SNARKs in Relativized Worlds

Thank you for many of the slides!

Megan Chen
Joint work with Alessandro Chiesa, Nicholas Spooner
Eurocrypt 2022 (ePrint: 2022/383)
Our setting: “streaming” verification of $t$-step NP computations
Our setting: “streaming” verification of $t$-step NP computations

**Goal:** Prove correctness of a $t$-step non-deterministic computation:

Given $F, z_0, z_t,$
Our setting: “streaming” verification of $t$-step NP computations

**Goal:** Prove correctness of a $t$-step non-deterministic computation:

Given $F, z_0, z_t$, check that $\exists z_1, \ldots, z_{t-1}, w_0, \ldots, w_{t-1} : \forall i \in [t], F(z_i, w_i) = z_{i+1}$
Our setting: “streaming” verification of $t$-step NP computations

**Goal:** Prove correctness of a $t$-step non-deterministic computation:

Given $F, z_0, z_t$, check that $\exists z_1, \ldots, z_{t-1}, w_0, \ldots, w_{t-1} : \forall i \in [t], F(z_i, w_i) = z_{i+1}$

**Option 1:** Monolithic proof

\[ F^t(z_0; w_0, \ldots, w_{t-1}) = z_t \]
Our setting: “streaming” verification of $t$-step NP computations

**Goal:** Prove correctness of a $t$-step non-deterministic computation:
Given $F, z_0, z_t$, check that $\exists z_1, \ldots, z_{t-1}, w_0, \ldots, w_{t-1} : \forall i \in [t], F(z_i, w_i) = z_{i+1}$

**Option 1:** Monolithic proof

\[ F^t(z_0; w_0, \ldots, w_{t-1}) = z_t \]

**Issues:**
- (Typically) requires prover memory $\Omega(t)$
- Proving $t + 1$ steps requires recomputing entire proof
Our setting: “streaming” verification of \( t \)-step NP computations

**Goal:** Prove correctness of a \( t \)-step non-deterministic computation:

Given \( F, z_0, z_t \), check that \( \exists z_1, \ldots, z_{t-1}, w_0, \ldots, w_{t-1} : \forall i \in [t], F(z_i, w_i) = z_{i+1} \)

**Option 1:** Monolithic proof

\[
F^t(z_0; w_0, \ldots, w_{t-1}) = z_t
\]

**Issues:**
- (Typically) requires prover memory \( \Omega(t) \)
- Proving \( t + 1 \) steps requires recomputing entire proof

**Option 2:** Incrementally verifiable computation (IVC) [Valiant08]
Our setting: “streaming” verification of $t$-step NP computations

**Goal:** Prove correctness of a $t$-step non-deterministic computation:

Given $F, z_0, z_t$, check that $\exists z_1, \ldots, z_{t-1}, w_0, \ldots, w_{t-1} : \forall i \in [t], F(z_i, w_i) = z_{i+1}$

**Option 1:** Monolithic proof

\[ F^t(z_0; w_0, \ldots, w_{t-1}) = z_t \]

**Issues:**

- (Typically) requires prover memory $\Omega(t)$
- Proving $t + 1$ steps requires recomputing entire proof

**Option 2:** Incrementally verifiable computation (IVC) [Valiant08]

**Proof-carrying data (PCD)** [CT10, BCCT13]: generalizes path graph to DAG
Our setting: “streaming” verification of $t$-step NP computations

Applications include:

- “Succinct” blockchains
- SNARKs with low space complexity
- Verifiable delay functions
- Byzantine agreement
- ZK cluster computing
- Verifiable image editing
- Enforcing language semantics across trust boundaries

Proof-carrying data (PCD) [CT10, BCCT13]: generalizes path graph to DAG
Defining incrementally verifiable computation (IVC)

\[
\text{Adversarial) Completeness: If } \% \& = \odot, \% = 1, \text{ then } \% \& = \odot, \% = 1.
\]

Proof of knowledge: For \( \text{*} \rightarrow (\&, \%) \text{ s.t. } \% \& = \odot, \% = 1, \text{*} \text{ must "know" (can be made to produce) a complete transcript of the computation so far.}
\]

Efficiency: \( |P| = |z| \)

Slide from Nicholas Spooner
Defining incrementally verifiable computation (IVC)

Defining IVC

(Adversarial) Completeness:

\[ \text{If } \alpha \neq 1, \text{ then } \alpha' = 1. \]

Proof of knowledge:

For * → (\& : ) s.t. \( \alpha \neq 1 \), * must "know" (can be made to produce) a complete transcript of the computation so far.

Efficiency:

\[ | \text{ } | = | \text{ } | \]
Defining incrementally verifiable computation (IVC)
Defining incrementally verifiable computation (IVC)

\[ \text{(Adversarial) Completeness:} \]
\[ \text{If } \pi' = 1, \text{ then } \pi = 1. \]

\[ \text{Proof of knowledge:} \]
\[ \text{For } * \rightarrow (\pi', \pi) \text{ s.t. } \pi' = 1, * \text{ must "know" (can be made to produce) a complete transcript of the computation so far.} \]

\[ \text{Efficiency:} \]
\[ |\pi'| = |\pi|. \]
Defining incrementally verifiable computation (IVC)

How to instantiate IVC?
IVC from SNARK

The IVC Prover...

$\exists w_i \forall z_i, w_i \in F(z_i, w_i) = z_i + 1$

IVC. $\mathcal{P}_F$

SNARK = Succinct, Non-interactive Argument of Knowledge
IVC from SNARK

The IVC Prover...

Runs a SNARK $P$, proving the computation $R$: $\exists w$ s.t. $F(z_i, w_i) = z_i + 1$ $V(z_i, \pi_i) = 1$
The IVC Prover...

Runs a SNARK $P$, proving the computation $R$:

For all $i$, $\exists w_i$ s.t.
IVC from SNARK

The IVC Prover...

Runs a SNARK $P$, proving the computation $R$:

For all $i$, $\exists w_i$ s.t.

- $F(z_i, w_i) = z_{i+1}$ and
IVC from SNARK

The IVC Prover...

Runs a SNARK $P$, proving the computation $R$:

For all $i$, $\exists w_i$ s.t.

- $F(z_i, w_i) = z_{i+1}$ and
- SNARK $V(z_i, \pi_i) = 1$
Prior works: IVC instantiations from SNARKs

Approach 1: CRS + knowledge (extraction) assumptions

[Groth10; GennaroGPR13; BitanskyCILOP13; Ben-SassonCTV14; BitanskyCCGLRT14; Groth16; GrothKMMM18]

SNARK.\(P\)

IVC.\(\mathcal{P}_F\)

\(w_i\)

\(z_i\)

\(\pi_i\)

\(z_{i+1}\)

\(\pi_{i+1}\)
Prior works: IVC instantiations from SNARKs

**Approach 1:** CRS + knowledge (extraction) assumptions

[Groth10; GennaroGPR13; BitanskyCIOP13; Ben-SassonCTV14; BitanskyCCGLRT14; Groth16; GrothKMMM18]

**Approach 2:** SNARKs in ROM

[Micali00; Ben-SassonCS16; ChiesaOS20; ChiesaHMMVW20]
Prior works: IVC instantiations from SNARKs

**Approach 1:** CRS + knowledge (extraction) assumptions

[Groth10; GennaroGPR13; BitanskyCIO13; Ben-SassonCTV14; BitanskyCCGLRT14; Groth16; GrothKMMM18]

**Approach 2:** SNARKs in ROM

[Micali00; Ben-SassonCS16; ChiesaOS20; ChiesaHMMVW20]

**Benefits:**
- Transparent / universal setup
- Efficiency improvements
Prior works: IVC instantiations from SNARKs

Approach 1: CRS + knowledge (extraction)
assumptions

[Groth10; GennaroGPR13; BitanskyCiOP13; Ben-SassonCTV14; BitanskyCCGLRT14; Groth16; GrothKMMM18]

Approach 2: SNARKs in ROM

[Micali00; Ben-SassonCS16; ChiesaOS20; ChiesaHMMVW20]

Benefits:
• Transparent / universal setup
• Efficiency improvements

Issues:
• SNARK verifier makes oracle queries, but SNARK is for non-oracle computations.
Prior works: IVC instantiations from SNARKs

**Approach 1:** CRS + knowledge (extraction) assumptions

[Groth10; GennaroGPR13; BitanskyCIOP13; Ben-SassonCTV14; BitanskyCCGLRT14; Groth16; GrothKMMM18]

**Approach 2:** SNARKs in ROM

[Micali00; Ben-SassonCS16; ChiesaOS20; ChiesaHMMVW20]

**Benefits:**
- Transparent / universal setup
- Efficiency improvements

**Issues:**
- SNARK verifier makes oracle queries, but SNARK is for non-oracle computations.
  - [ChiesaOS20; …] Heuristically instantiate \( \rho \)
Issues with heuristic RO instantiation

Theoretical:

• Requires non-blackbox use of oracle; this breaks the RO abstraction.

• Security flaws may be in the heuristic step [GoldwasserK03; CanettiGH04].

Practical:

• Flexibility: Oracle must be instantiated as a circuit: can’t use MPC, hardware token.

• Efficiency: SNARKs about SHA2, BLAKE are expensive!
Issues with heuristic RO instantiation

Theoretical:
• Requires non-blackbox use of oracle; this breaks the RO abstraction.

Practical:
• No flexibility: Oracle must be instantiated as a circuit: can't use MPC, hardware token.
• Inefficient: SNARKs about SHA2, BLAKE are expensive!
Issues with heuristic RO instantiation

**Theoretical:**
- Requires non-blackbox use of oracle; this breaks the RO abstraction.
- Security flaws may be in the heuristic step [GoldwasserK03; CanettiGH04].

**Practical:**
- Flexibility: Oracle must be instantiated as a circuit: can't use MPC, hardware token.
- Efficiency: SNARKs about SHA2, BLAKE are expensive!
Issues with heuristic RO instantiation

**Theoretical:**
- Requires non-blackbox use of oracle; this breaks the RO abstraction.
- Security flaws may be in the heuristic step [GoldwasserK03; CanettiGH04].

**Practical:**

\[ w_i \rightarrow R \rightarrow F \rightarrow z_{i+1} \]
\[ z_i \rightarrow V \rightarrow F \rightarrow z_{i+1} \]
\[ \pi_i \rightarrow V \rightarrow F \rightarrow \pi_{i+1} \]
Issues with heuristic RO instantiation

Theoretical:
- Requires non-blackbox use of oracle; this breaks the RO abstraction.
- Security flaws may be in the heuristic step [GoldwasserK03; CanettiGH04].

Practical:
- **No flexibility**: Oracle must be instantiated as a circuit: can’t use MPC, hardware token.

![Diagram of RO instantiation](image)
Issues with heuristic RO instantiation

**Theoretical:**
- Requires non-blackbox use of oracle; this breaks the RO abstraction.
- Security flaws may be in the heuristic step [GoldwasserK03; CanettiGH04].

**Practical:**
- **No flexibility:** Oracle must be instantiated as a circuit: can’t use MPC, hardware token.
- **Inefficient:** SNARKs about SHA2, BLAKE are expensive!
Research question

Is there an oracle model $O$ such that

1. there are SNARKs in the $O$ model; and
2. the SNARK can prove statements about $O$?
Research question

Is there an oracle model \( O \) such that

1. there are SNARKs in the \( O \) model; and
2. the SNARK can prove statements about \( O \)?

Having \( O \) means we can build IVC.
Research question

Is there an oracle model $O$ such that

1. there are SNARKs in the $O$ model; and
2. the SNARK can prove statements about $O$?

Impossible when $O$ is the random oracle!
Our results

**Define:** low-degree random oracle (LDRO)
Our results

Define: low-degree random oracle (LDRO)

Correctness of $\mathsf{NP}^\hat{\mathsf{P}}$ computation

SNARK in LDROM for LDROM computations

Define: low-degree random oracle (LDRO)
Our results

Define: low-degree random oracle (LDRO)

SNARK in LDROM for LDROM computations

Correctness of \[ \text{NP}^{\hat{\rho}} \] computation

\[ = \]

Correctness of \[ \text{NP} \] computation

\[ + \]

Succinct verification of \( M \)'s \( \hat{\rho} \) queries
Our results

Define: low-degree random oracle (LDRO)

Correctness of $\mathsf{NP}^\mathcal{O}$ computation

$=$

Correctness of $\mathsf{NP}$ computation

$+$

Succinct verification of $M$'s $\mathcal{O}$ queries

SNARK in LDROM for LDROM computations

$=$

SNARK in LDROM for non-oracle computations

$+$

Define: low-degree random oracle (LDRO)
Our results

Define: low-degree random oracle (LDRO)

Correctness of \( \text{NP}^\hat{\rho} \) computation = Correctness of NP computation + Succinct verification of \( M \)'s \( \hat{\rho} \) queries

SNARK in LDROM for LDROM computations = SNARK in LDROM for non-oracle computations + NI query reduction for LDROM queries

Our results: Define a new oracle model (LDRO) which is useful for constructing SNARKs with succinct verification for non-oracle computations. The correctness of computations in LDROM is reduced to the combination of the correctness of computations in NP, verification of \( M \)'s queries, and an NI query reduction for LDROM queries. This approach uses ideas from Kalai and Raz (2008).
Our results

Define: low-degree random oracle (LDRO)

Correctness of $\text{NP}^\hat{\rho}$ computation

SNARK in LDROM for LDROM computations

= =

Correctness of NP computation

SNARK in LDROM for non-oracle computations

+ +

Succinct verification of $M$’s $\hat{\rho}$ queries

NI query reduction for LDROM queries

Uses ideas from [KalaiRaz08].

Define:

Our results

Correctness of $\text{NP}^\hat{\rho}$ computation

SNARK in LDROM for LDROM computations

= =

Correctness of NP computation

SNARK in LDROM for non-oracle computations

+ +

Succinct verification of $M$’s $\hat{\rho}$ queries

NI query reduction for LDROM queries

Uses ideas from [KalaiRaz08].
Random oracle

$\mathcal{R}O : \{0,1\}^m \rightarrow \mathbb{F}$
Random oracle

\[ \text{RO}(0, 0, 1) = y \in \mathbb{F} \]

\[ \{0,1\}^3 \]

\[ \text{RO} : \{0,1\}^m \rightarrow \mathbb{F} \]
Low-degree random oracle ($\hat{\rho}$)

$\mathcal{RO}(0, 0, 1) = y \in \mathbb{F}$

$\mathcal{RO} : \{0, 1\}^m \rightarrow \mathbb{F}$
Low-degree random oracle ($\hat{\rho}$)

$m$-variate polynomials over $\mathbb{F}$, individual degree $\leq d$, evaluated over $\mathbb{F}^m$

Random $\hat{\rho} \in \mathbb{F}^{\leq d}[X_1, \ldots, X_m]$ s.t.:

$\text{RO}(0, 0, 1) = y \in \mathbb{F}$

$\text{RO} : \{0, 1\}^m \rightarrow \mathbb{F}$
Low-degree random oracle \((\hat{\rho})\)

\[\text{Random } \hat{\rho} \in \mathbb{F}^{\leq d}[X_1, \ldots, X_m] \text{ s.t.:}\]

- Points in Boolean hypercube agrees with random oracle

\[\mathbb{F}^{\leq d}[X_1, \ldots, X_m] = \text{m-variate polynomials over } \mathbb{F}, \text{ individual degree } \leq d, \text{ evaluated over } \mathbb{F}^m\]
Low-degree random oracle ($\hat{\rho}$)

Random $\hat{\rho} \in \mathbb{F}^{\leq d}[X_1, \ldots, X_m]$ s.t.:

- Points in Boolean hypercube agrees with random oracle
- Is low degree (e.g. $d = O(1)$).

$m$-variate polynomials over $\mathbb{F}$, individual degree $\leq d$, evaluated over $\mathbb{F}^m$. 

$RO(0, 0, 1) = y \in \mathbb{F}$
Low-degree random oracle ($\hat{\rho}$)

- Points in Boolean hypercube agrees with random oracle
- Is low degree (e.g. $d = O(1)$)
- Can query ANY point in $\mathbb{F}_3^m$

$m$-variate polynomials over $\mathbb{F}$, individual degree $\leq d$, evaluated over $\mathbb{F}^m$

Random $\hat{\rho} \in \mathbb{F}^{\leq d}[X_1, \ldots, X_m]$ s.t.:

- Points in Boolean hypercube agrees with random oracle
- Is low degree (e.g. $d = O(1)$)
- Can query ANY point in $\mathbb{F}^m$
Low-degree random oracle ($\hat{\rho}$)

Random $\hat{\rho} \in \mathbb{F}^{\leq d}[X_1, \ldots, X_m]$ s.t.:

- Points in Boolean hypercube agrees with random oracle
- Is low degree (e.g. $d = O(1)$).
- Can query ANY point in $\mathbb{F}^m$

Is $\hat{\rho}$ simulatable (can do lazily sampling) and programmable?
Lemma: There is perfect, stateful simulation of LDROs.

Lazy sampling of LDRO
Lazy sampling of LDRO

**Lemma:** There is perfect, stateful simulation of LDROs.

**Lazy sampling strategy:**

Given a query $x$, check if $y = \hat{\rho}(x)$ is already determined.
Lazy sampling of LDRO

**Lemma:** There is perfect, stateful simulation of LDROs.

**Lazy sampling strategy:**

Given a query $x$, check if $y = \hat{\rho}(x)$ is already determined.

**Succinct constraint detection** algorithm exists for low-degree polynomials [Ben-SassonCFGRS17]
Lazy sampling of LDRO

**Lemma:** There is perfect, stateful simulation of LDROs.

**Lazy sampling strategy:**

Given a query $x$, check if $y = \hat{\rho}(x)$ is already determined.

- If yes, use determined $y$.

Succinct constraint detection algorithm exists for low-degree polynomials
[Ben-SassonCFGRS17]
Lazy sampling of LDRO

Lemma: There is perfect, stateful simulation of LDROs.

Lazy sampling strategy:

Given a query \( x \), check if \( y = \hat{\rho}(x) \) is already determined.

- If yes, use determined \( y \).
- If no, sample \( y \leftarrow_R \mathbb{F} \).

Succinct constraint detection algorithm exists for low-degree polynomials [Ben-SassonCFGRS17]
How to (heuristically) instantiate the LDRO?
How to (heuristically) instantiate the LDRO?

1. Since LDRO has stateful simulation, use trusted party or MPC protocol.
How to (heuristically) instantiate the LDRO?

1. Since LDRO has stateful simulation, use trusted party or MPC protocol.

2. Obfuscate (or embed in hardware token)

\[ P(x_1, \ldots, x_m) = \sum_{\overrightarrow{a} \in [d]^m} F(\overrightarrow{a}) \cdot x_1^{a_1} \cdots x_m^{a_m} \]

where \( F \) is the structured PRF by [BenabbasGennaroVahlis11].
How to (heuristically) instantiate the LDRO?

1. Since LDRO has stateful simulation, use trusted party or MPC protocol.

2. Obfuscate (or embed in hardware token)

\[ P(x_1, \ldots, x_m) = \sum_{\overline{a} \in [d]^m} F(\overline{a}) \cdot x_1^{a_1} \cdots x_m^{a_m} \]

where \( F \) is the structured PRF by [BenabbasGennaroVahlis11].

3. **Future work:** arithmetize an existing “strong” hash function?
How do we efficiently verify LDRO queries?

Step 1: Recall [KalaiRaz08]'s interactive query reduction protocol

Step 2: Make [KalaiRaz08] SNARK-friendly
How do we efficiently verify LDRO queries?

**Step 1:** Recall [KalaiRaz08]’s *interactive* query reduction protocol
How do we efficiently verify LDRO queries?

**Step 1:** Recall [KalaiRaz08]’s *interactive* query reduction protocol

**Step 2:** Make [KalaiRaz08] SNARK-friendly
What is query reduction?

**Goal:** verify polynomial queries

\[
\{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F}
\]
What is query reduction?

**Goal:** verify polynomial queries

\[ \{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F} \]

**Idea:** Verifier has help from a prover
- [KalaiRaz08] gives an IP for this task
- Only requires 1 query to \( \hat{\rho} \)
Interactive query reduction protocol

Input: \( \{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F} \)

Goal: Check \( \hat{\rho}(x_i) = y_i \ \forall \ i \in [n] \) with 1 verifier query
[KalaiRaz08] Interactive query reduction protocol

Input: \( \{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F} \)

Goal: Check \( \hat{\rho}(x_i) = y_i \ \forall \ i \in [n] \) with 1 verifier query

Compute a curve \( g \) s.t. \( g(b_i) = x_i \ \forall \ i \in [n] \).

Prover

\[ g \]

Verifier

\[ g \]

\( b_1, \ldots, b_n \in \mathbb{F} \)
[KalaiRaz08] Interactive query reduction protocol

Input: \( \{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F} \)

Goal: Check \( \hat{\rho}(x_i) = y_i \ \forall \ i \in [n] \) with 1 verifier query

Compute a curve \( g \) s.t. \( g(b_i) = x_i \ \forall \ i \in [n] \).

Prover

\[
g
f := \hat{\rho} \circ g
\]

Verifier

\[
f
\]

Fact:

\[
\text{deg}(f) = nmd
\]
[KalaiRaz08] Interactive query reduction protocol

Input: \( \{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F} \)

Goal: Check \( \hat{\rho}(x_i) = y_i \ \forall i \in [n] \) with 1 verifier query

Compute a curve \( g \) s.t. \( g(b_i) = x_i \ \forall i \in [n] \).

Fact: \( \deg(f) = nmd \)
[KalaiRaz08] Interactive query reduction protocol

**Input:** \( \{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F} \)

**Goal:** Check \( \hat{\rho}(x_i) = y_i \ \forall \ i \in [n] \) with 1 verifier query

Compute a curve \( g \) s.t. \( g(b_i) = x_i \ \forall \ i \in [n] \).

**Prover**

\[
\begin{align*}
g \\
f := \hat{\rho} \circ g
\end{align*}
\]

**Verifier**

\[
\begin{align*}
f \\
f(b_i) = y_i \ \forall \ i \in [n] \quad \text{Check } f
\end{align*}
\]

Fact:

\[
\deg(f) = nmd
\]
[KalaiRaz08] Interactive query reduction protocol

Input: \( \{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{F}^m \times \mathbb{F} \)

Goal: Check \( \hat{\rho}(x_i) = y_i \ \forall i \in [n] \) with 1 verifier query

Compute a curve \( g \) s.t. \( g(b_i) = x_i \ \forall i \in [n] \).

Prover

\[
g
f := \hat{\rho} \circ g
\]

Verifier

\[
g
\beta \leftarrow \mathbb{F}
\]

Fact:

\[
\deg(f) = nmd
\]

\[
f(b_i) = y_i \ \forall i \in [n]
\]

\[
\hat{\rho}(g(\beta)) = f(\beta)
\]

\[
\hat{\rho}(g(\beta)) \vdash f(\beta)
\]
[KalaiRaz08] Interactive query reduction protocol

Input: \{ (x_1, y_1), \ldots, (x_n, y_n) \} \in \mathbb{F}^m \times \mathbb{F}

Goal: Check \( \hat{\rho}(x_i) = y_i \ \forall i \in [n] \) with 1 verifier query

- Soundness: \( \frac{nmd}{|\mathbb{F}|} \)
- Communication: \( O(nmd) \)

Prover

\[
g \\
f := \hat{\rho} \circ g
\]

Verifier

\[
g \\
\beta \leftarrow \mathbb{F} \\
f(\beta) \overset{?}{=} y_i \ \forall i \in [n] \\
\hat{\rho}(g(\beta)) \overset{?}{=} f(\beta)
\]

Fact:
\[
\text{deg}(f) = nmd
\]
Problem: Verifier is randomized (samples $\beta$)
Fix: “Fiat-Shamir” transform

SNARK-friendly [KalaiRaz08]: de-randomize $V$

Compute a curve $g$ s.t. $g(b_i) = x_i \forall i \in [n]$. 

Verifier:

- $g$
- $f, \beta$ 
- $f(b_i) \overset{?}{=} y_i \forall i \in [n]$
- $\hat{\rho}(g(\beta)) \overset{?}{=} f(\beta)$

Prover:

- $g$
- $f := \hat{\rho} \circ g$
- $\beta = \hat{\rho}(g, f)$
- $b_1, \ldots, b_n \in \mathbb{F}$
SNARK-friendly [KalaiRaz08]: de-randomize V

Problem: Verifier is randomized (samples $\beta$)
Fix: “Fiat-Shamir” transform

Compute a curve $g$ s.t. $g(b_i) = x_i \forall i \in [n]$. 

\begin{align*}
b_1, \ldots, b_n &\in \mathbb{F} \\
g &:= \hat{\rho} \circ g \\
\beta &= \hat{\rho}(g, f) \\
f &= \hat{\rho}(g, f) \\
\hat{\rho}(g(\beta)) &= f(\beta) \\
\hat{\rho}(g, f) &= \beta
\end{align*}
SNARK-friendly [KalaiRaz08]: succinct oracle queries

**Problem:** $|f|$ linear in the number of queries.

**Fix:** Use a hash function / compressing commitment

Compute a curve $g$ s.t. $g(b_i) = x_i \forall i \in [n]$.

\[ g \text{ is specified by } x_1, \ldots, x_n \]

---

**Prover**

\[
\begin{align*}
g & \in \mathbb{F} \\
 f & := \hat{\rho} \circ g \\
 \beta & = \hat{\rho}(g,f) \\
\end{align*}
\]

**Verifier**

\[
\begin{align*}
g & \\
 f, \beta & \leftarrow \hat{\rho}(g,f) \\
 f(b_i) & \overset{?}{=} y_i \forall i \in [n] \\
 \hat{\rho}(g(\beta)) & \overset{?}{=} f(\beta) \\
 \hat{\rho}(g,f) & \overset{?}{=} \beta \\
\end{align*}
\]
**SNARK-friendly [KalaiRaz08]: succinct oracle queries**

**Problem:** $|f|$ linear in the number of queries.

**Fix:** Use a hash function / compressing commitment

Compute a curve $g$ s.t. $g(b_i) = x_i \forall i \in [n]$.

---

**Prover**

\[
g, f := \hat{\rho} \circ g, \\
\beta = \hat{\rho}(g, f)
\]

\[b_1, \ldots, b_n \in \mathbb{F}\]

**Verifier**

\[
g, f, \beta \overset{?}{=} f(b_i) = y_i \forall i \in [n] \\
\hat{\rho}(g(\beta)) \overset{?}{=} f(\beta) \\
\hat{\rho}(g, f) \overset{?}{=} \beta
\]

$g$ is specified by $x_1, \ldots, x_n$
SNARK-friendly [KalaiRaz08]: succinct oracle queries

**Problem:** $|f|$ linear in the number of queries.

**Fix:** Use a hash function / compressing commitment

Compute a curve $g$ s.t. $g(b_i) = x_i \forall i \in [n]$.

- **Prover**
  - $g$
  - $f := \hat{\rho} \circ g$
  - $h = \text{Hash}((g, f))$
  - $\beta = \hat{\rho}(h)$

- **Verifier**
  - $g$
  - $f(b_i) \overset{?}{=} y_i \forall i \in [n]$
  - $\hat{\rho}(g(\beta)) \overset{?}{=} f(\beta)$
  - $\hat{\rho}(g, f) \overset{?}{=} \beta$

$g$ is specified by $x_1, \ldots, x_n$
SNARK-friendly [KalaiRaz08]: succinct oracle queries

Problem: $|f|$ linear in the number of queries.

Fix: Use a hash function / compressing commitment

Compute a curve $g$ s.t. $g(b_i) = x_i \forall i \in [n]$.

Prover

\[
g
\]
\[
f := \hat{\rho} \circ g
\]
\[
h = \text{Hash} ((g,f))
\]
\[
\beta = \hat{\rho}(h)
\]

Verifier

\[
g
\]
\[
h = \text{Hash} ((g,f))
\]
\[
f(b_i) \overset{?}{=} y_i \forall i \in [n]
\]
\[
\hat{\rho}(g(\beta)) \overset{?}{=} f(\beta)
\]
\[
\hat{\rho}(g,f) \overset{?}{=} \beta
\]

$g$ is specified by $x_1, \ldots, x_n$
SNARK-friendly [KalaiRaz08]: succinct oracle queries

**Problem:** $|f|$ linear in the number of queries.

**Fix:** Use a hash function / compressing commitment

Compute a curve $g$ s.t. $g(b_i) = x_i \forall i \in [n]$. 

---

**Prover**

$g$

$f := \hat{\rho} \circ g$

$h = \text{Hash}((g, f))$

$\beta = \hat{\rho}(h)$

---

**Verifier**

$g$

$h = \text{Hash}((g, f))$

$f(b_i) \overset{?}{=} y_i \forall i \in [n]$

$\hat{\rho}(g(\beta)) \overset{?}{=} f(\beta)$

$\hat{\rho}(h) \overset{?}{=} \beta$

$g$ is specified by $x_1, \ldots, x_n$
Review

We created a scheme that checks LDROM queries efficiently and non-interactively.

**Soundness for NI query reduction?**

- **Bad event:**
  
  Adversary outputs \( f \not\equiv \hat{\rho} \circ g \)
  
  s.t. \( \beta = \hat{\rho}(\text{Hash}(f, g)) \) and \( f(\beta) = (\hat{\rho} \circ g)(\beta) \).

- **Proof:** uses a new LDROM forking lemma
Other results

**Define:** low-degree random oracle (LDRO)

Correctness of $\text{NP}^{\hat{\rho}}$ computation

= \quad \text{Correctness of NP computation} + \text{Succinct verification of } M\text{'s } \hat{\rho} \text{ queries}

SNARK in LDROM for LDROM computations

= \quad \text{SNARK in LDROM for non-oracle computations} + \text{NI query reduction for LDROM queries}

Uses ideas from [KalaiRaz08].
Define: low-degree random oracle (LDRO)

Other results

Correctness of $\text{NP}^\hat{\rho}$ computation

SNARK in
LDROM for LDROM computations

SNARK in
LDROM for non-oracle computations

[Sicili00] SNARK using LDROM.
Soundness: proved with LDRO forking lemma

NI query reduction for LDROM queries

Succinct verification of $M$'s $\hat{\rho}$ queries

Uses ideas from [KalaiRaz08].

[Micali00] SNARK using LDROM.

Uses ideas from [KalaiRaz08].
Define: low-degree random oracle (LDRO)

Other results

- Correctness of \( NP^{\hat{\rho}} \) computation
  - SNARK in LDROM for LDROM computations
  - ZK

- Correctness of \( NP \) computation
  - SNARK in LDROM for non-oracle computations
  - ZK

- Succinct verification of \( M^{\text{\hat{\rho}}} \) queries
  - NI query reduction for LDROM queries
  - ZK

[Micali00] SNARK using LDROM.

Soundness: proved with LDRO forking lemma

Uses ideas from [KalaiRaz08].
**Define:** low-degree random oracle (LDRO)

- Correctness of \( \mathbb{NP}^\hat{\rho} \) computation
- Correctness of \( \mathbb{NP} \) computation
- Succinct verification of \( M \)'s \( \hat{\rho} \) queries

Other results

- SNARK in LDROM for LDROM computations

- SNARK in LDROM for non-oracle computations

- NI query reduction for LDROM queries

[Micali00] SNARK using LDROM.

Soundness: proved with LDRO forking lemma

Uses ideas from [KalaiRaz08].
Thanks!

https://ia.cr/2022/383