Testudo

Matteo Campanelli	Protocol Labs
Nicolas Gailly	Lagrange (work done at Protocol Labs)
Rosario Gennaro	CUNY (work done at Protocol Labs)
Philipp Jovanovic	University College of London
Mara Mihali	Aztec (work done at Protocol Labs)
Justin Thaler	A16Z

What is a SNARK?

Prover

Let F be a 2-input function, and x a public input

∃ w such F(x,w)=y

Verifier

Π

Properties of SNARKs

- If computing **F** takes **T** prover should be **~O(T)**
- Π should be short
- Verifier should run in **o(T)**

Requirements for practical SNARKs

- linear or quasilinear prover
- logarithmic proof size (or smaller)
- logarithmic verifier (or smaller)

In many cases:

- verification happens on-chain so every operation matters 💰

Requirements for practical SNARKs

- linear or quasilinear prover
- logarithmic proof size (or smaller)
- logarithmic verifier (or smaller)

In many cases:

- verification happens on-chain so every operation matters 💰
    - **Groth16** (an improved version of **G**GPR13) still remains
        - the cheapest proof system to verify with constant number of operations
        - smallest proof size: constant
            - < 200 bytes
            - exact size depends on the curve

But we’re not fully happy with Groth16… 😕

- *function-specific* trusted setup
  - any updates to the circuit require a new ceremony *which we want to avoid at all costs*
- trusted setup is linear in the size of the circuit
  - for large circuits ⇒ requires **GBs** of storage
  - **limits** the *maximum* size of a circuit
- Groth16 uses FFTs on the prover’s side
  - hard to parallelise
  - put strain on the prover for large circuits due to memory requirements

On the other hand

- *Transparent SNARKs*
  - Have no trusted setup (Spartan, Starks)
- *Universal setup SNARKs*
  - Have a trusted setup that depends only on the size of the circuit
  - Plonk, Hyperplonk, etc.
- However those have longer proofs and verification time than Groth16
  - But usually a faster prover

Let's build a SNARK with the same proof size and verification cost as Groth16, but also a *universal* and *small* trusted setup!

General idea of **Testudo**

- Prover wants to prove that ***F(x,w)=y***
  - Run a fast transparent SNARK **P** that the prover knows ***w***
    - to get a long proof ***Π***
  - Then run a Groth16 proof that the prover knows a correct ***Π***
    - That satisfies the verification algorithm ***V(Π)=1***
    - This yields a shorter (constant) proof *Π'*
- Trusted setup is short and universal
  - The computation over which we run Groth16 is the verification algorithm of the transparent SNARK
    - Which is logarithmic in the size of ***F***
    - And works for any ***F***
- Ideas inspired by other 2-level recursion works [Belling et al. '22]
  - Independently ZKBridge developed a similar approach
  - Specific computations, we are the first to use it for generic computations

Technical contributions

- We started with Spartan as the underlying transparent SNARK
  - But that has $O(\sqrt{N})$ proofs and we wanted shorter proofs to feed into Groth16
- Changed Spartan to use a new polynomial commitment for the witness
  - Started with the PST'13 polynomial commitment for multivariate polynomials
    - Log-size proofs
    - But that requires a $O(N)$ trusted setup (where N is the size of the R1CS)
      - Too long for us
- Modified PST'13 to achieve $O(\sqrt{N})$ trusted setup
  - Using ideas from the MIPP protocol [B+21]
- Implemented this over a cycle of curve to achieve greater efficiency

Refresher: rank-1 constraint system (R1CS)

- R1CS generalises the circuit satisfiability problem
- Consider finite field $\mathbb{F}$, matrices A, B, C and vector z
  - A, B, C model the actual circuit and are public
  - z is the private witness
- An R1CS instance is satisfiable, iff the following relation holds

$$(A \cdot z) \circ (B \cdot z) = C \cdot z$$

Why R1CS

- Introduced as *Quadratic Span Programs* in **G**GPR13
  - A form of arithmetization of generic computations
  - Groth16 and Spartan uses R1CS to generate proofs
- There are alternative ways to encode a computation
  - E.g. higher order constraints a la Plonk
- Motivation came from improving the SNARK used by the Filecoin protocol
  - Filecoin relies on storage providers proving that they are committing a certain amount of memory to the network
  - Uses a large *Proof of Space* too long to post on chain
  - The Groth16 SNARK is used to compress it to constant size
    - Run over an R1CS of size about 2^30
- Building a new R1CS based SNARK would give us a “drop-in” replacement for Groth16 in Filecoin

Spartan

- Assume we have a circuit $C = |N| = 2^n$
- Recall $(A \cdot z) \circ (B \cdot z) = C \cdot z$ tells whether R1CS instance is satisfiable

Spartan

- Assume we have a circuit $C = |N| = 2^n$
- Recall $(A \cdot z) \circ (B \cdot z) = C \cdot z$ tells whether R1CS instance is satisfiable
- Represent ***matrices as functions***

  $A, B, C : \{0,1\}^{logn} \times \{0,1\}^{logn} \rightarrow \mathbb{F}$ and $z : \{0,1\}^{logn} \rightarrow \mathbb{F}$

  $\downarrow$

$A(i,j) = A_{i,j}$

Spartan

- Assume we have a circuit $C = |N| = 2^n$
- Recall $(A \cdot z) \circ (B \cdot z) = C \cdot z$ tells whether R1CS instance is satisfiable
- Represent ***matrices as functions***

  $A, B, C : \{0,1\}^{logn} \times \{0,1\}^{logn} \to \mathbb{F}$ and $z : \{0,1\}^{logn} \to \mathbb{F}$
- Using MLE extensions over the boolean hypercube

$$F(x) = \sum_{y \in \{0,1\}^n} \tilde{A}(x,y) \cdot \tilde{z}(y) \times \sum_{y \in \{0,1\}^n} \tilde{B}(x,y) \cdot \tilde{z}(y) - \sum_{y \in \{0,1\}^n} \tilde{C}(x,y) \cdot \tilde{z}(y)$$

Spartan

- Assume we have a circuit $C = |N| = 2^n$
- Recall $(A \cdot z) \circ (B \cdot z) = C \cdot z$ tells whether R1CS instance is satisfiable
- Represent ***matrices as functions***

  $A, B, C : \{0,1\}^{logn} \times \{0,1\}^{logn} \to \mathbb{F}$ and $z : \{0,1\}^{logn} \to \mathbb{F}$
- Using MLE extensions

$$F(x) = \sum_{y \in \{0,1\}^n} \tilde{A}(x,y) \cdot \tilde{z}(y) \times \sum_{y \in \{0,1\}^n} \tilde{B}(x,y) \cdot \tilde{z}(y) - \sum_{y \in \{0,1\}^n} \tilde{C}(x,y) \cdot \tilde{z}(y)$$

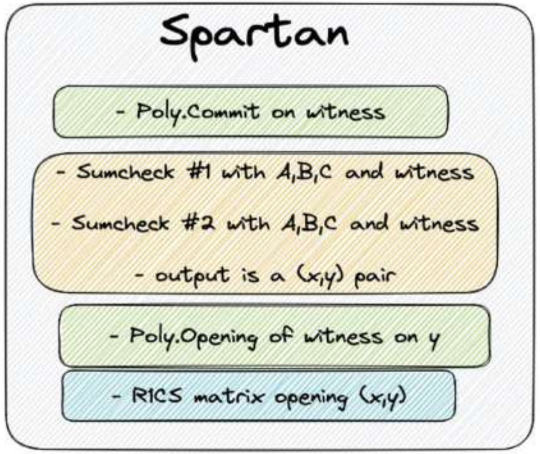
- Then, we can define the following sumcheck instance

$$0 = \sum_{x \in \{0,1\}^n} \tilde{eq}(x,t) \cdot F(x)$$

↓

1 if $x = t$ otherwise 0

Spartan



- **V** ask for evaluation at (r_y, r_x)
- P
  - commits to witness $z(y)$ with polynomial commitment scheme
    - proof size and verification between and $O(\sqrt{n})$ and $O(\log(n))$
      - depends on whether we allow a trusted setup
  - uses *computation commitments* for $\tilde{A}(x,y), \tilde{B}(x,y), \tilde{C}(x,y)$
    - generating this is part of the public setup
    - commitment exploits the sparsity of the R1CS matrices
      - ensures prover time remains quasilinear (at most $O(n \log n)$)
  - responds with evaluations and proof of opening for the 4 polynomials

PST13

$p(x_1, \ldots, x_n)$ - multilinear polynomial with n where $n = \log N = \log |C|$

Trusted Setup:

$t_1, ..., t_n$ and CRS is N elements $g^{\chi_i(t)}$ where $N = 2^n$

Commitment

$$Com(p) = g^{p(\vec{t})} = C$$

Opening

for $\vec{a} = [a_1, ...a_n]$ and $y = p(\vec{a})$

$p(\vec{x}) - y = \sum_i (x_i - a_i) q_i(\vec{x})$

proof is $\vec{w} = [w_1 \ldots w_n]$ where $w_i = g^{q_i(\vec{t})}$

Verification

$$e(Cg^{-y}, g) = \Pi_i e(g^{t_i - a_i}, w_i)$$

PST13

$p(x_1, \dots, x_n)$ - multilinear polynomial with n where $n = \log N = \log |C|$

Trusted Setup:

$t_1, ..., t_n$ and CRS is N elements $g^{\chi_i(t)}$ where $N = 2^n$

Commitment

$$Com(p) = g^{p(\vec{t})} = C$$

Opening

for $\vec{a} = [a_1, ...a_n]$ and $y = p(\vec{a})$

$$p(\vec{x}) - y = \sum_i (x_i - a_i) q_i(\vec{x})$$

proof is $\vec{w} = [w_1 \dots w_n]$ where $w_i = g^{q_i(\vec{t})}$

logarithmic proof size

Verification

$$e(Cg^{-y}, g) = \Pi_i e(g^{t_i - a_i}, w_i)$$

PST13

$p(x_1, \ldots, x_n)$ - multilinear polynomial with n where $n = \log N = \log |C|$

Trusted Setup:

$t_1, ..., t_n$ and CRS is N elements $g^{\chi_i(t)}$ where $N = 2^n$

Commitment

$Com(p) = g^{p(\vec{t})} = C$

Opening

for $\vec{a} = [a_1, ...a_n]$ and $y = p(\vec{a})$

$p(\vec{x}) - y = \sum_i (x_i - a_i) q_i(\vec{x})$

proof is $\vec{w} = [w_1 \ldots w_n]$ where $w_i = g^{q_i(\vec{t})}$

Verification

$e(Cg^{-y}, g) = \Pi_i e(g^{t_i - a_i}, w_i)$ logarithmic verification cost

PST

$p(x_1, \ldots, x_n)$ - multilinear polynomial with n where $n = \log N = \log |C|$

Trusted Setup:

$t_1, ..., t_n$ and CRS is N elements $g^{\chi_i(t)}$ where $N = 2^n$

Commitment

back to a O(N) circuit-size dependent trusted setup

$Com(p) = g^{p(\vec{t})} = C$

Opening

for $\vec{a} = [a_1, ...a_n]$ and $y = p(\vec{a})$

$p(\vec{x}) - y = \sum_i (x_i - a_i) q_i(\vec{x})$

proof is $\vec{w} = [w_1 \ldots w_n]$ where $w_i = g^{q_i(\vec{t})}$

Verification

$e(Cg^{-y}, g) = \Pi_i e(g^{t_i - a_i}, w_i)$

PST

$p(x_1, \ldots, x_n)$ - multilinear polynomial with n where $n = \log N = \log |C|$

Trusted Setup:

$t_1, \ldots, t_n$ and CRS is N elements $g^{\chi_i(t)}$ where $N = 2^n$

Commitment

$$Com(p) = g^{p(\vec{t})} = C$$

Opening

for $\vec{a} = [a_1, \ldots a_n]$ and $y = p(\vec{a})$

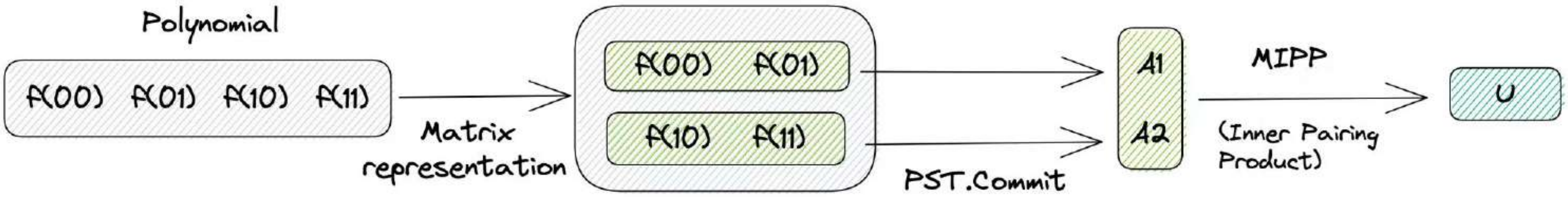
$p(\vec{x}) - y = \sum_i (x_i - a_i) q_i(\vec{x})$ *log|C|* polynomial divisions, the largest one dominates the cost 

proof is $\vec{w} = [w_1 \ldots w_n]$ where $w_i = g^{q_i(\vec{t})}$

Verification

$$e(Cg^{-y}, g) = \Pi_i e(g^{t_i - a_i}, w_i)$$

sqrt-PST

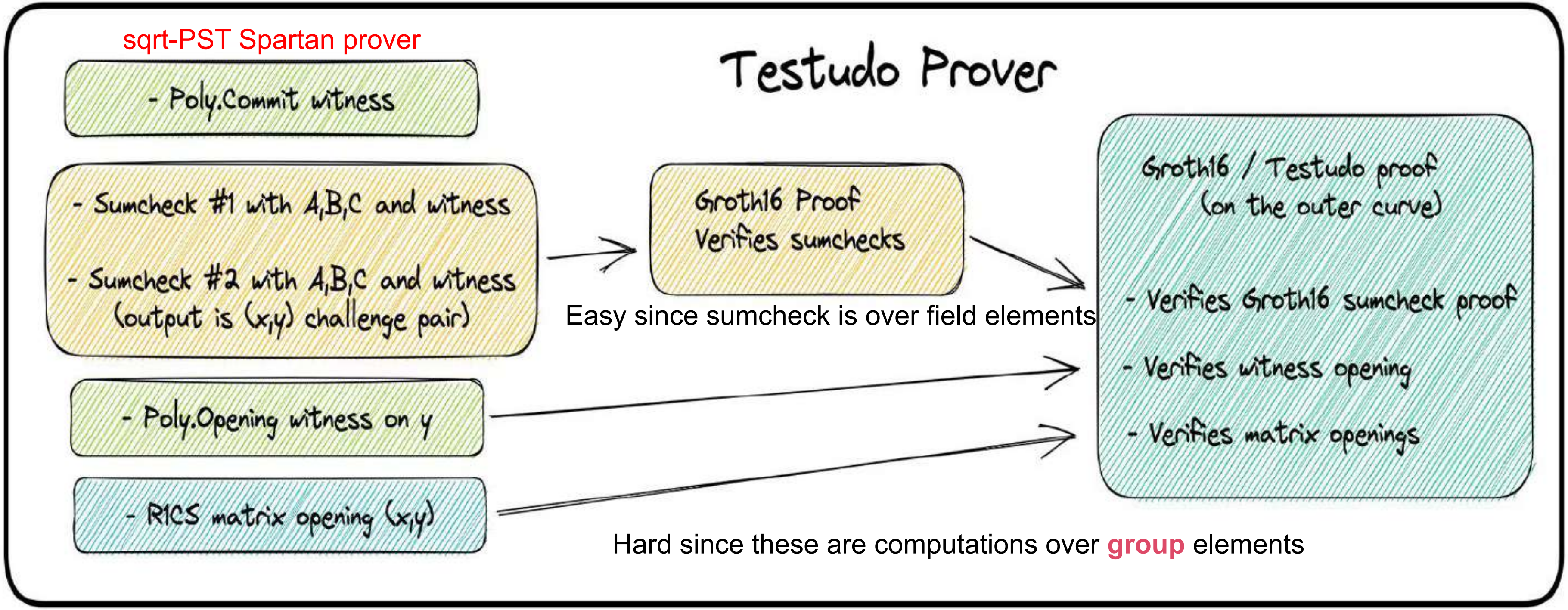


Trusted setup is O(√N) because

- polynomials now have √N terms (PST trusted setup)
- MIPP commits to √N group elements (MIPP trusted setup)
- B+'21 shows how to do this for univariate KZG commitments
  - We generalize it to multivariate PST commitments

Let’s build a SNARK with the same proof size and verification cost as Groth16, but also a *universal* and *small* trusted setup!

Testudo: PST + Spartan + Groth16

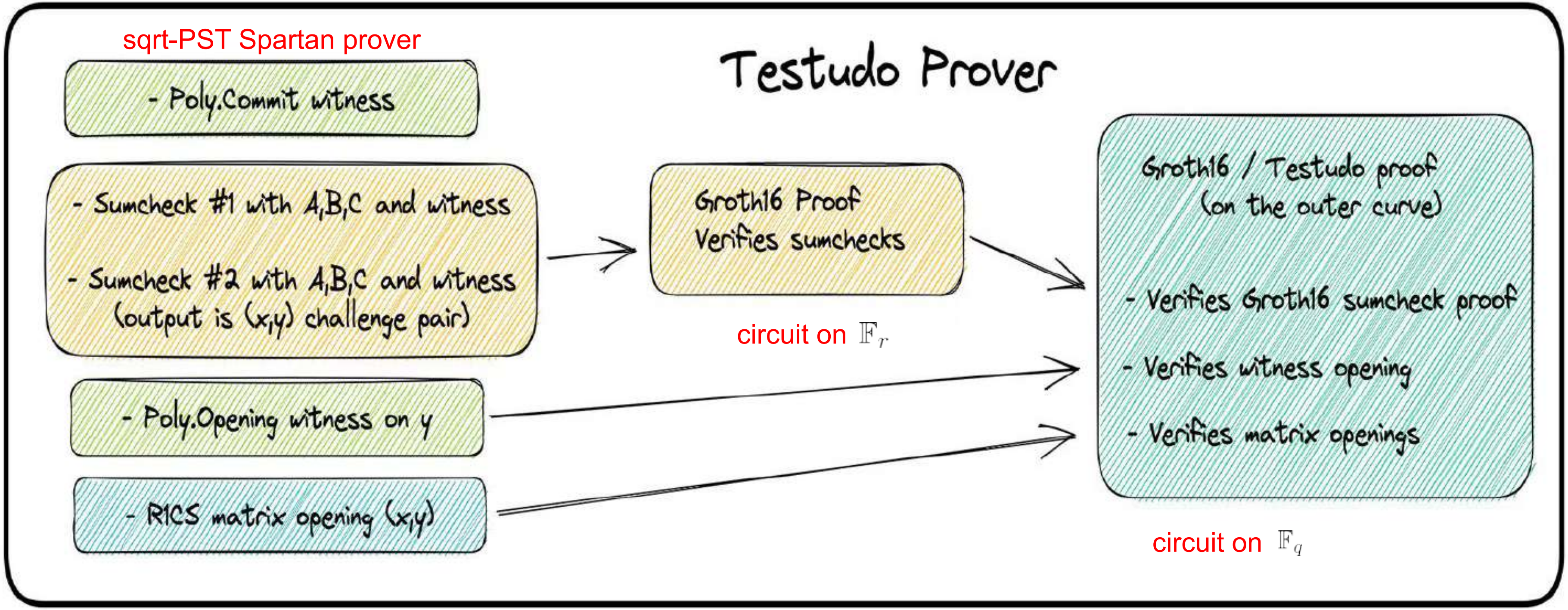


2-chain of Pairing Equipped curve

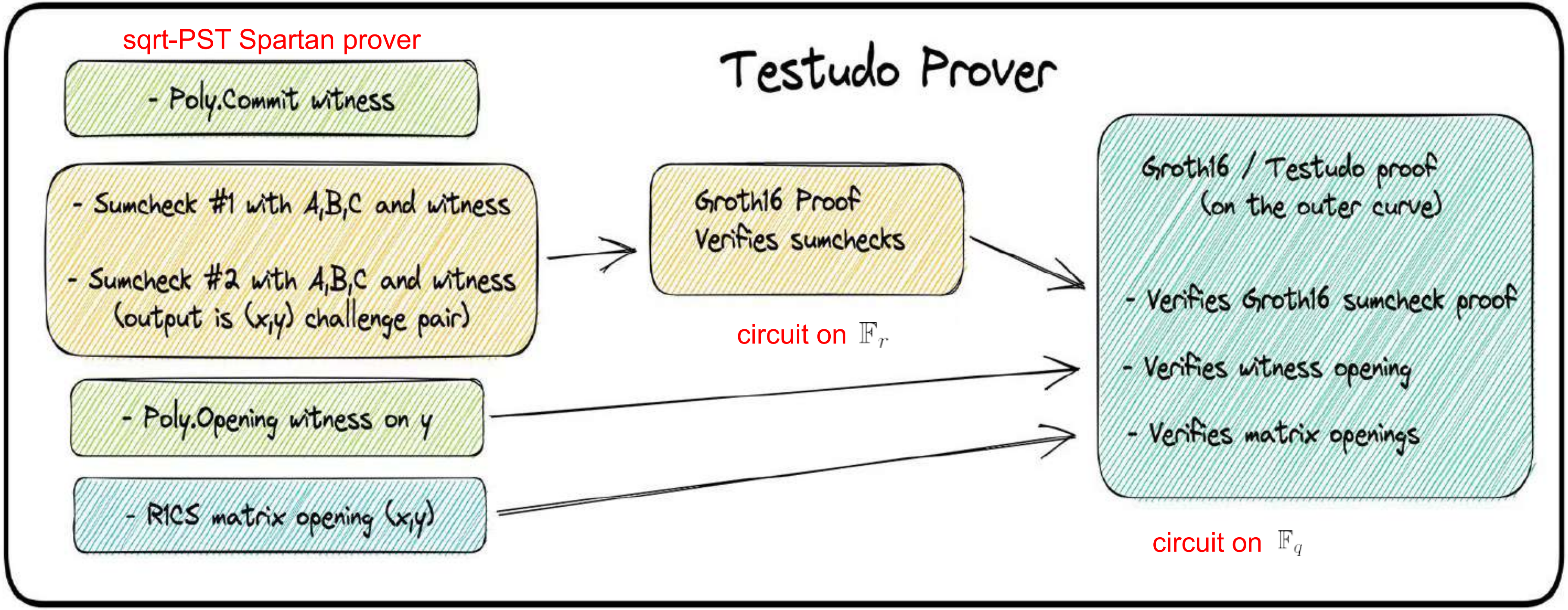
Consider one elliptic curve $E_1(\mathbb{F}_q)$ of order r (so scalar field $\mathbb{F}_r$)

- we can write a circuit with ***only scalar operations*** in $\mathbb{F}_r$ and prove on $\mathbb{F}_q$
- What if we want operations on points in the circuit?
    - if we have a second curve $E_2(\mathbb{F}_t)$ of order q, we can prove circuit with point arithmetic using E_2
- **BLS12-377** and **BW6-761** are such a pair!

Testudo: PST + Spartan + Groth16

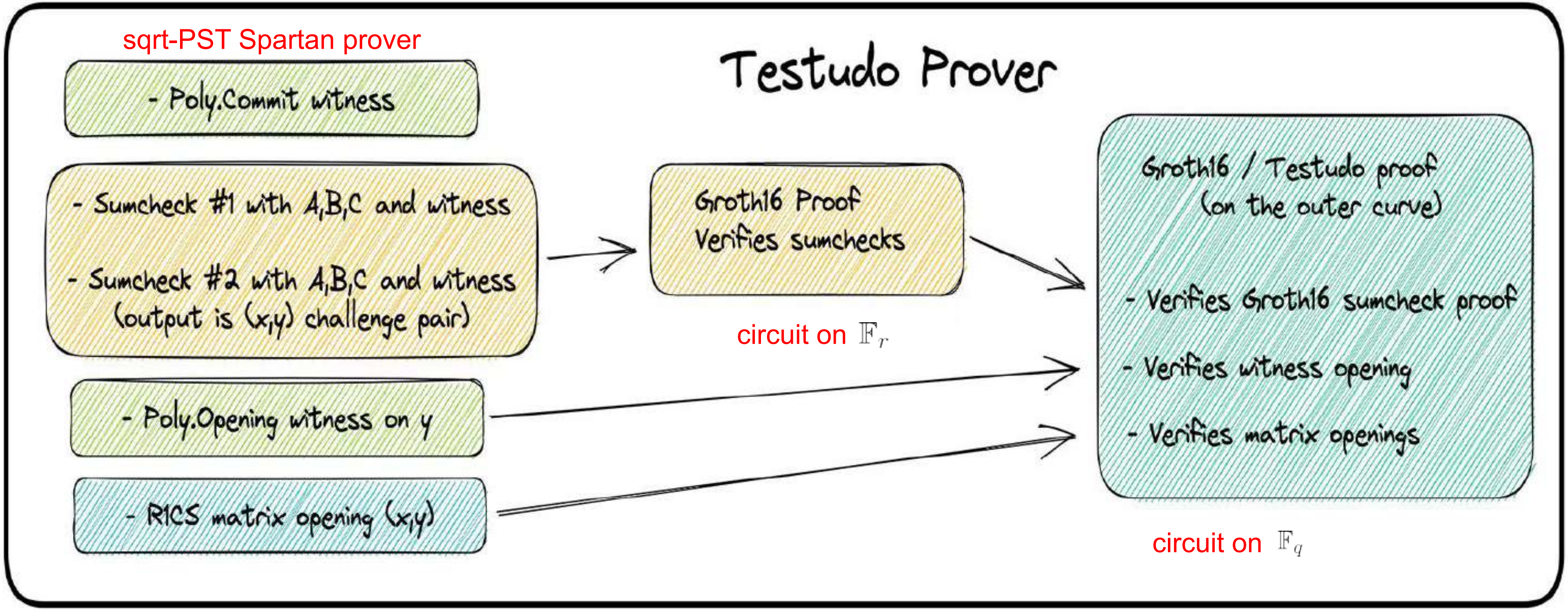


Testudo: PST + Spartan + Groth16



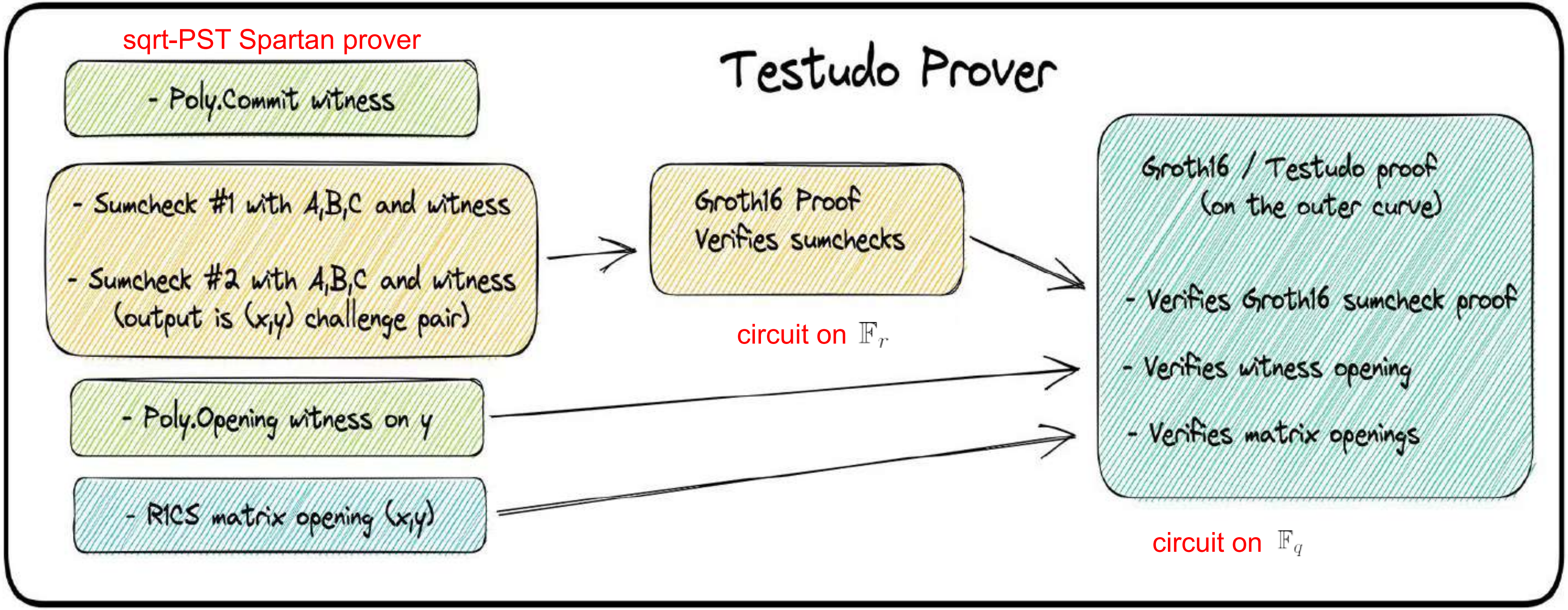
PST trusted setup + setup for ***fixed* circuit** ⇒ universal trusted setup

Testudo: PST + Spartan + Groth16



Circuit for Groth16 proof **< 10 mil constraints** => Groth16 proof generation adds **only few seconds** to proving time

Testudo: PST + Spartan + Groth16



Verifier is just one Groth16 proof verification on $\mathbb{F}_t$

Computation Commitment

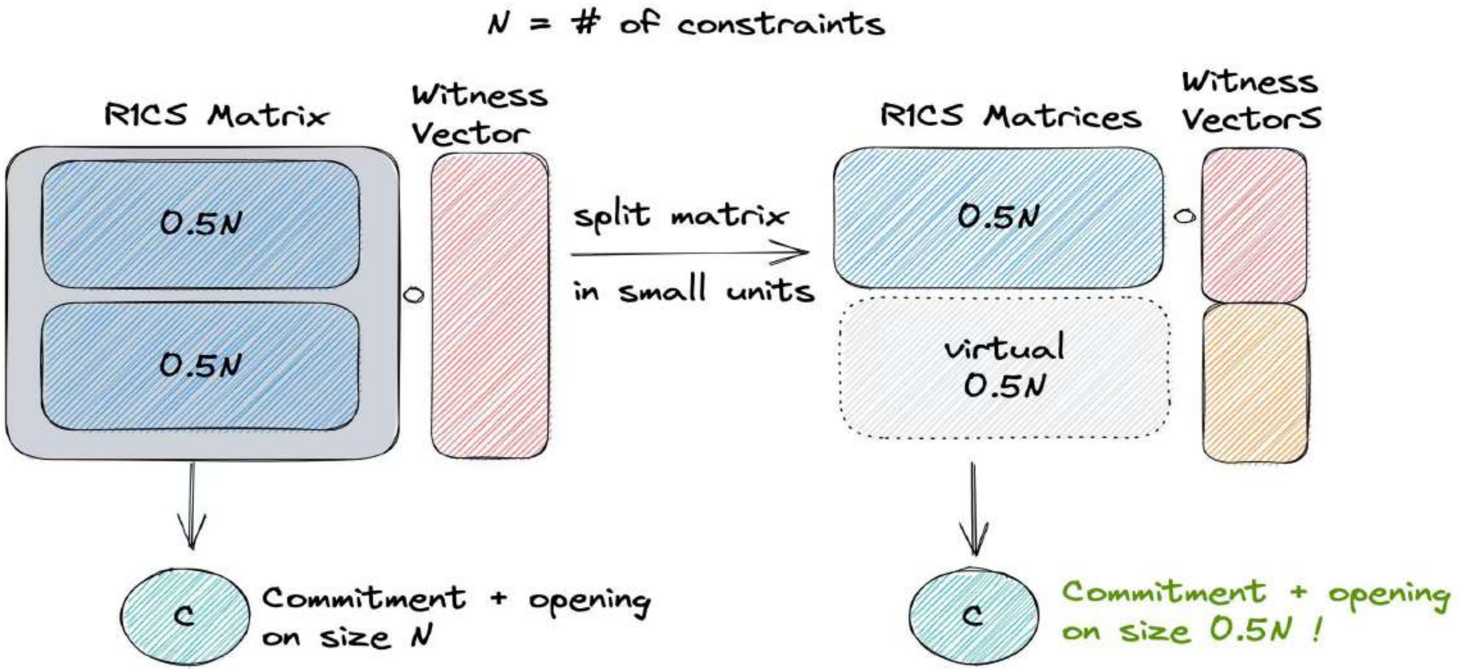
- A lot of prover time is spent on the *computational commitment*

Computational Commitment

- A lot of prover time is spent on the *computational commitment*
- This can be reduced by exploiting **data-parallel computation**

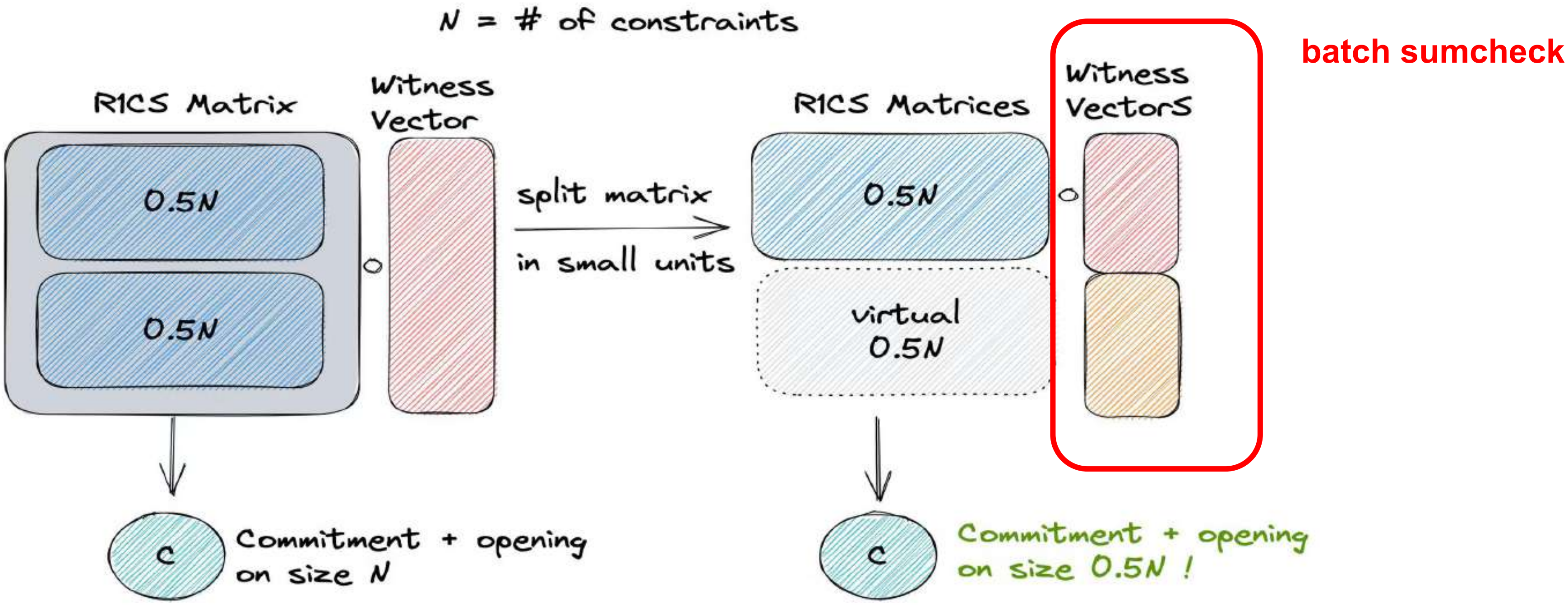
Computational Commitment

- A lot of prover time is spent on the *computational commitment*
- This can be reduced by exploiting **data-parallel computation**



Computational Commitment

- A lot of prover time is spent on the *computational commitment*
- This can be reduced by exploiting **data-parallel computation**



Further Steps

- Explore other ways to improve the computation commitment
  - would trusted setups help here?
- Use Dory instead of PST
  - Slower prover but much much smaller trusted setup
- Finish implementation
- Full comparison with Plonk/Hyperplonk
  - Hard to do meaningfully because of different arithmetizations
- Testudo on BLS12-381
  - more popular curve in the space, Filecoin uses this
  - lacks a “sister” curve that allows the same Groth16 compression
    - existing options do not support FFT
  - option 1: could leave the polynomial openings in the clear for proof
  - option 2: leverage another proof system for the outer circuit


Questions?