Groth16 still lives

Exploring the trade-offs of modern zk-proof systems

François Garillot, Mysten Labs

Who am I?

- A cryptographic engineer, working on Sui,
- Sui is:
    - A high-throughput L1 blockchain and smart contracts platform,
    - With a low-latency fast lane for transactions that do not need to go through consensus,
- One question on my mind:

"What does it mean to be a zk-friendly blockchain?"

Some Zero-Knowledge primitives in Sui

- Developing the fastcrypto library:

  https://github.com/mystenlabs/fastcrypto

- Standing on the shoulders of giants,
- Curated and benchmarked fast implementations,
- Offers implementations of:
  - Bulletproofs,
  - Pedersen hashing
  - A Groth16 verifier on BLS12-381
- Exposed as Move primitives on Sui.

This talk

- More questions than answers,
- Inspired by Georgios' talk at ZKSummit ZK0x04, right after SNARKtember (Marlin, Plonk, Halo, DARK, …),
- This talk gave a tour of the recent changes, a description of the trade-offs, and asked a few questions
- Came at the right moment to enshrine a modern era:
  - *No more circuit-specific setups,*
  - *The birth of new recursion methods*

Non-Interactive Argument of Knowledge

Public arithmetic circuit: $C(x, w) \to \mathbb{F}$

x public inputs in $\mathbb{F}^n$,

w secret witness in $\mathbb{F}^m$

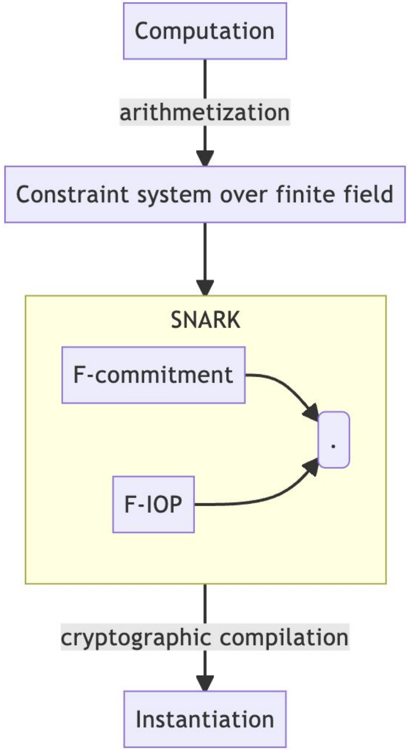
- *Preprocessing*: $S(C) \to (pp, vp)$ public parameters
- *Proof* for $R_C : \{(x, w), C(x, w) = 0\}$
- *Prover*: $pp, x, w \mapsto \pi$
- *Verifier*: $\pi, vp, x \mapsto \{\text{accept}, \text{reject}\}$

Split setup -> Universal setup

- *generator / initializer*: produce global parameters depending on the security parameter,
- *indexer*, produces (pp, vp) from the global parameters and the circuit C.

SNARKtember-ish examples: Sonic, Marlin, Plonk.

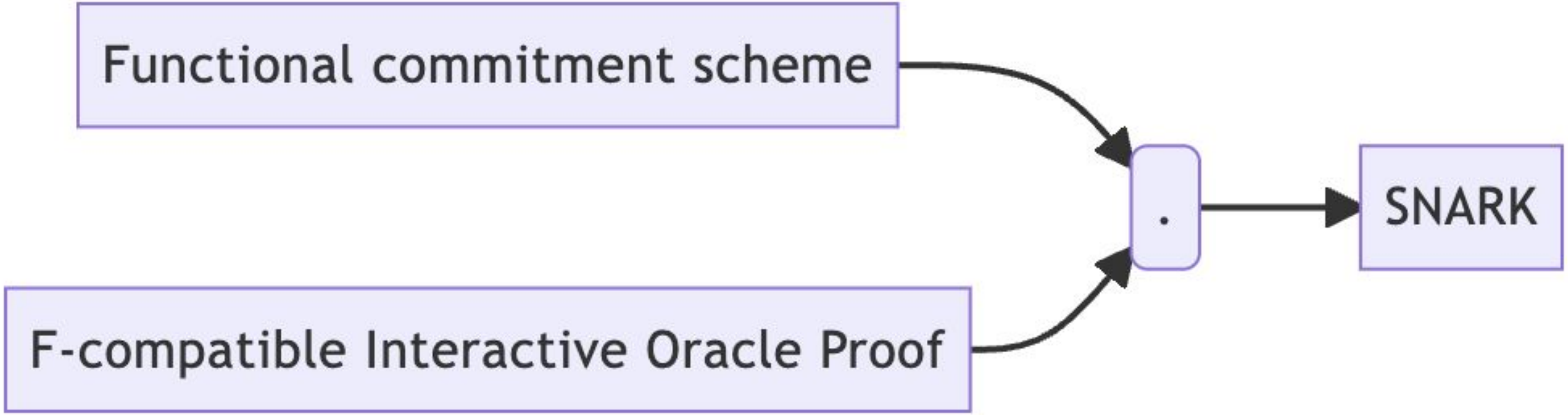
Context



Function-family IOPs (from Boneh):

- The proof scheme picks a family of functions (Polynomials, Multilinear Polynomials, Vectors, IPAs) and an associated *functional commitment.*
- In the later *Interactive Oracle Proof*, the *prover* sends functions from F as oracles to the *verifier*,
- The *verifier* queries these oracles, being assured they are to functions from F,

Function family IOPs



Function family IOPs

Commitments	F-IOP	System
KZG (Poly)	Plonk	Barretenberg, Jellyfish
Bulletproofs (IPA)	Sonic	Halo
Bulletproofs (IPA)	Plonk	Halo2, Plonky, Kimchi
FRI (Poly)	Plonk	Plonky2

F-commitments methods and their implications

- Pairing-based commitment (e.g. KZG) :
  - Universal trusted setup,
  - small proof sizes,
- Discrete-log-based commitment (e.g. Bulletproofs):
  - transparent setup,
  - proof sizes remain small in practice (despite change in asymptotics), but
  - slower verifier,
- Hashing-based :
  - proof sizes get much larger,
  - transparent system.

PLONK and its Arithmetic

Some places that run a variant of PLONK in production:

Zcash*, Polygon Zero (formerly known as Mir Protocol), Aztec Network, Dusk, MatterLabs* (zksync), Astar, Mina*, Anoma, Expresso Systems.

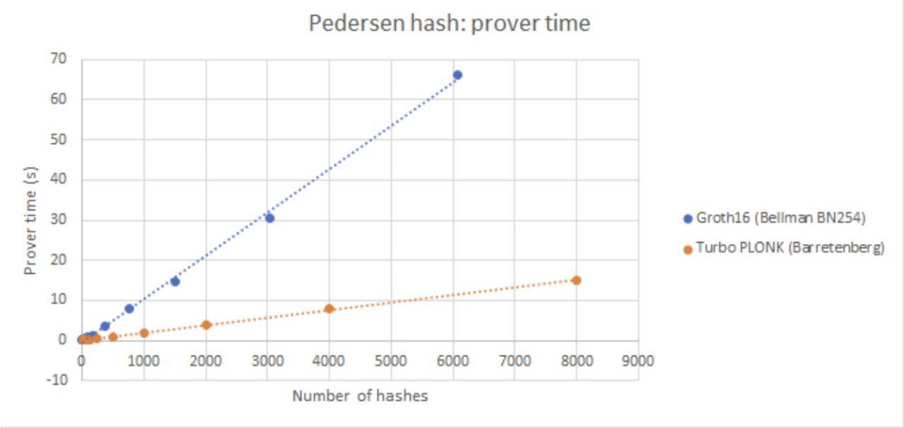
Some places that run a variant of Groth16:

Aleo, Filecoin, Celo.

(apologies if you are not mentioned, keeping track is hard)

Plonk and its arithmetic

- TurboPlonk: Plonk with custom gates, (here for custom scalar multiplication)



- UltraPlonk / PLONKup : Plonk with *lookup arguments*

Plonk with lookup arguments, for a table of size N

- plookup, Caulk, Caulk+, flookup, Baloo
- m the number of lookups, N the size of the table

Scheme	Preprocessing	Proof size	Prover work		Verifier work
			group	field	
Plookup [GW20]	–	$5\mathbb{G}_1, 9\mathbb{F}$	$O(N)$	$O(N \log N)$	2P
Halo2 [BGH20]	–	$6\mathbb{G}_1, 5\mathbb{F}$	$O(N)$	$O(N \log N)$	2P
Caulk [ZBK+22]	$O(N \log N)$	$14\mathbb{G}_1,\ 1\mathbb{G}_2,\ 4\mathbb{F}$	$15m$	$O(m^2 + m \log(N))$	4P
Caulk+ [PK22]	$O(N \log N)$	$7\mathbb{G}_1, 1\mathbb{G}_2, 2\mathbb{F}$	$8m$	$O(m^2)$	3P
Flookup [GK22]	$O(N \log^2 N)$	$7\mathbb{G}_1, 1\mathbb{G}_2, 4\mathbb{F}$	$O(m)$	$O(m \log^2 m)$	3P
This work: Ba*loo*	$O(N \log N)$	$12\mathbb{G}_1, 1\mathbb{G}_2, 4\mathbb{F}$	$14m$	$O(m \log^2 m)$	5P

Plonk with lookup arguments: performance

VERI-ZEXE : improves on ZEXE (see also Aleo)

	Setup		Execute^L			
			$\mathcal{R}_\Phi$ (outer circuit)			
Tx. Dim.	**Time** (s)	**SRS size** (MB)	**Constraints**	**Prover** (s)	**Time** (s)	**Memory** (GB)
snarkVM			R1CS			
2×2	176.8	5,254.2	4,235,068	138.5	151.4	16.6
3×3	246.0	7,056.6	6,330,496	202.7	223.0	20.5
4×4	370.1	10,454.9	8,447,588	293.2	321.1	27.1
verizexe			UltraPlonk			
2×2	11.8	33.1	87,176	13.1	16.9	6.5
3×3	18.4	66.2	126,076	24.7	29.2	8.5
4×4	19.1	66.2	141,492	24.8	32.4	9.1

Hashing

Around SNARKtember: new functions

- Poseidon and
- Rescue

were fighting for the Starkware hash challenge. Rescue won.

Yet, today, systems that use a variant of the Poseidon hash:

Aleo, Anoma, Dusk, Filecoin, Penumbra, Polygon Zero, zkSync, Mina

But has Poseidon won?

Hashing with lookup arguments

- Halo2: Sinsemilla
- Reinforced Concrete

	Performance			
	Zero knowledge			Native
	R1CS eq-s	Plookup reg. gates	Area-degree product	(μs)
Poseidon	243	633	9495	19
Rescue	288	480	7200	480
Rescue-Prime	252	420	6300	415
Feistel-MiMC	1326	1326	19890	38
Griffin	96	186	2790	115
Neptune	228	1137	17055	20
SHA-256	27534	3000	60000	0.32
Blake2s	21006	2000	40000	0.21
Pedersen hash	869		13035	54
SINSEMILLA		510	1530	137
Reinforced Concrete-BN/BLS	-	**378**	**5670**	**3.4**
Reinforced Concrete-ST	-	**360**	**5400**	**1.09**

Advantages:

- performance
- Reasoning about security

Approaches that tremendously help

- Penumbra's parameter generation module
- SAFE API (Khovratovich, Aumasson, Quine)

But what about recursion?

Before Halo, we could do:

- Cycles of pairing-friendly elliptic curves (Coda): encode the verifier in a proof statement,
- Layer-1 recursion (then Zexe, now SnarkVM),

Halo: *delay the verification of commitment openings, and encode them in the statement being proven. On the next layer, we can amortize the verification of those*

Halo -> PCD from accumulation Schemes -> PCD w/o Succint arguments -> Nova

The upshot: recursion uses smaller curves, some not pairing-friendly.

Arithmetization: the next frontier?

Approaches to arithmetization:

- DSL (e.g. Circom, SnarkJS, Noir, Leo, Alucard)
- IR (e.g. VampIR)
- The "embedding", or "direct" approach (e.g. Geb, Lurk)

And the Von Neuman approach (emulating a machine):

- All the zk-VMs (see e.g. the Anoma benchmarks for some (not E-)VMs)

Here, R1CS still dominates the majority of the tooling.

Hardware acceleration: another frontier?

- Two hard operations to parallelize: MSM and FFT,

- HyperPlonk : Plonk defined on the boolean hypercube,
- Nova: requires no FFTs

Tooling for a SNARK system

- *Commitment* to a *family of functions*,
  - Affects proof size, trusted setup, prover and verifier time
- Choosing a *corresponding IOP* (affects the arithmetization)
- For lookup arguments, any polynomial commitment should do (?)
- Choose a hash function
  - And a good implementation thereof!
- Choose a recursion scheme
  - Universal, updateable setups have a concurrency issue,
  - Most recursion schemes in practice use a transparent scheme for commitments,
  - "Power to the people" may help coordinate an update for L1 recursion.