Lookup Arguments and Symmetric Crypto

Dmitry Khovratovich (Ethereum Foundation)

STAP'23 (22 April 2023, Lyon)

Ethereum Foundation

- Non-profit organization doing research on Ethereum-related topics
- Cryptography, consensus, network security
- EF Crypto Group: 10 cryptographers https://crypto.ethereum.org/team
    - Zero-knowledge, VDFs, data availability, PQ designs, hash functions, secret leader election, MPC, lattice-based crypto…
    - In symmetric crypto:
        - Design
        - Cryptanalysis
        - Bounties
        - Collaboration with other divisions and research groups outside ETH
        - Interns are welcome

Zero knowledge…

and verifiable computation

Zero-knowledge proofs and verifiable computation

You know that something specific has occurred…

Zero-knowledge proofs and verifiable computation

You know that something specific has occurred…

…even if you don’t know what exactly.

Verifiable computation

Native computation

$$y = f(x_1, x_2, \dots, x_n)$$

Prove that you compute:

- Variables $y, x_1, x_2, \dots, x_n$
- Polynomial equations $F_i(y, x_1, x_2, \dots, x_n)$

which altogether imply $y = f(x_1, x_2, \dots, x_n)$

Verifiable computation

Native	Equation
$y = x^2$	$x^2 - y = 0$
$y = \sqrt{x};$	$y^2 - x = 0$
$y \leftarrow x^{(4p-3)/5} \bmod p$	$y^5 = x$

Verifiable computation

Progress in ZKSnarks. Computation of length N

- 1990: PCP theorem: every NP statement can be checked in logarithmic time
- 2000s: proof for computation of length N can be composed in subquadratic time

Verifiable computation

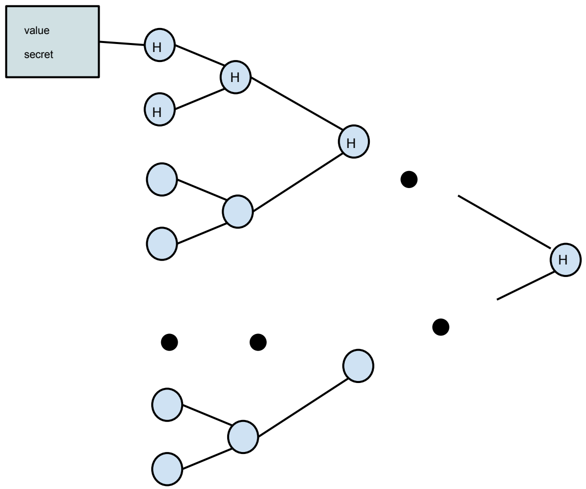
Progress in ZKSnarks. Computation of length N

- Pinocchio (2012): Prover O(N), const size proof
  - with trusted setup
  - Verification needs a few pairings
- Zcash (2014) and SHA-256
  - Private cryptocurrency
  - To spend one proves a path in the SHA-256 Merkle tree
  - Proof for 32 SHA-256 calls took 42 seconds!
- Groth16 (2016): even smaller proof and faster prover

ZCash Story

How ZCash works:

- All my coins are in a Merkle tree, with leafs being commitment to a secret.
- To spend a coin anonymously I have to prove there is a leaf whose secret I know.
- Original tree was built on SHA-256.



ZCash Story

How ZCash works:

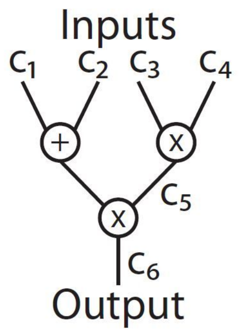
- All my coins are in a Merkle tree, with leafs being commitment to a secret.
- To spend a coin anonymously I have to prove there is a leaf whose secret I know.
- Original tree was built on SHA-256.
- A proof took >40 seconds to generate.

Verifiable computation

Progress in ZKSnarks. Computation of length N

- Pinocchio (2012): Prover O(N), const size proof
- Zcash (2014) and SHA-256
- Groth16 (2016): even smaller proof and faster prover
- Bulletproofs (2016): O(N) prover and verifier, no trusted setup needed
- STARKs (2018): polyLog-time verifier, no trusted setup
- Plonk (2019): setup once and for all
- Recursive schemes:
  - Aurora
  - Nova
  - Halo 1/2

Arithmetic circuits



From x86 to finite field arithmetic

- All arithmetic operations done modulo p
- Bitwise operations are nonexistent

Possible fields:

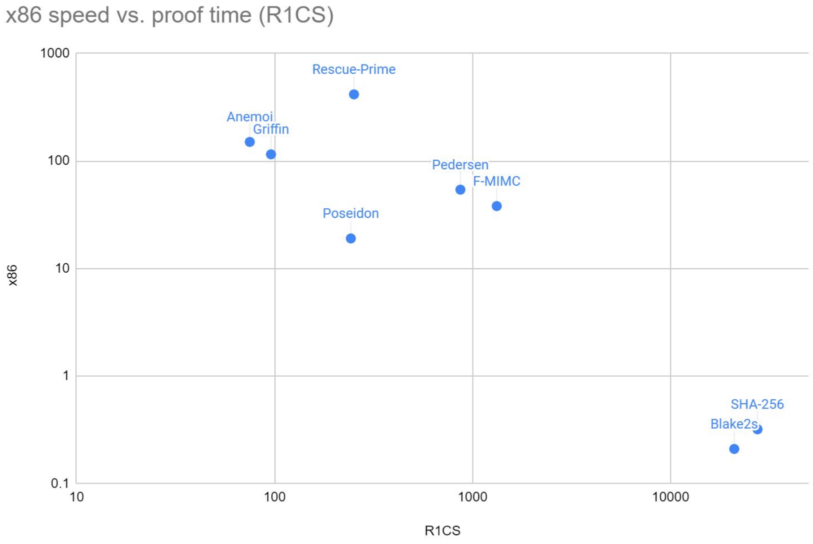
- Scalar field (size of prime order group) of elliptic curves BN254, BLS12-381; all about 2^{254} in size.

Supported operations:

- Addition
- Multiplication by variable or by constant
- Iterative computations are better than vonNeumann architecture

Problem

Hash	ZK time	x86 time	Cryptanalysis invested
BLAKE2	100	1	10
Poseidon	1	100	1
Rescue	1	1000	1
Pedersen	4	500	50

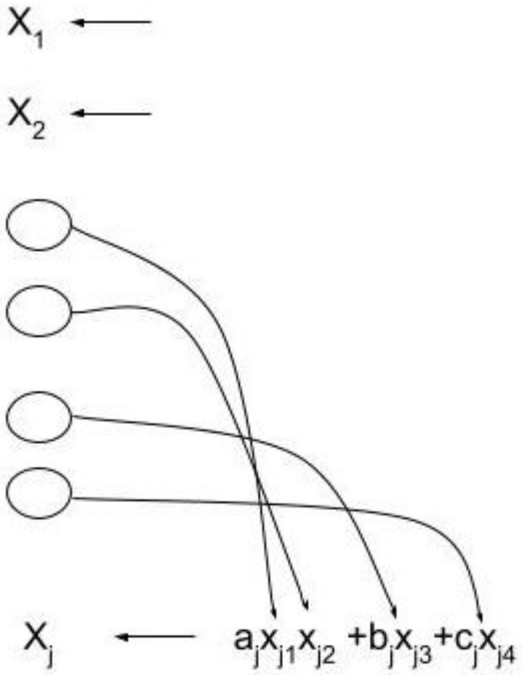


Modern zero-knowledge proof systems

How Plonk works

- Consider a deterministic arithmetic algorithm
    - Step j:

$$x_j \leftarrow a_j x_{i_{j,1}} x_{i_{j,2}} + b_j x_{i_{j,3}} + c_j x_{i_{j,4}}$$



How Plonk works

- Consider a deterministic arithmetic algorithm
  - Step j: $x_j \leftarrow a_j x_{i_{j,1}} x_{i_{j,2}} + b_j x_{i_{j,3}} + c_j x_{i_{j,4}}$
- Encode as a table
- Reinterpret in polynomials

Step\Column	1	2	3	4	5	6	7	8	9
1	x_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{2,4}$	a_1	b_1	c_1	d_1
2									
j	x_j	$x_{j,1}$	$x_{j,2}$	$x_{j,3}$	$x_{j,4}$	a_j	b_j	c_j	d_j
N									

Index\Poly	$f(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_a(x)$	$f_b(x)$	$f_c(x)$	$f_d(x)$
ω	x_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{2,4}$	a_1	b_1	c_1	d_1
ω^2									
ω^j	x_j	$x_{j,1}$	$x_{j,2}$	$x_{j,3}$	$x_{j,4}$	a_j	b_j	c_j	d_j
ω^N									

How Plonk works

- Consider a deterministic arithmetic algorithm
    - Step j: $x_j \leftarrow a_j x_{i_{j,1}} x_{i_{j,2}} + b_j x_{i_{j,3}} + c_j x_{i_{j,4}}$
- Encode as a table
- Reinterpret in polynomials

Index\Poly	$f(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_a(x)$	$f_b(x)$	$f_c(x)$	$f_d(x)$
ω	x_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{2,4}$	a_1	b_1	c_1	d_1
ω^2									
ω^j	x_j	$x_{j,1}$	$x_{j,2}$	$x_{j,3}$	$x_{j,4}$	a_j	b_j	c_j	d_j
ω^N									

- Algorithm is correct iff

$$f(x) \equiv f_a(x) f_1(x) f_2(x) + f_b(x) f_3(x) + f_c(x) f_4(x)$$

- Such equations are easy to check. Costs to create a proof are ~9N group operations

How Plonk works

- Consider a deterministic arithmetic algorithm

$$x_j \leftarrow a_j x_{i_{j,1}} x_{i_{j,2}} + b_j x_{i_{j,3}} + c_j x_{i_{j,4}}$$

- Encode as a table

Index\Poly	$f(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_a(x)$	$f_b(x)$	$f_c(x)$	$f_d(x)$
ω	x_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{2,4}$	a_1	b_1	c_1	d_1
ω^2									
ω^j	x_j	$x_{j,1}$	$x_{j,2}$	$x_{j,3}$	$x_{j,4}$	a_j	b_j	c_j	d_j
ω^N									

- Reinterpret in polynomials

- Algorithm is correct iff

$$f(x) \equiv f_a(x) f_1(x) f_2(x) + f_b(x) f_3(x) + f_c(x) f_4(x)$$

- Such equations are easy to check.
  - Commit to all polynomials
  - Open all at random point
  - Check equation at this point.

$$C, C_1, C_2, C_3, C_4 \leftarrow Commit(f, f_1, f_2, f_3, f_4)$$
$$\lambda \leftarrow H(C, C_1, C_2, C_3, C_4)$$
$$\pi \leftarrow Proof(y, y_1, y_2, y_3, y_4 : \text{ values of } f, f_1, f_2, f_3, f_4 \text{ at } \lambda)$$

Lookup Argument

Lookup Argument

- Original table T
- Witness
- Claim $\forall x \; f_1(x) \in T$

Table T
a_1
a_2
…
a_k

Index\Poly	$f(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_a(x)$	$f_b(x)$	$f_c(x)$	$f_d(x)$
ω	x_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{2,4}$	a_1	b_1	c_1	d_1
ω^2									
ω^j	x_j	$x_{j,1}$	$x_{j,2}$	$x_{j,3}$	$x_{j,4}$	a_j	b_j	c_j	d_j
ω^N									

Cached quotients lookup [EFG23]

- Rational function equation (1)

$$\forall x \in S: \; f(x) \in T \;\text{ iff }\; \exists \{m_t\}: \sum_{t \in T} \frac{m_t}{X+t} \equiv \sum_{x \in S} \frac{1}{X+f(x)}$$

- Example

$$\{1,0,1\} \subseteq \{0,1,2,3\} \;\Leftrightarrow\; \frac{1}{X} + \frac{2}{X+1} + \frac{0}{X+2} + \frac{0}{X+3} \equiv \frac{1}{X+1} + \frac{1}{X} + \frac{1}{X+1}$$

Cached quotients lookup [EFG23]

- Rational function equation (1)

$$\forall x \in S: \; f(x) \in T \;\text{ iff }\; \exists \{m_t\}: \sum_{t \in T} \frac{m_t}{X+t} \equiv \sum_{x \in S} \frac{1}{X+f(x)}$$

- Example

$$\{1,0,1\} \subseteq \{0,1,2,3\} \;\Leftrightarrow\; \frac{1}{X} + \frac{2}{X+1} + \frac{0}{X+2} + \frac{0}{X+3} \equiv \frac{1}{X+1} + \frac{1}{X} + \frac{1}{X+1}$$

- Poly equation is derived from (1)
- Costs 8|S| group operations (assuming |T| log |T| preprocessing)

Table T
a_1
a_2
...
a_k

Index\Poly	$f(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_a(x)$	$f_b(x)$	$f_c(x)$	$f_d(x)$
ω	x_1	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{2,4}$	a_1	b_1	c_1	d_1
ω^2									
ω^j	x_j	$x_{j,1}$	$x_{j,2}$	$x_{j,3}$	$x_{j,4}$	a_j	b_j	c_j	d_j
ω^N									

Cost summary

- Arithmetic operations
  - If all gates are the same $x_j \leftarrow a_j x_{i_{j,1}} x_{i_{j,2}} + b_j x_{i_{j,3}} + c_j x_{i_{j,4}}$ then N gates cost 9N
- Custom gates of degree D of E variables cost ~D*E per gate
- Lookups
  - K lookups from table of size W cost 8K (assuming W log W preprocessing)

Symmetric design

SHA-256 (per 512 bits)

- As arithmetic circuit: 26000 constraints
- With lookups: 1700 gates (lookups of size 2^{10} and polys of degree 6)

AES-128:

- With lookups: 2000 gates per 128 bits

Poseidon:

- 280 gates per 512 bits
- 1/100 of SHA-256 speed on x86

Reinforced Concrete:

- 300 gates with lookups per 512 bits
- 1/10 of SHA-256 speed on x86

Primitives for verifiable computation

Features

- Small arithmetic circuits
- Reasonably fast on x86
- Scalability
- Security

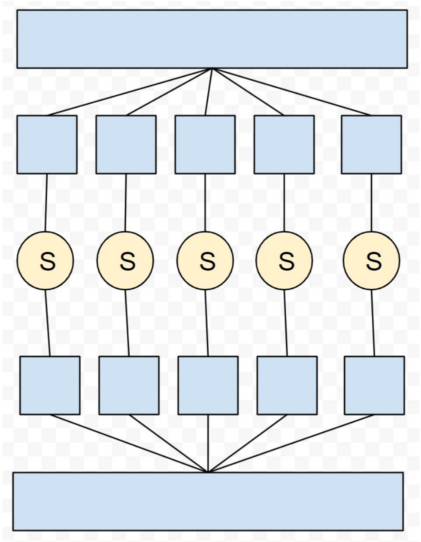
What security means

- Only real attacks: collision/preimage/key recovery
- Resistance to statistical attacks
- Resistance to algebraic attacks: high overall degree, many high-degree equations.
- Getting high-degree:
    - Exponentiation to high degree (Rescue)
    - Many rounds (Poseidon)
    - Both is expensive

Advantages of lookups

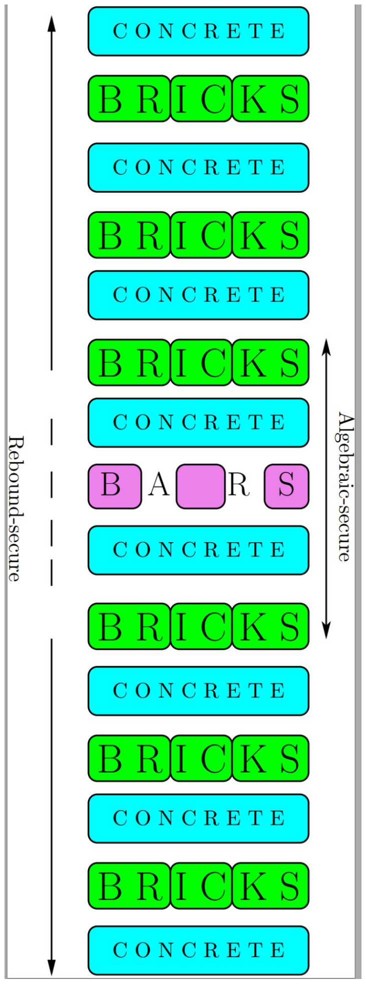
What Sboxes/lookups provide

- High algebraic degree
- So fewer rounds
- So faster



Reinforced Concrete

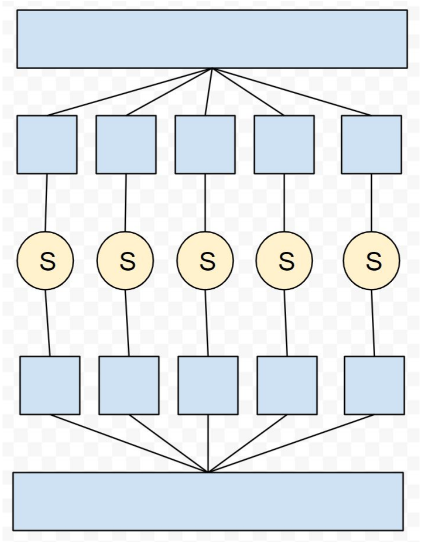
- Few rounds with simple and fast algebraic functions
- One lookup layer (BARS)



Field mismatch

The problem:

- Lookup is done on the smaller domain
- Field is not a product of subdomains
- What happens at the last step?

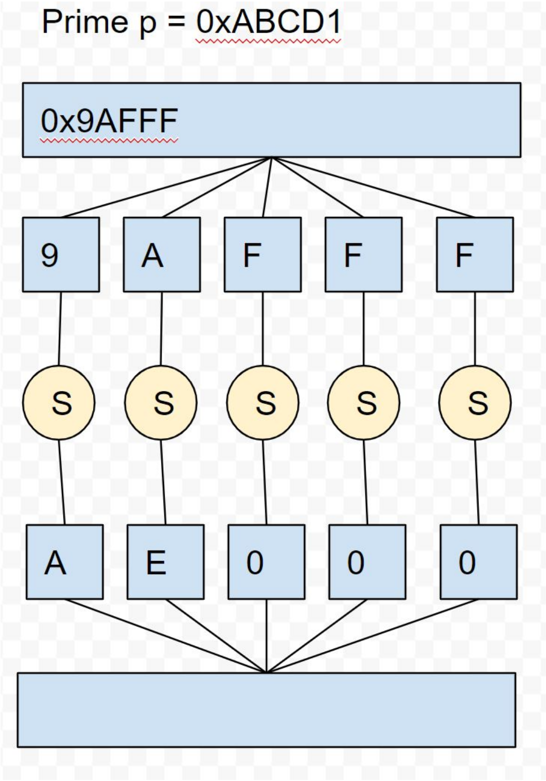


Field mismatch

The problem:

- Lookup on small domain
- Field is bigger

Example overflow:



Small fields

Goldilocks $p = 2^{64} - 2^{32} + 1$

Multiplication mod p is fast on x86: regular multiplication and a few shifts

Field elements from 0 to p-1=0xFFFFFFFF00000000

Small fields

Goldilocks $p = 2^{64} - 2^{32} + 1$

Multiplication mod p is fast on x86: regular multiplication and a few shifts

Field elements from 0 to p-1

$p-1$	**0xFFFF**	**0xFFFF**	**0x0000**	**0x0000**
Decomposition	a_1	a_2	a_3	a_4
S-box	$S(a_1)$	$S(a_2)$	$S(a_3)$	$S(a_4)$

Small fields

Goldilocks $p = 2^{64} - 2^{32} + 1$

Multiplication mod p is fast on x86: regular multiplication and a few shifts

Field elements from 0 to p-1

$p-1$	0xFFFF	0xFFFF	0x0000	0x0000
Decomposition	a_1	a_2	a_3	a_4
S-box	$S(a_1)$	$S(a_2)$	$S(a_3)$	$S(a_4)$

Exercise: if S has fixed points 0 and 0xFFFF, overflow does not happen, i.e.

$$S(a_1)2^{48} + S(a_2)2^{32} + S(a_3)2^{16} + S(a_4) < p$$

Designs: Tip5, RC64 (tomorrow at permutation-based crypto).

Sboxes in a less nice field

$$p = 769 = \mathrm{0x301}$$

Can't do:

$$x_1, x_2, x_3 \in Z_{16}; \quad x_i \to f(x_i).$$

Overflows for many cases and almost all f.

Solution

C1	C2	C3
0	$a_0 < 19$	0
1	$a_1 < 19$	0
$\cdots$	$\cdots$	0
18	$a_{18} < 19$	0
19	19	0
$\cdots$	$\cdots$	0
24	24	0
25	25	1
26	26	1
$\cdots$	$\cdots$	1
29	29	1

Native		Circuit	
p=769		**p = 30x25+19**	
Input x		Input x	
Decomposition	$x_2 = x \bmod 25$ $x_1 = (x - x_2)/25$	Variables	x_1, x_2
		Equations	$x = 25x_1 + x_2$
Sbox (left)	$y_i \leftarrow S(x_i)$	Lookups	$(x_1, y_1, *) \in T$ $(x_2, y_2, 0) \in T$
Composition	$y = 25y_1 + y_2$	Variables	y_1, y_2
		Equations	$y = 25y_1 + y_2$

General rule

$$p - 1 < s_1 s_2 \cdots s_l$$

Embed

$$x \in \mathbb{F}_p \to (x_1, x_2, \ldots, x_l)$$
$$x = x_1 \cdot s_2 s_3 \cdots s_l + x_2(s_3 s_4 \cdots s_l) + \ldots x_{l-1} s_l + x_l$$

How to guarantee no overflow?

General rule

$$p - 1 < s_1 s_2 \cdots s_l$$

Embed

$$x \in \mathbb{F}_p \to (x_1, x_2, \dots, x_l)$$
$$x = x_1 \cdot s_2 s_3 \cdots s_l + x_2(s_3 s_4 \cdots s_l) + \dots x_{l-1} s_l + x_l$$

How to guarantee no overflow?

$$y = S(x_1) \cdot s_2 s_3 \cdots s_l + S(x_2)(s_3 s_4 \cdots s_l) + \dots S(x_{l-1}) s_l + S(x_l)$$

We have y<p if $(x_i \le a_i) \implies S(x_i) \le a_i$ where

$$p - 1 = a_1 \cdot s_2 s_3 \cdots s_l + a_2(s_3 s_4 \cdots s_l) + \dots a_{l-1} s_l + a_l$$

Native and circuits

2 parts:

- Native computation
  - Decompose
  - Apply sboxes
  - Compose
- Circuit proof
  - Proof of decomposition
  - Proof of sboxes
  - Proof of composition

Native		Circuit	
p=769		**p = 30x25+19**	
Input x		Input x	
Decomposition	$x_2 = x \bmod 25$ $x_1 = (x - x_2)/25$	Variables	x_1, x_2
$x = 25x_1 + x_2$		Equations	
Sbox	$y_i \leftarrow S(x_i)$	Lookups	$(x_1, y_1, *) \in T$ $(x_2, y_2, 0) \in T$
Composition		Variables	y_1, y_2
$y = 25y_1 + y_2$		Equations	

Sbox design

2 parts:

- Native computation
- Circuit proof

Circuit proof vs native:

- *Completeness*: every valid computation should be represented by circuit witness
- *Soundness*: every valid witness implies a valid computation

Soundness and completeness

Inversion: `if x!=0 then y=1/x`

```
else y=0
```

In circuit

$$
\begin{aligned}
xy &= z \\
z(1-z) &= 0 \\
(1-z)(x-y) &= 0
\end{aligned}
$$

Sound?

Complete?

Questions?