**MIRAGE**: Succinct Arguments for Randomized Algorithms with Applications to Universal zk-SNARKs

**Ahmed Kosba** (Alexandria University)

**Dimitrios Papadopoulos** (Hong Kong University of Science and Technology)

**Charalampos Papamanthou** (University of Maryland)

**Dawn Song** (UC Berkeley)
Zero-Knowledge Proofs [GMR85]

- The zero knowledge proof $\pi$ should be convincing without leaking any information about $w$.

\[ y = F(x, w) \]
zk-SNARKs

- Zero knowledge succinct non-interactive arguments of knowledge
  - Main advantage: Very short proofs and fast verification
- In this talk, we consider QAP-based zk-SNARKs [GGPR13, PGHR13, Groth16], which provide succinct constant-size proofs.
  - This was attractive for many applications.
Challenges of zk-SNARKs in Practice

• Challenge 1: High proof computation cost
  • This led to several works on efficient circuit representations, SNARK-friendly cryptography, back end optimizations, distributed zk-SNARK proof computation, etc.
  • Examples include: Pantry [BFR+13], libsnark [BCTV14a], Scalable SNARKs [BCTV14b], TrueSet [KPP+14], Buffet [WSH+15], Ad-SNARK [BBFR15], Geppetto [CFH+15], C0C0 [KZM+16], [FFG+16], xJsnark [KPS18], DIZK [WZC+18]

• Challenge 2: Trusted setup per computation
  • The prover and verifier need access to a common reference string that is generated in a trusted manner.
  • If done insecurely, the prover can cheat.
# ZK Proof Systems

| Trusted setup per computation | [GGPR13], Pinocchio [PGHR13], [Groth16] | • Succinct proofs (128 to 288 bytes)  
• Efficient verification |
|------------------------------|----------------------------------------|-----------------------------------------------------------------------------------|
| No trusted setup             | Ligero [AHIV17], zk-STARKs [BBHR18],  
Bulletproofs [BBBPWM18], 
Hyrax [WTSTW18], Aurora [BCRSVW18], 
Virgo [ZXZS20], .. | • The proof size and/or the verification effort are increased. |

**Approach 1: Universal Circuits**  
vnTinyRAM [BCTV14]  
• Maintains succinct proofs and efficient verification  
• However, it has *quasilinear* circuits.  
• Very high proof computation cost.

**Approach 2: Universal Updatable CRS**  
[GKMMM18], Sonic [MBKM19]  
Concurrent: PLONK [GWC19], MARLIN [CHMMVW19]  
• In Sonic (unhelped) mode, proof is 1.1 KB.  
• Concurrent work: 448 bytes – 1 KB.
Our Contributions

• We address the previous two challenges via
  • Enabling randomized verification in zk-SNARK circuits.
  • Making universal circuits more efficient.

• In comparison with other universal ZK proof systems,
  • Universal circuit is linear instead of vnTinyRAM's quasilinear circuit.
  • Succinct proofs and efficient verification (Proof size = 160 bytes)
  • Proof size is 7x less than Sonic (unhelped), and 2.8x less than concurrent work.

• Limitations:
  • CRS is not updatable
  • Proof computation overhead is high in comparison with per-circuit preprocessing zk-SNARKs
QAP-based zk-SNARK Circuits

```c
int compute(int[] input, int[] witness)
{
    //
    return result;
}
```

How to support randomized algorithms?

Constraints

\[ c_5 = c_3 \cdot c_4 \]
\[ c_6 = c_5 \cdot (c_1 + c_2) \]

...
Why Randomized Algorithms?

• Many problems can be solved more efficiently using randomized algorithms. Examples include:
  • Polynomial identity testing
  • Primality testing

• In the case of universal zk-SNARK circuits, randomization can help with verifying permutations efficiently.
Randomized Verification in the Circuit

If we allow the prover to choose the randomness, or if the prover knows it before computing the solution, the prover can cheat.
Randomized Verification in the Circuit

• Naïve solution:

This solution will have a very high cost, due to calling the hash function in the circuit.

Question: Can we support randomized verification without having to pay this cost?
Randomized Verification in the Circuit

• We modify the Groth16 zk-SNARK protocol to support randomization
  • The prover adds one group element to the zk-SNARK proof. (Total proof size: 160 bytes)
  • The verifier will do one extra pairing, and apply hash function calls on part of the zk-SNARK proof.

• Intuition (simplified):
  • In a zk-SNARK protocol, the prover computes group elements as functions of all wires in the circuit.
  • These group elements can act as commitments.
  • We force the prover to do the computation of the proof over two stages.
  • We utilize the first part of the zk-SNARK proof to produce the randomness needed for the rest of the circuit.
How to make Universal Circuits more efficient?
Universal zk-SNARK Circuits

What is a universal circuit?

Example: A simple universal circuit that supports two multiplication operations and two addition operations.

- **MUL**: \( (id_1, v_1), (id_2, v_2), (id_3, v_3) \)
- **MUL**: \( (id_4, v_4), (id_5, v_5), (id_6, v_6) \)
- **ADD**: \( (id_7, v_7), (id_8, v_8), (id_9, v_9) \)
- **ADD**: \( (id_{10}, v_{10}), (id_{11}, v_{11}), (id_{12}, v_{12}) \)
Universal zk-SNARK Circuits

The circuit must

1. Verify correctness
   Example: assert \((v_1 * v_2 = v_3)\)

2. Verify consistency
   Example:
   - If \((id_1 = id_8)\), assert \((v_1 = v_8)\)

To implement (2) efficiently, this requires checking permutations in the circuit.
Universal zk-SNARK Circuits

• To verify permutations, previous approaches, e.g., vnTinyRAM, use a permutation network. This has an $O(n \log n)$ overhead, where $n$ is the number of operations.

• Using our modified zk-SNARK, we reduce this cost to $O(n)$.

• We explore other issues related to universal circuit design in the paper.
Evaluation

• Comparison with custom zk-SNARK circuits and vnTinyRAM
  • We use vnTinyRAM results from [WSH+15]

• Scale of supported applications under nearly similar circuit costs:

<table>
<thead>
<tr>
<th>Matrix multiplication O(m^3) operations</th>
<th>Universal circuit?</th>
<th>Supported Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffet, xJsnark</td>
<td>×</td>
<td>m = 188</td>
</tr>
<tr>
<td>vnTinyRAM</td>
<td>✓</td>
<td>m = 7</td>
</tr>
<tr>
<td>MIRAGE</td>
<td>✓</td>
<td>m = 41</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Merge sort O(m log m) operations</th>
<th>Universal circuit?</th>
<th>Supported Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>xJsnark</td>
<td>×</td>
<td>m = 600</td>
</tr>
<tr>
<td>vnTinyRAM</td>
<td>✓</td>
<td>m = 32</td>
</tr>
<tr>
<td>MIRAGE</td>
<td>✓</td>
<td>m = 200</td>
</tr>
</tbody>
</table>

We reduce the gap between the universal circuit approaches and the custom circuits.
Evaluation

• Privacy-preserving smart contracts.
  • In HAWK [KMS+16], a trusted setup is needed per smart contract.
  • Instead, MIRAGE's universal circuit can be used.
    • Cryptographic keys will be generated once in a trusted manner.
    • For any new computation, a publicly verifiable custom verification key (32 bytes) will be pushed to the blockchain. (This does not require a trusted setup)

<table>
<thead>
<tr>
<th>Auction (6 parties)</th>
<th>Universal Setup</th>
<th>Universal PK</th>
<th>Universal VK</th>
<th>Custom PK</th>
<th>Custom VK</th>
<th>Proof time</th>
<th>Proof size</th>
<th>Verification time</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAWK</td>
<td>×</td>
<td>N/A</td>
<td>57.8 MB</td>
<td>3.9 KB</td>
<td>10.3 sec</td>
<td>128 B</td>
<td>1.5 ms</td>
<td></td>
</tr>
<tr>
<td>This work</td>
<td>✓</td>
<td>1.8 GB</td>
<td>473 KB</td>
<td>N/A</td>
<td>322 sec</td>
<td>160 B</td>
<td>2.1 ms</td>
<td></td>
</tr>
</tbody>
</table>

Cost of universality

Succinct proof and minimal verification overhead
Conclusions and Future Directions

• We presented MIRAGE, which enables
  • Verification of randomized algorithms in zk-SNARK circuits
  • Linear-sized universal circuits

• Future directions:
  • More optimization for universal circuits
  • Explore scalability options
  • Integrate randomization in zk-SNARK compilers (for non-universal circuits)
Thank you!

ahmed.kosba@alexu.edu.eg