Complexity-Preserving Sublinear-Arguments from Symmetric Key Primitives

Laasya Bangalore
Muthu Venkitasubramaniam
Georgetown University

Rishabh Bhadauria
Carmit Hazay
Bar-Ilan University
Succinct (ZK) Arguments [Kil99,Mic00]

Given a language $L$

Completeness: $\forall x \in L, \langle P, V \rangle(x) = 1$ w.p. 1

Soundness: $\forall x \notin L, \forall$ PPT $P^*$, $\langle P^*, V \rangle(x) = 1$ w.p. $\leq \text{negl}(k)$

Zero-knowledge: $\forall x \in L, \forall$ PPT $V^*$, $\exists$ PPT $S$, s.t. $\{\text{view}_{V^*}\langle P, V^* \rangle(x)\} \approx \{S(x)\}$
Main Question

Do there exist complexity-preserving public-coin succinct* arguments for NP from minimal assumptions?
Main Question

Do there exist complexity-preserving public-coin succinct* arguments for NP from minimal assumptions?

This Talk YES*  
* sublinear
Main Question

Do there exist complexity-preserving public-coin succinct* arguments for NP from minimal assumptions?

This Talk: YES*
* sublinear

Time-preserving
Space-preserving

Complexity-preserving
Main Question

Do there exist complexity-preserving public-coin succinct* arguments for NP from minimal assumptions?

This Talk

YES*

* sublinear

Taxonomy of Arguments

• Succinct* vs Non-succinct
• Interactive vs Non-interactive
• Trusted setup vs No setup (transparent)
• ZK vs (only) Integrity
• Public-Key Crypto vs (only) Symmetric-Key Crypto

Time-preserving
Space-preserving

Complexity-preserving
A SNARKy background

1992: Sublinear ZK for NP [Kil92]
2000: CSProofs [Mic00]
2007: IKO [IKO07], IVC [V07]
2008: IVC [Val08]
2010: Short-PB-NIZKA [Groth10], Preprocessing-Verifiable-Computation [GGP10]
2012: QSP [GGPR12], EfficientPCP [IMS12], Succinct-NIArguments [BCIO12]
2013: Pinocchio [PGHR13], SNARKs-for-C [BCGT13], ZK-vonNeumann [BCTV13]
2014: Geppetto [CFHKNPZ14], CyclesOfCurves [BCTV14]
2015: IP4Muggles [GKR15], SNARKs-for-MapReduce [CTV15]
2016: ZKBoo [GMO16], BulletproofsPrequel [BCGP16], Groth16 [Groth16], HybIntZK [CGM16]
2017: Ligero [AHIV17], ZKB++ and Picnic [CDGORRSZ17] (discuss alongside ZKBoo), Hyrax [WTSW], zk-vSQL [ZGKPP17] (can add to existing vSQL section), Bulletproofs [BBPFWM17], SnarkySigs [GM17]
2018: Aurora [BCRSVW18], FRIs [BBHR18], ZKStarks [BBHR18], Picnic2 [KKW18], vRAM [ZGKPP18], DIZK [WZCPS18], UpdatableNIZK [AKM18], HybNIZK [AGM18]
2019: Fractal [COS19], Halo [BGH19], Plonk [GWC19], RedShift [KPV19], Spartan [Setty19], DeepFRI [BGKS19], LatticeZKPs [ESL19], SubversionResistant [Bag19], Darks [BSF19], LatticeSnarkArithmetic [Nit19], ZKPSetMembership [BCFGD19]
2020: Halolfinite [BDFG20], Quarks (Xiphos and Kopis) [SL20], Dory [Lee20], Wolverine [WYKW20], Bulletproofs+ [CHJKS20], SPARKS [EFKP20], Plookup [GW20], SuperSonic [BFS20], CompressedSigma [AC20], LatticeKviaOTC [LKS20], GeneralizedCompressedSigma [ACR20], PVZKfromBlockchain [SSV20], LinePointZK [DIO20], PublicCoinZKTime&Space [BHRRS20], Dory [Lee20], DoublyEfficientIP [ZLWZX20], PQSnarks4Rsis-Rlwe [BCOS20], ZAPsAlgebraicLangs [CH20]
2021: Manta [CXZ21], Nova [KST21], Rinocchio [GNS21], Limbo [DGOT21], QuickSilver [YWW21], Limbo [GOT21], IntRange [CKLR21], SubexpDDH [JL12], Cerberus [LST21], ConstOverZKRamProgs [FKLW21]
2022: NIZK Multiple Verifiers [YW22], Feta [BJOSS22], GOTzilla [BCHM22], ZK UNSAT [LHPT22]

Credits: ZKProof.org
A SNARKy background

(Uniform) RAM program that takes time $T$ and uses space $S$

- Succinct
- Public-coin or publicly verifiable
- Time-preserving
- Space-preserving
- “Black-box” in the underlying assumption
A SNARKy background

(Uniform) RAM program that takes time $T$ and uses space $S$
- Succinct
- Public-coin or publicly verifiable
- Time-preserving
- Space-preserving
- “Black-box” in the underlying assumption

[BC12] Designated Verifier Sublinear Arguments
[HR18] Non-interactive Sublinear Arguments
A SNARKy background

(Uniform) RAM program that takes time $T$ and uses space $S$

• Succinct
• Public-coin or publicly verifiable
• Time-preserving
• Space-preserving

❌ “Black-box” in the underlying assumption

[WTSTW18,XZZPS19,S20,SL20,KMP20,BCG20a,BCG20,...]
All require Prover to use space proportional to $T$!
A SNARKy background

(Uniform) RAM program that takes time $T$ and uses space $S$

- Succinct
- Public-coin or publicly verifiable
- Time-preserving
- Space-preserving
- “Black-box” in the underlying assumption

[Val08, BCCT13] Complexity preserving via recursive composition

[EFKP20] SNARKs for parallel RAM comp. using CRH and SNARKs

[BGH21, BCMS20, COS20] “Heuristic” assumptions
A SNARKy background

(Uniform) RAM program that takes time $T$ and uses space $S$

- Succinct
- Public-coin or publicly verifiable
- **Time-preserving**
- Space-preserving
- “Black-box” in the underlying assumption

[BBHR18,BFHVXZ20] *Folklore*
A SNARKy background

(Uniform) RAM program that takes time $T$ and uses space $S$

- Succinct
- Public-coin or publicly verifiable
- Time-preserving
- Space-preserving
- “Black-box” in the underlying assumption

[BHRRS20] Based on hardness of discrete log
[BHRRS21] Based on hardness assumptions on hidden order groups
A SNARKy background

(Uniform) RAM program that takes time T and uses space S

- Succinct
- Public-coin or publicly verifiable
- Time-preserving
- Space-preserving
- “Black-box” in the underlying assumption

[BHRRS20] Based on hardness of discrete log
[BHRRS21] Based on hardness assumptions on hidden order groups

Main Question: Can we get based on symmetric cryptography?
(Uniform) RAM program $M(x,w)$

- Runs in time $T(|x|)$
- Uses space $S(|x|)$
- It is of constant size

Modelled as a RAM

- Input Tape - Linear Access
- Work Tape - RAM Access

Space = Size of work tape

$\begin{align*}
P & \quad \text{has } x, w \text{ on (two) input tapes with linear access.} \\
V & \quad \text{has } x \text{ on input tape with linear access.}
\end{align*}$
Main Result – Upper Bound

**Theorem:** Assume CRH exists. ∀ NP language verified by a uniform RAM machine running in time $T$ and space $S$ ∃ public-coin (ZK) arg. with soundness $\text{negl}(k)$ such that:

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{P}$</td>
<td>$\tilde{O}(T)$</td>
<td>$\tilde{O}(S)$</td>
</tr>
</tbody>
</table>

where the CRH is used in a black-box way and $\tilde{O}(\cdot)$ ignores $\text{poly}(\log(T), k)$
**Main Result – Upper Bound**

**Theorem:** Assume CRH exists. ∀ NP language verified by a uniform RAM machine running in time $T$ and space $S$ ∃ public-coin (ZK) arg. with soundness $\text{negl}(k)$ such that:

where the CRH is used in a black-box way and $\tilde{O}(\cdot)$ ignores $\text{poly}(\log(T), k)$
Main Result – Upper Bound

**Theorem:** Assume CRH exists. \( \forall \) NP language verified by a uniform RAM machine running in time \( T \) and space \( S \) \( \exists \) public-coin (ZK) arg. with soundness \( \text{negl}(k) \) such that:

\[
\begin{align*}
\text{Time} & : P & \tilde{O}(T) \\
\text{Space} & : V & \tilde{O}(\frac{T}{S} + S) & \tilde{O}(1)
\end{align*}
\]

where the CRH is used in a black-box way and \( \tilde{O}(\quad) \) ignores \( \text{poly}(\log(T), k) \)
**Main Result – Upper Bound**

**Theorem:** Assume CRH exists. ∀ NP language verified by a uniform RAM machine running in time $T$ and space $S$ ∃ public-coin (ZK) arg. with soundness $\text{negl}(k)$ such that:

<table>
<thead>
<tr>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tilde{O}(T)$</td>
<td>$\tilde{O}(S)$</td>
</tr>
<tr>
<td>$\tilde{O}\left(\frac{T}{S} + S\right)$</td>
<td>$\tilde{O}(1)$</td>
</tr>
</tbody>
</table>

Proof Length is $\tilde{O}\left(\frac{T}{S}\right)$

where the CRH is used in a black-box way and $\tilde{O}(\cdot)$ ignores $\text{poly}(\log(T), k)$
Main Result – Upper Bound

**Theorem:** Assume CRH exists. ∀ NP language verified by a uniform RAM machine running in time $T$ and space $S$ ∃ public-coin (ZK) arg. with soundness $\text{negl}(k)$ such that:

<table>
<thead>
<tr>
<th>Time</th>
<th>Space</th>
<th>Complexity preserving</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>$\tilde{O}(T)$</td>
<td></td>
</tr>
<tr>
<td>$V$</td>
<td>$\tilde{O}\left(\frac{T}{S} + S\right)$</td>
<td></td>
</tr>
<tr>
<td>Proof Length is $\tilde{O}\left(\frac{T}{S}\right)$</td>
<td>$\tilde{O}(1)$</td>
<td>Sublinear</td>
</tr>
</tbody>
</table>

where the CRH is used in a black-box way and $\tilde{O}(\quad)$ ignores $\text{poly}(\log(T), k)$
Theorem: Assume CRH exists. ∀ NP language verified by a uniform RAM machine running in time $T$ and space $S$ $\exists$ public-coin (ZK) arg. with soundness $\text{negl}(k)$ such that:

- Time:
  - Prover: $\tilde{O}(T)$
  - Verifier: $\tilde{O}\left(\frac{T}{S} + S\right)$

Proof Length is $\tilde{O}\left(\frac{T}{S}\right)$

where the CRH is used in a black-box way and $\tilde{O}(\cdot)$ ignores $\text{poly}(\log(T), k)$

Our lower bound shows why it’s hard to further improve the proof length.
Ligero [AHIV17]
Proof Schematic
Given a circuit $C$
Prove that $\exists z$, such that $C(z) = 1$
Extended witness $\bar{w}$
Extended witness $\bar{w}$

\[ a \cdot b \geq X \cdot \# \text{gates} \]
Compute Merkle Root

Root( )
\[ \text{Root}(v_1, v_2, v_3, \ldots, f_1, f_2, f_3, \ldots, i_1, i_2, i_3, \ldots) \]
Proof Length: $O(b + \kappa \cdot a)$

Prover Computation: $O(a)$ FFTs of $O(b)$

Set $a = T/S$ and $b = S$
Compute Merkle Root

Compute Row Aggregates

Decommit Columns

Root( ), \(f_1, f_2, f_3, \ldots\)

\(i_1, i_2, i_3, \ldots\)

Proof Length:
\(\tilde{O}\left(\frac{T}{S} + S\right)\)

Prover Computation:
\(\tilde{O}(T)\)

Set \(a = T/S\) and \(b = S\)
What about space-efficiency?

- Compute Merkle Root
- Compute Row Aggregates
- Decommit Columns

$O(S)$ space if the matrix can be computed row by row
What about space-efficiency?

- Compute Merkle Root
- Compute Row Aggregates
- Decommit Columns

$O(S)$ space if the matrix can be computed row by row

[BCGT13] RAM program $\Rightarrow$ succinct circuits $\Rightarrow$ Transcript generated in time $T$ and space $S$
Computing Merkle Root Space Efficiently
Computing Merkle Root Space Efficiently
Computing Merkle Root Space Efficiently
Computing Merkle Root Space Efficiently
Computing Merkle Root Space Efficiently
Computing Merkle Root Space Efficiently
Computing Merkle Root Space Efficiently
Computing Merkle Root Space Efficiently
Computing Merkle Root Space Efficiently
Computing Merkle Root Space Efficiently
Computing Merkle Root Space Efficiently
Computing Merkle Root Space Efficiently

\[ S \]

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{merkle_tree}
\caption{Merkle tree representation.}
\end{figure}
Computing Merkle Root Space Efficiently
Computing Merkle Root Space Efficiently

S
What about space-efficiency of Row Aggregates?

- **Code test** - the prover encoded each row correctly
- **Quadratic test** - the prover computed multiplication gates correctly
- **Linear test** - the prover computed “linear” gates correctly

Compute Merkle Root

Compute Row Aggregates

Decommit Columns
What about space-efficiency of Row Aggregates?

- **Code test** - the prover **encoded** each row correctly
- **Quadratic test** - the prover computed **multiplication** gates correctly
- **Linear test** - the prover computed "linear" gates correctly

- **Compute Merkle Root**
- **Compute Row Aggregates**
- **Decommit Columns**
Linear gates were correct

Linear constraints can be expressed as $A\bar{w} = \bar{b}$

Check if $\bar{q}$ is a valid codeword encoding $\bar{r}^T\bar{b}$ and agrees on revealed columns.
Linear gates were correct

Linear constraints can be expressed as $A\bar{w} = \bar{b}$

Check if $\bar{q}$ is a valid codeword encoding $\bar{r}^T\bar{b}$ and agrees on revealed columns

Main Challenge: How to compute $\bar{r}^T A$ efficiently?
Linear gates were correct

Linear constraints can be expressed as $A\bar{w} = \bar{b}$

Check if $\bar{q}$ is a valid codeword encoding $\bar{r}^T\bar{b}$ and agrees on revealed columns

Main Challenge: How to compute $\bar{r}^T A$ efficiently?

Size of $r$: $\tilde{O}(T)$

Size of $A$: $\tilde{O}(T) \times \tilde{O}(T)$
Linear gates were correct

Linear constraints can be expressed as $A\bar{w} = \bar{b}$

Check if $\bar{q}$ is a valid codeword encoding $\bar{r}^T \bar{b}$ and agrees on revealed columns

Main Challenge: How to compute $\bar{r}^T A$ efficiently?

Naive Approach: $\tilde{O}(T^2)$ time & $\tilde{O}(T^2)$ space.

Size of $A$: $\tilde{O}(T) \times \tilde{O}(T)$

Size of $r$: $\tilde{O}(T)$
How to compute $\bar{r}^T A$ efficiently?

\[ r \]

\[ r \text{ requires } O(T) \text{ space!} \]
How to compute $\bar{r}^T A$ efficiently?

$r$

- Generate $r$ from a seed
- Required space: $\tilde{O}(1)$
How to compute $\bar{r}^T A$ efficiently?

$\bar{r}$ requires $O(T)$ space!

- Generate $r$ from a seed
- Required space: $\tilde{O}(1)$

$A$ is a large matrix!
How to compute $\bar{r}^T A$ efficiently?

- Generate $\bar{r}$ from a seed
- Required space: $\tilde{O}(1)$

$A$ is a large matrix!

- $A$ is sparse and structured
How to compute $\bar{r}^T A$ efficiently?

- $A$ is a large matrix!
- $A$ is sparse and structured
- $[BCGT13]$ Uniform RAM machine $\Rightarrow$ Succinct Circuit $\Rightarrow$ Succinct matrix

$r$ requires $O(T)$ space!

- Generate $r$ from a seed
- Required space: $\tilde{O}(1)$
How to compute $\bar{r}^T A$ efficiently?

$r$ requires $O(T)$ space!

- Generate $r$ from a seed
- Required space: $\tilde{O}(1)$

$A$ is a large matrix!

- $A$ is sparse and structured
- [BCGT13] Uniform RAM machine $\Rightarrow$ Succinct Circuit $\Rightarrow$ Succinct matrix
- Non-zero elements in each column can be computed in $\tilde{O}(1)$ time without storing $A$. 
How to compute $\bar{r}^T A$ efficiently?

$r$ is a large matrix!

- Generate $r$ from a seed
- Required space: $\tilde{O}(1)$
- $A$ is sparse and structured
- [BCGT13] Uniform RAM machine => Succinct Circuit => Succinct matrix
- Non-zero elements in each column can be computed in $\tilde{O}(1)$ time without storing $A$.

$r$ requires $O(T)$ space!

$r^T A$ can be computed row-by-row in time $\tilde{O}(T)$ and space $\tilde{O}(S)$
How to compute $\bar{r}^T A$ efficiently?

$r$ requires $O(T)$ space!

- Generate $r$ from a seed
- Required space: $\tilde{O}(1)$

Hence $r^T A$ can be computed row-by-row in time $\tilde{O}(T)$ and space $\tilde{O}(S)$

- $A$ is sparse and structured
- [BCGT13] Uniform RAM machine => Succinct Circuit => Succinct matrix
- Non-zero elements in each column can be computed in $\tilde{O}(1)$ time without storing $A$. 

$P$ is complexity-preserving.
Main Result – Lower Bound*

**Theorem:** Assume CRH exists. ∀ NP language verified by a uniform RAM machine running in time $T$ and space $S$ ∃ public-coin (ZK) arg. with soundness $\text{negl}(k)$ such that:

<table>
<thead>
<tr>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tilde{O}(T)$</td>
<td>$\tilde{O}(\frac{T}{S} + S)$</td>
</tr>
</tbody>
</table>

Proof Length is $\tilde{O}\left(\frac{T}{S}\right)$

where the CRH is used in a black-box way and $\tilde{O}(\ )$ ignores $\text{poly}(\log(T), k)$.
Current Techniques*: Most ZK-SNARKs rely on Codes...

RAM

Time $T$
Space $S$

Transcript

Size $T$

Encoded Transcript

Using Codes with Constant Relative Distance
Current Techniques*: Most ZK-SNARKs rely on Codes...

RAM

Time $T$
Space $S$

Transcript

Size $T$

Encoded Transcript

Using Codes with Constant Relative Distance

“Lower bound” on the encoding algorithm

provides evidence
Current Techniques*: Most ZK-SNARKs rely on Codes...

Using Codes with Constant Relative Distance

“Lower bound” on the encoding algorithm

provides evidence

“Hard” to reduce the proof length below $\tilde{O}\left(\frac{T}{S}\right)$
Lower Bound*

**Theorem:** Let $C$ be $[n, m, \delta m]$ code over $\mathbb{F}$ then any $r$-pass Encode RAM algorithm requires space

$$S > \frac{\delta n}{r} \log |\mathbb{F}|$$
Lower Bound*

**Theorem**: Let $C$ be $[n, m, \delta m]$ code over $\mathbb{F}$ then any $r$-pass Encode RAM algorithm requires space

$$S > \frac{\delta n}{r} \log|\mathbb{F}|$$

[YRST02] Proved the lower bound for 1-pass algorithm
Lower Bound*

**Theorem:** Let $C$ be $[n, m, \delta m]$ code over $\mathbb{F}$ then any $r$-pass Encode RAM algorithm requires space

$$S > \frac{\delta n}{r} \log |\mathbb{F}|$$

[YRST02] Proved the lower bound for 1-pass algorithm

Setting $n = T$ and $|\mathbb{F}| = \log T$ we get that $\delta < \frac{S \cdot r}{T \cdot \log T}$

=>$\text{Minimum number of oracle queries is } \tilde{O} \left( \frac{1}{\delta} \right) = \tilde{O} \left( \frac{T}{S} \right)$
Summary

• First construction of complexity-preserving sublinear (ZK) arguments from CRH in a black-box way

• Improving the communication will require new techniques – i.e., getting around constant-distance codes.

Thank You
Additional Slides
Warmup Lower Bound

Simplifying assumptions:
1. Only one pass over the message
2. No work tape
3. Codeword head is independent of the contents of the message
Warmup Lower Bound

Simplifying assumptions:
1. Only one pass over the message
2. No work tape
3. Codeword head is independent of the contents of the message