

$$(\bar{q_L}, \bar{q_R}, \bar{q_O}, \bar{q_M}, \bar{q_{const}}, \bar{S}_{\sigma_1}, \bar{S}_{\sigma_2}, \bar{S}_{\sigma_3}, \bar{a}, \bar{b}, \bar{c}, \bar{z}, \bar{z}_\omega, \bar{t_1}, \bar{t_2}, \bar{t_1}_\omega, \bar{t_2}_\omega, \bar{t_0})$$

verifying the system of constraints

$$\begin{cases} \bar{t_0} \cdot Z_H(\mathfrak{z}) = \bar{q_L}\bar{a} + \bar{q_R}\bar{b} + \bar{q_O}\bar{c} + \bar{q_M}\bar{a}\bar{b} + \bar{q_C} + PI(\mathfrak{z}) \quad (1) \\ \bar{t_1} \cdot Z_H(\mathfrak{z}) = L_1(\mathfrak{z})(\bar{z} - 1) \quad (2) \\ \bar{t_2} \cdot Z_H(\mathfrak{z}) = \big[(\bar{a} + \beta\mathfrak{z} + \gamma)(\bar{b} + k_1\beta\mathfrak{z} + \gamma)(\bar{c} + k_2\beta\mathfrak{z} + \gamma)\bar{z} \\ \qquad -(\bar{a} + \beta\bar{S}_{\sigma_1} + \gamma)(\bar{b} + \beta\bar{S}_{\sigma_2} + \gamma)(\bar{c} + \beta\bar{S}_{\sigma_3} + \gamma)\bar{z}_\omega\big] \quad (3) \end{cases}$$

in addition to a constraint defining ms (4).

Note: $\beta, \gamma, \mathfrak{z}$ have already been set and $Z_H(X)$, $L_1(X)$ are public.

Fiat-Shamir Array Inputs Not Transcribed - Steps 4–6

FAINT: Malicious $\mathcal{P}'_{PLONK}$ Steps 4–6

4. $\mathcal{P}'_{FFLONK}$ solves for the vector of scalars

$$(\bar{q_L}, \bar{q_R}, \bar{q_O}, \bar{q_M}, \bar{q_{const}}, \bar{S}_{\sigma_1}, \bar{S}_{\sigma_2}, \bar{S}_{\sigma_3}, \bar{a}, \bar{b}, \bar{c}, \bar{z}, \bar{z}_\omega, \bar{t_1}, \bar{t_2}, \bar{t_1}_\omega, \bar{t_2}_\omega, \bar{t_0})$$

verifying the system of constraints

$$\begin{cases} \bar{t_0} \cdot Z_H(\mathfrak{z}) = \bar{q_L}\bar{a} + \bar{q_R}\bar{b} + \bar{q_O}\bar{c} + \bar{q_M}\bar{a}\bar{b} + \bar{q_C} + PI(\mathfrak{z}) \quad (1) \\ \bar{t_1} \cdot Z_H(\mathfrak{z}) = L_1(\mathfrak{z})(\bar{z} - 1) \quad (2) \\ \bar{t_2} \cdot Z_H(\mathfrak{z}) = \big[(\bar{a} + \beta\mathfrak{z} + \gamma)(\bar{b} + k_1\beta\mathfrak{z} + \gamma)(\bar{c} + k_2\beta\mathfrak{z} + \gamma)\bar{z} \\ \qquad -(\bar{a} + \beta\bar{S}_{\sigma_1} + \gamma)(\bar{b} + \beta\bar{S}_{\sigma_2} + \gamma)(\bar{c} + \beta\bar{S}_{\sigma_3} + \gamma)\bar{z}_\omega\big] \quad (3) \end{cases}$$

in addition to a constraint defining ms (4).

Note: $\beta, \gamma, \mathfrak{z}$ have already been set and $Z_H(X)$, $L_1(X)$ are public.

5. $\mathcal{P}'_{FFLONK}$ fills in the above computed evaluations and completes π'_{FFLONK}.

Fiat-Shamir Array Inputs Not Transcribed - Steps 4–6

FAINT: Malicious $\mathcal{P}'_{PLONK}$ Steps 4–6

4. $\mathcal{P}'_{FFLONK}$ solves for the vector of scalars

$$(\bar{q_L}, \bar{q_R}, \bar{q_O}, \bar{q_M}, \bar{q_{const}}, \bar{S}_{\sigma_1}, \bar{S}_{\sigma_2}, \bar{S}_{\sigma_3}, \bar{a}, \bar{b}, \bar{c}, \bar{z}, \bar{z}_\omega, \bar{t_1}, \bar{t_2}, \bar{t_1}_\omega, \bar{t_2}_\omega, \bar{t_0})$$

verifying the system of constraints

$$\begin{cases} \bar{t_0} \cdot Z_H(\mathfrak{z}) = \bar{q_L}\bar{a} + \bar{q_R}\bar{b} + \bar{q_O}\bar{c} + \bar{q_M}\bar{a}\bar{b} + \bar{q_C} + PI(\mathfrak{z}) \quad (1) \\ \bar{t_1} \cdot Z_H(\mathfrak{z}) = L_1(\mathfrak{z})(\bar{z} - 1) \quad (2) \\ \bar{t_2} \cdot Z_H(\mathfrak{z}) = \big[(\bar{a} + \beta\mathfrak{z} + \gamma)(\bar{b} + k_1\beta\mathfrak{z} + \gamma)(\bar{c} + k_2\beta\mathfrak{z} + \gamma)\bar{z} \\ \qquad -(\bar{a} + \beta\bar{S}_{\sigma_1} + \gamma)(\bar{b} + \beta\bar{S}_{\sigma_2} + \gamma)(\bar{c} + \beta\bar{S}_{\sigma_3} + \gamma)\bar{z}_\omega\big] \quad (3) \end{cases}$$

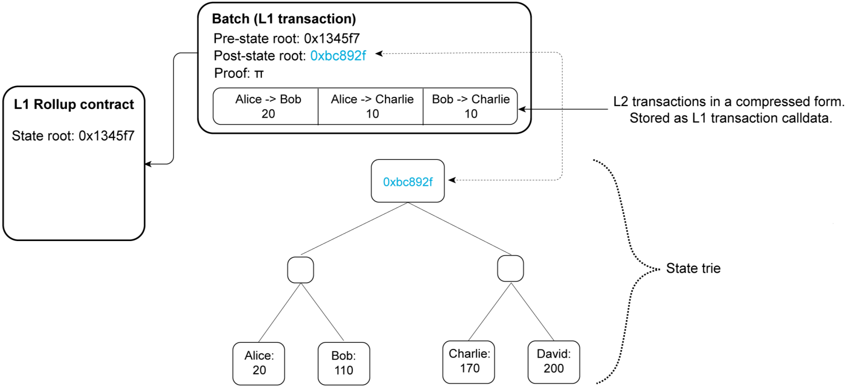
in addition to a constraint defining ms (4).

Note: $\beta, \gamma, \mathfrak{z}$ have already been set and $Z_H(X)$, $L_1(X)$ are public.

5. $\mathcal{P}'_{FFLONK}$ fills in the above computed evaluations and completes π'_{FFLONK}.
6. Deviating $\mathcal{V}'_{FFLONK}$ accepts false proof π'_{FFLONK} as valid with probability 1!

Implications

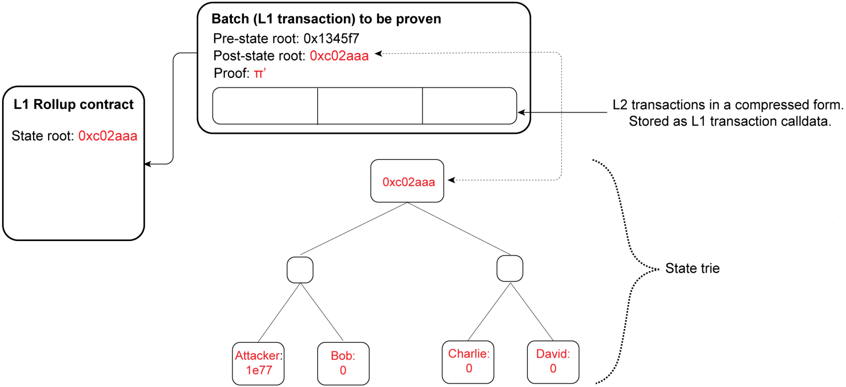
Let $\mathcal{P}'$ be a malicious SNARK prover interacting with a faulty verifier $\mathcal{V}$ as described above. Then $\mathcal{P}'$ can set itself as the owner of all the assets by changing the Merkle root (part of the **PI**) and steal all user funds.



Based on: `https://vitalik.eth.limo/general/2021/01/05/rollup.html`

Implications

Let $\mathcal{P}'$ be a malicious SNARK prover interacting with a faulty verifier $\mathcal{V}$ as described above. Then $\mathcal{P}'$ can set itself as the owner of all the assets by changing the Merkle root (part of the **PI**) and steal all user funds.



Based on: `https://vitalik.eth.limo/general/2021/01/05/rollup.html`

Attack on Statistical Zero-Knowledge

What Is ZK, Again? - The Intuition

An honest prover convinces any curious verifier of the validity of a statement on a secret witness, without disclosing even one bit of the witness.

Attack on Statistical Zero-Knowledge

What Is ZK, Again? - The Intuition

An honest prover convinces any curious verifier of the validity of a statement on a secret witness, without disclosing even one bit of the witness.

What If the Witness Is a Polynomial $\boldsymbol{a(X)}$ in a PCS?

Blind $\boldsymbol{a(X)}$ with $\boldsymbol{blind(X)}$, where $\boldsymbol{deg(blind(X))} \geq$ the number of opening points for the commitment to $\boldsymbol{a(X)}$. Ensures statistical ZK.

Attack on Statistical Zero-Knowledge

What Is ZK, Again? - The Intuition

An honest prover convinces any curious verifier of the validity of a statement on a secret witness, without disclosing even one bit of the witness.

What If the Witness Is a Polynomial $\boldsymbol{a(X)}$ in a PCS?

Blind $\boldsymbol{a(X)}$ with $\boldsymbol{blind(X)}$, where $\boldsymbol{deg(blind(X))} \geq$ the number of opening points for the commitment to $\boldsymbol{a(X)}$. Ensures statistical ZK.

What If the Witness is a Vector of Polynomials (in a SNARK)?

Blind each polynomial as before. *Are we then done?*

For some SNARKs and efficiency considerations, other polynomials need blinding.

Attack on Statistical Zero-Knowledge (cont.)

Attack Example: Missing Blinding of Shards in PLONK Prover

- PLONK prover Round 3, quotient polynomial:
  $t(X) = t_{lo}(X) + X^{n+2} \cdot t_{mid}(X) + X^{2n+4} \cdot t_{hi}(X)$.
- Missing blinding:

$$t'_{lo} = t_{lo}(X) + \rho_1 \cdot X^{n+2}$$
$$t'_{mid}(X) = t_{mid}(X) - \rho_1 + \rho_2 \cdot X^{n+2}$$
$$t'_{hi}(X) = t_{hi}(X) - \rho_2,$$

  where ρ_1, ρ_2 are random coefficients in $\mathbb{F}_p$.
- Attack on statistical ZK: Marek Sefranek, *How (Not) to Simulate PLONK*, SCN 2024.

Conclusions

On the Soundness Vulnerabilities

We introduced LCA and FAINT, two new types of soundness attacks on specific incorrect implementations of the FS transform for KZG-based SNARKs.

- LCA exploits that the last FS transform challenge is incorrectly computed as independent from some KZG-based SNARK proof components.
- FAINT exploits the fact the length of certain proof components is unchecked.

On the Zero-Knowledge Vulnerability

We highlighted a subtle attack on zero-knowledge encountered in a SNARK implementation.

Mind your blindings and your Fiat-Shamir-s!

Thank you!