ZCON 2 Lite

Presented by the Zcash Foundation

Halo 2 and Orchard

Deirdre Connolly

Daira Hopwood

Sean Bowe

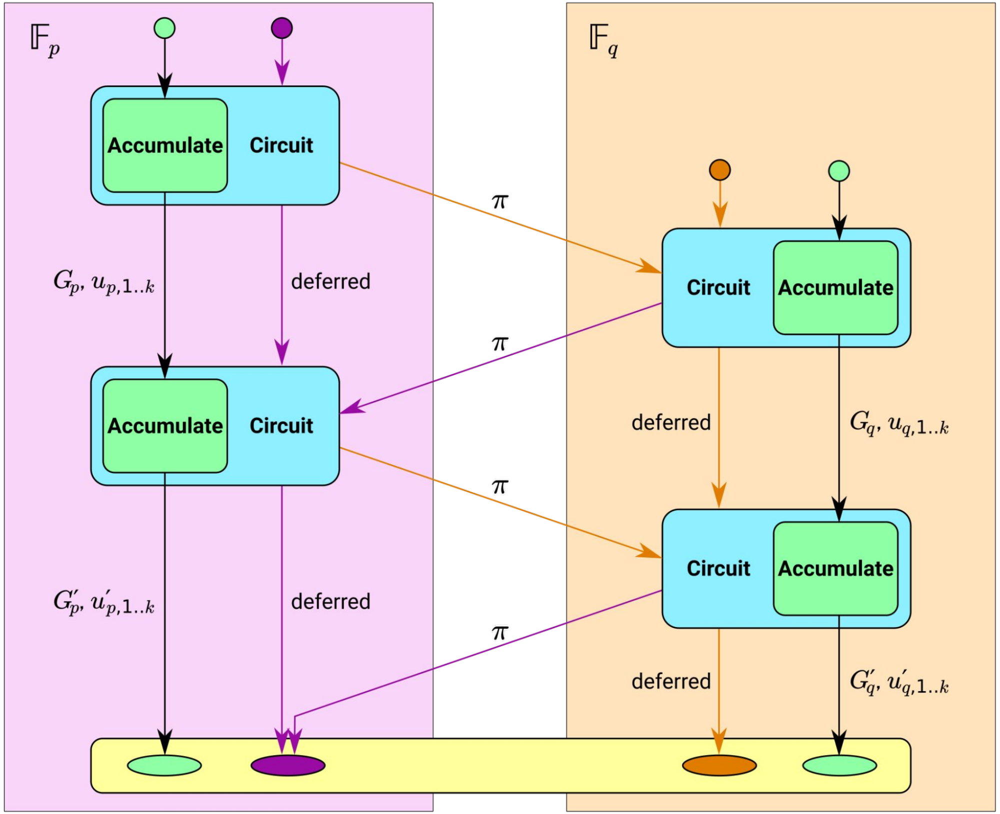
What are Recursive Proofs?

- **Proofs of knowledge** can be used to show you performed a computation
- But what if *that* computation is "verifying a proof of knowledge"?
  - This is what is known as **proof composition**.
- But what if the inner proof's computation is the same as the outer proof's computation?
  - This is what is known as **recursive proof composition**.
- Recursive proofs can be used to show practically unlimited amounts of computation have been correctly performed.
  - This leads to a variety of scaling solutions.

Background

- Recursive proofs finally became a thing we could actually do.
  - Prior to ECC's discovery of **Halo** (late 2019), they were bulky and inefficient, and required trusted setups.
- Recursive proofs help address scalability issues.
- More efficient SNARK constructions emerged after Halo appeared which we could borrow ideas from.
- This led the ECC team toward **Halo 2** — essentially the original Halo, but a more PLONK-like construction with heavy optimization.

What's Halo?

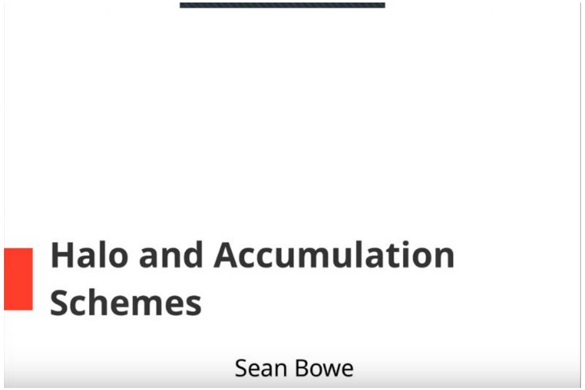


$\mathbb{F}_p$
$\mathbb{F}_q$
Accumulate
Circuit
Accumulate
Circuit
Circuit
Accumulate
Circuit
Accumulate
$G_p, u_{p,1..k}$
deferred
$G'_p, u'_{p,1..k}$
deferred
π
π
π
π
deferred
$G_q, u_{q,1..k}$
deferred
$G'_q, u'_{q,1..k}$

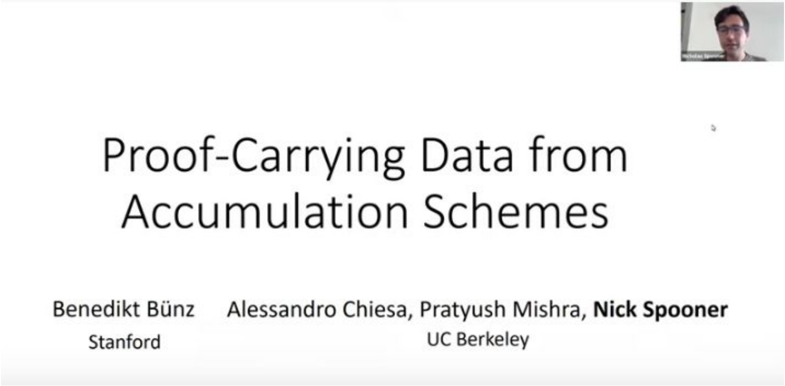
What's Halo 2?

- Recursive zk-SNARK proving system built on the cyclic prime-order Pasta curves: Pallas and Vesta.
- Uses the UltraPLONK arithmetization from PLONK to express the circuits, instead of previously used R1CS.
  - Supports higher degree polynomial constraints, letting us constrain more at once.
- The Pasta curve cycle allows a proof produced by one curve to be efficiently verified in the circuit of the other curve. See Daira's ZK Study Club presentation.
- Some checks can be deferred to the later verifier, allowing recursion, which allows smaller total proof sizes on the blockchain overall. [1]
- Inner-product argument removes the need for a trusted setup:
  - When using Groth16, we need to encode the circuit into the Structured Reference String. Halo's inner product argument, similar to Bulletproofs, does not need this.

Learn more about Halo



https://youtu.be/yn_j3QJDZ9s



https://youtu.be/UNwlBq1FQ3E

Why not just “Sapling on Halo 2”?

- We need to change the curves:
  - (Efficient) recursion needs a 2-cycle of prime-order curves [1]
  - We don’t need pairings, and therefore the curves can be smaller [2]
  - Gives the opportunity for curve optimization (constant-time hash-to-curve [3] etc.)
- Linear verification time would make proof verification too slow without further optimization.
- Halo 2 allows for efficient “lookup arguments”, which led us to design Sinsemilla, a collision-resistant hash that is more efficient with lookups.
- We wanted to remove hashes that are expensive in a circuit (BLAKE2s [4]).
- “Action” transfers are simpler and leak a bit less metadata [5] than separate Spends and Outputs.

So what is Orchard?

- A shielded payment protocol, like Sapling.
- Take all the good parts of Sapling, drop a couple of less good ones, and optimize for Halo 2!
- What is the same as Sapling?
  - Support for viewing keys
  - Support for efficient signing on hardware wallets
  - Support for diversified addresses
  - How balance checking works
  - Note encryption
  - The note commitment tree is a binary tree of the same height (32)

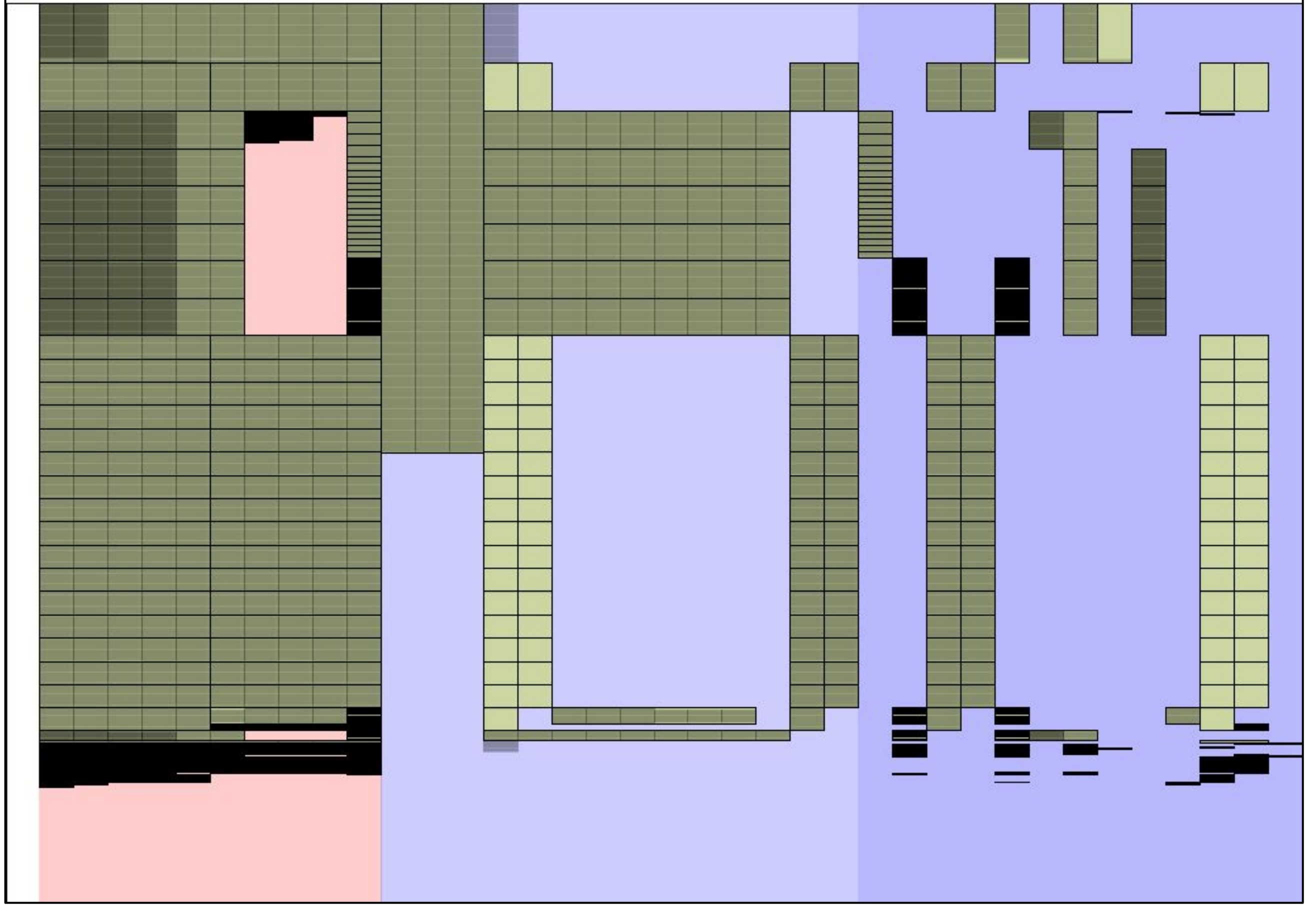
How is Orchard different from Sapling?

- Optimized for Halo 2 / UltraPLONK arithmetization.
- Actions, rather than Spends and Outputs.
- Different nullifier computation (allows dropping BLAKE2s in the circuit).
- All diversifiers are valid, using the new hash-to-curve function.
- Unified Addresses.
- A full viewing key + ask is sufficient for proving (there is no nsk).
- Note commitments are unique in a valid chain.

Orchard circuit performance

- No “heavyweight” hash functions in the circuit.
- Optimized to make use of custom gates and lookups in UltraPLONK.
- Circuit size is below 2048 rows × 10 advice columns.
  - The Sapling Spend circuit was ~96500 R1CS constraints, and the Output circuit was ~7600 constraints.
  - Fits below Halo 2’s recursion threshold.
  - ZSAs would also fit within 2048 rows.
- Action circuit verification time is ~20ms, with scope for improvement.
  - Both proving and verification can be efficiently multi-threaded.
  - Can’t give you a benchmark for proving yet; current implementation is dominated by some inefficiencies in witness generation.

Orchard Action Circuit



Sinsemilla hash function

- Replaces Bowe–Hopwood Pedersen hashes in the note Merkle tree and in non-homomorphic commitments (NoteCommit and $\mathsf{Commit}^{\mathsf{ivk}}$).
- Uses lookups to process 10 bits at once (rather than 3 bits as in Bowe–Hopwood).
  - There is scope for further optimization, but we opted for simplicity.
  - We do 2 Sinsemilla hashes in parallel in the Orchard circuit's 10 columns.
- Basically a vector Pedersen hash "on its side".
- Security tightly reducible to a standard vector Pedersen hash, hence to DLP.
- Implemented using incomplete curve additions.
  - Failure implies finding a discrete logarithm.

Security properties

- **Balance:** can I forge money?
- **Spend Authentication:** can I spend someone else’s money?
- **Note Privacy:** can I gain information about notes sent in-band?
- **Note Privacy (OOB):** can I gain information about notes sent out-of-band, only from the public block chain?
- **Spend unlinkability:** given the IVK but not the FVK for an address, can I (maybe the sender) detect spends of notes sent to that address?
- **Faerie Resistance:** can I cause notes to be accepted that may be unspendable?

Security assumptions

	Sapling	**Orchard**
Balance Preservation	DL_{Jubjub}	DL_{Pallas}
Spend Authentication	RedDSA unforgeability (DL_{Jubjub})	RedDSA unforgeability (DL_{Pallas})
Note Privacy	$HashDH_{Jubjub}$(BLAKE2b)	$HashDH_{Pallas}$(BLAKE2b)
Note Privacy (OOB)	Perfect	Near perfect ‡
Spend Unlinkability	PRF(BLAKE2s)	$DDH\dagger_{Pallas}$ or PRF(Poseidon)
Faerie Resistance	CR(BLAKE2s) and DL_{Jubjub}	DL_{Pallas}

† We also assume that $\{PRF_{nk}(x) : nk \in \mathbb{F}\}$ for any x gives an adequate range for DDH_{Pallas}.

‡ Statistical distance $<2^{-167.8}$ from perfect.

Balance preservation – Sapling (informal)

- The critical property is that there can only be one nullifier for a given note commitment. If there were more than one, we can forge / double-spend.
- A note commitment cm binds (g_d, pk_d, v, ρ) for the note, given pos.
- Check pk_d = [ivk] g_d. There is only one such ivk for given (g_d, pk_d).
- Check ivk = CRH^{ivk} (ak, nk). This binds ak and nk.
- Check nf = PRF_{nk}(ρ).
- nf deterministically depends only on fields bound to the note (and the position which is fixed for a given commitment).

Balance preservation – Orchard (informal)

- The critical property is that there can only be one nullifier for a given note commitment. If there were more than one, we can forge / double-spend.
- A note commitment cm binds (g_d, pk_d, v, ρ, ψ) for the note (no pos).
- Check pk_d = [ivk] g_d. There is only one such ivk for given (g_d, pk_d).
- Check ivk = $\mathrm{Commit}^{\mathrm{ivk}}_{\mathrm{rivk}}$ (ak, nk). This binds ak and nk.
- Check nf = $\mathrm{Extract}_{\mathbb{P}}([(\mathrm{PRF}_{nk}(\rho) + \psi) \bmod q_{\mathbb{P}}]\, \mathcal{G} + cm)$.
- nf deterministically depends only on fields bound to the note.

Security assumptions

	Sapling	**Orchard**
Balance Preservation	DL_{Jubjub}	DL_{Pallas}
Spend Authentication	RedDSA unforgeability (DL_{Jubjub})	RedDSA unforgeability (DL_{Pallas})
Note Privacy	$HashDH_{Jubjub}$(BLAKE2b)	$HashDH_{Pallas}$(BLAKE2b)
Note Privacy (OOB)	Perfect	Near perfect ‡
Spend Unlinkability	PRF(BLAKE2s)	$DDH\dagger_{Pallas}$ or PRF(Poseidon)
Faerie Resistance	CR(BLAKE2s) and DL_{Jubjub}	DL_{Pallas}

† We also assume that $\{PRF_{nk}(x) : nk \in \mathbb{F}\}$ for any x gives an adequate range for DDH_{Pallas}.

‡ Statistical distance $<2^{-167.8}$ from perfect.

Note Privacy / Unlinkability – Orchard (informal)

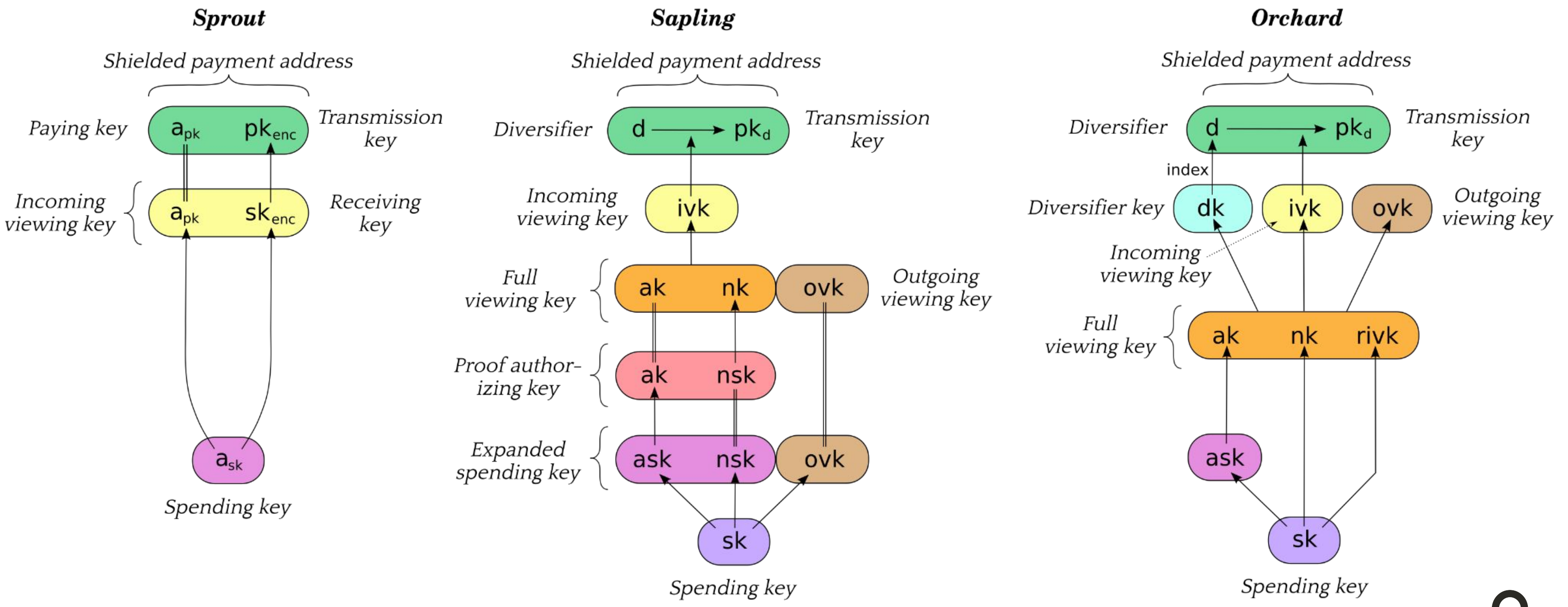
- Basically the main thing we need to worry about is leakage of information via note ciphertexts, note commitments, or nullifiers.
- Note ciphertexts are encrypted with ChaCha20-Poly1305, as in Sapling.
- Note commitments are perfectly blinded.
- $\mathrm{nf} = \mathrm{Extract}_P([(\mathrm{PRF}_{nk}(\rho) + \psi) \bmod q_P]\,\mathcal{G} + \mathrm{cm})$.
    - This construction is designed so that if *either* Poseidon is a secure PRF, *or* Poseidon is non-trivial and assuming DDH on Pallas, nf will look like a randomly distributed Pallas *x*-coordinate.

Faerie Gold – Orchard (informal)

- In a Faerie Gold attack†, an adversary can create two notes, only one of which can be spent, but the recipient sees both as valid.
- This can happen if two notes have the same nullifier.
- In Orchard, ρ for a new note is copied from the nullifier of the note spent in the same Action description, which ensures it is unique.
- Then, the new note's nullifier is effectively a (short) Pedersen hash with input that includes ρ, so it is unique to that note:
  - $\mathsf{nf} = \mathrm{Extract}_{\mathbb{P}}([(\mathrm{PRF}_{nk}(\rho) + \psi) \bmod q_{\mathbb{P}}]\, \mathcal{G} + \mathrm{cm})$.

† Thanks Zooko for finding this attack, and for the name!

Keys and Addresses

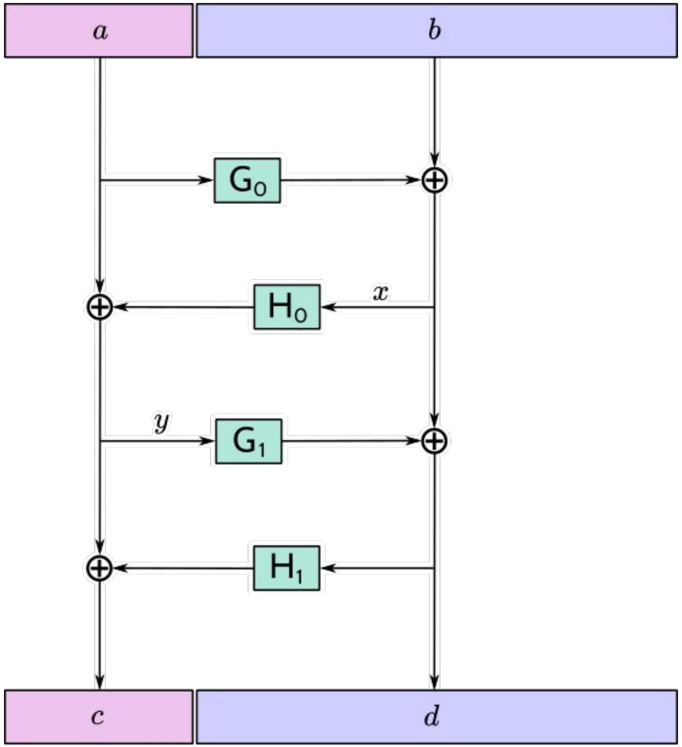


Unified Addresses

- If we added a new Orchard address type, we'd have 4 valid types of Zcash addresses to juggle: Orchard, Sapling, Sprout[1], and Transparent.
- What if we could have a bundle of addresses that a user presents as valid to pay them, with the latest shielded one as top priority?
- This idea led to Unified Addresses, that combine any/all Zcash "classic" addresses (except Sprout[1]) into one encoding.
- Supports 'just Orchard' or 'just Sapling', requires at least one shielded option.
- See ZIP 316 for full details.

Unified Addresses – F4Jumble

- Concating multiple classic addresses together could be more vulnerable to address replacement attacks, if users compare only a subset of the address.
- To combine addresses together, came up with an unkeyed 4-round Feistel network, used when encoding and decoding Unified Addresses.
- Then encode with Bech32m.



Orchard Launch 🚀

Targeted for NU5.

Testnet activation planned for July 2021

(delayed to August).

Mainnet activation planned for October 2021

(delayed to November).

Thank you! Questions?