Deep dive on Halo 2

Daira Hopwood (@feministPLT)
Ying Tong Lai (@therealyingtong)

https://github.com/daira/halographs
(deepdive.odp)

Outline

- Constructing 2-cycles of elliptic curves
  - Obstacles to finding cycles of curves
  - Complex multiplication to the rescue
  - 2-adicity
- Halo 2
  - Polynomial circuits and PLONK
  - Polynomial commitments
  - Recursion and proof-carrying data
- Optimizations
  - Scalar multiplication
  - Fiat–Shamir and duplex sponges
  - Hashes (Sinsemilla and Rescue).

Elliptic curves

- An elliptic curve in affine form is:
  - a *group* of points (x, y)
  - over a *field of definition* $\mathbb{F}_p$
  - satisfying some equation, say $y^2 = x^3 + ax + b$,
  - with (for this equation) a “point at infinity” as the group identity.
- If the group has a prime order q, we’ll name the curve $E_{p \to q}$.
- Because it’s a group, we can add points, or multiply them by a scalar.
- The *scalar field* is $\mathbb{F}_q$.
- We’ll assume that a proof system using $E_{p \to q}$ efficiently supports circuits with arithmetic over $\mathbb{F}_q$.
- A cycle of elliptic curves is a pair $E_{p \to q}$ and $E_{q \to p}$.

Motivation for cycles

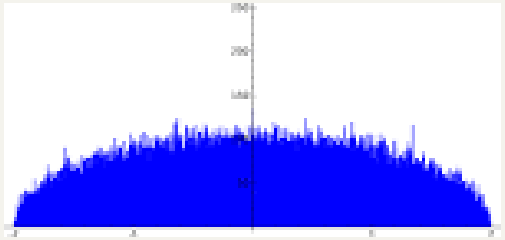
- A proof system using $E_{p \to q}$ directly supports arithmetic over $\mathbb{F}_q$.
- "Wrong-field" arithmetic has an overhead of over 1000 times.
  - This is using the sum of residues method for reduction: Zcash #4093
- Can't we solve this using lookups (e.g. Plookup)?
  - No; lookups help but wrong-field arithmetic probably still has an overhead of 10-100 times.
  - Nothing here conflicts with using lookups for other things in the same proof system.
- As we'll see later, it's not quite sufficient for recursive proofs to just do arithmetic in the field of definition of the other proof system. We'll need two instances of the proof system, on $E_{p \to q}$ and on $E_{q \to p}$.

The Tweedle cycle

- The Halo paper gives a pair of curves:
  - $E_{p \to q} : y^2 = x^3 + 5$ is called Tweedledum.
  - $E_{q \to p} : y^2 = x^3 + 5$ is called Tweedledee.
  - $p = 2^{254} + \texttt{0x38AA1276C3F59B9A14064E200000001}$
  - $q = 2^{254} + \texttt{0x38AA127696286C9842CAFD400000001}$
  - Both have 126-bit Pollard rho security, maximal embedding degree, and 2-adicity ≥ 33.
  - They have cubic endomorphisms (we'll explain what that means).
  - $\gcd(p - 1, 5) = \gcd(q - 1, 5) = 1$
- We're going to explain the construction that found them, and some generalizations of it that allow finding more complicated graphs of curves.

Constructing cycles – the problem

- By the Hasse bound, the order of an elliptic curve over $\mathbb{F}_p$ lies in the range $[\, p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}\,]$. For Tweedle this range is of size $\sim 2^{129}$.
- The Sato–Tate conjecture* concerns the distribution of the order in this range. We don't need to go into detail, but here's a picture:

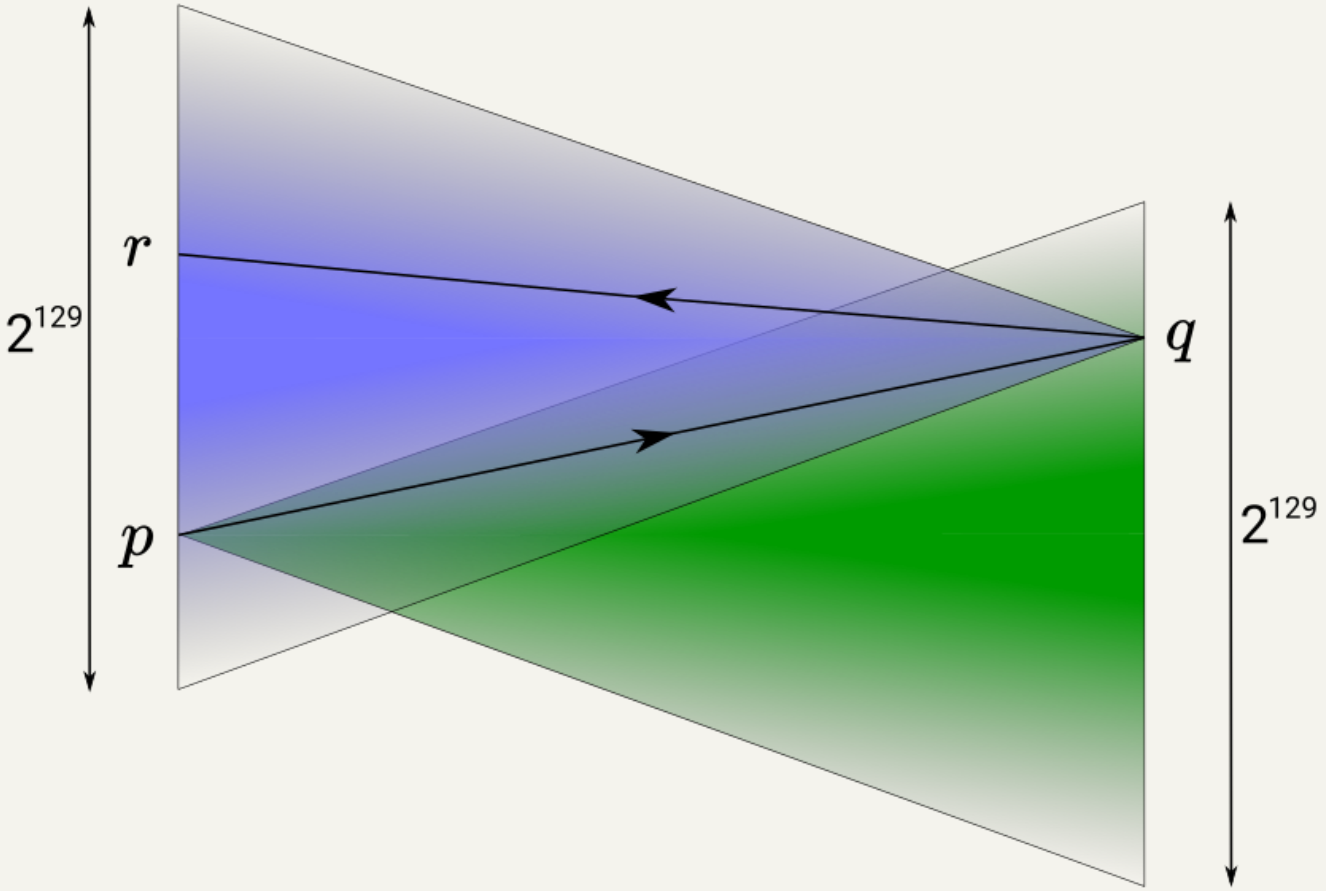


* proven by Taylor et al in 2006.

- That is, the order n could be anywhere in the range.
- And (if n is a prime q) when we construct a curve $E_{q \to r}$ it could also have order anywhere in its Hasse range.
- So, it's exceptionally unlikely that $E_{q \to r}$ has order p.

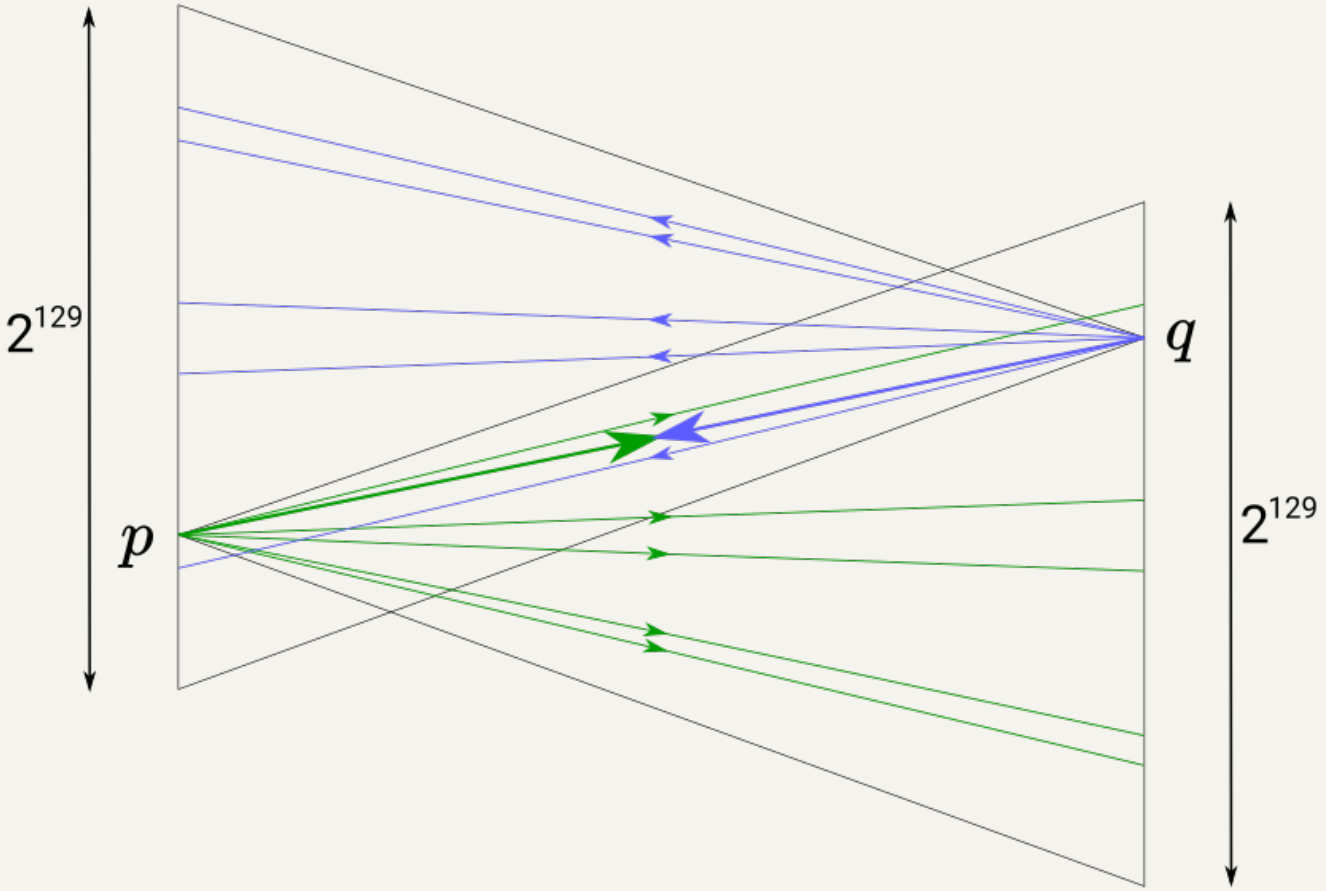
Constructing cycles – the problem

- By the Hasse bound, the order of an elliptic curve over $\mathbb{F}_p$ lies in the range $[\, p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}\,]$. For Tweedle this range is of size $\sim 2^{129}$.



Constructing cycles – the solution

- Suppose we were able to restrict the orders to a small number of possibilities, one of which was guaranteed to form a cycle...



CM curves to the rescue

- Katherine Stange and Joe Silverman noticed that CM curves have precisely that property [SS2011].
- What is a CM curve?
  - This is not intended to be a fully precise definition, just to give intuition and make the concept less mysterious.
- All curves have an “endomorphism ring”. An endomorphism is a group homomorphism (meaning that it preserves the group structure) from the curve group to itself.
  - Why is it a ring? Because you can compose and “add” endomorphisms, and there is an identity endomorphism.
- An example of an endomorphism in an elliptic curve group is scalar multiplication by a constant integer.
- We can think of endomorphisms on elliptic curves as being generalized scalars.

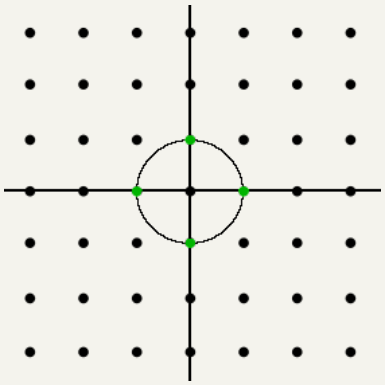
Complex multiplication

- All endomorphisms of an ordinary elliptic curve *over a finite field* are equivalent to scalar multiplication by an integer.
- But an elliptic curve over $\mathbb{F}_p$ is a reduction of a curve with the same equation over the complex numbers $\mathbb{C}$.
- "Complex multiplication" refers to scalar-multiplying points in the curve over $\mathbb{C}$ by complex numbers.
  - E.g. consider $E : y^2 = x^3 + ax$ and let $[i]\ (x, y) = (-x, iy)$. Then $[i^2]\ (x, y) = [-1]\ (x, y) = (x, -y)$, which is the same as applying the $[i]$ map twice.
  - So these scalars are numbers in a complex lattice L, such as $\mathbb{Z}[i]$ or $\mathbb{Z}[\sqrt[3]{1}]$.
- Are yous still with me? If not then don't worry because it all gets a bit simpler again when we map back to finite fields.

Structure of the endomorphism ring

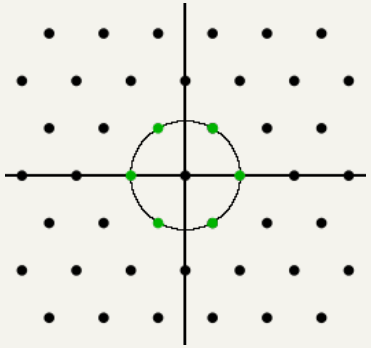
- We said that a generalized scalar can be a number in a complex lattice.
- An elliptic curve over $\mathbb{C}$ has CM if that lattice has more than one dimension (i.e. it's bigger than $\mathbb{Z}$). The lattice structure depends on something called the curve's j-invariant.
- For example, j = 1728 gives a quadratic lattice ($\mathbb{Z}[i]$, also called the Gaussian integers):

  The example we saw on the previous slide is of this case.



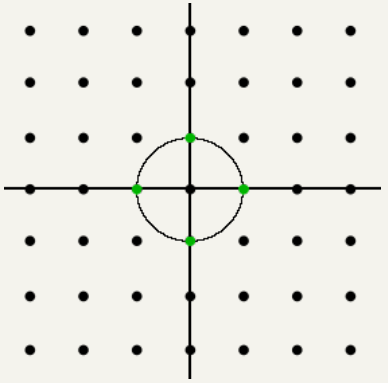
- And j = 0 gives a hexagonal lattice ($\mathbb{Z}[\sqrt[3]{1}]$, also called the Eisenstein integers):

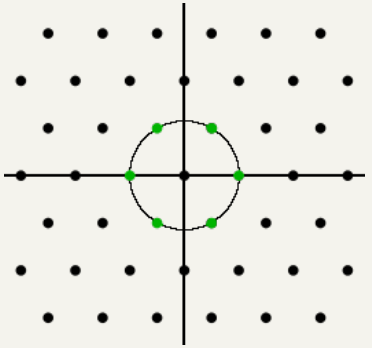
  This case turns out to be really nice for cryptography. This is the case that secp256k1, used in Bitcoin, falls into.



Moving from $E_{\mathbb{C}\to L}$ to $E_{p\to q}$

- The lattice L is a ring, and its units are the elements that have multiplicative inverses. The roots of unity are evenly distributed around the circle of radius 1 in the complex plane.
- The units of L will correspond to possible orders of E_p as we vary the curve coefficients.
- Remember that each point in L is a generalized scalar. But a complex root s can also have a corresponding root s' in $\mathbb{F}_q$.
- Then when we move to the curve over $\mathbb{F}_p$, complex multiplication by s corresponds to ordinary scalar multiplication by the integer s'.
- Also, the units that form a basis for the lattice correspond to endomorphisms that provide “shortcuts” to scalar multiplication in the curve over $\mathbb{F}_p$.





Security of CM curves

- There are only 13 elliptic curves over $\mathbb{C}$ with complex multiplication, up to isomorphism. Whether a curve has CM depends only on its equation.
- Even though CM curves are a small fraction of all elliptic curves, there's no reason to believe they are any weaker. All known pairing-friendly curve constructions produce CM curves, and they are very well-studied.
- Note that isomorphism between curves over $\mathbb{C}$ isn't what determines the security of the Discrete Log Problem; that depends mainly on the sizes of p and q.
- Curves with the same curve equation over different fields are not isomorphic.

But what does it all mean?

- What happens when we map a CM curve back to $\mathbb{F}_p$ is that it can only have a small number of possible orders.
- How many orders it can have is dependent on the number of units in the lattice we saw earlier, which is in turn determined by the j-invariant.
  - For details see On Orders of Elliptic Curves over Finite Fields.
- Curves $y^2 = x^3 + b$ (with no x or x^2 terms) have j-invariant 0.
- These curves are interesting because, over $\mathbb{F}_p$,
  - they have 6 possible orders;
  - they have efficiently computable endomorphisms;
  - it's usually easy to solve the CM norm equation.

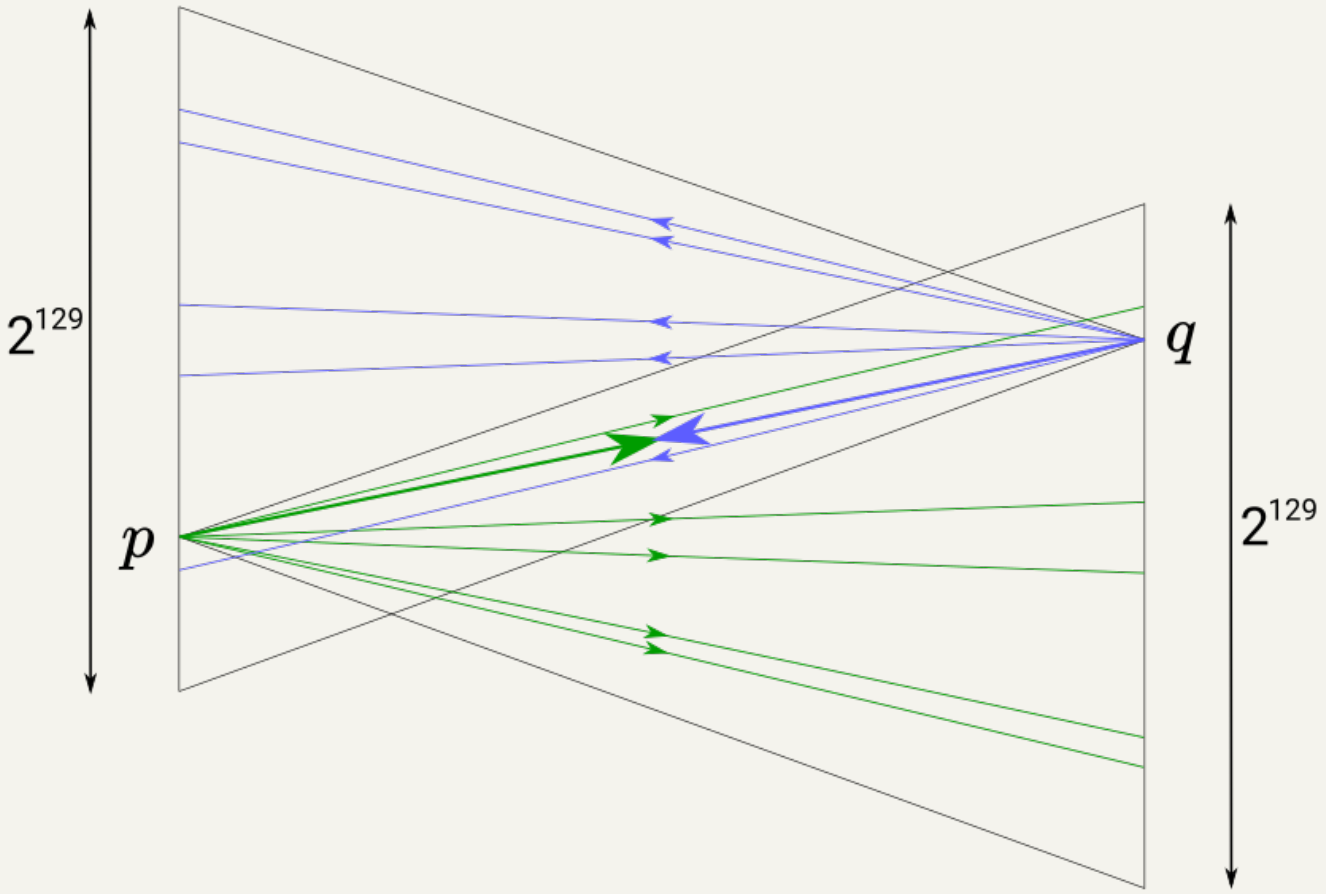
The CM norm equation

- $|D|V^2 = 4p - T^2$
- $|D|$ is the absolute fundamental discriminant (D is negative).
- p is the field size.
- V and T are integers.
- V and T determine the trace t, where $q = p + 1 - t$.
- In fact $\pm T$ are two of the possible traces.
- The Tweedle curves were found using this construction:
  - set $|D| = 3$, pick V and T, then

    $p = \frac{1}{4}(|D|V^2 + T^2)$ and $q = p + 1 - T$.
- The reason for this approach is that we can choose V and T so that *both* curves in the cycle have high 2-adicity.

Constructing cycles – the solution

- For each possible order of $E_{p\to q}$ one of the possible orders of $E_{p\to q}$ is p.
- This is proven by Stange and Silverman, and again with a more elementary proof in the accompanying notes. For more detail on CM see Chenal's thesis.



But why?

- The norm equation helps to explain why CM curves form cycles.

The CM norm equation of $E_{p \to q}$ is $4p = |D|V^2 + T_p^2$ for integers V and T_p.

Theorem: One of the possible orders for E_q is p.

Case for $q = p + 1 \pm T_p$

We have

$$\begin{aligned} 4q &= 4p + 4 \pm 4T_p \\ &= |D|V^2 + T_p^2 \pm 4T_p + 4 \\ &= |D|V^2 + (T_p \pm 2)^2 \end{aligned}$$

This is a norm equation for E_q, with the same $|D|$ and V, and with $T_q = T_p \pm 2$.

In the positive case we have $T_q = T_p + 2$ and $q + 1 - T_q = (p + 1 + T_p) + 1 - (T_p + 2) = p$.

In the negative case we have $T_q = T_p - 2$ and $q + 1 + T_q = (p + 1 - T_p) + 1 + (T_p - 2) = p$.

That is, one of the possible orders of E_q, specifically $q + 1 \mp T_q$, is p in both cases. $\square$

2-adicity

- Protocols that use Lagrange basis, or that need to efficiently multiply polynomials, benefit from $\mathbb{F}^*_{p,\,q}$ having a “large enough” multiplicative subgroup of size z^c. The simplest option is z = 2.
- In other words, we need $p \equiv 1$ and $q \equiv 1 \pmod{2^c}$.

We can freely choose V_p and T_p. So choose $\frac{V_p-1}{2}$ and $\frac{T_p-1}{2}$ as multiples of 2^c. Then

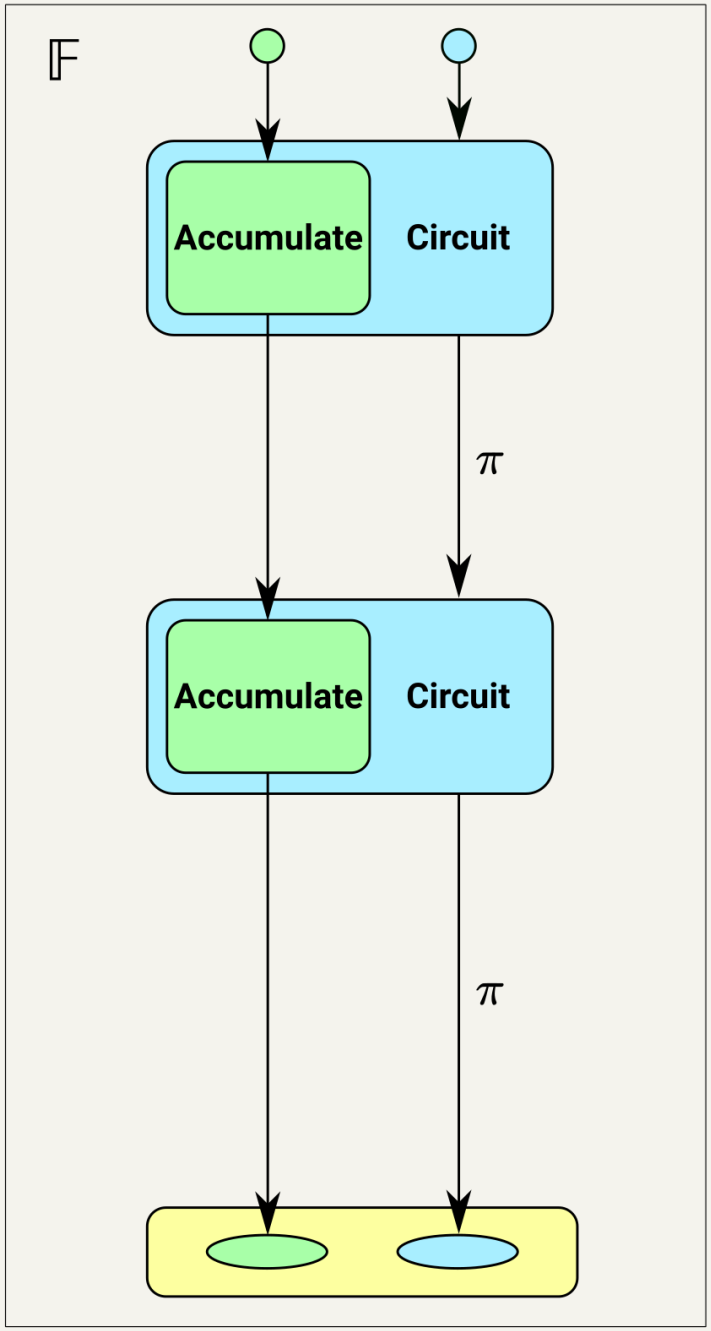
$$\begin{aligned} 4p &= (3(V_p-1)^2 + 6(V_p-1) + 3) + ((T_p-1)^2 + 2(T_p-1) + 1) \\ &= 3(V_p-1)^2 + 6(V_p-1) + (T_p-1)^2 + 2(T_p-1) + 4 \\ p &= 3\left(\frac{V_p-1}{2}\right)^2 + 3\frac{V_p-1}{2} + \left(\frac{T_p-1}{2}\right)^2 + \frac{T_p-1}{2} + 1 \end{aligned}$$

So $p-1$ will be a multiple of 2^c, and so will $q-1$ for $q \in \left\{p+1-T_p,\, p+1-\frac{3V_p}{2}+\frac{T_p}{2}\right\}$.

The story so far

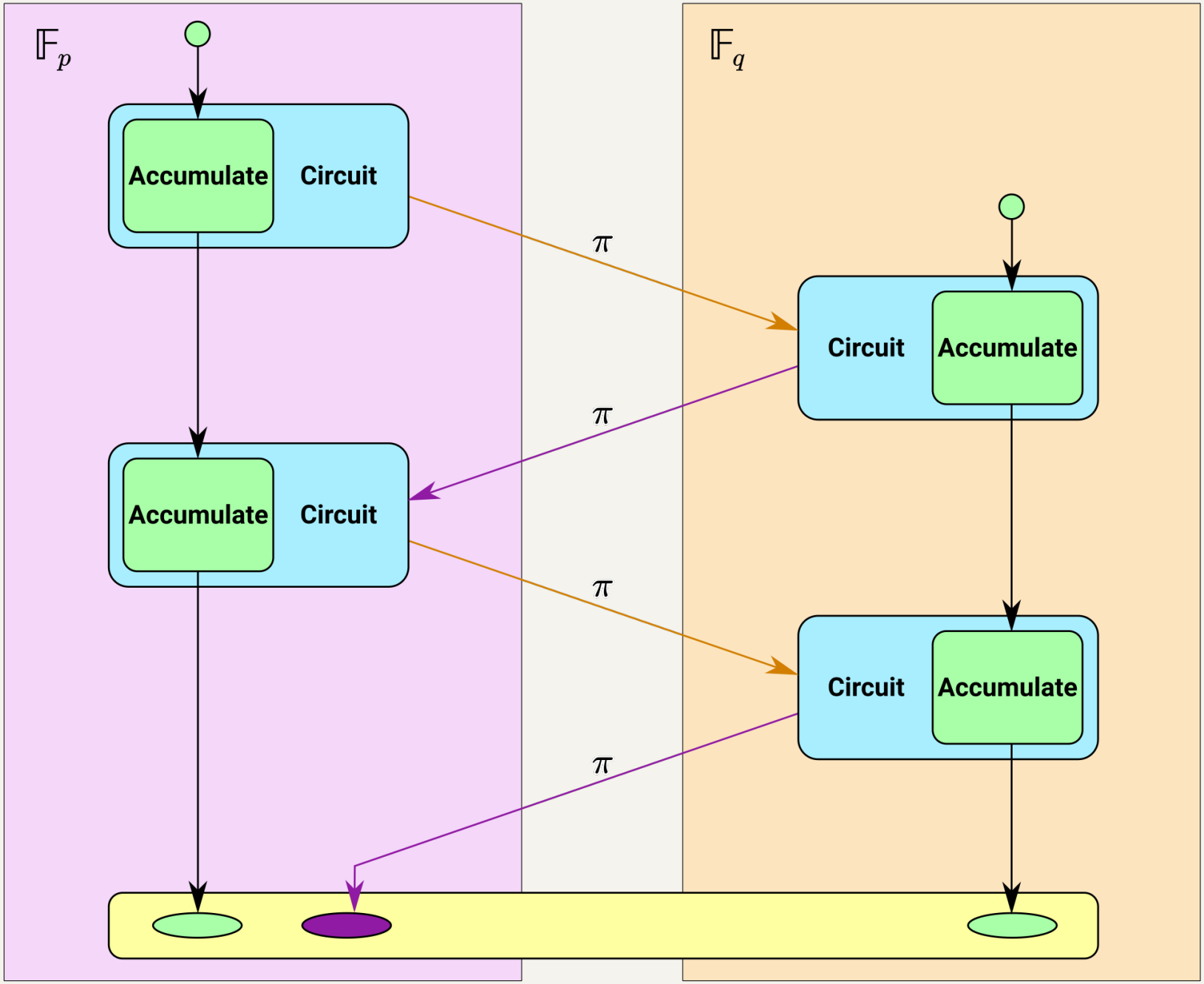
- We defined what a graph of elliptic curves is.
- We explained the difficulty with constructing cycles of curves, and how to solve it by using curves with complex multiplication.
- We described (approximately) what complex multiplication is, and how the units of a lattice defined by the endomorphism ring correspond to the possible orders of curves over a given field.
- Along the way, we discussed endomorphisms and how they correspond to “shortcuts” for scalar multiplication.
  - That part will be useful for an optimization we’re going to describe in this section of the talk.

Accumulation schemes

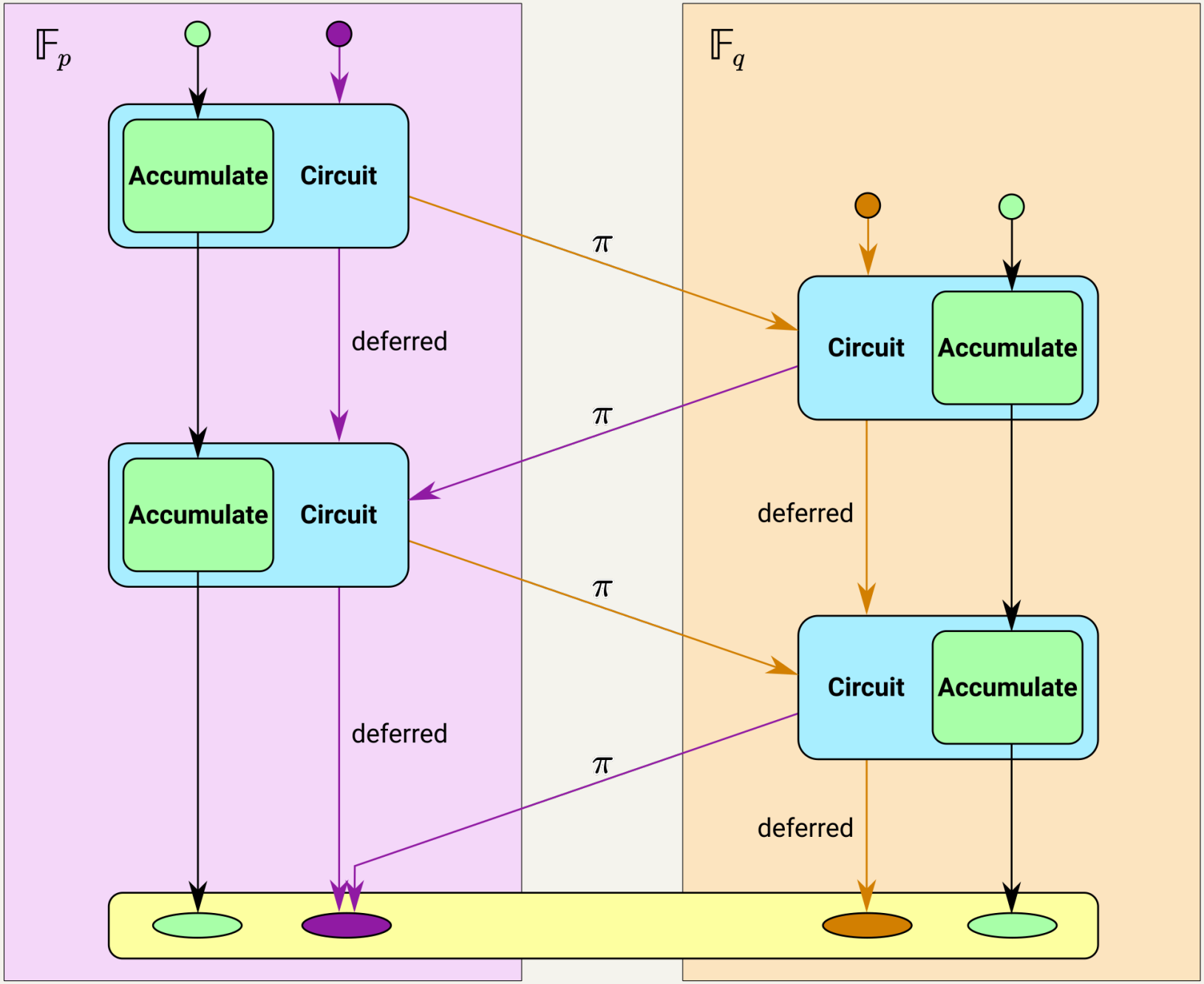


- This was covered in Sean's talk, so I'll just go over the basic idea quickly.
- In an accumulation scheme, it is expensive to "decide" validity of an instance, but cheap to combine two (or more) instances.
- This is how the idea is described in [BCMS2020] (ia.cr/2020/499).
- In practice, accumulation schemes and proof systems use arithmetic in a particular field.
- So if we tried to literally implement what is shown in the diagram, we'd have to do wrong-field arithmetic in the circuit.

Accumulation schemes on 2-cycles

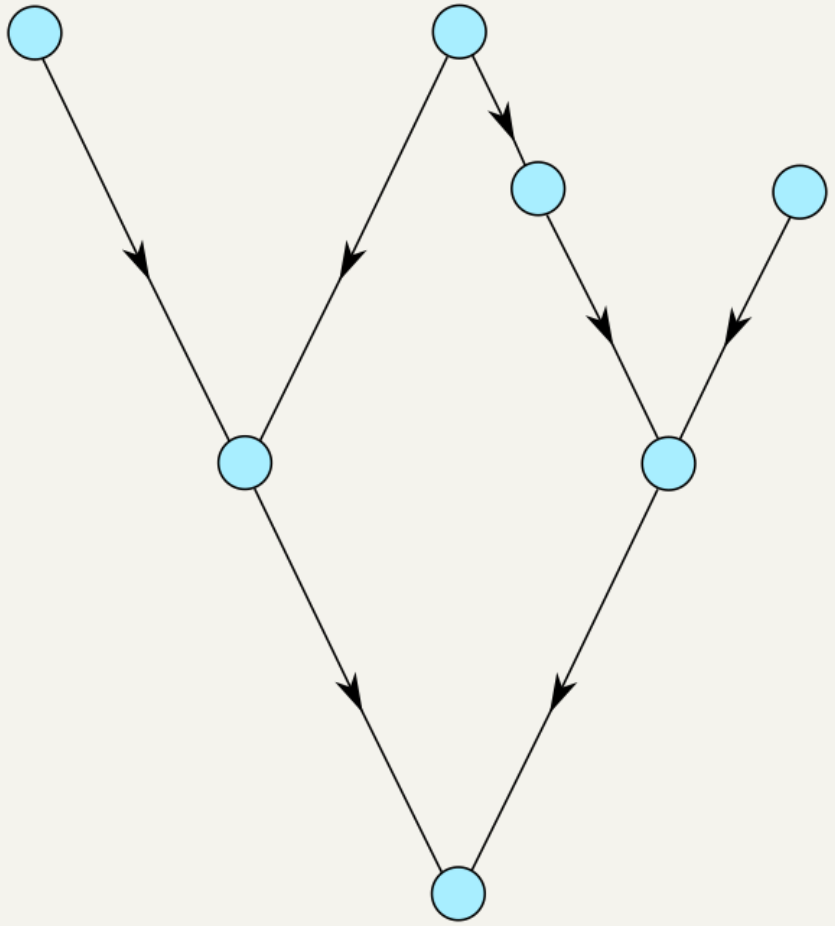


Accumulation schemes on 2-cycles



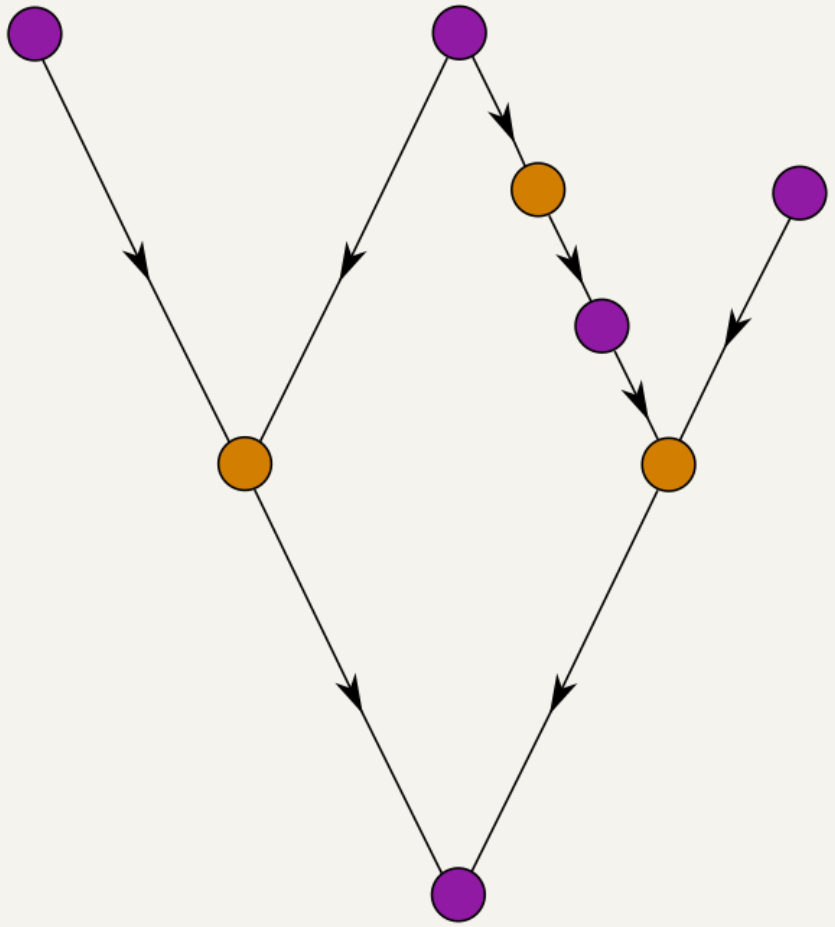
Encoding general Proof-Carrying Data

- Let’s consider how this goes for general PCD, where we have a directed acyclic graph of proofs.



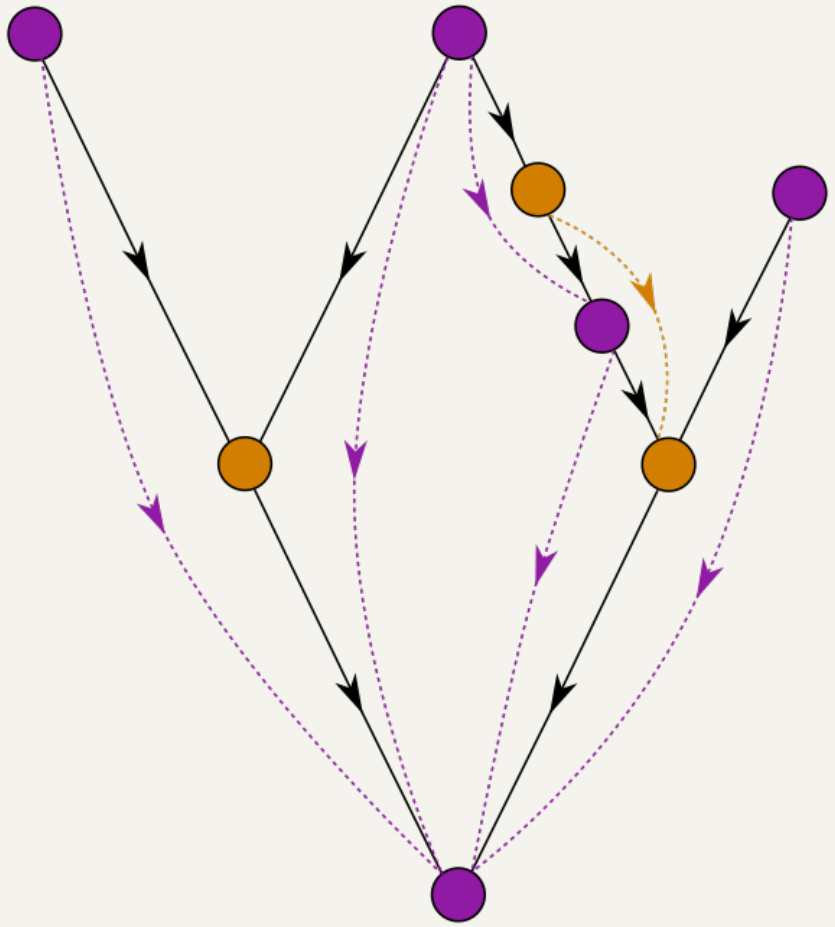
Encoding general Proof-Carrying Data

- We might need to add extra wrapper proofs so that $\mathbb{F}_p$ and $\mathbb{F}_q$ proofs alternate on all paths.



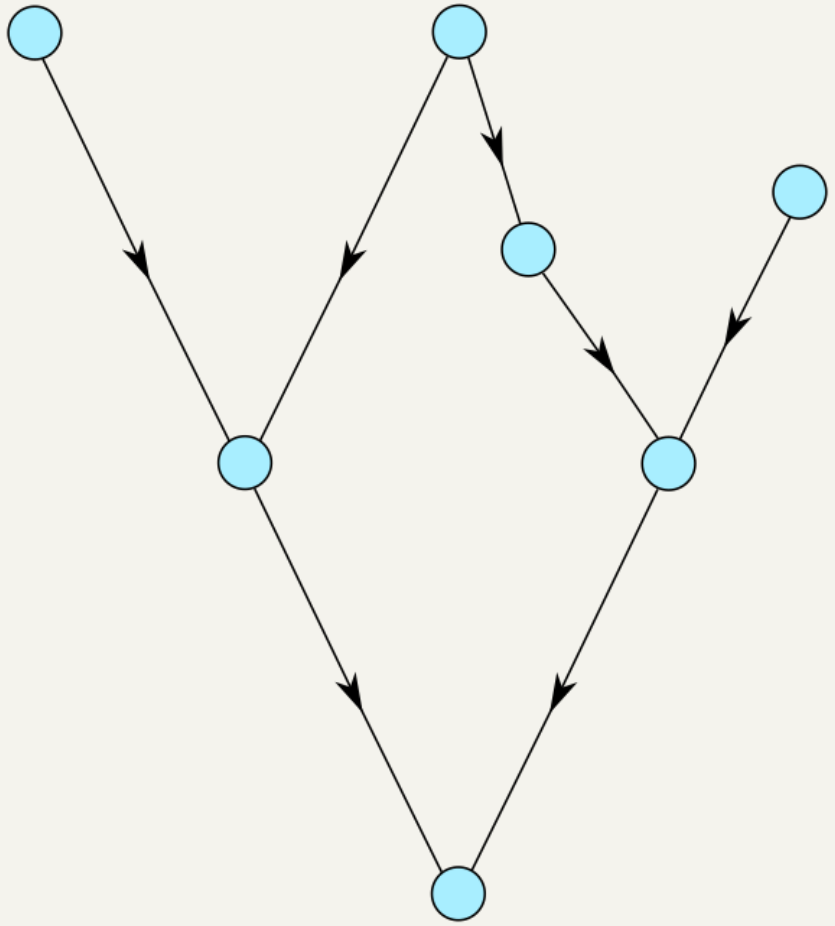
Encoding general Proof-Carrying Data

- We might need to add extra wrapper proofs so that $\mathbb{F}_p$ and $\mathbb{F}_q$ proofs alternate on all paths. Here we also show deferreds.



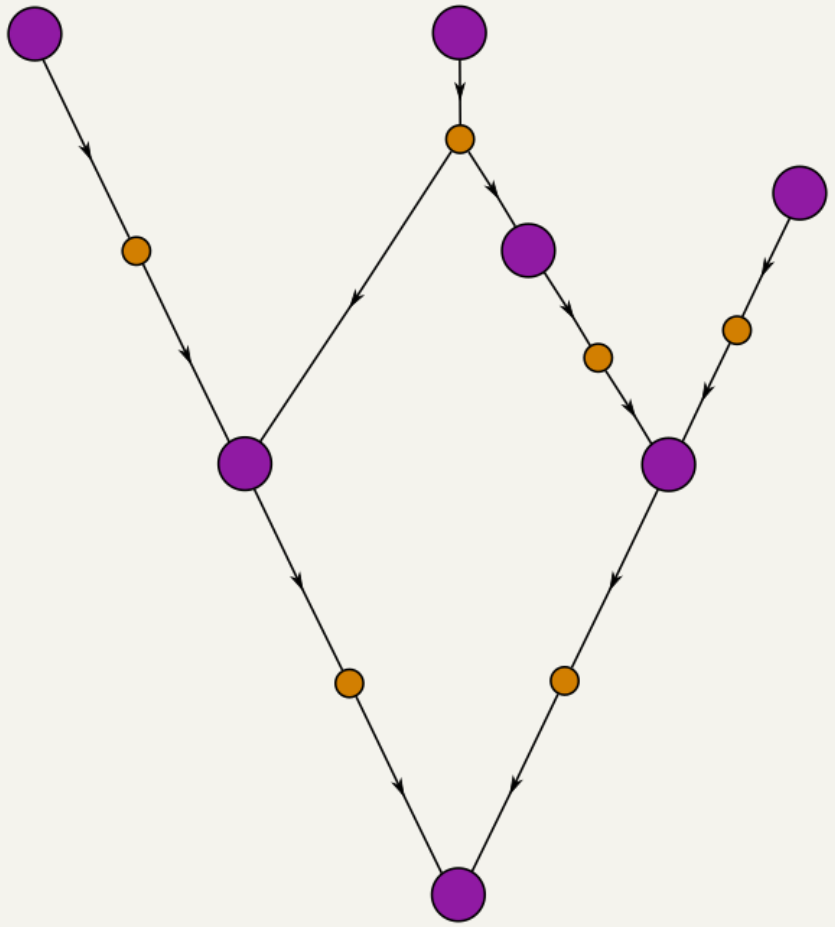
Encoding general Proof-Carrying Data

- Another option is to add wrapper proofs after every node.



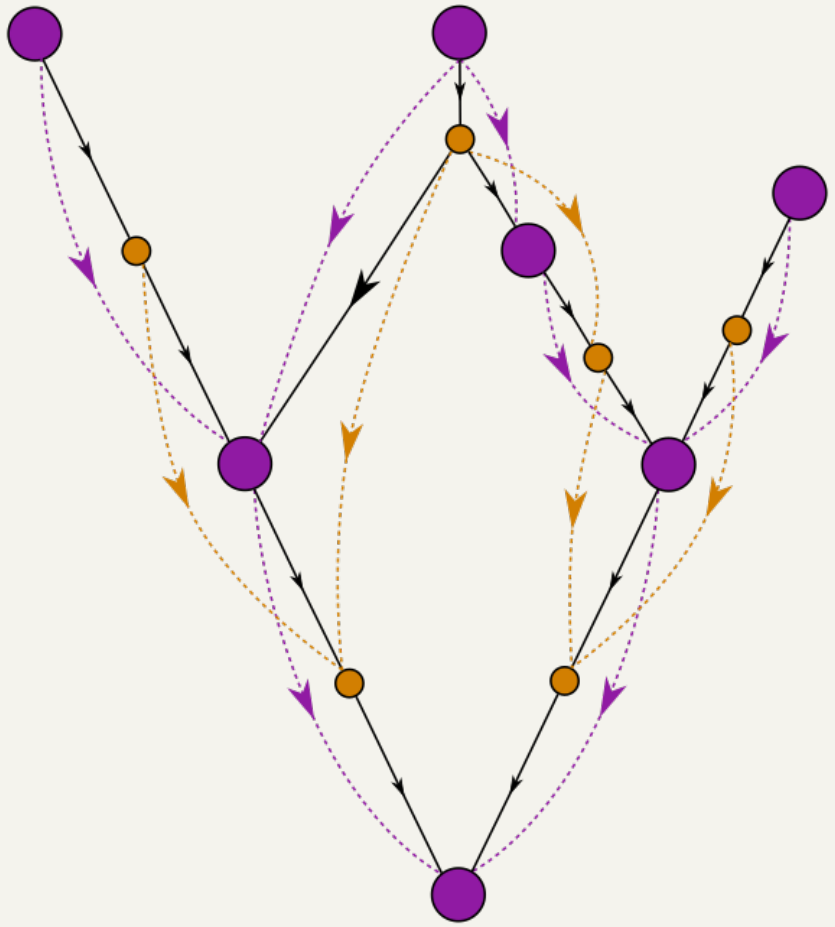
Encoding general Proof-Carrying Data

- Another option is to add wrapper proofs after every node, like this.



Encoding general Proof-Carrying Data

- The data flow for deferreds in PCD might look complicated, but it's fairly straightforward in practice: a proof just needs to include a lists of deferreds added in that proof, and a list of those added in its antecedent proof(s).



Halo 2

- Halo 2 is like Halo but with a different arithmetization

Halo 2
Polynomial circuits and PLONK
Polynomial commitments
Recursion and proof-carrying data

Halo and Sonic arithmetization

- Any generic proof system needs an “arithmetization”, which is a way of expressing statements as relations between variables.
- Halo uses the same arithmetization as Sonic, which is slightly different from R1CS. I’ll call it the “Sonic arithmetization”.
  - It’s similar to the one in [BCCGP2016] and Bulletproofs, but “BCCGP” is too hard to pronounce.
- In R1CS, each constraint is $A \times B = C$, where A, B, and C are linear combinations.
- In Sonic, we have:
  - multiplicative constraints $u \times v = w$, where each of u, v, and w are *fresh* variables.
  - linear constraints, $A = 0$ where A is a linear combination.
- In practice, we can translate fairly easily between R1CS and Sonic arithmetization, with roughly the same number of multiplications.

The Halo verification circuit

- Practical circuits tend to be dominated by a small number of operations that are repeated many times. In the Halo verification circuit, we need:
  - Multi-scalar multiplications
  - A hash function, to generate Fiat–Shamir challenges.
- We also need to include the “application” circuit(s), and so any optimizations to the proof system should be made with that in mind.

Scalar multiplication in circuits

- Halo uses prime-order short Weierstrass curves, $E : y^2 = x^3 + b$.
- We start with a variation on the scalar multiplication algorithm in Zcash #3924, which takes 6 muls per scalar bit.
- This is based on an idea from Kirsten Eisentrager, Kristin Lauter, and Peter Montgomery. Instead of computing [2] $A + P$ directly in a double-and-add algorithm, we compute $(A + P) + A$.
- To avoid needing two constraints for the conditional, we actually compute $(A \pm P) + A$.
- The λ for the outer addition can be computed from the λ for the inner one, saving one mul.
- Naively we need 4 muls for the doubling, 3 for the addition, and 2 for the conditional. This technique costs 3 for the inner addition, 2 for the outer addition, and 1 for conditional negation.

Optimized scalar multiplication

- It turns out we can improve 6 muls/bit to 3.5 muls/bit, by using the endomorphism we dicussed earlier.
- We only require the multiplications to be of some scalar with at least 128 bits of entropy (depending on the verifier challenges).
- The basic idea is to add one of $\{ -P,\ P,\ -\phi(P),\ \phi(P) \}$ at each step. This takes 7 muls for every 2 bits of entropy (the extra mul is to conditionally apply ϕ).
- This has the effect of multiplying by $\zeta a + b$, where a and b depend on random $\mathbf{r}$.

— ALGORITHM 1 —

Inputs: $\mathbf{r} \in \{0,1\}^\lambda, P \in E \setminus \{\mathcal{O}\}$

$\mathsf{Acc} := [2](\phi(P) + P)$

for i from $\lambda/2 - 1$ down to 0:

$$\text{let } S_i = \begin{cases} [2\mathbf{r}_{2i} - 1]P, & \text{if } \mathbf{r}_{2i+1} = 0 \\ \phi\big([2\mathbf{r}_{2i} - 1]P\big), & \text{otherwise} \end{cases}$$

$\mathsf{Acc} := (\mathsf{Acc} + S_i) + \mathsf{Acc}$

Output Acc

Optimized scalar multiplication

- We want the mapping $\mathbf{r} \to \zeta a + b$ not to lose any entropy, i.e. to be 1-to-1.
- To prove this we show that $\mathbf{r} \to (a, b)$ and $(a, b) \to \zeta a + b$ are both 1-to-1.
- For the first part:

Lemma 2. *For* $k \geq 0$, $(\mathbf{c}, \mathbf{d}) \in M_k \mapsto \left(\sum_{j=0}^{k-1} \mathbf{c}_j 2^j, \sum_{j=0}^{k-1} \mathbf{d}_j 2^j\right)$ *is injective.*

Proof (sketch). If $(\mathbf{c}, \mathbf{d})$ and $(\mathbf{c}', \mathbf{d}')$ coincide on a prefix of length m, then the statement reduces to a smaller instance of the lemma with that prefix deleted and k reduced by m. So we need only consider the case $(\mathbf{c}_0, \mathbf{d}_0) \neq (\mathbf{c}'_0, \mathbf{d}'_0)$ and show that the resulting sums always differ. In fact they always differ modulo 4:

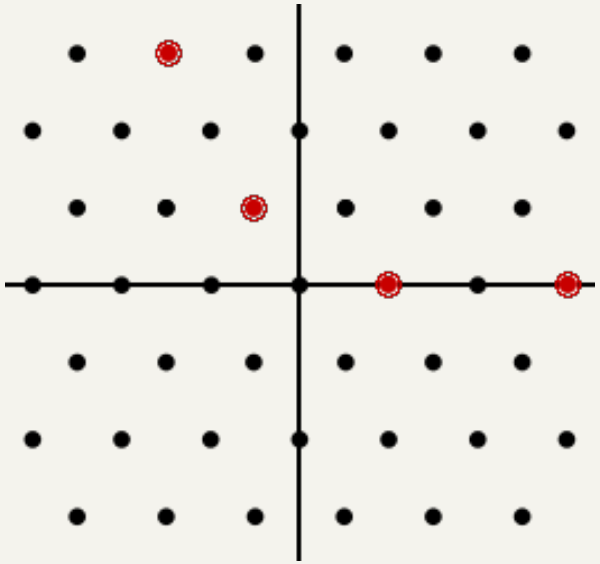
$$\begin{aligned}(\mathbf{c}_0, \mathbf{d}_0) \neq (\mathbf{c}'_0, \mathbf{d}'_0) \implies & \left(\sum_{j=0}^{k-1} \mathbf{c}_j 2^j \bmod 4, \sum_{j=0}^{k-1} \mathbf{d}_j 2^j \bmod 4\right) \\ & \neq \left(\sum_{j=0}^{k-1} \mathbf{c}'_j 2^j \bmod 4, \sum_{j=0}^{k-1} \mathbf{d}'_j 2^j \bmod 4\right)\end{aligned}$$

Therefore, it suffices to verify this property exhaustively for $k = 1$ and $k = 2$ [26, `injectivitylemma.py`], since $j \geq 2$ terms do not affect the sums modulo 4. □

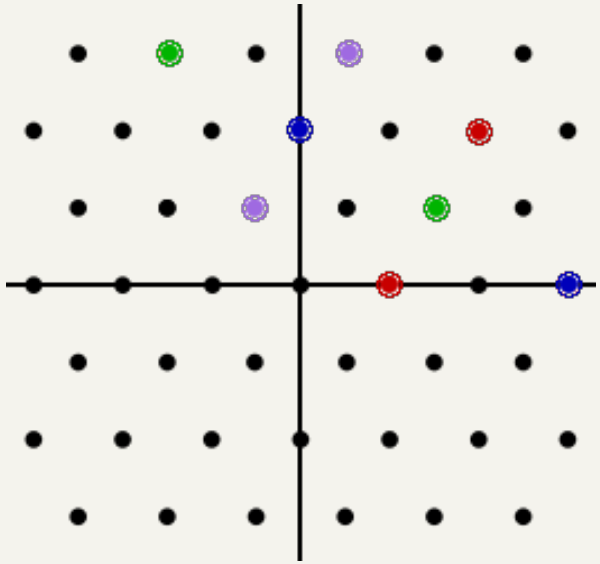
Optimized scalar multiplication

- We want the mapping $\mathbf{r} \to \zeta a + b$ not to lose any entropy, i.e. to be 1-to-1.
- To prove this we show that $\mathbf{r} \to (a, b)$ and $(a, b) \to \zeta a + b$ are both 1-to-1.
- For the first part:

$k = 1$

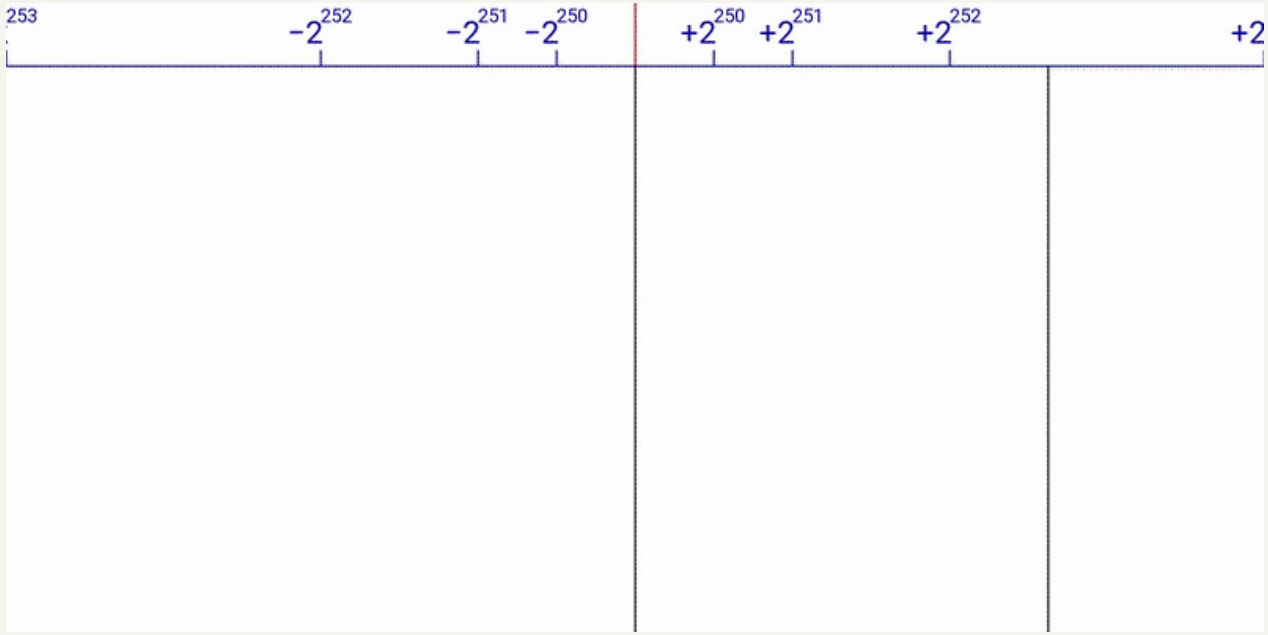


$k = 2$



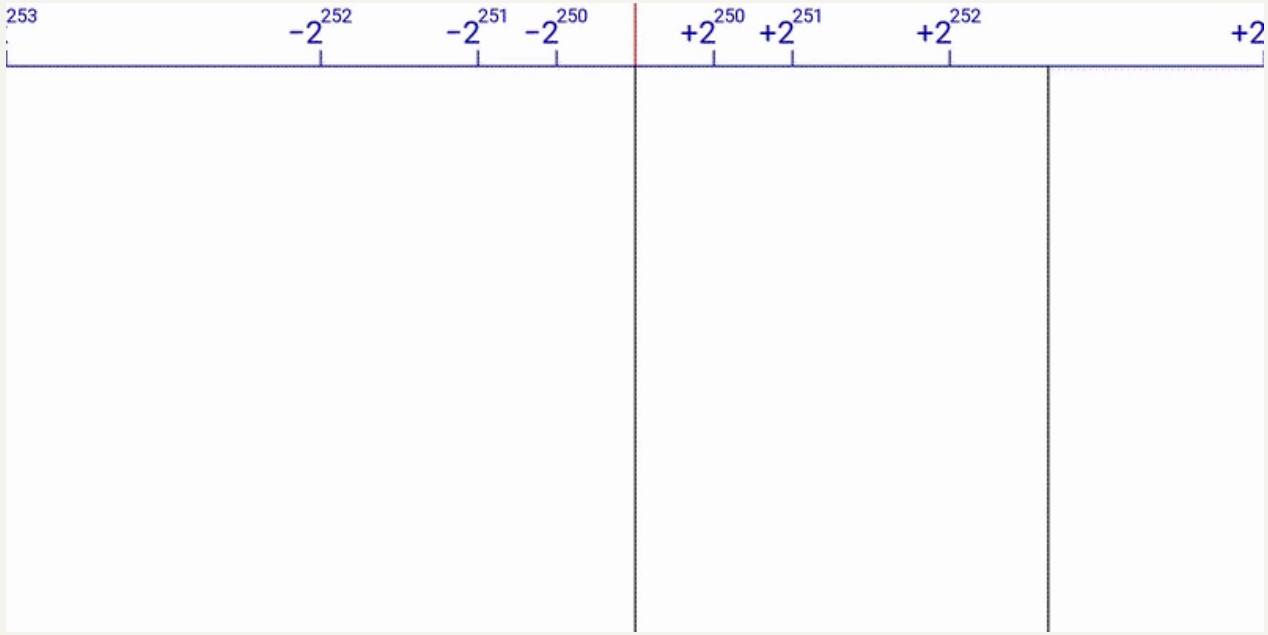
Optimized scalar multiplication

- We want the mapping $\mathbf{r} \to \zeta a + b$ not to lose any entropy, i.e. to be 1-to-1.
- To prove this we show that $\mathbf{r} \to (a, b)$ and $(a, b) \to \zeta a + b$ are both 1-to-1.
- For the second part, we find the smallest distance between two values of $\zeta a \pmod q$ for a in $A = [0, 2^{65} + 2^{64})$. If this is $\geq 2^{65} + 2^{64}$, then a copy of A will "fit between the gaps" of ζA, and so $(a, b) \to \zeta a + b$ must be 1-to-1.
- `checksumsets.py` does this check and also generates this video:



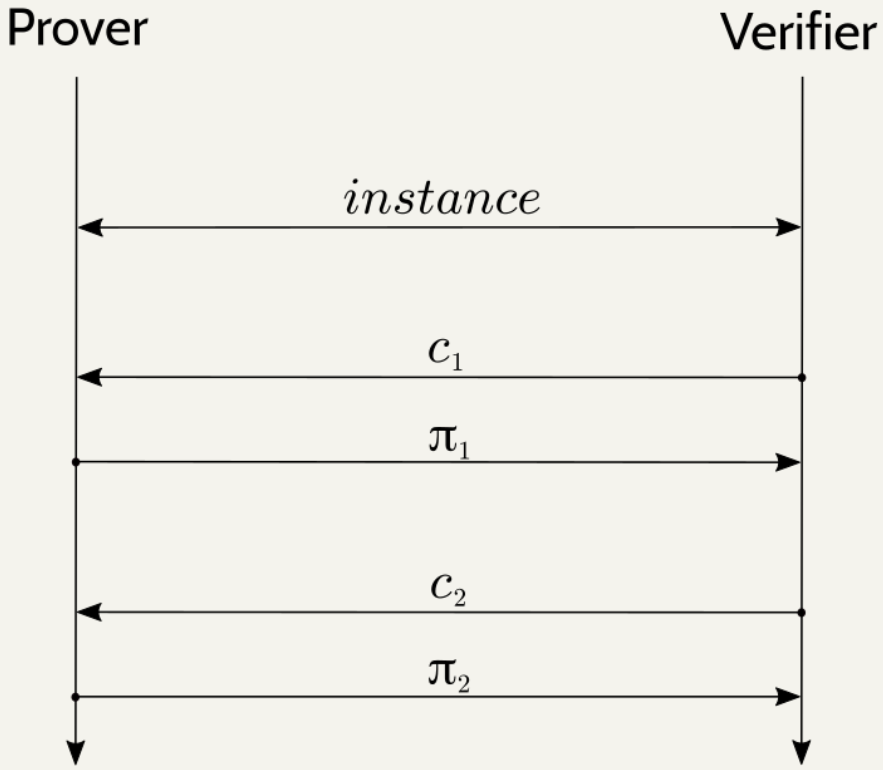
Optimized scalar multiplication

- We want the mapping $\mathbf{r} \to \zeta a + b$ not to lose any entropy, i.e. to be 1-to-1.
- To prove this we show that $\mathbf{r} \to (a, b)$ and $(a, b) \to \zeta a + b$ are both 1-to-1.
- For the second part, we find the smallest distance between two values of $\zeta a \pmod q$ for a in $A = [0, 2^{65} + 2^{64})$. If this is $\geq 2^{65} + 2^{64}$, then a copy of A will "fit between the gaps" of ζA, and so $(a, b) \to \zeta a + b$ must be 1-to-1.
- `checksumsets.py` does this check and also generates this video:



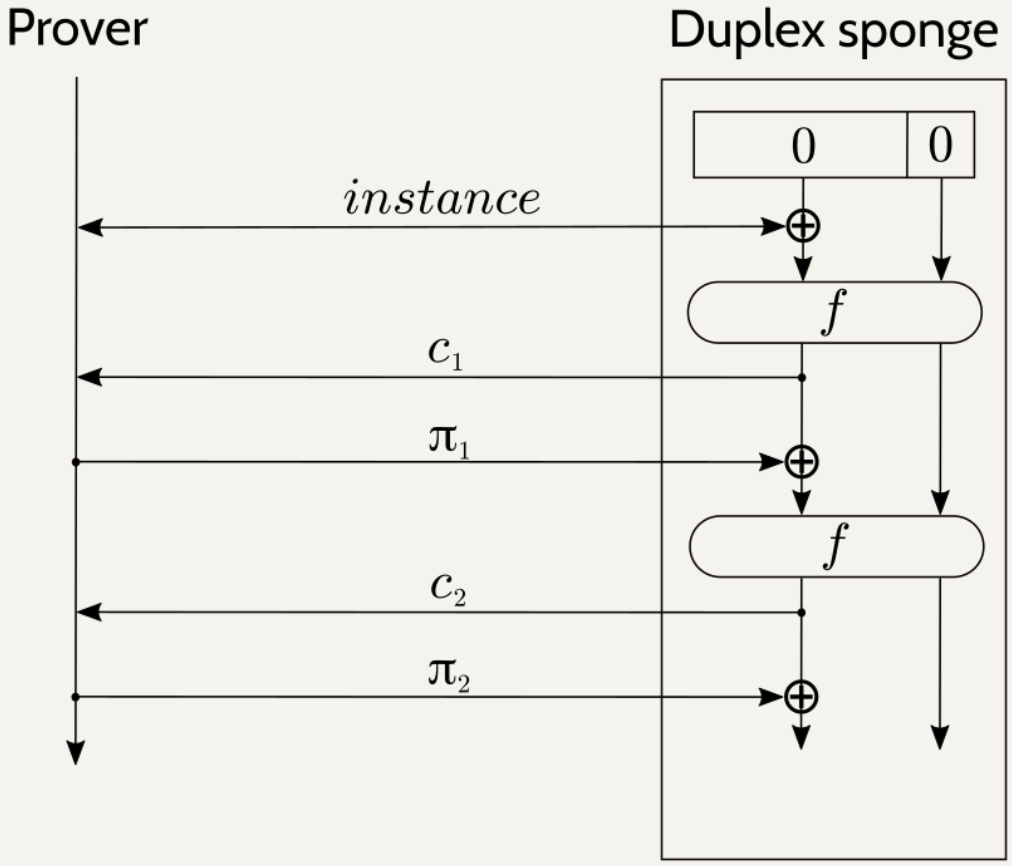
Fiat–Shamir and duplex sponges

- The Fiat–Shamir construction takes an interactive public-coin protocol, ...



Fiat–Shamir and duplex sponges

- The Fiat–Shamir construction takes an interactive public-coin protocol, and replaces the verifier with a hash function.



- Using a duplex sponge basically halves the number of f evaluations relative to other hash constructions.

Optimizations for any duplex sponge

- Use addition in the field for ⊕, rather than XOR.
- Compress the absorbed inputs.
  - There's a way of probabilistically compressing two curve points to three field elements that is *much* less expensive than standard point compression (see accompanying notes).

Theorem: Let $E/\mathbb{F}_p : y^2 = x^3 + b$ be an elliptic curve with a large number of points $\#E$. Except with negligible probability over random choices of P_1 and P_2, $x_1^3 \neq x_2^3$ and $x_1^3 + b \neq 0$ and $x_2^3 + b \neq 0$ and $y_1 + y_2 \neq 0$, and in that case $(x_1, x_2, y_1 + y_2)$ determines P_1 and P_2.

- Pick a "rate" that is just large enough that we only need one f evaluation per round.
  - For the inner product argument we need $\lg(N)$ rounds, each of which absorbs two curve points and squeezes out one challenge.

Algebraic hashes

- To instantiate f in the duplex sponge, we need a permutation that is efficient in the circuit.
- Rescue is a permutation designed to be efficient in circuits over a prime field.
  - We also considered Poseidon, but recent attacks have wounded it (eprint 2020/188), so it looks like Rescue has a better security/cost trade-off.
- Optimization that we can use with Rescue:
  - Choose curves with gcd(p–1, 5) = 1 so that $x \mapsto x^5$ is a permutation.
  - $x \mapsto x^3$ cannot be a permutation for curves with the endomorphism (and the latter gives us more of a performance advantage).

Questions?

Daira Hopwood
 @feministPLT
@daira#0512 on Discord
https://github.com/daira/halographs