One out of billion within one second: ZK-friendly hash functions Poseidon and Starkad

Dmitry Khovratovich
with Arnab Roy, Lorenzo Grassi, Christian Rechberger, Sebastian Ramacher, Markus Schofnegger

Ethereum Foundation and Dusk Network
and University of Bristol and Graz University

Real World Cryptography, 10 Jan 2020

Introduction

Private cryptocurrency spending:

1. Sign a transaction $h = H(\mathrm{K}, \mathrm{MetaData})$;
2. h is added to Merkle tree T of valid coins.
3. After a while, spend o by proving that
   - $h \in T$;
   - $h = H(\mathrm{K}, \mathrm{MetaData})$ for K you know;

h is referred to in zero knowledge using SNARK (Pinocchio, Groth16, Sonic, Plonk, Marlin) or STARK or Bulletproofs.

The most computationally expensive is to prove

$$h \in T.$$

Zcash 1.0: 45 seconds for a proof because SHA-256 was used for the tree.

Problems with traditional hash functions

Traditional collision-resistant functions are not quite suited for SNARKs (and STARKs) as their circuits are too complex and slow in SNARK/STARK-friendly fields. Why?

Problems with traditional hash functions

Traditional collision-resistant functions are not quite suited for SNARKs (and STARKs) as their circuits are too complex and slow in SNARK/STARK-friendly fields. Why?

How all such proofs are constructed:

1. Express the proof verification algorithm as a circuit over some field ($GF(p)$ with 256/384-bit p for SNARKs/Bulletproofs, $GF(2^n)$ with $n = 32/64/128$ for STARKs);
2. In SNARKs, a trusted party creates a setup for fast polynomial commitments (*proving key*).
3. In Bulletproofs/STARKs, the proving key is just the circuit itself.
4. For each proof, combine the actual execution trace with the proving key.

Proof generation time depends on the circuit size, width, degree.

Hash functions we need

- Operate in a big prime or big binary field;
- Best in certain metrics (circuit size or degree-size product);
- Secure.

Hash functions for Zero-knowledge proof systems

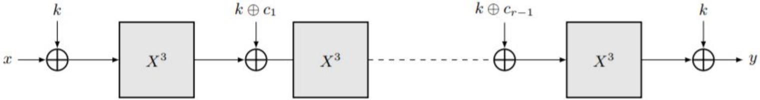
Finite field friendly designs are different from those optimized for x86 (i.e. for binary rings):

- Blake2b is one of the fastest hashes on x86 but its bitwise functions make it very slow in ZK (20-30,000 constraints or a huge AET). Same for SHA-3.
- Pedersen hash with curve points B_1, B_2 is

$$h(X, Y) = ([X]B_1 + [Y]B_2)_{\mathrm{x-coord}}$$

  has many problems: homomorphism, length-extension attack, low preimage security.

MIMC over $GF(p)$ or $GF(2^n)$:

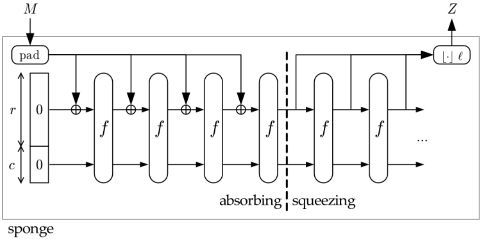


1. Raise to the power of 3;
2. Add constant;
3. Go to step 1.

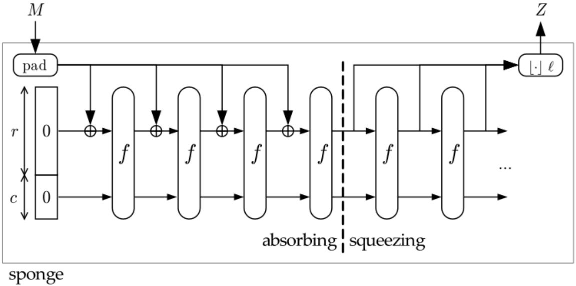
$\approx n \log_3 2$ steps are needed to achieve maximum degree.
Non-trivial to generalize for a wider state.

Poseidon and Starkad

Let us work in a finite field $\mathbb{F}$:



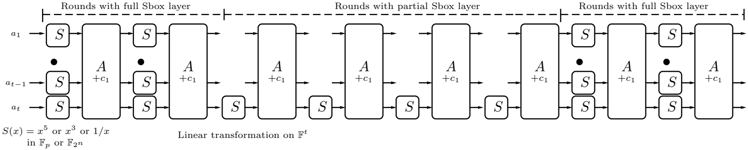
- Bijective transformation f of width $r + c$ field elements;
- r message $\mathbb{F}$ elements are added per call;
- Subset of c elements left untouched (for 128-bit security level and 256-bit fields $c = 1$).
- Permutation should behave like random one up to 2^{128} queries.

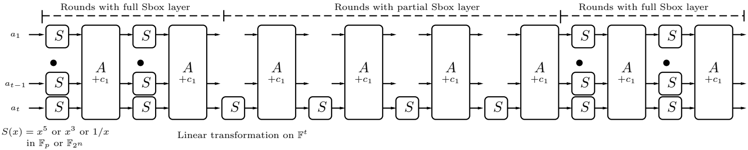


Advantages

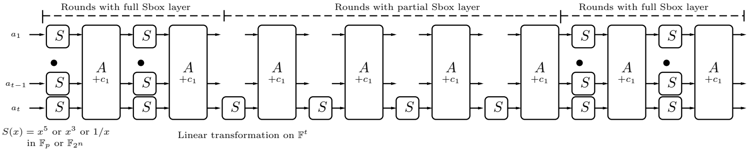
- No key schedule;
- Simpler analysis for many attacks
- Well-known SPN approach (many rounds of nonlinear S-boxes + linear mixing) fits well.

Substituion-Permutation Network:





- S-boxes are R1CS/AET friendly, so low-degree polynomials (x^3, x^5, or $1/x$);
- Linear transform is finite field matrix multiplication;
- In middle rounds – only one S-box! Why?



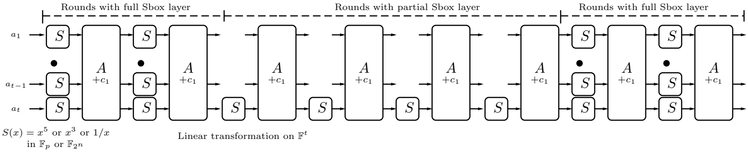
- Checked 10+ methods from 1990 to 2018;
- For finite field designs the most efficient method is algebraic (Groebner, interpolation, etc.);
- Algebraic methods stop working when the permutation has high (2^{128} in our case) degree of its inputs.
- Apparently, the degree grows as good if only one S-box is used.
- 8 outer rounds have S-boxes everywhere to prevent statistical attacks (differential etc.).

Results

Outcome

- Design suitable for both binary and prime fields;
- Most of analysis apply to all fields simultaneously or with simple changes;
- Simple pseudocode (except for round constants, they have elaborate one-time setup);
- Low-degree exponent S-boxes, so expect reasonable non-ZK performance;
- Available implementations: Rust, Go, Sage, C++, Circom.
- Support of Merkle trees with various arities (2:1, 4:1, 8:1).
- Long message support (padding!).
- Authenticated encryption.

Instances



Poseidon:

- Prime field $\mathbb{F}_p$;
- S-box is x^5 for many popular curves;

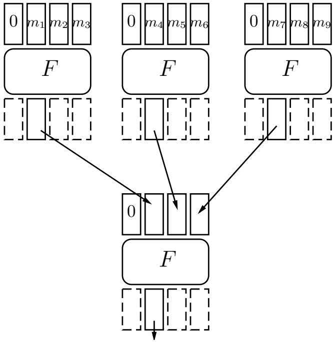
Starkad:

- Binary field $\mathbb{F}_{2^n}$;
- S-box is x^3;

Sponges on trees:

- For arity $t : 1$ use $(t+1)$-wide permutation;
- Fix one element.
- Take out one element.

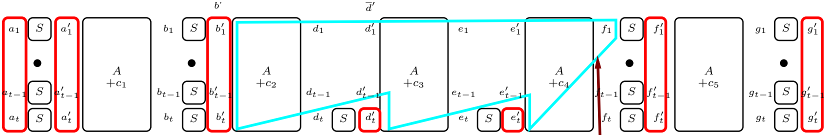
3:1 tree:



In Zero Knowledge

In SNARKs

Algebraic constraints:



Input variables: $a_1, a_2, \ldots, a_t$

Output variables: $g'_1, g'_2, \ldots, g'_t$

Constraint variables: $a'_1, a'_2, \ldots, a'_t$
$b'_1, b'_2, \ldots, b'_t$
d'_t
e'_t
$f'_1, f'_2, \ldots, f'_t$

Notation: $A[i,:]$ – i-th row of A
$A[:,j]$ – j-th column of A

Constraints: $a_i \cdot a'_i = 1,\ i = 1, 2, \ldots, t$
$(\langle A[i,:], \overline{a}' \rangle + c_1) \cdot b'_i = 1,\ i = 1, 2, \ldots, t$
$(\langle A[t,:], \overline{b}' \rangle + c_2) \cdot d'_t = 1$
$(\langle B[t,:], \overline{b}' || d'_t \rangle + c_3) \cdot e'_t = 1$
$(\langle C[i,:], \overline{b}' || d'_t || e'_t \rangle + c_4) \cdot f'_i = 1,\ i = 1, 2, \ldots, t$
$(\langle A[i,:], \overline{f}' \rangle + c_5) \cdot g'_i = 1,\ i = 1, 2, \ldots, t$

B and C are matrices specially derived from A

Relate through S-box only.

252-bit x^5 S-boxes (Ristretto), Merkle tree of 2^{30} elements, 127-bit collision resistance.

Poseidon				
Arity	Width	R_F	R_P	Total constraints
2 : 1	3	8	55	7110
4 : 1	5	8	56	4320
8 : 1	9	8	57	3870
Pedersen hash				
510	171	–	–	43936
Rescue				
2 : 1	3	22	–	11880
4 : 1	5	14	–	6300
8 : 1	9	10	–	5400

Bulletproofs performance to prove 1 out of 2^{30} set:

Field	Arity	Merkle 2^{30}-tree ZK proof Bulletproofs time Prove	Merkle 2^{30}-tree ZK proof Bulletproofs time Verify	R1CS Constraints
POSEIDON hash				
BLS12-381	2:1	16.8s	1.5s	7110
	4:1	13.8s	1.65s	4320
	8:1	11s	1.4s	3870
BN254	2:1	11.2s	1.1s	7110
	4:1	9.6s	1.15s	4320
	8:1	7.4s	1s	3870
Ristretto	2:1	8.4s	0.78s	7110
	4:1	6.45s	0.72s	4320
	8:1	5.25s	0.76s	3870

Plonk

Plonk [GWC19] is a new SNARK using universal trusted setup and Kate commitments.
Poseidon permutation with x^5 of width w in Plonk:

- Standard Plonk: quadratic Bulletproof-like constraints. $11(w(w+6)+3)R$ exponentiations, and proof has 7 $\mathbb{G}$ and 7 $\mathbb{F}$ elements.
- Tailored Plonk:
  - Define a polynomial for each S-box line;
  - Avoid permutation arguments.
  - $(w+11)R$ exponentiations, proof is $((w+3)\mathbb{G}_1, 2w\mathbb{F})$.
  - 25-40x increase in performance.

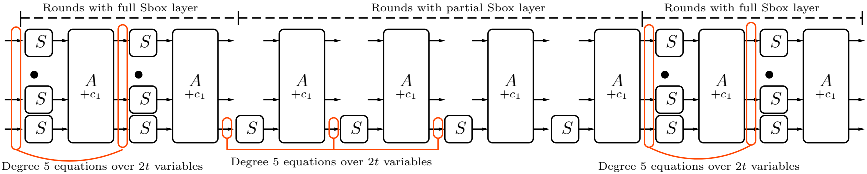
RedShift [KPV19] is a post-quantum trustless SNARK using Reed-Solomon commitments.

Proof is $c_\lambda \log d^2$ where d is the degree of circuit polynomials and $c_\lambda \approx 2.5$KB for 120-bit security.

2^{30}-size Merkle tree based on a Poseidon permutation of width 5 in RedShift:

- Standard RedShift: quadratic Bulletproof-like constraints.
- Tailored RedShift (same way as Plonk).
  - Polynomials of degree $15wR = 4800$ for the entire tree;
  - Total proof around 12 KB.

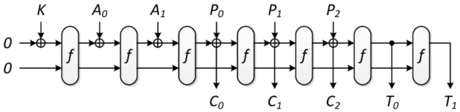
Algebraic execution trace:



- Input variables and S-box inputs only.
- Trace of width $t =$ width of permutation:
  - For full rounds – linear relations between simple S-box ouptuts (degree 3 of inputs) and S-box inputs of the next round;
  - For partial rounds – polynomial of degree 3 over $2t$ S-box inputs.

Encryption

Verifiable authenticated encryption can be implemented with ECDH and SpongeWrap:



1. $\mathbb{F}$ is a scalar field of the ZK proof system.
2. Let recipient have a key on an elliptic curve $\mathcal{E}(\mathbb{F})$.
3. Diffie-Hellman: create a shared secret keypoint K on $\mathcal{E}$.
4. Select nonce N and run 5-wide Poseidon in SpongeWrap with $(0, len, K_x, K_y, N)$ as input.
5. Add 4 plaintext $\mathbb{F}$ elements per permutation call.

The last 3 steps form a SNARK circuit.

Projects that plan to use our design:

- Sovrin: zero-knowledge revocation check with statuses stored in the Merkle tree;
- Dusk Network: zero-knowledge proof of stake;
- Loopring DEX Protocol.
- CODA Protocol.

Projects that plan to use our design:

- Sovrin: zero-knowledge revocation check with statuses stored in the Merkle tree;
- Dusk Network: zero-knowledge proof of stake;
- Loopring DEX Protocol.
- CODA Protocol.

JOIN!

Other resources

Website https://www.poseidon-hash.info/.

Parameter generator: (appears soon).

Questions?