POSEIDON: A New Hash Function for Zero-Knowledge Proof Systems

Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, Markus Schofnegger

USENIX Security ’21
Motivation – Hash Functions in Zero-Knowledge Protocols

- Private cryptocurrency spending
  - Sign transaction \( h = H(k, m) \), where \( k \) is secret
  - Add \( h \) to Merkle tree \( T \) of coins
  - Later prove that
    \[
    (1) \quad h \in T \\
    (2) \quad h = H(k, m)
    \]
- \( h \) is used in a zero-knowledge context (SNARK, STARK, Bulletproofs, …)
- Proving that \( h \in T \) is in general expensive
Traditional Hash Functions

- So why not just use e.g. SHA-256?
  - Too expensive (almost one minute for proofs in early Zcash)
- Proof procedure
  - Express proof verification algorithm as circuit over some field
  - Proof generation time depends on circuit size, width, degree
- Traditional hash functions not well-suited
  - Mainly optimized for e.g. performance on a certain architecture
  - Design something new
Which properties are we looking for?

- Operate in big finite field
  - E.g., field of \( \approx 256 \) bits
- Consider new metrics
  - Degrees
  - Size of circuit
- Cryptographic security
The POSEIDON \( \pi \) Permutation

- Based on HADES strategy [GLR+20]
- Mixture of full nonlinear and partial nonlinear rounds
- Fixed MDS matrix as linear layer
- “Efficient” S-box
  - Low-degree polynomial
  - E.g., \( x^3 \) or \( x^5 \)
- Flexible design
  - Different field sizes, number of S-boxes, …
The **POSEIDON Hash Function**

- Sponge hash construction
  - $r$ message elements are added per call
  - $c$ elements remain untouched
- $\mathcal{P}$ is the **POSEIDON**$^\pi$ permutation
  - Width of $r + c$ elements
- Adjust $c$ according to security level and field size
Cryptanalysis

- Hash-function-specific cryptanalysis
  - Keyless setting, CICO, preimage, …
- Evaluated many strategies from the last couple of decades
- Algebraic attacks most promising
  - Interpolation, Gröbner bases, …
- Statistical attacks prevented by external rounds
Sponges in Merkle Trees

- For arity $t$, use permutation of size $t + 1$
- Fix one element, take out one element

- Arities e.g. 2, 4, 8
R1CS for POSEIDON$^\pi$

- Single S-boxes in most rounds
- Optimized constraint representation
  - Include linear layer and round constants in fewer constraints
- For $R_F$ full and $R_P$ partial rounds:
  \[ 3tR_F + 3R_P \] constraints for POSEIDON$^\pi$ with $x^5$

- Merkle tree with $2^m$ elements:
  \[
  \frac{m}{\log_2(\text{arity} - 1)} \text{ levels}
  \]
Benchmarks

- Focus on security level of 128 bits
- Comparison in two directions
  - R1CS constraints, hashing performance
  - Prove leaf knowledge in Merkle tree of $2^{30}$ elements
- Result:
  - Very low number of R1CS constraints
  - Proof verification in $< 1$ second with Ristretto
  - Up to 15x hashing performance of comparable competitors
## Benchmarks – R1CS

<table>
<thead>
<tr>
<th></th>
<th>Poseidon-128</th>
<th>Rescue-x⁵</th>
<th>SHA-256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merkle tree arity</td>
<td>Width</td>
<td>Total constraints</td>
<td>Width</td>
</tr>
<tr>
<td>2:1</td>
<td>3</td>
<td>7290</td>
<td>3</td>
</tr>
<tr>
<td>4:1</td>
<td>5</td>
<td>4500</td>
<td>5</td>
</tr>
<tr>
<td>8:1</td>
<td>9</td>
<td>4050</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Merkle tree arity** refers to the ratio of the number of inputs to the number of outputs in the R1CS circuit. **Width** is the number of bits per input or output channel. **Total constraints** is the total number of arithmetic and Boolean constraints in the circuit.
### Benchmarks – Runtime

<table>
<thead>
<tr>
<th>Merkle tree arity</th>
<th>Width</th>
<th>Plain time / call</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>POSEIDON-128</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2:1</td>
<td>3</td>
<td>0.033 ms</td>
</tr>
<tr>
<td>4:1</td>
<td>5</td>
<td>0.08 ms</td>
</tr>
<tr>
<td>8:1</td>
<td>9</td>
<td>0.259 ms</td>
</tr>
<tr>
<td><strong>Rescue-x^5</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2:1</td>
<td>3</td>
<td>0.525 ms</td>
</tr>
<tr>
<td>4:1</td>
<td>5</td>
<td>0.555 ms</td>
</tr>
<tr>
<td>8:1</td>
<td>9</td>
<td>1.03 ms</td>
</tr>
</tbody>
</table>
Implementations

- Available in various languages
  - Rust, Go, Sage, C++
- Circuit implementations in Bulletproofs and Circom
- Reference implementations available online
  - [https://extgit.iaik.tugraz.at/krypto/hadeshash](https://extgit.iaik.tugraz.at/krypto/hadeshash)
  - Use version 1.1
POSEIDON in the Wild

- Already used in various protocols
  - Filecoin\(^1\): Merkle tree proofs
  - Dusk Network\(^2\): securities trading
  - Sovrin [Lod19]: Merkle-tree-based revocation
  - Loopring\(^3\): private trading on Ethereum
  - Semaphore\(^4\), Tornado Cash\(^5\), Hermez\(^6\), …

\(^1\)https://github.com/filecoin-project/neptune
\(^2\)https://github.com/dusk-network/Poseidon252
\(^3\)https://tinyurl.com/y7tl537o
\(^4\)https://semaphore.appliedzkp.org/
\(^5\)https://tornado.cash/
\(^6\)https://hermez.io/
Thank you!

More info: poseidon-hash.info
Contact: team@poseidon-hash.info
References
