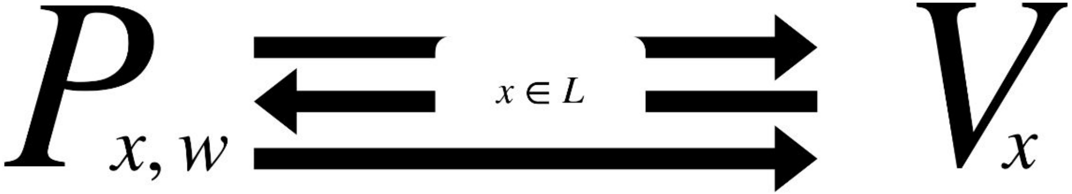
Security of Fiat-Shamir transformation

Michał Zając
Nethermind

Zero-knowledge proofs



x — statement, e.g. "Circuit C executed on input a outputs b"

w — witness, e.g., some private inputs to the program P, intermediate values of C's evaluation

Completeness Honest V always accepts a proof from a prover P

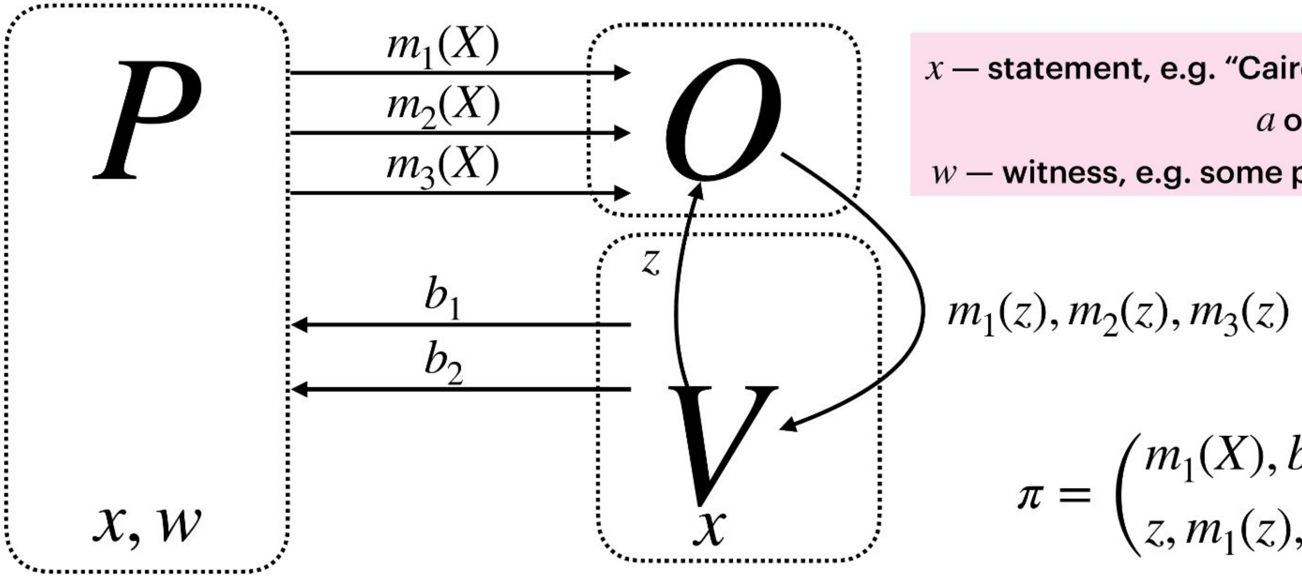
Soundness Probability that P^* makes V accept false statement is negligible

Knowledge soundness If P^*'s proof is accepted by V, then P knows w

Zero-knowledge V learns nothing new about x, except that $Rel(x, w)$

How SNARKs are built?

From PIOP to SNARKs



x — statement, e.g. “Cairo program P executed on input a outputs b”

w — witness, e.g. some private inputs to the program P

$$\pi = \begin{pmatrix} m_1(X), b_1, m_2(X), b_2, m_3(X), \\ z, m_1(z), m_2(z), m_3(z) \end{pmatrix}$$

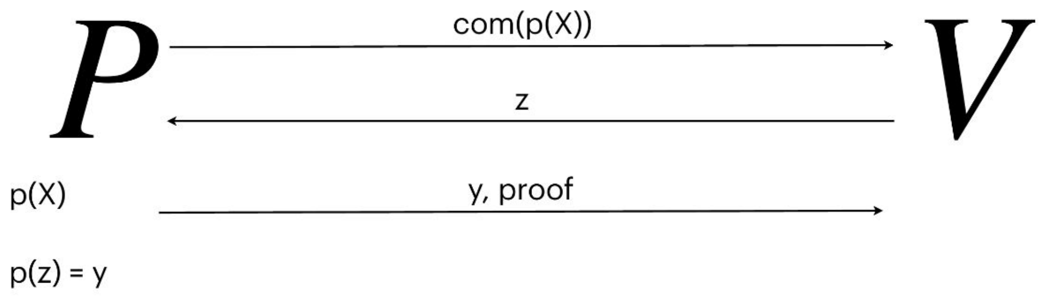
m1(X) * m2(X) = m3(X)
m1(z) * m2(z) = m3(z)

Problem 1 this proof is huge and we rely on O

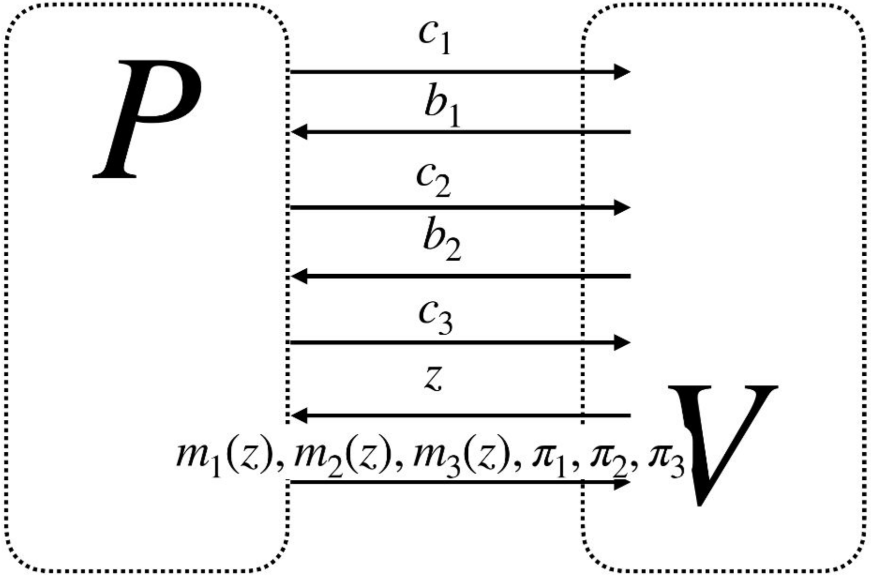
Polynomial commitments

Allow the **prover** to send a **short digest** that represents the polynomial **without reveling it**

Allow the **verifier** to **verifiably evaluate** the polynomial represented by the digest



From PIOP to SNARKs. Replace $\mathcal{O}$

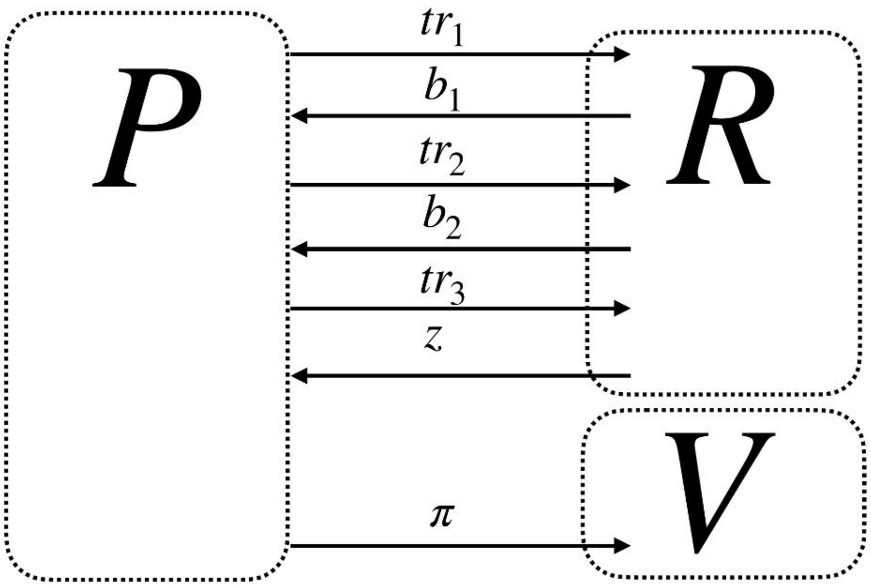


c_i polynomial commitment to polynomial $m_i(X)$
π_i proof for the polynomial commitment

$$\pi = \begin{pmatrix} c_1, b_1, c_2, b_2, c_3, \\ z, m_1(z), m_2(z), m_3(z) \\ \pi_1, \pi_2, \pi_3 \end{pmatrix}$$

Problem 2 the protocol is interactive

From PIOP to SNARKs. Fiat—Shamir transformation



c_i polynomial commitment to polynomial $m_i(X)$

π_i proof for the polynomial commitment

R random oracle

$tr_1 = x, c_1 \qquad b_1 = R(tr_1)$

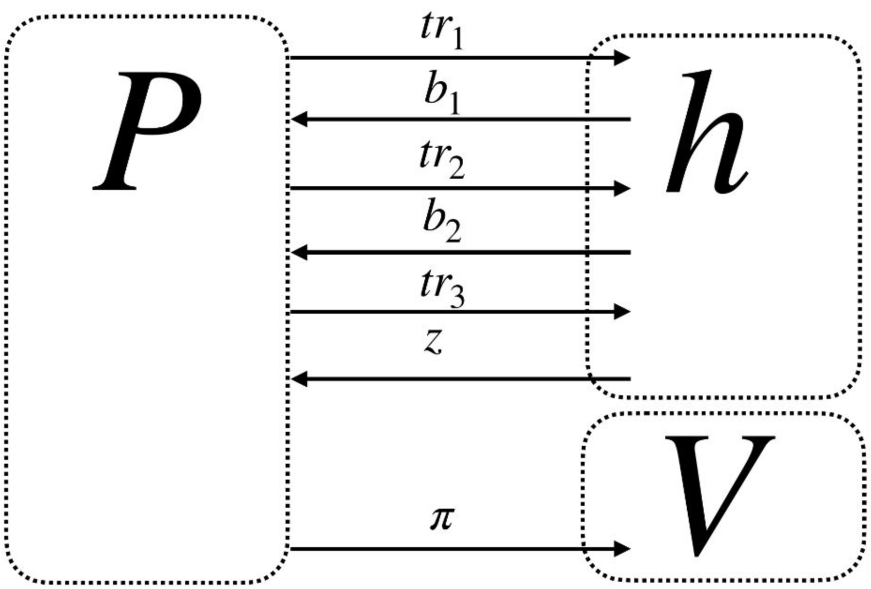
$tr_2 = tr_1, b_1, c_2 \qquad b_2 = R(tr_2)$

$tr_3 = tr_2, c_2, b_2 \qquad z = R(tr_3)$

$$\pi = \begin{pmatrix} c_1, b_1, c_2, b_2, c_3, \\ z, m_1(z), m_2(z), m_3(z) \\ \pi_1, \pi_2, \pi_3 \end{pmatrix}$$

Problem 3 random oracle doesn't exist

From PIOP to SNARKs. Removing random oracle



c_i polynomial commitment to polynomial $m_i(X)$

π_i proof for the polynomial commitment

H hash function

$$tr_1 = x, c_1 \qquad b_1 = h(tr_1)$$

$$tr_2 = tr_1, b_1, c_2 \qquad b_2 = h(tr_2)$$

$$tr_3 = tr_2, c_2, b_2 \qquad z = h(tr_3)$$

$$\pi = \begin{pmatrix} c_1, b_1, c_2, b_2, c_3, \\ z, m_1(z), m_2(z), m_3(z) \\ \pi_1, \pi_2, \pi_3 \end{pmatrix}$$

Problem 4 how secure is such protocol?

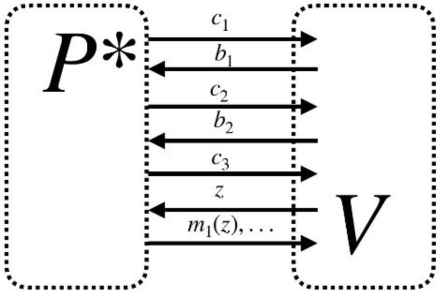
Fiat—Shamir transformation security

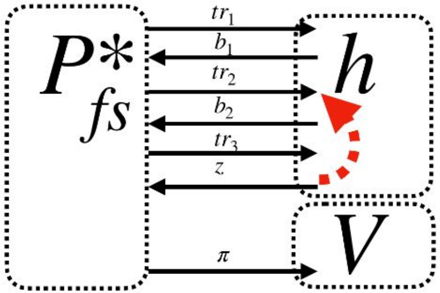
The power of rewinding

Intuition

interactive: the attacker must give only *good answers*

non-interactive: the attacker can cheat by finding a challenge from a lucky set





Lucky set Set of challenges B such that if P^* gets $b \in B$ as a challenge then V accepts and Π enters *lucky state*

V accepts **only if** the protocol is in the *lucky state*

ϵ — probability that P^* makes V accept a false statement

η — the probability that Π gets into the lucky state at i-th round

$(1-\eta)^k$ — the probability that Π doesn't get into the lucky state in one of the k rounds

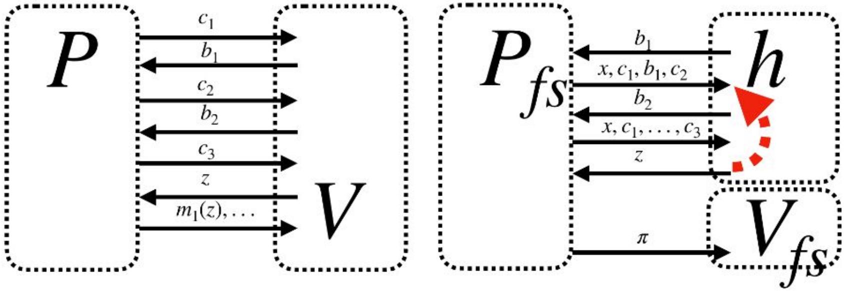
$$\epsilon = 1 - (1-\eta)^k \approx k \cdot \eta$$

How to compare the security?

ϵ - probability that P^* convinces V on a false statement (a.k.a. P^* breaks the soundness)

η - probability that P^* gets a challenge from a lucky set

Q - number of hashes P^*_{fs} can make (a.k.a. P^*_{fs}'s running time or *attack budget*)



$1 - \eta$: probability that P^* **doesn't get a challenge from a lucky set** in a particular round

$(1-\eta)^{Q/k}$: probability that P^* doesn't get a challenge from a lucky set in a particular round **in Q/k trials**

$\prod_{i=1}^{k}(1-\eta)^{Q/k}$: probability that P^* doesn't get a challenge from a lucky set in a particular round in Q/k trials **for each of k rounds**

$$\epsilon_{fs} = 1 - \prod_{i=1}^{k}(1-\eta)^{Q/k} = 1 - (1-\eta)^{Q} \approx \eta \cdot Q$$

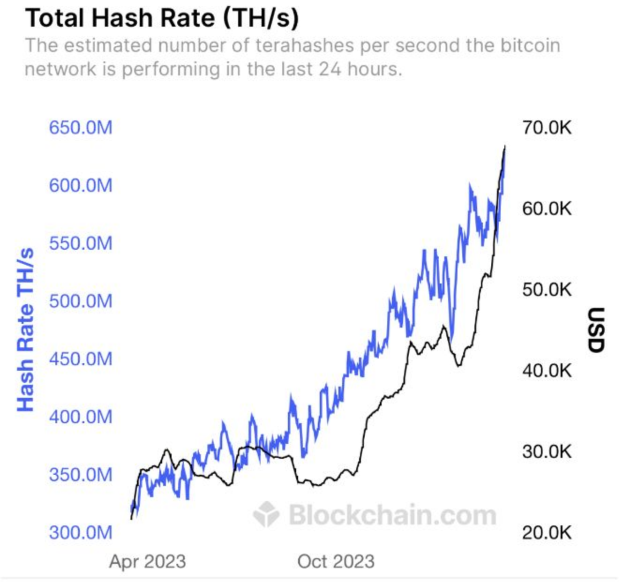
But what's the security?

$$\epsilon \approx 2^{-100}$$

What's Q?

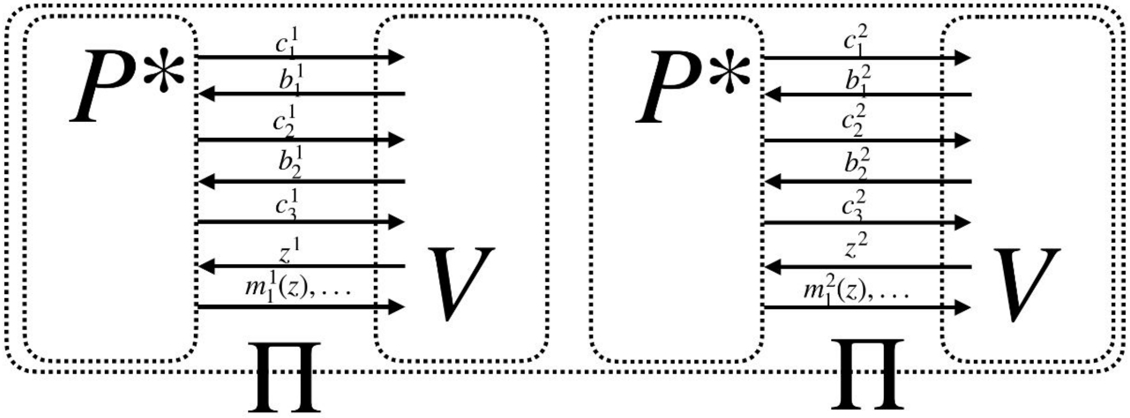
$$600M * 10^{12} \approx 6 * 2^{69} \ [h/s]$$
$$2^{85} \ [h/day]$$

$$\epsilon' \approx 2^{-100} \cdot Q = 2^{-15}$$



Bootstrapping security with parallel repetition

Parallel repetition

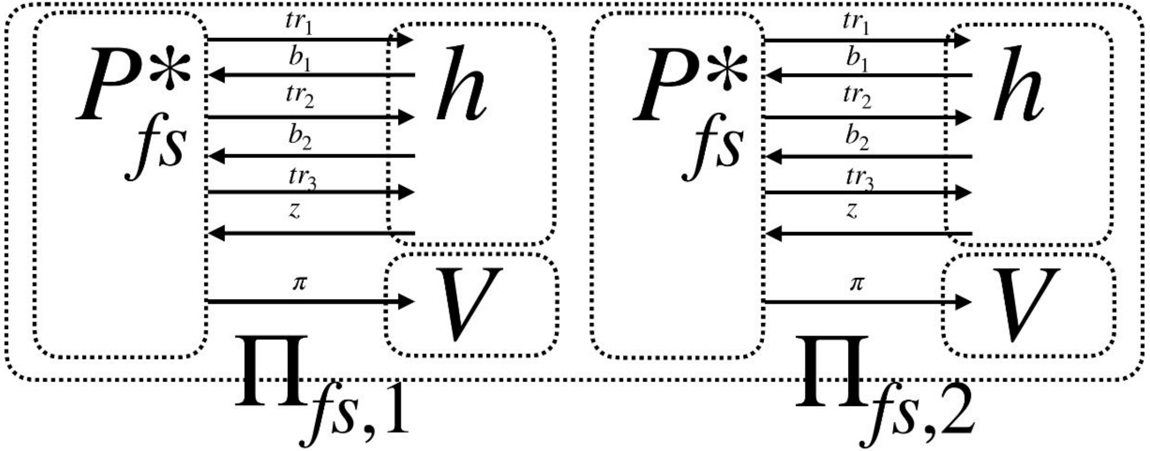


ϵ probability of breaking the security of Π

ϵ_2 - probability of breaking Π^2

$$\epsilon_2 = \epsilon \cdot \epsilon$$

Parallel repetition. FS transformation



transcripts of $\Pi_{fs,1}$, $\Pi_{fs,2}$ depend on each other

ϵ_{fs} probability of breaking the security of Π_{fs}

$\epsilon_{fs,2}$ - probability of breaking Π^2_{fs}

$$\cancel{\epsilon_{fs,2} = \epsilon_{fs} \cdot \epsilon_{fs}}$$

Lucky sets in FS transformation

Lucky set Set of challenges B such that if P^* gets $b \in B$ as a challenge then V accepts

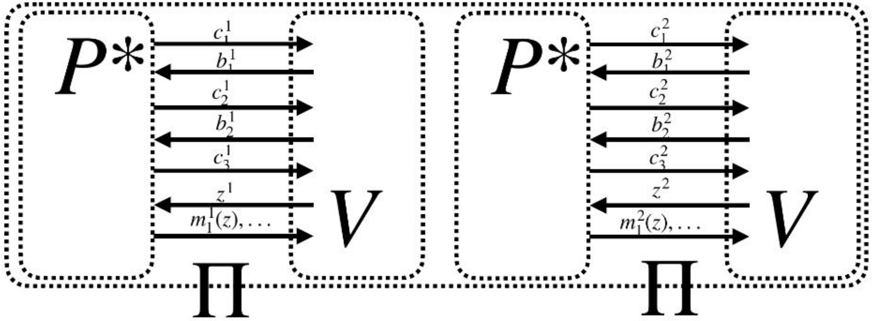
B_i^j — lucky set for i-th challenge in execution j

N — set of all challenges

η — max probability that Π_i enters *lucky state* in B_1^j

k — number of protocol rounds

t — number of executions



P^* needs to enter a *lucky state* in **all** executions to be successful

$$(1-(1-\eta)^k)\cdot \ldots \cdot (1-(1-\eta)^k) \approx (\eta \cdot k)^t$$

Example

$t = 8,\ k = 4,$

$\eta = 2^{-20}$

$$(2^{-20}\cdot 4)^8 = \left(2^{-18}\right)^8 = 2^{-144}$$

Lucky sets in FS transformation

Lucky set Set of challenges B such that if P^* gets $b \in B$ as a challenge then V accepts

B_i^j — lucky set for i-th challenge in execution j

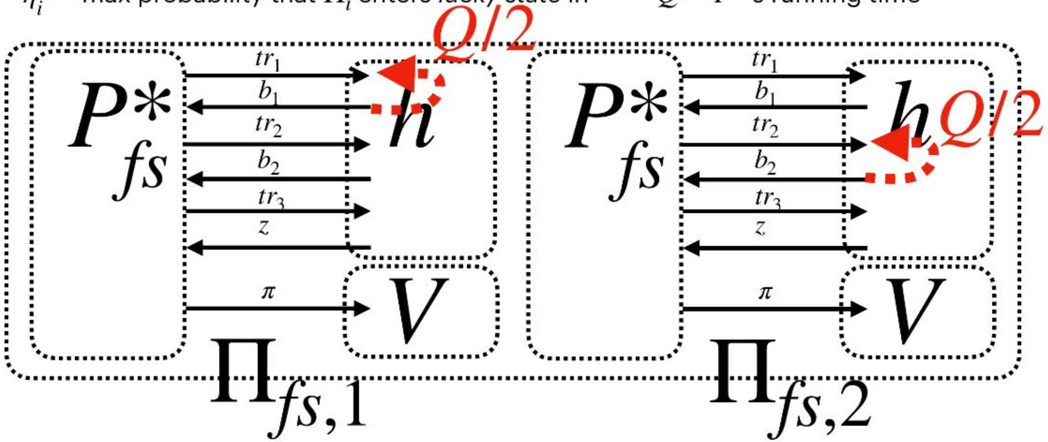
N — set of all challenges

η_i^j — max probability that Π_i enters *lucky state* in

k — number of protocol rounds

t — number of executions

Q — P^*'s running time



P^*'s strategy

(1) Send the first message

(2) Get the challenge b_1^1

(3) If $b_1^1 \notin B_1^1$, rewind to (1), propose a new tr_1^1, tr_1^2, get a new challenge b_1^1

(4) Repeat until $b_1^1 \in B_1^1$ found

(5) Send tr_2^2

(6) Get a challenge b_2^2

(7) If $b_2^2 \notin B_2^2$, rewind to (5), propose new tr_2^2 and get a new challenge b_2^2

(8) Repeat until $b_2^2 \in B_2^2$

Lucky sets in FS transformation

Lucky set Set of challenges B such that if P^* gets $b \in B$ as a challenge then V accepts

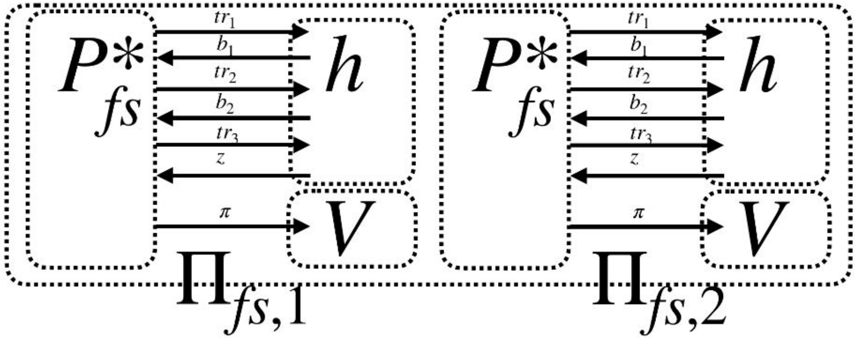
B_i^j — lucky set for i-th challenge in execution j

N — set of all challenges

η_i^j — max probability that Π_i enters *lucky state* in

k — number of protocol rounds

t — number of executions



P^* needs to enter a *lucky state* in **all** executions to be successful

$$(1-(1-\eta)^{Q/t}) \cdot \ldots \cdot (1-(1-\eta)^{Q/t}) \approx \left(Q/t \cdot \eta\right)^t$$

Example

$t = 10,\ k = 4,\ Q = 2^{20}$

$\eta = 2^{-20}$

$$\left(2^{20}/8 \cdot 2^{-20}\right)^8 = (1/8)^{10} = 2^{-30}$$

There is more…

How to do Fiat—Shamir correctly

Coordinated disclosure of vulnerabilitie... Girault, Bullet... PlonK

POST APRIL 13, 2022 LEAVE A COM...

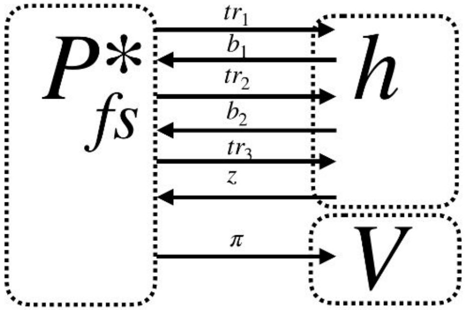
By Jim Miller

Trail of Bits is publicly disclosing critical vulnerabilities that break the soundness of multiple implementations of zero-knowledge proof systems, including PlonK and Bulletproofs. These vulnerabilities are caused by insecure implementations of the Fiat-Shamir transformation that allow malicious users to forge proofs for random statements.

We've dubbed this class of vulnerabilities Frozen Heart. The word *frozen* is an acronym for FoRging Of ZEro kNowledge proofs, and the Fiat-Shamir transformation is at the *heart* of most proof systems: it's vital for their practical use, and it's generally located centrally in protocols. We hope that a catchy moniker will help raise awareness of these issues in the cryptography and wider technology communities.

This is a coordinated disclosure: we have notified those parties who we ...ted the issues prior to our ... were affected:

TL;DR Some implementations of Fiat—Shamir transformation didn't include all the necessary information in the hash function's input.



Hash needs to be computed on **all the data** that the prover sent so far

Some more attacks

TL;DR don't deviate from the protocol description

By Oana Ciobotaru, Maxim Peter and Vesselin Velichkov

OpenZeppelin recently identified a critical vulnerability during an audit of Linea's PLONK verifier. At its core, the issue is similar in nature to some previously disclosed vulnerabilities (e.g., Frozen Heart, 00). The 'Last Challenge' vulnerability arises from the ability of a malicious prover to exploit the degrees of freedom introduced by an incorrect application of the Fiat-Shamir transform when computing the final PLONK challenge. A malicious prover exploiting this vulnerability could steal all the assets in the rollup by submitting a proof for an invalid state transition. While the issue was promptly communicated and fixed, we believe the specifics are worth sharing more broadly so that others may learn to recognize similar patterns and protect themselves accordingly.

Prior to describing the issue in detail below, we give a brief high-level introduction to Zero-Knowledge Succinct Non-interactive ARguments of Knowledge (zkSNARKs) and the Fiat-Shamir transform.

Be careful when you fine-tune your parameters