Transparent Time- and Space-Efficient Arguments From Groups of Unknown Order

Justin Holmgren
NTT Research

Alex Block
Purdue
Alon Rosen
IDC Herzliya
Ron Rothblum
Technion
Pratik Soni
CMU
Pre-Quantum Cryptography with Lattices

Justin Holmgren
NTT Research

Alex Block
Purdue

Alon Rosen
IDC Herzliya

Ron Rothblum
Technion

Pratik Soni
CMU
Pre-Quantum Cryptography with Lattices

Justin Holmgren
NTT Research

Alex Block
Purdue

Alon Rosen
IDC Herzliya

Ron Rothblum
Technion

Pratik Soni
CMU
Interactive Arguments

for an NP relation $R$ with corresponding language $L$
Interactive Arguments

for an NP relation $R$ with corresponding language $L$

Completeness:
For any $(x, w) \in R$,
Interactive Arguments

for an NP relation $R$ with corresponding language $L$

**Completeness:**
For any $(x, w) \in R$,

$$P(x; w) \quad \ldots \quad V(x)$$
Interactive Arguments
for an NP relation $R$ with corresponding language $L$

**Completeness:**
For any $(x, w) \in R$, polynomial time
Interactive Arguments
for an NP relation $R$ with corresponding language $L$

Completeness:
For any $(x, w) \in R$,

$P(x; w)$ \hspace{2cm} $\cdots$ \hspace{2cm} $V(x)$

polynomial time \hspace{2cm} nearly linear time
Interactive Arguments
for an NP relation $R$ with corresponding language $L$

**Completeness:**
For any $(x, w) \in R$,

**Soundness:**
For any $x \not\in L$, poly-size adversary $A$,
Interactive Arguments

for an NP relation $R$ with corresponding language $L$

Completeness:
For any $(x, w) \in R$,

- $P(x; w)$ in polynomial time
- $V(x)$ in nearly linear time

Soundness:
For any $x \notin L$, poly-size adversary $A$,

- $A$ in polynomial time
- $V(x)$ in nearly linear time
Two Desirable Properties for Interactive Arguments
Two Desirable Properties for Interactive Arguments

Public-Coin Verification:
Two Desirable Properties for Interactive Arguments

Public-Coin Verification:

- Uniformly random verifier messages
Two Desirable Properties for Interactive Arguments

Public-Coin Verification:

- Uniformly random verifier messages
- Acceptance depends deterministically on transcript
Two Desirable Properties for Interactive Arguments

Public-Coin Verification:

• Uniformly random verifier messages

• Acceptance depends deterministically on transcript

Necessary for decentralized verification (e.g. in blockchains)
Two Desirable Properties for Interactive Arguments

Public-Coin Verification:

- Uniformly random verifier messages
- Acceptance depends deterministically on transcript

Time- and Space-Efficient Prover:

[Image: Necessary for decentralized verification (e.g. in blockchains)]
Two Desirable Properties for Interactive Arguments

Public-Coin Verification:

- Uniformly random verifier messages
- Acceptance depends deterministically on transcript

Time- and Space-Efficient Prover:

- If $(x; w) \in R$ is decidable in time $T$ and space $S$, then prover runs in time $\approx T$ and space $\approx S$
Two Desirable Properties for Interactive Arguments

Public-Coin Verification:

- Uniformly random verifier messages
- Acceptance depends deterministically on transcript

Time- and Space-Efficient Prover:

- If \((x; w) \in R\) is decidable in time \(T\) and space \(S\), then prover runs in time \(\approx T\) and space \(\approx S\)
- Space can be as much of a bottleneck as time, but is often overlooked

Necessary for decentralized verification (e.g. in blockchains)
Prior Approaches for Time- and Space-Efficient Proving
Prior Approaches for Time- and Space-Efficient Proving

Approach 1: Recursive Composition [Valiant '08, BCCT '12]
Prior Approaches for Time- and Space-Efficient Proving

**Approach 1:** Recursive Composition [Valiant '08, BCCT '12]

- Large concrete overheads due to non-black-box crypto
Prior Approaches for Time- and Space-Efficient Proving

Approach 1: Recursive Composition [Valiant '08, BCCT '12]

• Large concrete overheads due to non-black-box crypto (or if brave: algebraic hash functions [BGH19, BCMS20, COS20])
Prior Approaches for Time- and Space-Efficient Proving

Approach 1: Recursive Composition [Valiant '08, BCCT '12]

- Large concrete overheads due to non-black-box crypto
  (or if brave: algebraic hash functions [BGH19, BCMS20, COS20])

- Soundness relies on exotic computational assumptions
Prior Approaches for Time- and Space-Efficient Proving

Approach 1: Recursive Composition [Valiant '08, BCCT '12]

- Large concrete overheads due to non-black-box crypto (or if brave: algebraic hash functions [BGH19, BCMS20, COS20])
- Soundness relies on exotic computational assumptions

Approach 2: Compiling IOPs with space-efficient provers
Prior Approaches for Time- and Space-Efficient Proving

Approach 1: Recursive Composition [Valiant '08, BCCT '12]

- Large concrete overheads due to non-black-box crypto (or if brave: algebraic hash functions [BGH19, BCMS20, COS20])
- Soundness relies on exotic computational assumptions

Approach 2: Compiling IOPs with space-efficient provers

- Until now: space-preserving compilers produced private-coin arguments [Bitansky-Chiesa '12, BHRRS '20]
Prior Approaches for Time- and Space-Efficient Proving

**Approach 1:** Recursive Composition [Valiant '08, BCCT '12]

- Large concrete overheads due to non-black-box crypto (or if brave: algebraic hash functions [BGH19, BCMS20, COS20])
- Soundness relies on *exotic computational assumptions*

**Approach 2:** Compiling IOPs with space-efficient provers

- Until now: space-preserving compilers produced *private-coin* arguments [Bitansky-Chiesa '12, BHRSS '20]
- **This work:** *public-coin* arguments, based on a *simple & falsifiable* "hidden order" assumption
Compiling IOPs to Arguments
Compiling IOPs to Arguments

IOP:

\[ \text{P}(x; w) \quad \rightarrow \quad \ldots \quad \leftarrow \quad \text{V}(x) \]
Compiling IOPs to Arguments

IOP:

\[ P(x; w) \]

\[ V(x) \]
Compiling IOPs to Arguments

IOP:

\[ P(x; w) \]

\[ V(x) \]

\[ \ldots \]
Compiling IOPs to Arguments

IOP:

\[ P(x; w) \]

\[ \pi \]

\[ V(x) \]
Compiling IOPs to Arguments

IOP:

\[ P(x; w) \]

\[ \pi \]

\[ \cdots \]

\[ V(x) \]
Compiling IOPs to Arguments

IOP:

\[ P(x; w) \quad \pi \]

Argument:

\[ P(x; w) \quad V(x) \]
Compiling IOPs to Arguments

IOP:

\[
P(x; w)
\]

\[
\pi
\]

\[
V(x)
\]

commit(\pi)

Argument:

\[
P(x; w)
\]

\[
commit(\pi)
\]

\[
V(x)
\]
Compiling IOPs to Arguments

IOP:

\[ P(x; w) \]

\[ \pi \]

commit(\(\pi\))

Argument:

\[ P(x; w) \]

\[ \pi \]

\[ V(x) \]
Compiling IOPs to Arguments

IOP:

\[ P(x; w) \]

\[ V(x) \]

Argument:

\[ \text{commit}(\pi) \]

\[ i_1, \ldots, i_k \]
Compiling IOPs to Arguments

**IOP:**

\[ P(x; w) \]

\[ \pi \]

\[ V(x) \]

**Argument:**

\[ \text{commit}(\pi) \]

\[ i_1, \ldots, i_k \]

\[ \pi_{i_1}, \ldots, \pi_{i_k} + \text{proof} \]

\[ V(x) \]
Compiling IOPs to Arguments

IOP:

\[ P(x; w) \]

\[ \pi \]

...\[ V(x) \]

Important Question:
Which IOP prover cost is most relevant to argument prover?

A. enumerate all of \( \pi \)
B. compute \( \pi_i \) given \( i \)
C. other?

Argument:

\[ P(x; w) \]

commit(\( \pi \))

\[ i_1, \ldots, i_k \]

\[ \pi_{i_1}, \ldots, \pi_{i_k} + proof \]

\[ V(x) \]
Compiling IOPs to Arguments

IOP: $\pi$

Argument: $\text{commit}(\pi)$

Important Question:
Which IOP prover cost is most relevant to argument prover?

A. enumerate all of $\pi$
B. compute $\pi_i$ given $i$
C. other?

Non-answer:
Depends on how "commit" and "proof" are instantiated...
Compiling IOPs to Arguments

Important Question:
Which IOP prover cost is most relevant to argument prover?

A. enumerate all of $\pi$
B. compute $\pi_i$ given $i$
C. other?

Non-answer:
Depends on how "commit" and "proof" are instantiated...

Why does this matter?
We know IOPs with time- & space-efficient provers in the sense of (B) but not (A).
Instantiations of Commit-and-Prove
Instantiations of Commit-and-Prove

A. Merkle commitments
Instantiations of Commit-and-Prove

A. Merkle commitments
   • Prover's work: ≈ enumerating all of $\pi$
Instantiations of Commit-and-Prove

A. Merkle commitments
   • Prover's work: $\approx$ enumerating all of $\pi$

B. Function commitments [BC '12]
Instantiations of Commit-and-Prove

A. Merkle commitments
   • Prover's work: \( \approx \) enumerating all of \( \pi \)

B. Function commitments [BC '12]
   • Prover's work: \( \approx \) computing \( \pi_i \) for a given \( i \).
Instantiations of Commit-and-Prove

A. Merkle commitments
   • Prover's work: \(\approx\) enumerating all of \(\pi\)

B. Function commitments [BC '12]
   • Prover's work: \(\approx\) computing \(\pi_i\) for a given \(i\).
   • Private coin proofs
Instantiations of Commit-and-Prove

A. Merkle commitments
   • Prover's work: \( \approx \text{enumerating all of } \pi \)

B. Function commitments [BC '12]
   • Prover's work: \( \approx \text{computing } \pi_i \text{ for a given } i. \)
   • Private coin proofs

C. For a "polynomial IOP" \( (\pi : \mathbb{F}_q^n \rightarrow \mathbb{F}_q \text{ is truth table of a multilinear polynomial}), \text{ can use a polynomial commitment [BFS19]} \)
Instantiations of Commit-and-Prove

A. Merkle commitments
   • Prover's work: \( \approx \text{enumerating all of } \pi \)

B. Function commitments [BC '12]
   • Prover's work: \( \approx \text{computing } \pi_i \text{ for a given } i \).
   • Private coin proofs

C. For a "polynomial IOP" (\( \pi : \mathbb{F}_q^n \rightarrow \mathbb{F}_q \) is truth table of a multilinear polynomial), can use a polynomial commitment [BFS19]
   • Polynomial commitments can be public-coin
Instantiations of Commit-and-Prove

A. Merkle commitments
   • Prover's work: \( \approx \) enumerating all of \( \pi \)

B. Function commitments [BC '12]
   • Prover's work: \( \approx \) computing \( \pi_i \) for a given \( i \).
   • Private coin proofs

C. For a "polynomial IOP" (\( \pi : \mathbb{F}_q^n \rightarrow \mathbb{F}_q \) is truth table of a multilinear polynomial), can use a polynomial commitment [BFS19]
   • Polynomial commitments can be public-coin
   • **This work:** Prover's work \( \approx \) enumerating description of \( \pi \) (not the whole truth table);
Instantiations of Commit-and-Prove

A. Merkle commitments
   - Prover's work: $\approx$ enumerating all of $\pi$

B. Function commitments [BC '12]
   - Prover's work: $\approx$ computing $\pi_i$ for a given $i$.
   - Private coin proofs

C. For a "polynomial IOP" ($\pi : \mathbb{F}_q^n \rightarrow \mathbb{F}_q$ is truth table of a multilinear polynomial), can use a polynomial commitment [BFS19]
   - Polynomial commitments can be public-coin
   - This work: Prover's work $\approx$ enumerating description of $\pi$ (not the whole truth table);
     (time- and space-) efficient for known IOPs (e.g. Clover [BTVW14])
Our Polynomial Commitment Efficiency Results
Our Polynomial Commitment
Efficiency Results

**Informal Theorem 1:** Assume a group of "unknown order". Then there is a polynomial commitment scheme with public-coin commit and prove protocols.
Our Polynomial Commitment Efficiency Results

**Informal Theorem 1:** Assume a group of "unknown order". Then there is a polynomial commitment scheme with public-coin commit and prove protocols.

Moreover, the committer/prover on (multi-linear) input $p$ is efficient given *streaming access* to $(p(x))_{x \in \{0,1\}^n}$. 
Informal Theorem 1: Assume a group of "unknown order". Then there is a polynomial commitment scheme with public-coin commit and prove protocols.

Moreover, the committer/prover on (multi-linear) input $p$ is efficient given streaming access to $(p(x))_{x \in \{0,1\}^n}$.

Informal Theorem 2: There are polynomial IOPs where the prover can compute relevant streams above (as well as all other IOP messages) with time- and space-efficiency.
No More Talking About (Fine-Grained) Efficiency
Our Polynomial Commitment Efficiency Results

Informal Theorem 1: Assume a group of "unknown order". Then there is a polynomial commitment scheme with public-coin commit and prove protocols. Moreover, the committer/prover on input $p$ is efficient (in both time and space) given multi-pass streaming access to values of $p$ on $\{0,1\}$.

Informal Theorem 2: There are polynomial IOPs where the prover can compute relevant streams above (as well as all other IOP messages) with time- and space-efficiency.
Polynomial Commitment
Blueprint / Sketch

[BFS19]: Basic framework, buggy instantiation.

They independently discovered bug
Polynomial Commitment

Blueprint / Sketch

\textbf{Commit}(p : \mathbb{F}_q^n \rightarrow \mathbb{F}_q):

Output \( h(p) \), where \( h \) is a "homomorphic CRHF" (more later)

[BFS19]: Basic framework, buggy instantiation.
They independently discovered bug
Polynomial Commit
Blueprint / Sketch

\textbf{Commit}(p : \mathbb{F}_q^n \rightarrow \mathbb{F}_q):

Output \( h(p) \), where \( h \) is a "homomorphic CRHF" (more later)

\textbf{Prove}( "I know a degree-\( d \) poly \( p \) s.t. \text{Commit}(p) = c \\
and \( p(x) = z" )
Polynomial Commitment

Blueprint / Sketch

Commit($p : \mathbb{F}_q^n \rightarrow \mathbb{F}_q$):
Output $h(p)$, where $h$ is a "homomorphic CRHF" (more later)

Prove( "I know a degree-$d$ poly $p$ s.t. abstractly: $f(p) = (c, z)$, where $f$ is a homomorphism

[BFS19]: Basic framework, buggy instantiation.
They independently discovered bug
Polynomial Commitment
Blueprint / Sketch

**Commit** \( (p : \mathbb{F}_q^n \rightarrow \mathbb{F}_q) \):

Output \( h(p) \), where \( h \) is a "homomorphic CRHF" (more later)

**Prove** ("I know a degree-\( d \) poly \( p \) s.t. abstractly: \( f(p) = (c, z) \), where \( f \) is a homomorphism

1. Split claim into similar sub-claims of smaller size
Polynomial Commitment

**Blueprint / Sketch**

**Commit**\( p : \mathbb{F}_q^n \rightarrow \mathbb{F}_q \):  
Output \( h(p) \), where \( h \) is a "homomorphic CRHF" (more later)

**Prove** "I know a degree-\( d \) poly \( p \) s.t. abstractly: \( f(p) = (c, z) \), where \( f \) is a homomorphism

1. Split claim into similar sub-claims of smaller size
2. Combine sub-claims to reduce number
Polynomial Commitment
Blueprint / Sketch

Commit: \( p : \mathbb{F}_q^n \rightarrow \mathbb{F}_q \):
Output \( h(p) \), where \( h \) is a "homomorphic CRHF" (more later)

Prove: "I know a degree-\( d \) poly \( p \) s.t.

abstractly: \( f(p) = (c, z) \),
where \( f \) is a homomorphism

1. Split claim into similar sub-claims of smaller size
2. Combine sub-claims to reduce number
3. Recurse
Polynomial Commit
Blueprint / Sketch

Commit($p : \mathbb{F}^n_q \rightarrow \mathbb{F}_q$):
Output $h(p)$, where $h$ is a "homomorphic CRHF" (more later)

Prove( "I know a degree-$d$ poly $p$ s.t."
abstractly: $f(p) = (c, z)$, where $f$ is a homomorphism

1. Split claim into similar sub-claims of smaller size
2. Combine sub-claims to reduce number
3. Recurse
Not today!

[BFS19]: Basic framework, buggy instantiation.
They independently discovered bug
From Many Claims to Fewer Claims?
From Many Claims to Fewer Claims?

**Initial Claims:** Knowledge of $f$-preimages of $y_1, \ldots, y_k$

(think of $f$ as an arbitrary homomorphism)
From Many Claims to Fewer Claims?

**Initial Claims:** Knowledge of $f$-preimages of $y_1, \ldots, y_k$
(think of $f$ as an arbitrary homomorphism)

**Flawed Protocol:**
From Many Claims to Fewer Claims?

**Initial Claims:** Knowledge of $f$-preimages of $y_1, \ldots, y_k$ (think of $f$ as an arbitrary homomorphism)

**Flawed Protocol:**

1. Let $y' = \sum_i r_i y_i$, for $r_i \leftarrow [2^\lambda]$ sampled by verifier
From Many Claims to Fewer Claims?

**Initial Claims:** Knowledge of $f$-preimages of $y_1, \ldots, y_k$
(think of $f$ as an arbitrary homomorphism)

**Flawed Protocol:***

1. Let $y' = \sum_i r_i y_i$, for $r_i \leftarrow [2^\lambda]$ sampled by verifier

2. Prover proves knowledge of $x' \in f^{-1}(y')$
From Many Claims to Fewer Claims?

**Initial Claims:** Knowledge of $f$-preimages of $y_1, \ldots, y_k$ 
(think of $f$ as an arbitrary homomorphism)

**Flawed Protocol:**

1. Let $y' = \sum_i r_i y_i$, for $r_i \leftarrow [2^\lambda]$ sampled by verifier

2. Prover proves knowledge of $x' \in f^{-1}(y')$

⚠️ Prover might know $x \in f^{-1}(2y_1)$, but not $x \in f^{-1}(y_1)$; could still win with probability $1/2$. 
From Many Claims to Fewer Claims?

Initial Claims: Knowledge of $f$-preimages of $y_1, \ldots, y_k$
(think of $f$ as an arbitrary homomorphism)

Flawed Protocol:

1. Let $y' = \sum_i r_i y_i$, for $r_i \leftarrow [2^\lambda]$ sampled by verifier

2. Prover proves knowledge of $x' \in f^{-1}(y')$

⚠️ Prover might know $x \in f^{-1}(2y_1)$, but not $x \in f^{-1}(y_1)$;
could still win with probability $1/2$.

Prover might only know $x_r \in f^{-1}(\langle r, y \rangle)$ when $r$ is in a lattice
$L \subseteq \mathbb{Z}^k$. Winning probability $\approx$ density of $L$ in $\mathbb{Z}^k$. 

From Many Claims to Fewer Claims?

**Initial Claims:** Knowledge of $f$-preimages of $y_1, \ldots, y_k$ (think of $f$ as an arbitrary homomorphism)

**Flawed Protocol:**

1. Let $y' = \sum_i r_i y_i$, for $r_i \leftarrow [2^\lambda]$ sampled by verifier

2. Prover proves knowledge of $x' \in f^{-1}(y')$

⚠️ Prover might know $x \in f^{-1}(2y_1)$, but not $x \in f^{-1}(y_1)$; could still win with probability $1/2$.

Prover might only know $x_r \in f^{-1}(\langle r, y \rangle)$ when $r$ is in a lattice $L \subseteq \mathbb{Z}^k$. Winning probability $\approx$ density of $L$ in $\mathbb{Z}^k$. 

[BFS19] show *computational* soundness for a specific $f$. 
From Many Claims to Fewer Claims: Our Protocol
From Many Claims to Fewer Claims: Our Protocol

Let $f : \mathbb{G} \to \mathbb{H}$ be an arbitrary homomorphism. Let $y = (y_1, \ldots, y_k) \in \mathbb{H}^k$ be arbitrary.
From Many Claims to Fewer Claims: Our Protocol

Let \( f : \mathbb{G} \rightarrow \mathbb{H} \) be an arbitrary homomorphism and \( y = (y_1, \ldots, y_k) \in \mathbb{H}^k \) be arbitrary.

Prover claims to know \( x = (x_1, \ldots, x_k) \in \mathbb{G}^k \) s.t. \( f(x_i) = y_i \).
From Many Claims to Fewer Claims: Our Protocol

Let \( f : \mathbb{G} \rightarrow \mathbb{H} \) be an arbitrary homomorphism and \( y = (y_1, \ldots, y_k) \in \mathbb{H}^k \) be arbitrary.

Prover claims to know \( x = (x_1, \ldots, x_k) \in \mathbb{G}^k \) s.t. \( f(x_i) = y_i \).
From Many Claims to Fewer Claims: Our Protocol

Let $f : \mathbb{G} \rightarrow \mathbb{H}$ be an arbitrary homomorphism and $y = (y_1, \ldots, y_k) \in \mathbb{H}^k$ be arbitrary.

Prover claims to know $x = (x_1, \ldots, x_k) \in \mathbb{G}^k$ s.t. $f(x_i) = y_i$.
From Many Claims to Fewer Claims: Our Protocol

Let \( f : \mathbb{G} \rightarrow \mathbb{H} \) be an arbitrary homomorphism and \( y = (y_1, \ldots, y_k) \in \mathbb{H}^k \) be arbitrary.

Prover claims to know \( x = (x_1, \ldots, x_k) \in \mathbb{G}^k \) s.t. \( f(x_i) = y_i \).

\[
\begin{align*}
P(x) & \quad A \leftarrow \{0,1\}^{k' \times k} \\
V(y) & \\
& \quad y' := A \cdot y
\end{align*}
\]
From Many Claims to Fewer Claims: Our Protocol

Let \( f : \mathbb{G} \to \mathbb{H} \) be an arbitrary homomorphism, \( y = (y_1, \ldots, y_k) \in \mathbb{H}^k \) be arbitrary.

Prover claims to know \( x = (x_1, \ldots, x_k) \in \mathbb{G}^k \) s.t. \( f(x_i) = y_i \).

\[
P(x) \quad \xrightarrow{\text{A} \leftarrow \{0,1\}^{k' \times k}} \quad \text{V(y)} \quad \xleftarrow{x' := A \cdot x} \quad \text{y' := A \cdot y}
\]
From Many Claims to Fewer Claims: Our Protocol

Let \( f : \mathbb{G} \to \mathbb{H} \) be an arbitrary homomorphism
\( y = (y_1, \ldots, y_k) \in \mathbb{H}^k \) be arbitrary.

Prover claims to know \( x = (x_1, \ldots, x_k) \in \mathbb{G}^k \) s.t. \( f(x_i) = y_i \).

\[ P(x) \quad A \leftarrow \{0, 1\}^{k' \times k} \quad x' := A \cdot x \]

\[ V(y) \quad y' := A \cdot y \]

Accept if \( y_i' = f(x_i') \) for all \( i \in [k'] \).
Soundness: Our Batch Extractor

\[ P(x) \]

\[ A \leftarrow \{0,1\}^{k' \times k} \]

\[ x' := A \cdot x \]

\[ V(y) \]

\[ y' := A \cdot y \]

Accept if

\[ y_i' = f(x_i') \]

for all \( i \in [k'] \)
Soundness: Our Batch Extractor

\[ P(x) \]

\[ A \leftarrow \{0,1\}^{k' \times k} \]

\[ x' := A \cdot x \]

\[ V(y) \]

\[ y' := A \cdot y \]

Accept if

\[ y'_i = f(x'_i) \]

for all \( i \in [k'] \)

Attempt 0:
Soundness: Our Batch Extractor

\[ P(x) \]

\[ A \leftarrow \{0,1\}^{k' \times k} \]

\[ x' := A \cdot x \]

\[ V(y) \]

\[ y' := A \cdot y \]

Accept if

\[ y_i' = f(x_i') \]

for all \( i \in [k'] \)

**Attempt 0:**
Get an accepting transcript \((A, x')\), hope \(A\) has an integer left-inverse.
Soundness: Our Batch Extractor

\[
\begin{align*}
P(x) & \\
A & \leftarrow \{0,1\}^{k'\times k} \\
x' & := A \cdot x \\
V(y) & \\
y' & := A \cdot y
\end{align*}
\]

Accept if \(y'_i = f(x'_i)\) for all \(i \in [k']\)

Attempt 0:
Get an accepting transcript \((A, x')\), hope \(A\) has an integer left-inverse. 

\(k' < k \ldots\)
Soundness: Our Batch Extractor

\[
\begin{align*}
P(x) & \quad A \leftarrow \{0,1\}^{k \times k} \\
x' & := A \cdot x
\end{align*}
\]

\[
\begin{align*}
V(y) & \\
y' & := A \cdot y
\end{align*}
\]

Accept if
\[
y'_i = f(x'_i) \quad \text{for all } i \in [k']
\]

\[
\begin{align*}
\text{Attempt 0:} & \\
& \text{Get an accepting transcript } (A, x'), \text{ hope } A \text{ has an integer left-inverse.}
\end{align*}
\]

Compute \( x = A^{-1} \cdot x' \). (correctness follows from homomorphism)
Soundness: Our Batch Extractor

\[ P(x) \]

\[ A \leftarrow \{0,1\}^{k' \times k} \]

\[ x' := A \cdot x \]

\[ V(y) \]

\[ y' := A \cdot y \]

Accept if
\[ y'_i = f(x'_i) \]
for all \( i \in [k'] \)

Attempt 1:
Soundness: Our Batch Extractor

\[ P(x) \quad \xrightarrow{A \leftarrow \{0,1\}^{k' \times k}} \quad V(y) \]

\[ x' := A \cdot x \]

\[ y' := A \cdot y \]

Accept if
\[ y'_i = f(x'_i) \]
for all \( i \in [k'] \)

Attempt 1:
Rewind until \( B \) accepting transcripts \( \rightarrow A \in \{0,1\}^{Bk' \times k}, x' \in \mathcal{G}^{Bk'} \).
Soundness: Our Batch Extractor

\( P(x) \)

\[ A \leftarrow \{0,1\}^{k' \times k} \]
\[ x' := A \cdot x \]

\( V(y) \)

\[ y' := A \cdot y \]

Accept if

\[ y_i' = f(x_i') \]

for all \( i \in [k'] \)

**Attempt 1:**

Rewind until \( B \) accepting transcripts \( \rightarrow A \in \{0,1\}^{Bk' \times k}, x' \in \mathbb{G}^{Bk'} \).

**Hope** \( A \) has an integer left inverse.
Soundness: Our Batch Extractor

\[ P(x) \quad A \leftarrow \{0,1\}^{k' \times k} \quad x' := A \cdot x \quad V(y) \quad y' := A \cdot y \]

Attempt 1:
Rewind until \( B \) accepting transcripts \( \rightarrow A \in \{0,1\}^{Bk' \times k}, x' \in \mathbb{G}^{Bk'} \).

Hope \( A \) has an integer left inverse.

Compute \( x = A^{-1} \cdot x' \).
### Soundness: Our Batch Extractor

Given a transcript $P(x)$, we attempt to verify that $x$ is a valid input by computing $x' = A^{-1} \cdot x$.

1. **Attempt 1:**
   - **Rewind until $B$ accepting transcripts**
   - **Hope** $A$ has an integer left inverse.
   - **Compute** $x = A^{-1} \cdot x'$

   **Accept if**
   
   $$y'_i = f(x'_i) \text{ for all } i \in [k']$$

   **:-) with all but negl$(k)$ probability**
   
   $$5k/k' \leq B \leq O(1)$$
Soundness: Our Batch Extractor

\[ P(x) \]

\[ A \leftarrow \{0,1\}^{k'\times k} \]

\[ x' := A \cdot x \]

\[ V(y) \]

\[ y' := A \cdot y \]

Accept if
\[ y'_i = f(x'_i) \]
for all \( i \in [k'] \)

Attempt 1:
Rewind until \( B \) accepting transcripts

Hope \( A \) has an integer left inverse.

Compute \( x = A^{-1} \cdot x' \).

Non-uniform distribution; accepting transcripts skewed

\(-\) with all but \( \text{negl}(k) \) probability
if \( 5k/k' \leq B \leq O(1) \)
**Soundness: Our Batch Extractor**

**Attempt 0:**
- Get an accepting transcript
- Hope has an integer left-inverse.

\[ (A, x') \]

\[ A^{-1} \cdot x' \]

\[ \text{Compute. (correctness follows from homomorphism)} \]

\[ x = A^{-1} \cdot x' \]

**Attempt 1:**
- Rewind until accepting transcripts

\[ A \leftarrow \{0,1\}^{k' \times k} \]

\[ x' := A \cdot x \]

**Accept if**
- \( y'_i = f(x'_i) \) for all \( i \in [k'] \)

\[ \forall i \in [k'], y'_i = f(x'_i) \]

\[ A \text{ has an integer left inverse.} \]

\[ \text{Hope} \]

\[ A \text{ is not too skewed} \]

\[ \text{Non-uniform distribution; accepting transcripts skewed} \]

\[ \text{:-) with all but negl}(k) \text{ probability} \]

\[ 5k/k' \leq B \leq O(1) \]
Soundness: Our Batch Extractor

\[ P(x) \]
\[ A \leftarrow \{0,1\}^{k \times k} \]
\[ x' := A \cdot x \]

\[ V(y) \]
\[ y' := A \cdot y \]

Accept if \( y'_i = f(x'_i) \) for all \( i \in [k'] \)

**Attempt 1:**
Rewind until \( B \) accepting transcripts

**Hope** \( A \) has an integer left inverse.

Compute \( x = A^{-1} \cdot x' \).

Non-uniform distribution; accepting transcripts skewed

so \( B\lambda > k \)

so \( A \) is not too skewed

\(-\) with all but \( \text{negl}(k) \) probability

if \( 5k/k' \leq B \leq O(1) \)
Soundness: Our Batch Extractor

$P(x)$

$A \leftarrow \{0,1\}^{k \times k}$

$x' := A \cdot x$

$V(y)$

$y' := A \cdot y$

Accept if $y'_i = f(x'_i)$ for all $i \in [k']$

Attempt 1:
Rewind until $B$ accepting transcripts

Hope $A$ has an integer left inverse.

Compute $x = A^{-1} \cdot x'$.

Non-uniform distribution; accepting transcripts skewed

so $B \lambda > k$

Non-uniform distribution; accepting transcripts skewed

so $A$ is not too skewed

:-) with all but negl$(k)$ probability

if $5k/k' \leq B \leq O(1)$

Bonus: can prove knowledge of "small" $x$ by bounds-checking $x'$;
extractor works because computed $A^{-1}$ has "small" entries ($2^{\text{poly}(k)}$)
Soundness: Our Batch Extractor

Attempt 0:

Get an accepting transcript $\mathcal{A}$, hope has an integer left-inverse $A$. Compute $x = A^{-1} \cdot x'$. (correctness follows from homomorphism)

Attempt 1:

Rewind until accepting transcripts, $B \rightarrow A \in \{0,1\}^{k' \times k}$, $x' \in \mathbb{G}^{Bk}$. Hope $A$ has an integer left inverse. Compute $x = A^{-1} \cdot x'$. :-) with all but $\text{negl}(k)$ probability if $5k/k' \leq B \leq O(1)$

Bonus: can prove knowledge of "small" $x$ by bounds-checking $x'$; extractor works because computed $A^{-1}$ has "small" entries $2^{\text{poly}(k)}$.

Non-uniform distribution; accepting transcripts skewed so $B \lambda > k$ so $A$ is not too skewed actually essential & improperly addressed in [BFS19]
Soundness: Our Batch Extractor

- **\( P(x) \)**
- **\( A \leftarrow \{0, 1\}^{k \times k} \)**
- **\( x' := A \cdot x \)**
- **\( V(y) \)**
- **\( y' := A \cdot y \)**
- **Accept if**
  \( y'_i = f(x'_i) \) for all \( i \in [k'] \)

**Attempt 1:**
Rewind until \( B \) accepting transcripts → \( A \in \{0, 1\}^{Bk' \times k}, x' \in \mathbb{G}^{Bk} \).

**Hope** \( A \) has an integer left inverse.
Compute \( x = A^{-1} \cdot x' \).

- Non-uniform distribution; accepting transcripts skewed
- \( B\lambda > k \) so \( A \) is not too skewed
- \( \negl(k) \) probability
  \(-\) with all but \( 5k/k' \leq B \leq O(1) \)

**Bonus:** can prove knowledge of "small" \( x \) by bounds-checking \( x' \);
extractor works because computed \( A^{-1} \) has "small" entries (2^{\text{poly}(k)})

We want to extract CRHF pre-images, but...

actually essential & improperly addressed in [BFS19]
Homomorphic CRHFs
Homomorphic CRHFs

- Let $\mathbb{G} = \langle g \rangle$ be a group where it is hard to find $x \neq 0$ s.t. $g^x = 1$ (any *multiple* of the order of $g$).
Homomorphic CRHFs

- Let $\mathbb{G} = \langle g \rangle$ be a group where it is hard to find $x \neq 0$ s.t. $g^x = 1$ (any multiple of the order of $g$).

- Hardness holds in generic group of unknown order
Homomorphic CRHFs

• Let $\mathbb{G} = \langle g \rangle$ be a group where it is hard to find $x \neq 0$ s.t. $g^x = 1$ (any *multiple* of the order of $g$).

• Hardness holds in generic group of unknown order

• Concrete candidates:
  • RSA group (private-coin setup)
  • Class groups of imaginary quadratic order (public-coin setup)
Homomorphic CRHFs

- Let $\mathbb{G} = \langle g \rangle$ be a group where it is hard to find $x \neq 0$ s.t. $g^x = 1$ (any *multiple* of the order of $g$).

  - Hardness holds in generic group of unknown order
  - Concrete candidates:
    - RSA group (private-coin setup)
    - Class groups of imaginary quadratic order (public-coin setup)

- Then $h(x) = g^x$ is a homomorphic CRHF from $\mathbb{Z}$ to $\mathbb{G}$
Homomorphic CRHFs: Domains beyond $\mathbb{Z}$
Homomorphic CRHFs: Domains beyond $\mathbb{Z}$

- We wanted an (additively) homomorphic CRHF mapping
  Commit: $\mathbb{Z}[x_1, \ldots, x_n] \to G$  (& extra property I am ignoring)
Homomorphic CRHFs: Domains beyond $\mathbb{Z}$

- We wanted an (additively) homomorphic CRHF mapping
  Commit: $\mathbb{Z}[x_1, \ldots, x_n] \rightarrow \mathbb{G}$  
  (& extra property I am ignoring)

- *Almost* follows from a $\mathbb{Z} \rightarrow \mathbb{G}$ homomorphic CRHF:
Homomorphic CRHFs: Domains beyond $\mathbb{Z}$

- We wanted an (additively) homomorphic CRHF mapping Commit: $\mathbb{Z}[x_1, \ldots, x_n] \rightarrow \mathbb{G}$ (& extra property I am ignoring)

- Almost follows from a $\mathbb{Z} \rightarrow \mathbb{G}$ homomorphic CRHF:
  - Homomorphically "embed" $\mathbb{Z}[x_1, \ldots, x_n]$ into $\mathbb{Z}$ by setting $x_i = q^i$. 
  

Homomorphic CRHFs: Domains beyond \( \mathbb{Z} \)

- We wanted an (additively) homomorphic CRHF mapping
  \[ \text{Commit} : \mathbb{Z}[x_1, \ldots, x_n] \to \mathbb{G} \] (& extra property I am ignoring)

- *Almost* follows from a \( \mathbb{Z} \to \mathbb{G} \) homomorphic CRHF:
  - Homomorphically "embed" \( \mathbb{Z}[x_1, \ldots, x_n] \) into \( \mathbb{Z} \) by setting \( x_i = q^i \).
  - Injective only on \( D := \{\text{small-coefficient multilinear polynomials}\} \) (each coefficient is a digit base-\( q \)).
Homomorphic CRHFs:
Domains beyond $\mathbb{Z}$

• We wanted an (additively) homomorphic CRHF mapping
  Commit: $\mathbb{Z}[x_1, \ldots, x_n] \rightarrow \mathbb{G}$  (& extra property I am ignoring)

• Almost follows from a $\mathbb{Z} \rightarrow \mathbb{G}$ homomorphic CRHF:
  • Homomorphically "embed" $\mathbb{Z}[x_1, \ldots, x_n]$ into $\mathbb{Z}$ by setting $x_i = q^i$.
  • Injective only on $D := \{small-coefficient~multilinear~polynomials\}$ (each coefficient is a digit base-$q$).
  • Thus $\mathbb{Z}[x_1, \ldots, x_n] \rightarrow \mathbb{G}$ composition is a CRHF only on $D$. 
Main Extraction Lemma
Lemma: Let $A \leftarrow \{0,1\}^{5n \times n}$. With all but $2^{-\Omega(n)}$ probability, $A$ has an integer left-inverse.
Lemma: Let $A \leftarrow \{0,1\}^{5n \times n}$. With all but $2^{-\Omega(n)}$ probability, $A$ has an integer left-inverse.
Lemma: Let $A \leftarrow \{0,1\}^{5n \times n}$. With all but $2^{-\Omega(n)}$ probability, $A$ has an integer left-inverse.

A taste of our proof:
Lemma: Let $A \leftarrow \{0,1\}^{5n \times n}$. With all but $2^{-\Omega(n)}$ probability, $A$ has an integer left-inverse.

A taste of our proof:

- Consider sequence of lattices $\{L_i\}$, where $L_i$ is generated by first $i$ rows of $A$
Lemma: Let $A \leftarrow \{0,1\}^{5n \times n}$. With all but $2^{-\Omega(n)}$ probability, $A$ has an integer left-inverse.

A taste of our proof:

• Consider sequence of lattices $\{L_i\}$, where $L_i$ is generated by first $i$ rows of $A$

• Show that $L_i$ rapidly approaches (and becomes) $\mathbb{Z}^n$
**Main Extraction Lemma**

**Lemma:** Let $A \leftarrow \{0,1\}^{5n \times n}$. With all but $2^{-\Omega(n)}$ probability, $A$ has an integer left-inverse.

**A taste of our proof:**

- Consider sequence of lattices $\{L_i\}$, where $L_i$ is generated by first $i$ rows of $A$.
- Show that $L_i$ rapidly approaches (and becomes) $\mathbb{Z}^n$.
  - Equivalently, $|\det(L_i)| \rightarrow 1$.

*5 is not tight, but unimportant today*
Main Extraction Lemma

**Lemma:** Let $A \leftarrow \{0,1\}^{5n \times n}$. With all but $2^{-\Omega(n)}$ probability, $A$ has an integer left-inverse.

A taste of our proof:

- Consider sequence of lattices $\{L_i\}$, where $L_i$ is generated by first $i$ rows of $A$
- Show that $L_i$ rapidly approaches (and becomes) $\mathbb{Z}^n$
  - Equivalently, $|\det(L_i)| \to 1$
  - We analyze prime factorization of $\det(L_i)$, show that each step kills enough prime powers with enough probability to deduce the lemma.

5 is not tight, but unimportant today
Conclusion
Conclusion

• First publicly verifiable arguments for NP that are time- and space-efficient and based on a simple complexity assumptions
Conclusion

- First publicly verifiable arguments for NP that are time- and space-efficient and based on a simple complexity assumptions

- Based on groups of unknown order, but very lattice-related techniques
Conclusion

• First publicly verifiable arguments for NP that are time- and space-efficient and based on a simple complexity assumptions

• Based on groups of unknown order, but very lattice-related techniques

• **Open:** from lattice assumptions, or in random oracle model
Conclusion

- First publicly verifiable arguments for NP that are time- and space-efficient and based on a simple complexity assumptions

- Based on groups of unknown order, but very lattice-related techniques

- Open: from lattice assumptions, or in random oracle model

- Found and fixed bug in DARK polynomial commitment
Conclusion

• First publicly verifiable arguments for NP that are time- and space-efficient and based on a simple complexity assumptions

• Based on groups of unknown order, but very lattice-related techniques

• **Open:** from lattice assumptions, or in random oracle model

• Found and fixed bug in DARK polynomial commitment

• Techniques likely more broadly applicable: we also improve Pietrzak's proof of exponentiation protocol to achieve statistical soundness in arbitrary groups
Questions?