Post-Quantum Designated-Verifier zkSNARKs from Lattices

David Wu
September 2021
Argument Systems

\[ x \in \{0,1\}^* \]

\[ \mathcal{L}_C = \{x : C(x, w) = 1 \text{ for some } w \} \]

Completeness: \[ \forall x \in \mathcal{L}_C : \Pr[P, V](x) = \text{accept} = 1 \]
“Honest prover convinces honest verifier of true statements”

Soundness: \[ \forall x \notin \mathcal{L}_C, \forall \text{ efficient } P^* : \Pr[P^*, V](1^\lambda, x) = \text{accept} = \text{negl}(\lambda) \]
“Efficient prover cannot convince honest verifier of false statement”
Argument Systems

$$x \in \{0, 1\}^*$$

$$L_C = \{x : C(x, w) = 1 \text{ for some } w\}$$

**accept if**

$$x \in L$$

Argument system is **succinct** if:

- Prover communication is $\text{poly}(\lambda + \log|C|)$
- Running time of $V$ is $\text{poly}(\lambda + |x| + \log|C|)$
Succinct Non-Interactive Arguments (SNARGs)

\[ \mathcal{L}_C = \{ x : C(x, w) = 1 \text{ for some } w \} \]

Additional properties of interest:

- **Proof of knowledge**: succinct non-interactive argument of knowledge (SNARK):
  
  “There exists an efficient extractor that can recover a witness from any prover that convinces an honest verifier”
Succinct Non-Interactive Arguments (SNARGs)

$\mathcal{L}_C = \{x : C(x, w) = 1 \text{ for some } w\}$

- Zero-knowledge: “Proof does not leak information about the prover’s witness”
- zkSNARK: zero-knowledge succinct non-interactive argument of knowledge
Succinct Non-Interactive Arguments (SNARGs)

For general NP languages, SNARGs are unlikely to exist in standard model [BP04, Wee05].

\[ \mathcal{L}_C = \{x : C(x, w) = 1 \text{ for some } w\} \]

\[ \pi = P(x, w) \]

For general NP languages, SNARGs are unlikely to exist in standard model [BP04, Wee05].
Succinct Non-Interactive Arguments (SNARGs)

Instantiation: “CS proofs” in the random oracle model [Mic94]

prover

\[(x, w)\]

verifier

\[\pi = P^{RO}(x, w)\]

Argument consists of a single message

random oracle (RO)

\[\text{accept if } V^{RO}(x, \pi) = 1\]
Succinct Non-Interactive Arguments (SNARGs)

Preprocessing SNARGs: allow “expensive” setup

Can consider publicly-verifiable and secretly-verifiable SNARGs

Setup($1^\lambda$)

common reference string (CRS)

verification state

$\sigma \leftarrow \tau$

(prover)

$x, w$

pi = P(\sigma, x, w)$

 verifier

accept if $V(\tau, x, pi) = 1$
Succinct Non-Interactive Arguments (SNARGs)  

Very active area of research (encompassing both theory and practice):  
PHGR13, BCI\(^*\)\(^{13}\), BCC\(^*\)\(^{16}\), Gro16, ZGK\(^*\)\(^{17}\), AHIV17, WTS\(^*\)\(^{18}\), GMNO18, BB\(^*\)\(^{18}\), BBHR19, BCR\(^*\)\(^{19}\), XZ\(^*\)\(^{19}\), LM19, CHM\(^*\)\(^{20}\), BFS20, SL20, Set20, COS20, CY21, GNS21, GMN21, GLS\(^*\)\(^{21}\), and many, many more…  

This talk: post-quantum constructions (specifically, from lattice-based assumptions)
### zkSNARK Constructions

<table>
<thead>
<tr>
<th>Construction</th>
<th>Prover Complexity</th>
<th>Proof Size</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Gro16]</td>
<td>$N \log N$</td>
<td>1</td>
<td>128 bytes</td>
</tr>
<tr>
<td>Marlin [CHM+20]</td>
<td>$N \log N$</td>
<td>1</td>
<td>704 bytes</td>
</tr>
<tr>
<td>Xiphos [SL20]</td>
<td>$N$</td>
<td>$\log N$</td>
<td>61 KB</td>
</tr>
<tr>
<td>Fractal [COS20]</td>
<td>$N \log N$</td>
<td>$\log^2 N$</td>
<td>215 KB</td>
</tr>
<tr>
<td>STARK [BBHR19]</td>
<td>$N \text{ polylog } N$</td>
<td>$\log^2 N$</td>
<td>127 KB*</td>
</tr>
<tr>
<td>[GMNO18]^†</td>
<td>$N \log N$</td>
<td>1</td>
<td>640 KB</td>
</tr>
</tbody>
</table>

Focus is on constructions with a *succinct* verifier

$N$: size of NP relation being verified ($N \approx 2^{20}$ for concrete values)

Asymptotic metrics are given up to $\text{poly}(\lambda)$ factors (for a security parameter $\lambda$)

*For a structured computation

Pre-Quantum

Post-Quantum

[^†]: designated-verifier
## zkSNARK Constructions

<table>
<thead>
<tr>
<th>Construction</th>
<th>Prover Complexity</th>
<th>Proof Size</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Gro16]</td>
<td>$N \log N$</td>
<td>1</td>
<td>128 bytes</td>
</tr>
<tr>
<td>Marlin [CHM+20]</td>
<td>$N \log N$</td>
<td>1</td>
<td>704 bytes</td>
</tr>
<tr>
<td>Xiphos [SL20]</td>
<td>$N$</td>
<td>$\log N$</td>
<td>61 KB</td>
</tr>
<tr>
<td>Fractal [COS20]</td>
<td>$N \log N$</td>
<td>$\log^2 N$</td>
<td>215 KB</td>
</tr>
<tr>
<td>STARK [BBHR19]</td>
<td>$N \text{polylog } N$</td>
<td>$\log^2 N$</td>
<td>127 KB*</td>
</tr>
<tr>
<td>[GMNO18]+</td>
<td>$N \log N$</td>
<td>1</td>
<td>640 KB</td>
</tr>
</tbody>
</table>

1000× gap between size of pre-quantum zkSNARKs and post-quantum ones

**This talk:** constructing shorter post-quantum zkSNARKs (via lattice-based assumptions)
## zkSNARK Constructions

<table>
<thead>
<tr>
<th>Construction</th>
<th>Prover Complexity</th>
<th>Proof Size</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Gro16]</td>
<td>(N \log N)</td>
<td>1</td>
<td>128 bytes</td>
</tr>
<tr>
<td>Marlin [CHM+20]</td>
<td>(N \log N)</td>
<td>1</td>
<td>704 bytes</td>
</tr>
<tr>
<td>Xiphos [SL20]</td>
<td>(N)</td>
<td>(\log N)</td>
<td>61 KB</td>
</tr>
<tr>
<td>Fractal [COS20]</td>
<td>(N \log N)</td>
<td>(\log^2 N)</td>
<td>215 KB</td>
</tr>
<tr>
<td>STARK [BBHR19]</td>
<td>(N \text{polylog} N)</td>
<td>(\log^2 N)</td>
<td>127 KB*</td>
</tr>
<tr>
<td>[GMNO18]†</td>
<td>(N \log N)</td>
<td>1</td>
<td>640 KB</td>
</tr>
<tr>
<td>This work</td>
<td>(N \log N)</td>
<td>1</td>
<td>16 KB</td>
</tr>
</tbody>
</table>

- \(\approx 10\times\) shorter proofs compared to previous post-quantum zkSNARKs for general NP relations
- Prover and verifier are concretely faster compared to most succinct pre-quantum construction [Gro16]
- Construction is designated-verifier (need secret key to check proofs) and has long CRS
Construction Overview

Follows the classic approach of combining an information-theoretic proof system (for NP) with a cryptographic compiler.

Examples:

PCP (or IOP) → zkSNARK

- hash function (or polynomial commitment) [Mic00, BCS16]
- linear-only encryption [BCIOP13, GGPR13]
Follows the classic approach of combining an information-theoretic proof system (for NP) with a cryptographic compiler.

**Starting point:** the [BCIOP13] compiler from linear PCPs to zkSNARKs
- Yields the most succinct pre-quantum zkSNARKs [GGPR13, Gro16]
- Basis of several lattice-based zkSNARKs [BISW17, GMNO18]
Linear Probabilistically-Checkable Proofs (LPCPs)

(\(x, w\)) \rightarrow \pi \in \mathbb{F}^m

linear PCP
“encoding” of statement/witness

\(q \in \mathbb{F}^m\)

\(q^T \pi \in \mathbb{F}\)

Verifier given oracle access to a linear function \(\pi \in \mathbb{F}^m\)

Several instantiations:

- 3-query LPCP based on the Walsh-Hadamard code: \(m = O(|C|^2)\) [ALMSS92]
- 4-query LPCP based on quadratic arithmetic programs: \(m = O(|C|)\) [GGPR13]
Linear Probabilistically-Checkable Proofs (LPCPs)

Oftentimes, verifier is oblivious: the queries $q$ do not depend on the statement $x$.

- $(x, w)$ is input.
- $\pi \in \mathbb{F}^m$ is a linear PCP "encoding" of statement/witness.
- $q^T \pi \in \mathbb{F}$ is checked, with verifier accepting/rejecting based on $q^T \pi$. 

[IKO07]
Linear Probabilistically-Checkable Proofs (LPCPs)

Equivalent view (if verifier is oblivious):

\[ Q \in \mathbb{F}^{m \times k} \]

\[ \pi \in \mathbb{F}^m \]

\[ Q^T \pi \in \mathbb{F}^k \]

\[ Q = \begin{pmatrix} q_1 & q_2 & q_3 & \cdots & q_k \end{pmatrix} \in \mathbb{F}^{m \times k} \]

pack all queries into single matrix
Oblivious verifier can “commit” to its queries ahead of time.

Honest prover takes \((x, w)\) and constructs linear PCP \(\pi \in \mathbb{F}^m\) and computes \(Q^T \pi\).

Two problems:

- Malicious prover can choose \(\pi\) based on queries.
- Malicious prover can apply different \(\pi\) to the different columns of \(Q\).
Oblivious verifier can “commit” to its queries ahead of time.

Honest prover takes \((x, w)\) and constructs linear PCP \(\pi \in \mathbb{F}^m\) and computes \(Q^T \pi\).

Step 1: Encrypt elements of \(Q\) using additively homomorphic encryption scheme.
Oblivious verifier can “commit” to its queries ahead of time

Honest prover takes \((x, w)\) and constructs linear PCP \(\pi \in \mathbb{F}^m\) and computes \(Q^T \pi\)

\[
Q = q_1 q_2 q_3 \ldots q_k
\]

homomorphic evaluation

SNARK proof

part of the CRS
Designated-verifier SNARK:
Decryption key needed to verify

If LPCP verification can be performed
directly on ciphertexts (e.g., with
pairing-based instantiations), then
SNARK is **publicly-verifiable**

Veriﬁer decrypts to learn
$Q^T \pi$ and runs linear PCP
decision procedure

Honest prover takes
$(x, w)$ and constructs
linear PCP $\pi \in \mathbb{F}^m$ and
computes $Q^T \pi$

Verifier decrypts to learn
$Q^T \pi$ and runs linear PCP
decision procedure

homomorphic evaluation

SNARK proof

$q_1^T \pi \quad \ldots \quad q_k^T \pi$
Oblivious verifier can “commit” to its queries ahead of time. Honest prover takes \((x, w)\) and constructs linear PCP \(\pi \in \mathbb{F}^m\) and computes \(Q^T \pi\).

Two problems:

- Malicious prover can choose \(\pi\) based on queries
- Malicious prover can apply different \(\pi\) to the different columns of \(Q\)
Oblivious verifier can “commit” to its queries ahead of time

Honest prover takes \((x, w)\) and constructs linear PCP \(\pi \in \mathbb{F}^m\) and computes \(Q^T \pi\)

\[ Q = q_1q_2q_3 \ldots q_k \]

part of the CRS

[BCIOP13] approach:
- Add a linear consistency check and view construction as a linear IP (LIP)
- Encrypt the LIP queries using a “linear-only” encryption scheme
Oblivious verifier can “commit” to its queries ahead of time.

Honest prover takes \((x, w)\) and constructs linear PCP \(\pi \in \mathbb{F}^m\) and computes \(Q^T\pi\).

\[Q = \begin{bmatrix} q_1 & q_2 & q_3 & \cdots & q_k \end{bmatrix}\]

Intuitively: an encryption scheme that only supports additive homomorphism.

- Encrypt the LIP queries using a “linear-only” encryption scheme.
Linear-Only Encryption

\[ x_1 \in \mathbb{F} \]
\[ x_2 \in \mathbb{F} \]
\[ \vdots \]
\[ x_n \in \mathbb{F} \]

adversary

\[ \alpha_1, \ldots, \alpha_m \in \mathbb{F} \]

extractor

\textbf{Requirement:} If \( \text{Decrypt}(sk, ct) \neq \bot \), then \( \text{Decrypt}(sk, ct) = \sum_{i \in [n]} \alpha_i x_i \)

\textbf{Intuition:} adversary’s strategy can be “explained” by a linear function
Oblivious verifier can “commit” to its queries ahead of time. Honest prover takes \((x, w)\) and constructs linear PCP \(\pi \in \mathbb{F}^m\) and computes \(Q^T\pi\).

All adversarial strategies can be explained by a linear function of the encrypted query components \(\Rightarrow\) soundness can now be based on the soundness of the linear PCP.
Oblivious verifier can “commit” to its queries ahead of time.

Honest prover takes \((x, w)\) and constructs linear PCP \(\pi \in \mathbb{F}^m\) and computes \(Q^T \pi\).

For zero-knowledge, require that LPCP is (honest-verifier) ZK and encryption scheme is circuit private (hides linear combination).

Rest of this talk: will not focus on ZK.

All adversarial strategies can be explained by a linear function of the encrypted query components \(\Rightarrow\) soundness can now be based on the soundness of the linear PCP.
**Candidate Linear-Only Encryption from Lattices**

**Conjecture:** Regev encryption is linear-only

- **KeyGen(1^λ):** Outputs a secret key \( s \in \mathbb{Z}_q^n \)

- **Encrypt(\( s, \mu \in \mathbb{Z}_p \)):** Sample random \( a \leftarrow \mathbb{Z}_q^n \), error \( e \leftarrow \chi \) and output \( ct = (a, s^T a + pe + \mu) \)

- **Decrypt(\( s, ct \)):** Write \( ct = (a, b) \) and output \( (b - s^T a \mod q) \mod p \)

**Correct as long as** \(|e| \leq \frac{q}{2p}\)**
Candidate Linear-Only Encryption from Lattices

Conjecture: Regev encryption is linear-only

KeyGen(1^λ): Outputs a secret key \( s \in \mathbb{Z}_q^n \)

Encrypt(\( s, \mu \in \mathbb{Z}_p \)): Sample random \( a \leftarrow \mathbb{Z}_q^n \), error \( e \leftarrow \chi \) and output
\[
ct = (a, s^T a + pe + \mu)
\]

Decrypt(\( s, ct \)): Additive homomorphism:
\[
\begin{align*}
ct_1 &= (a_1, s^T a_1 + pe_1 + \mu_1) \\
ct_2 &= (a_2, s^T a_2 + pe_2 + \mu_2)
\end{align*}
\]
Then:
\[
ct_1 + ct_2 = (a_1 + a_2, s^T (a_1 + a_2) + pe_1 + e_2 + (\mu_1 + \mu_2))
\]

Homomorphic operations increase noise growth
**Conjecture:** Regev encryption is linear-only

KeyGen$(1^λ)$: Outputs a secret key $s ∈ ℤ_q^n$

Encrypt$(s, μ ∈ ℤ_p)$: Sample random $a ← ℤ_q^n$, error $e ← χ$ and output
$ct = (a, s^T a + pe + μ)$

Decrypt$(s, ct)$: Write $ct = (a, b)$ and output
$(b - s^T a \mod q) \mod p$

While Regev encryption can be extended to obtain FHE, existing constructions require additional components or different message embedding

*Can we get more homomorphism from vanilla Regev?*
Concrete Efficiency of Basic Instantiation

\[ Q = q_1 q_2 q_3 \ldots q_k \]

homomorphic evaluation

linear combinations of length \( m \) over \( \mathbb{F}_p \)

Amount of homomorphism determines scheme parameters

Using quadratic arithmetic programs (for verifying circuit \( C \)):

- \( k = 4 \)
- \( m = O(|C|) \)
- soundness \( \approx \frac{2|C|}{|\mathbb{F}_p|} = \frac{2|C|}{p} \)
Concrete Efficiency of Basic Instantiation

\[ Q = q_1 q_2 q_3 \cdots q_k \]

homomorphic evaluation

linear combinations of length \( m \) over \( \mathbb{F}_p \)

Amount of homomorphism determines scheme parameters

SNARK proof

Using quadratic arithmetic programs (for verifying circuit \( C \)):

- \( k = 4 \)
- \( m = O(|C|) \)
- soundness \( \approx \frac{2|C|}{|\mathbb{F}_p|} = \frac{2|C|}{p} \)

Need to choose encryption modulus \( q \) to support this amount of homomorphism:

\[ \frac{q}{2p} > p \cdot m \cdot B \]

where \( B \) is the initial noise term
Concrete Efficiency of Basic Instantiation

For a circuit with \( m = 2^{20} \) gates and requiring 128 bits of soundness, we require:
- \( p > 2^{148} \), so \( q > 2^{300} \)
- At 128 bits of security, lattice dimension \( n > 10^4 \), so a single Regev ciphertext is \textbf{over 350 KB} (longer than other post-quantum constructions based on IOPs)
- Proof contains \( k \) ciphertexts, so proof is even longer

\textbf{Alternatively:} Use a small plaintext field \( \mathbb{F}_p \) and amplify soundness via parallel repetition
- \( p \approx 2^{20} \) and \( q \approx 2^{100} \): single ciphertext is 45 KB
- Need many copies in this case (\( \approx 128 \) copies), so proof is again very long

[GMNO18]: use an instantiation where \( p = 2^{32} \) \textit{without} soundness amplification
- Proofs are already 640 KB (and provide \( \approx 15 \) bits of provable soundness for verifying computations of size \( 2^{16} \))

New techniques needed to reduce proof size
Revisiting the Bitansky et al. Compiler

Oblivious verifier can “commit” to its queries ahead of time

Honest prover takes \((x, w)\) and constructs linear PCP \(\pi \in \mathbb{F}^m\) and computes \(Q^T \pi\)

\[
Q = \begin{bmatrix}
q_1 \\
q_2 \\
q_3 \\
\vdots \\
q_k
\end{bmatrix}
\]

Key idea: Instead of encrypting each component of \(Q\) individually, encrypt rows instead.
Linear-Only Vector Encryption

\[ v_1 \in \mathbb{F}^k \]
\[ v_2 \in \mathbb{F}^k \]
\[ \vdots \]
\[ v_m \in \mathbb{F}^k \]

plaintext space is a \textit{vector} space
Linear-Only Vector Encryption

\[ \boldsymbol{v}_1 \in \mathbb{F}^k \]
\[ \boldsymbol{v}_2 \in \mathbb{F}^k \]
\[ \vdots \]
\[ \boldsymbol{v}_m \in \mathbb{F}^k \]

\[ \sum_{i \in [m]} \alpha_i \boldsymbol{v}_i \in \mathbb{F}^k \]

supports homomorphic vector addition

**Linear-only**: scheme only supports linear homomorphism
From Linear PCPs to Preprocessing SNARGs

Honest prover takes \((x, w)\) and constructs linear PCP \(\pi \in \mathbb{F}^m\) and computes \(Q^T \pi\)

Verifier decrypts to learn \(Q^T \pi\) and runs linear PCP decision procedure

\[Q = \begin{array}{c}
q_1 \\
q_2 \\
q_3 \\
\vdots \\
q_k \\
\end{array}\]

\[ct_1, ct_2, ct_3, \ldots, ct_m\]

Homomorphic evaluation

\[Q^T \pi\]

SNARK proof
From Linear PCPs to Preprocessing SNARGs

Honest prover takes \((x, w)\) and constructs linear PCP \(\pi \in \mathbb{F}^m\) and computes \(Q^T \pi\).

Proof is a single vector encryption ciphertext

Allows direct compilation from linear PCPs to SNARKs (without extra linearity check from [BCIOP13])
Candidate Linear-Only Vector Encryption

**Conjecture:** Regev encryption is linear-only

**KeyGen**($1^\lambda$): Outputs a secret key $s \in \mathbb{Z}_q^n$

**Encrypt**($s, \mu \in \mathbb{Z}_p$): Sample random $a \leftarrow \mathbb{Z}_q^n$, error $e \leftarrow \chi$ and output

$$ct = (a, s^T a + pe + \mu)$$

**Decrypt**($s, ct$): Write $ct = (a, b)$ and output

$$(b - s^T a \mod q) \mod p$$

**Key observation:** the same vector $a \in \mathbb{Z}_q^n$ can be reused with many different secret keys

Amortized/vectorized variant of Regev encryption [PVW08]
Conjecture: Vectorized Regev encryption [PVW08] is linear-only

KeyGen($1^\lambda$): Outputs a secret key $s \in \mathbb{Z}_q^n$

Encrypt($s, \mu \in \mathbb{Z}_p$): Sample random $a \leftarrow \mathbb{Z}_q^n$, error $e \leftarrow \chi$ and output $ct = (a, s^T a + pe + \mu)$

Decrypt($s, ct$): Write $ct = (a, b)$ and output $(b - s^T a \mod q) \mod p$
Conjecture: **Vectorized** Regev encryption [PVW08] is linear-only

KeyGen($1^\lambda$): Outputs a secret key $S \in \mathbb{Z}_q^{n \times k}$

Encrypt($s, \mu \in \mathbb{Z}_p$): Sample random $a \leftarrow \mathbb{Z}_q^n$, error $e \leftarrow \chi$ and output $ct = (a, s^T a + pe + \mu)$

Decrypt($s, ct$): Write $ct = (a, b)$ and output $(b - s^T a \mod q) \mod p$
Conjecture: **Vectorized** Regev encryption [PVW08] is linear-only

**KeyGen**($1^\lambda$): Outputs a secret key $S \in \mathbb{Z}_q^{n \times k}$

**Encrypt**($S, \mu \in \mathbb{Z}_p^k$): Sample random $\alpha \leftarrow \mathbb{Z}_q^n$, error $e \leftarrow \mathbb{X}_k^k$ and output $ct = (\alpha, S^T\alpha + p e + \mu)$

**Decrypt**($s, ct$): Write $ct = (\alpha, b)$ and output $\left( b - s^T\alpha \mod q \right) \mod p$
Conjecture: **Vectorized** Regev encryption \([\text{PVW08}]\) is linear-only

KeyGen\((1^\lambda)\): Outputs a secret key \(S \in \mathbb{Z}_q^{n \times k}\)

Encrypt\((S, \mu \in \mathbb{Z}_p^k)\): Sample random \(a \leftarrow \mathbb{Z}_q^n\), error \(e \leftarrow \chi^k\) and output
\[
ct = (a, S^T a + pe + \mu)
\]

Decrypt\((S, ct)\): Write \(ct = (a, v)\) and output
\[
(v - S^T a \mod q) \mod p
\]

\(|ct| = (n + k) \log q\) Would be \(k(n + 1) \log q\) using vanilla Regev

Ciphertext size is **additive** in the vector dimension
Candidate Linear-Only Vector Encryption

**Conjecture:** Vectorized Regev encryption [PVW08] is linear-only

KeyGen\( (1^\lambda) \): Outputs a secret key \( S \in \mathbb{Z}_q^{n \times k} \)

Encrypt\( (S, \mu \in \mathbb{Z}_p^k) \): Sample random \( a \leftarrow \mathbb{Z}_q^n \), error \( e \leftarrow \chi^k \) and output
\[
ct = (a, S^T a + pe + \mu)
\]

Decrypt\( (S, ct) \): Write \( ct = (a, v) \) and output
\[
(v - S^T a \mod q) \mod p
\]

\(|ct| = (n + k) \log q\)

Can use modulus switching [BV11, BGV12] to reduce ciphertext size after homomorphic evaluation: \((n + k) \log q \rightarrow (n + k) \log q'\)

Ciphertext size is **additive** in the vector dimension
Lattice-Based zkSNARKs using Vector Encryption

\[ Q = \begin{bmatrix} ct_1 \\ ct_2 \\ \vdots \\ ct_m \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ \cdots \\ q_k \end{bmatrix} \]

homomorphic evaluation

linear combinations of length \( m \) over \( \mathbb{F}_p \)

common reference string

Using quadratic arithmetic programs (for verifying circuit \( C \)):

- \( k = 4 \)
- \( m = O(|C|) \)
- soundness \( \approx \frac{2|C|}{|\mathbb{F}_p|} = \frac{2|C|}{p} \)

SNARK proof

[BI\textsubscript{SW}17, IS\textsubscript{W}21]
Previously techniques to achieve small soundness:
1. Use large $p$ (to ensure LPCP soundness); or
2. Use small $p$ and parallel repetition to amplify soundness

Our approach: parallel repetition of LPCP to amplify soundness:
- Define LPCP to be $t$ independent sets of queries
- Accept only if all $t$ sets accept
- Requires $kt$ LPCP queries and provides soundness $\left(\frac{|C|}{2p}\right)^t$

With vanilla [BCIOP13], same proof size as parallel repetition

With vector encryption, proof is always a single vector encryption ciphertext and $|ct|$ is additive in vector dimension (not multiplicative)

Setting $p \approx 2^{28}$, proof size is 29 KB (with a CRS of size 2.7 GB) for verifying circuit of size $2^{20}$
Further Compression via Extensions Fields

\[ Q = \begin{pmatrix} q_1 & q_2 & q_3 & \cdots & q_k \\ \end{pmatrix} \]

- Homomorphic evaluation
- Linear combinations of length \( m \) over \( \mathbb{F}_p \)
- SNARK proof

\[ Q^T \pi \]

Recall: Noise growth in ciphertexts scales with
- Length \( m \) of linear combination
- Magnitude of coefficients in linear combination \( p \)

Can we further reduce \( p \)?

Idea: use an extension field of small characteristic
Further Compression via Extensions Fields

Suppose $\mathbb{F} = \mathbb{F}_{p^k}$ where $k > 1$

Can still instantiate using quadratic arithmetic programs

Two approaches to compile to a SNARK:

- Compile LPCP over $\mathbb{F}_{p^k}$ to a LPCP over $\mathbb{F}_p$, apply linear-only vector encryption over $\mathbb{F}_p$
  
  Recall that $\mathbb{F}_{p^k} \cong \mathbb{F}_p^k$; field operations in $\mathbb{F}_{p^k}$ are linear transformations over $\mathbb{F}_p^k$
  
  Transformation increases number of queries and query dimension by $k$

- Apply linear-only vector encryption over $\mathbb{F}_{p^k}$
  
  Work over a polynomial ring $R = \mathbb{Z}[x]/\Phi_m(x)$ where $m$ is chosen so that $R/pR \cong \mathbb{F}_{p^k}$
  
  Consider Regev encryption over $R$ (using module lattices)
Further Compression via Extensions Fields

Suppose $\mathbb{F} = \mathbb{F}_{p^k}$ where $k > 1$

Can still instantiate using quadratic arithmetic programs

Two approaches to compile to a SNARK:

- Compile LPCP over $\mathbb{F}_{p^k}$ to a LPCP over $\mathbb{F}_p$
  
  Recall that $\mathbb{F}_{p^k} \cong \mathbb{F}_p^k$; field operations in $\mathbb{F}_{p^k}$ are linear transformations over $\mathbb{F}_p$.
  
  Transformation increases number of queries and query dimension by $k$.

- Apply linear-only vector encryption over $\mathbb{F}_{p^k}$

  Work over a polynomial ring $R = \mathbb{Z}[x]/\Phi_m$ where $\Phi_m$ is a cyclotomic polynomial.

  Consider Regev encryption over $R$ (using module lattices).

In both settings: coefficients of prover’s linear combination have magnitude $\approx p$ while field has size $p^k$. 

\[(x, w) \quad \rightarrow \quad \pi \in \mathbb{F}^m\]

linear PCP
Further Compression via Extensions Fields

\((x, w)\) \rightarrow \pi \in \mathbb{F}^m

This work: consider \textbf{quadratic} extension fields

- \(R = \mathbb{Z}[x]/(x^2 + 1)\) and set \(p = 3 \mod 4\) so \(R_p = R/pR \cong \mathbb{F}_{p^2}\)
- Choose ciphertext \(q\) to be a power of 2
  - All arithmetic operations can be implemented using 128-bit arithmetic
  - Low degree means polynomial arithmetic only slightly more expensive
Further Compression via Extensions Fields

This work: consider quadratic extension fields

- \( R = \mathbb{Z}[x]/(x^2 + 1) \) and set \( p = 3 \mod 4 \) so 
  \[ R_p \cong \mathbb{F}_{p^2} \]
- Choose ciphertext \( q \) to be a power of 2
  - All arithmetic operations can be implemented using 128-bit arithmetic
  - Low degree means polynomial arithmetic only slightly more expensive
- Choose \( p = 2^t \pm 1 \) so \( \mathbb{F}_{p^2} \) has \( 2^{t+1} \)-th roots of unity (for efficient implementation of LPCP prover)

Higher-degree extension makes polynomial arithmetic more costly (or need non-power-of-two modulus to exploit FFTs)
Further Compression via Extensions Fields

Working over extension field reduces noise accumulation ⇒ smaller lattice parameters ⇒ concretely shorter proofs
Further Compression via Extensions Fields

- Slightly more expensive homomorphic operations over extension field, but smaller lattice parameters
- Smaller field \( \Rightarrow \) more LPCP queries for soundness amplification \( \Rightarrow \) higher prover cost

Schemes have comparable prover costs
Using the extension field increases CRS size but decreases proof size

- CRS consists of “compressed” ciphertexts where random component is derived from a PRF (i.e., $ct = (a, v)$ where $a$ is random and $v = S^T a + pe + \mu$)
- Proof consists of full ciphertexts

[see paper for more microbenchmarks]
# Concrete Comparison with zkSNARKs

<table>
<thead>
<tr>
<th>Construction</th>
<th>Size</th>
<th>Time</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CRS</td>
<td>Proof</td>
<td>Setup</td>
</tr>
<tr>
<td>[Gro16]</td>
<td>199 MB</td>
<td>128 bytes</td>
<td>72 s</td>
</tr>
<tr>
<td>Ligero [AHIV17]</td>
<td>–</td>
<td>14 MB</td>
<td>–</td>
</tr>
<tr>
<td>Aurora [BCR+19]</td>
<td>–</td>
<td>169 KB</td>
<td>–</td>
</tr>
<tr>
<td>Fractal [COS20]</td>
<td>11 GB</td>
<td>215 KB</td>
<td>116 s</td>
</tr>
<tr>
<td>This work</td>
<td>5.3 GB</td>
<td>16.4 KB</td>
<td>2240 s</td>
</tr>
<tr>
<td>This work</td>
<td>1.9 GB</td>
<td>20.8 KB</td>
<td>877 s</td>
</tr>
</tbody>
</table>

All benchmarks collected on same hardware for verifying NP relation of size $2^{20}$
## Concrete Comparison with zkSNARKs

<table>
<thead>
<tr>
<th>Construction</th>
<th>Size</th>
<th>Proof</th>
<th>Time</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CRS</td>
<td>Proof</td>
<td>Setup</td>
<td>Prover</td>
</tr>
<tr>
<td>[Gro16]</td>
<td>199 MB</td>
<td>128 bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ligero [AHIV17]</td>
<td>–</td>
<td>14 MB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aurora [BCR+19]</td>
<td>–</td>
<td>169 KB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fractal [COS20]</td>
<td>11 GB</td>
<td>215 KB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This work</td>
<td>5.3 GB</td>
<td>16.4 KB</td>
<td>877 s</td>
<td>56 s</td>
</tr>
<tr>
<td>This work</td>
<td>1.9 GB</td>
<td>20.8 KB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All benchmarks collected on same hardware for verifying NP relation of size $2^{20}$.
## Concrete Comparison with zkSNARKs

<table>
<thead>
<tr>
<th>Construction</th>
<th>Size</th>
<th>Time</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CRS</td>
<td>Proof</td>
<td>Setup</td>
</tr>
<tr>
<td>[Gro16]</td>
<td>199 MB</td>
<td>72 s</td>
<td>79 s</td>
</tr>
<tr>
<td>Ligero [AHIV17]</td>
<td>11 GB</td>
<td>184 s</td>
<td>68 s</td>
</tr>
<tr>
<td>Aurora [BCR+19]</td>
<td>5.3 GB</td>
<td>2240 s</td>
<td>877 s</td>
</tr>
<tr>
<td>Fractal [COS20]</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>This work</td>
<td>1.9 GB</td>
<td>877 s</td>
<td>56 s</td>
</tr>
<tr>
<td>This work</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

- Prover cost is essentially cost of LPCP prover and computing a linear combination.
- 1.2× faster than pairing-based SNARKs.
- Slower than schemes like Ligero based on MPC-in-the-head (which does not have succinct verification).
- If we consider restricted computations, can have much faster provers (e.g., ethSTARK [BBHR19]).

All benchmarks collected on same hardware for verifying NP relation of size $2^{20}$.
## Concrete Comparison with zkSNARKs

<table>
<thead>
<tr>
<th>Construction</th>
<th>Size CRS</th>
<th>Proof</th>
<th>Setup</th>
<th>Proof</th>
<th>Time Verifier</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Gro16]</td>
<td>199 MB</td>
<td>72 s</td>
<td>79 s</td>
<td>128 bytes</td>
<td>3.4 ms</td>
<td>Pairings</td>
</tr>
<tr>
<td>Ligero [AHIV17]</td>
<td>11 GB</td>
<td>116 s</td>
<td>184 s</td>
<td>215 KB</td>
<td>9.5 ms</td>
<td>Random Oracle</td>
</tr>
<tr>
<td>Aurora [BCR+19]</td>
<td>5.3 GB</td>
<td>2240 s</td>
<td>68 s</td>
<td>16.4 KB</td>
<td>1.2 ms</td>
<td>Lattices</td>
</tr>
<tr>
<td>Fractal [COS20]</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.4 ms</td>
<td>Post-Quantum</td>
</tr>
<tr>
<td>This work</td>
<td>1.9 GB</td>
<td>877 s</td>
<td>56 s</td>
<td>20.8 KB</td>
<td>6.3 s</td>
<td>Lattices</td>
</tr>
<tr>
<td>This work</td>
<td>38 s</td>
<td>22 s</td>
<td>–</td>
<td>14 MB</td>
<td>–</td>
<td>Post-Quantum</td>
</tr>
</tbody>
</table>

Lattice-based SNARKs have very lightweight verification: computing a matrix-vector product and rounding. Well-suited for lightweight or energy-constrained devices.

All benchmarks collected on same hardware for verifying NP relation of size $2^{20}$. 
## Concrete Comparison with zkSNARKs

<table>
<thead>
<tr>
<th>Construction</th>
<th>Size</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Gro16]</td>
<td>199 MB</td>
<td>CRS</td>
</tr>
<tr>
<td>Ligero [AHIV17]</td>
<td>14 MB</td>
<td>Proof</td>
</tr>
<tr>
<td>Aurora [BCR+19]</td>
<td>169 KB</td>
<td>Setup</td>
</tr>
<tr>
<td>Fractal [COS20]</td>
<td>11 GB</td>
<td></td>
</tr>
<tr>
<td>This work</td>
<td>5.3 GB</td>
<td></td>
</tr>
<tr>
<td>This work</td>
<td>1.9 GB</td>
<td></td>
</tr>
</tbody>
</table>

**Time**
- **Prover**
  - 72 s
  - 116 s
  - 116 s
  - 2240 s
  - 877 s

**Verifier**
- 3.4 ms
- 9.5 ms
- 6.3 ms
- 1.2 ms
- 0.4 ms

**Limitations of lattice-based SNARKs:**
- Require expensive trusted setup (need to encrypt large number of vectors)
- Resulting construction is designated-verifier (other schemes are publicly-verifiable)
- Resulting CRS is large (lattice ciphertexts still large, even with compression)

All benchmarks collected on same hardware for verifying NP relation of size $2^{20}$
Summary

Directly compile linear PCPs to SNARKs using linear-only vector encryption

Instantiate linear-only vector encryption from vectorized Regev encryption

\[ Q = q_1 q_2 q_3 \ldots q_k \]

Work over extension fields for better concrete efficiency

[see paper for further optimizations]
Publicly-verifiable SNARKs from lattice-based assumptions

Constructions with short proofs but expensive verifiers are known from lattices [BBC+18, BLNS20]

Concretely-efficient designated-verifier SNARKs with reusable soundness from lattices

Thank you!

https://eprint.iacr.org/2021/977

https://github.com/lattice-based-zkSNARKs/lattice-zksnark