Understanding the Security of Zcash

Daira-Emma Hopwood (ze/hir)

<daira-emma@electriccoin.co>
@daira.bsky.social

Slides at https://github.com/daira/zcash-security

A statement of solidarity

- I could not present a talk in the US without expressing solidarity with everyone affected by recent attacks on bodily autonomy, especially over abortion rights and trans healthcare, and on Native American sovereignty.

- **Bodily autonomy and privacy are intertwined**, and neither should be negotiable.

- **There is no privacy without financial privacy**.

    - This affects, for example, crowdfunding funds required for an abortion or to travel for one.

Goals and Disclaimers

- Zcash is based on Zerocash, described in an academic paper [BCGGMTV2014] with proofs of several important security properties. However:

  - The proofs had several holes (which we took into account before Zcash was launched).

  - Some of the Zerocash security definitions have modelling deficiencies (they don't give the adversary sufficient power, for example to force rollbacks, insert multiple transactions, or front-run a transaction in the mempool).

  - Zcash has three shielded protocols (Sprout, Sapling, and Orchard), all different from each other and from Zerocash.

- **The primary goal of this talk is to give you an idea of what needs to be proved**. We won't have time for actual proofs (and I don't have proofs for everything), but we will sketch some security arguments.

- We concentrate almost exclusively on Orchard, occasionally mentioning differences for Sapling.

- Feel free to try to attack the protocol at the points where you think I'm handwaving too much!

- Polite request to follow the security disclosure process linked from https://z.cash/security if you believe you have found a bug.

Out of scope for this talk

- Eventual ledger consistency
  - I'll touch on consequences of rollbacks for other security properties.
- Network-level privacy leaks via metadata
- Shielded multisig using FROST
- Spend authorization for hardware wallets
- Collaborative transaction creation
- Transparent parts of the protocol
  - Security flaws in those parts would in practice affect Balance and Spendability.

- The symbol ‼ indicates some caveat, which I may or may not have time to explain.

Versions of the shielded protocol

- Zerocash (2014 paper):

  - The protocol described in the Zerocash paper was never deployed as-is. It is broken for Balance: an "InternalH attack" on binding of note commitments, with cost ~2^{64} SHA-256 evaluations, was published by us in April 2016. It is also broken for Spendability due to Faerie Gold attacks (explained later).

- Sprout (October 2016):

  - Sprout used an unsound version of [BCTV2014] proofs at launch. It was repaired with the Sapling network upgrade. It is similar to Zerocash, but integrates differently with the Bitcoin layer, changes the note commitment scheme and some other details, and has a complicated "interstitial treestate" mechanism for transferring value inside a transaction.

- **Sapling** (October 2018):

  - This upgrade changed algorithms to be more circuit-efficient, by taking advantage of elliptic curve cryptography on an "embedded curve", Jubjub. As a result, the Sapling Spend circuit is ~20 times smaller than the original Sprout JoinSplit circuit. It has a completely different key and address structure that supports viewing keys and diversified addresses. Sapling uses value commitments as part of ensuring balance.

- **Orchard** (May 2022):

  - Orchard switches to the Pasta curves and the Halo 2 proving system, which is capable of efficient recursion. Algorithms were changed again for efficiency with these curves and proofs. Other changes include tweaks to the key/address structure and the method of computing nullifiers (to avoid using a heavyweight hash algorithm in the circuit).

Privacy in cryptocurrencies

- Why do we need a strong definition of privacy?

    - Intersection attacks (combining information from multiple higher-than-expected probability links) are very powerful, especially for active adversaries.

    - Preventing addresses or amounts from appearing on the blockchain does not help much by itself: the transfer graph is the information we want to protect.

    - "Plausible deniability" is not going to help when fascists and/or cops are at your door, because they will have other information that they can use to confirm guesses.

    - The nightmare scenarios that privacy advocates have been warning about for decades are real. You could be hauled to jail if someone finds out that you have bought mifepristone, say, in a state that outlaws abortion. You could have your trans kids taken away, if someone finds out that you or they have bought hormones or puberty blockers in a state that outlaws gender-affirming treatment.

- Cryptocurrencies with privacy based primarily on ring signatures are trivially insecure, because any ring entry gives a higher-than-expected probability link.

- The practical use of Zcash still imposes significant usability hurdles for fully shielded transactions, and that's a legitimate criticism.

Kinds of adversary

- Cryptography is complicated. We are interested in which security properties survive various cryptographic breaks:

    - A "standard adversary" can perform only feasible classical computations, and has no special ability to break things.

    - A "knowledge-soundness-breaking adversary" (KS-breaking) is assumed to be able to find proofs of statements for which it does not know the witness, in the Groth16 and/or Halo 2 systems used by Zcash.

    - A "discrete-logarithm-breaking adversary" (DL-breaking) is assumed to be able to find discrete logs for arbitrary elements of the cryptographic groups used by Zcash (Jubjub, BLS12-381 $\mathbb{G}_{\{1,2,T\}}$, Pallas, Vesta).

    - A "quantum adversary" is an adversary able to perform large-scale quantum computations, such as instances of Shor's algorithm for discrete logs, or Grover's algorithm for preimage search.

- In terms of power, Quantum adversary > DL-breaking adversary > KS-breaking adversary > Standard adversary.

- Even if it turns out that large-scale quantum computers cannot be built, we might still care about DL-breaking adversaries, because discrete-log-based crypto is often more complicated to reason about and to implement correctly, to protect from side-channel attacks, etc., than symmetric crypto.

- Proof systems are especially complicated. Privacy in Zcash should not depend on any soundness property of the proof system (but may depend on zero knowledge).

- In these slides, I'll write security properties that are intended to hold against quantum adversaries (and therefore also other types of adversary) in green. Properties written in black are intended to hold against standard adversaries.

How Zcash works

- We start with a blockchain protocol that provides eventual ledger consensus.

- There is a key structure: Spending key → Full viewing key → Incoming viewing key → {Addresses}
(→ means one-way and collision-resistant).

- Shielded value is held in notes, which follow a UTXO-like model.

- A note has a value and an address. Only the holder of the!! spending key for that address can spend it.

- Privacy in UTXO-based cryptocurrencies can be compromised:

  - by linking outputs and their corresponding spends (that is, gaining information about links in the transfer graph);

  - by gaining information about specific inputs or outputs;

  - by network (meta)data when a transaction is submitted by its sender or recognised by its receiver.

- The challenge is to be able to hide links in the transfer graph while preventing double spends.

- We define two cryptovalues associated with a note: a commitment and a nullifier.

- The commitment appears on the ledger at the point a note is created, and the nullifier appears where it is spent. To prevent double-spends, a given nullifier cannot appear twice.

How Zcash works, continued

- How does a receiver know which transactions are sending notes to it?

  - Addresses have *incoming viewing keys*.<mark>!!</mark>

  - Note plaintexts are public-key encrypted to the key in the destination address.

  - Each receiver trial-decrypts *all* transactions using its incoming viewing key.

  - The encryption scheme is *key-private*, so that the ciphertext does not reveal information about the key/address.

  - Broadcasting of transactions on the ledger, with key-private encrypted outputs, acts as an anonymous communication channel from senders to recipients.

  - Zcash doesn't preclude sending notes out-of-band. That's out of scope for this talk (but might be useful in future to reduce blockchain size requirements).

Sapling-specific detail

- Transactions have a Spend proof for each shielded input, and an Output proof for each shielded output.

  - Spend proofs are essential to the idea of the protocol: we need to prove in zero knowledge a rather complicated statement involving showing a path in the commitment tree, deriving the nullifier via a PRF, etc. This requires the full power of zk-SNARKs. Output proofs are more of a technical convenience.

- Spend and Output proofs don't directly check that values balance; they do so via homomorphic value commitments (like in Confidential Transactions, Monero, or Mimblewimble).

- The nullifier of a note depends on its position.

- The proving system makes use of a particular scalar field efficient in the circuit. We rely on that heavily for optimization, using Pedersen hashes in an elliptic curve over that field, etc.

- Viewing keys allow selectively revealing the content of transactions.

  - We also know how to define payment disclosures but that is unimplemented.

- "Diversified addresses" are a way to derive multiple addresses that cannot be linked to each other, but allow decryption using the same incoming/full viewing key.

Orchard-specific detail

- Transactions have Action proofs that cover one shielded input and one shielded output.

- Action proofs don't directly check that values balance; they do so via homomorphic value commitments (to input value – output value).

- Different method of preventing Faerie Gold attacks to Sapling, making use of the fact that Actions have both an input and an output.

- The proving system makes use of a particular scalar field efficient in the circuit.

  - Unlike Sapling we use prime-order short Weierstrass curves, Pallas and Vesta.

- Viewing keys and diversified addresses work as in Sapling.

  - We also know how to define payment disclosures but that is unimplemented.

- Outside scope of this talk but important in practice:

  - Unified addresses, which allow combining protocol addresses for different pools into a single user-level address.

  - Protocol cleanups we didn't have time to do for Sapling, such as using diversified addresses in hierarchical key derivation, and properly segregating internal and external keys. (Also backported to Sapling components of Unified Addresses!)

Security conditions — a first try

- We can specify some conditions that are definitely necessary for this approach to be secure:
  - Only intended key derivations are feasible, and they are collision-resistant:

    Spending key → Full viewing key → Incoming viewing key → {Addresses}
  - [Balance, View consistency, Spendability] A note commitment tree root binds a set of committed notes.
  - [Balance, View consistency, Spendability] Nodes correctly maintain the note commitment trees and nullifier sets as append-only.
  - [Balance] A spend must prove the note was committed in a previous output, and has the nullifier being revealed.
  - [Balance] Each committed output note must have only one nullifier.
  - [Balance] In each transaction, total value of committed output notes must be no greater than that of spent notes.
  - [View consistency] Information that is successfully decrypted using a viewing key is consistent with that used in other security definitions.
  - [Privacy] Only a holder of a full viewing key for a note can recognize when it is spent.
    - The creator of a note necessarily knows its address, but may not know its full viewing key.
  - [Privacy] Only a holder of an incoming viewing key can recognize / decrypt outputs encrypted to it.
  - [Nonmalleability] Given an honestly created transaction *Tx*, it must not be possible to transplant components of *T* into another valid transaction without already knowing the private information used to create *Tx*.
- However, these don't clearly lay out the adversary's capabilities. How do we know they are sufficient?

Security properties

- Can the adversary construct a valid blockchain where:
  - [Balance] Money is created[!!]?
  - [View consistency] Information obtained by use of viewing keys is inconsistent with a transaction's Effects[!!]?
- Given an oracle for a victim (that privately holds spending keys but is otherwise controlled by the adversary), can the adversary construct a valid blockchain where:
  - [Spendability] An honestly created transaction's Effects[!!] are not as intended?
- Given an oracle for a victim that takes some adversary-influenced[!!] and some private steps, can the adversary construct a valid blockchain where:
  - [Privacy] The adversary gains information of the victim that it shouldn't have?
  - [Nonmalleability] The adversary causes effects that depend on such information? (Caveat: the adversary may create an alternative tx with the same effects as an honest one and, if the honest one is not v5, a different txid.)
- [Diversified address unlinkability] Given an oracle that generates a set of diversified addresses for each of two ivks, can an adversary challenged with another address, for one of the ivks at random, determine which?
- Simplification: Spendability + Balance ⇒ Spend authorization
  - If someone else spends my notes, then assuming Balance, I cannot spend them.

Modelling issues

- The definitions on the last slide present a difficulty in how to define the Effects of a (valid) transaction.

- Suppose we say the Effects are to spend notes with given nullifiers, and create notes with given note commitments.

  - Value commitments, anchors, rk, signatures, enable flags, and proofs only affect whether a transaction is valid, not its Effects.

- What goes wrong with this definition?

  - There might be multiple openings for a note commitment. In fact, *any* input is an opening for some randomness. Also, that randomness can be found for any input by a DL-breaking adversary.

  - At least in Zcash, the security properties that depend on Effects (Balance, View consistency, and Spendability) can't hold against a DL-breaking adversary in any case. But the problem is that we cannot meaningfully talk about "the" opening of a note commitment. There are similar problems with nullifiers.

- It's quite possible for an attack to depend critically on generating transactions that an honest user would never generate, so we must allow the adversary to submit its own "raw" transactions.

Modelling issues

- For simplicity we only model one pool (Sapling or Orchard)[!!]. A more complete model would allow for use of multiple anchors[!!] and pools in a transaction.

- Similarly to Zerocash and [BMM2016], we model ledger interaction via an oracle, which accepts the following requests:
    - CreateAccountHandle() → AccountHandle
    - DeriveAddress(AccountHandle, DiversifierIndex) → Address
    - CreateHonestTransaction(*chain*: ValidChain, *anchor*: Anchor, *spends*: [(AccountHandle, TxHandle, OutputIndex)], *outputs*: [(Amount, Address, Memo)], *expiry*: Option<Height>) → Option<TxHandle>
    - CreateRawTransaction(*serialized*: [u8], *justifications*: [Satisfied(Witness) | Replayed(TxHandle, int) | Invalid]) → Option<TxHandle>
    - GetEncoding(TxHandle) → [u8]

    where
    - type ValidChain = [{*header*: [u8], *transactions*: [TxHandle]}] | (all consensus rules hold)

- The oracle keeps track of mappings AccountHandle → SpendingKey and TxHandle → ([u8], TxWitnesses) that the adversary cannot access directly.

- CreateHonestTransaction checks that all its arguments are valid, and that each note in *spends* is spendable by its account with the given *anchor* and *chain*. If so, it returns a TxHandle for the transaction and hidden witnesses used to create it. Its parameters are meant to capture all ways for an adversary to influence honest transaction creation.

- CreateRawTransaction takes a transaction encoding and justification for each proof it contains. It enforces that for each decoded *instance* and **valid**[!!] *proof*:
    - the *justification* is Satisfied(*witness*) where *instance* and *witness* satisfy the corresponding statement; or
    - the *justification* is Replayed(*handle*, *n*) and *instance* matches the decoded instance at index *n* in the transaction with the given *handle*.

- This models the fact that a Standard adversary[!!] would need to know witnesses in order to create valid proofs for fresh instances[!!], by knowledge soundness. We also allow the adversary to replay instances from previous transactions, but do not require proofs to be the same, capturing the possibility of malleating proofs.

- Compare the functional interface here to the one used in the Zerocash paper, which has *imperative* operations Mint, Pour, Receive, Insert. This is not just a matter of preferred style; it also allows the modelling of rollbacks and front-running. An adversary can (realistically) see an honest transaction before it is mined in the chain, and mine its own transactions first, or after doing a rollback. It can also attempt to re-mine honest transactions at a different position.

- It is up to the definition of security properties, rather than the oracle interface, to constrain which calls are allowed in each security game.

Modelling issues

- We also add operations to selectively reveal information:
  - RevealIncomingViewingKey(AccountHandle) → IncomingViewingKey
  - RevealFullViewingKey(AccountHandle) → FullViewingKey
  - RevealSpendingKey(AccountHandle) → SpendingKey
  - RevealWitnesses(TxHandle) → [Witness]
  - RevealReceivedOutputs(TxHandle, AccountHandle) → [OutputIndex]
  - RevealNotePlaintext(TxHandle, AccountHandle, OutputIndex) → [u8]

- RevealIncomingViewingKey and RevealFullViewingKey will be used later in the definition of Spendability. They can be also used to model intentional disclosure of viewing keys.

- All of these operations also allow modelling situations where information has been compromised, or side channel attacks. For example, a side channel attack against trial note decryption could be modelled using RevealReceivedOutputs. We won't consider this further in this talk.

Modelling issues

- This approach of annotating the ledger with witnesses in order to define transaction Effects, cannot prove security against KS-breaking adversaries.

- However, the Balance, View consistency, and Spendability properties do essentially depend on knowledge soundness of the zk-SNARK, and (not by coincidence) those are also the ones for which we need to know the witnesses associated with each transaction.

- We can prove Privacy and Nonmalleability separately in a way that does not depend on knowledge soundness.

- We do not need (and should not define) oracle requests to compute things that the adversary can compute itself. For example, if the adversary has an IncomingViewingKey then it can receive notes sent to that ivk itself and see all of their fields.

Defining Effects

- The Effects of a transaction in the context of a valid block chain can be determined from its witnesses. In more detail:

  - The Spend-side witnesses directly specify which notes we spend in each transfer.

  - The Output-side witnesses directly specify which notes we output in each transfer.

- So, for a valid block chain it's a simple matter to define which notes are unspent, and therefore what the balance is.

  …

Defining Effects

- The effects of a transaction in the context of a valid block chain can be determined from its witnesses. In more detail:

    - The Spend-side witnesses directly specify which notes we spend in each transfer.

    - The Output-side witnesses directly specify which notes we output in each transfer.

- So, for a valid block chain it's a simple matter <mark>!! !! !! !! !!</mark> to define which notes are unspent, and therefore what the balance is.

    ...

Security definition for Balance

- Define a linearization of transactions in a valid chain. Let Outputs(i) be the set[!!] of positioned[!!] notes that appear in witnesses of outputs in transactions preceding index i. Let NonZeroSpends(i) be the multiset[!!] of non-zero-valued positioned notes that appear in witnesses of spends in transactions preceding index i. Let Issuance(i) be the intended issuance for the relevant block height when i indexes a coinbase transaction, and zero otherwise. Let PoolValueBalance(i) be the total value of notes in Outputs(i) – NonZeroSpends(i).

  - We assume it is a consensus rule that a block only has one coinbase transaction. For simplicity we ignore transparent transfers here except for issuance, and we ignore multiple shielded pools (but it would be fairly easy to extend the definition to handle these).

- **Security game:** The adversary wins if, using only CreateRawTransaction, it outputs a valid chain containing a transaction Tx at index i for which:

  - NonZeroSpends(i+1) is not a subset[!!] of Outputs(i); or

  - PoolValueBalance(i+1) > PoolValueBalance(i) + Issuance(i); or

  - the adversary outputs an ivk and an output of Tx such that Receive applied to that ivk and note ciphertext returns an output note that does not match (in all note fields) the corresponding witness.

- Note that the last condition is essential. The receiver of a note doesn't determine its value by knowing the witness of the output; it determines it by trial decrypting and then checking the decrypted note against the output note commitment. If the note commitment scheme is binding, we can infer that the decrypted note *is* the note for which the Output or Action circuit checked the note commitment against the witness.

  - Sapling and Orchard actually have two Receive functions, one for IVKs and one for FVKs. Both of them check the commitment.

- The above definition allows that Receive can fail to return an output note because the note ciphertext is incorrect, i.e. it only requires the note to match *if* it can be decrypted by *some* incoming viewing key. This is intentional; undecryptable output notes can result in loss of funds, but that is not a Balance violation.

- We allow the adversary to provide any ivk, because we do not want *any* user to believe that they have received funds when they have not.

Security argument for Balance

- The set of notes that have appeared as witnesses in outputs is not definitionally the same as the set of notes committed to by the note commitment tree. That is part of what we will need to prove, based on the Output/Action statement.

- Similarly, the multiset of non-zero-valued notes that have appeared as witnesses in spends is not definitionally the same as the multiset of notes whose nullifiers have been revealed.

  - Actually it isn't the same in practice, but only because of zero-valued notes. We exclude those from NonZeroSpends(i) because they are not required by the Spend or Action statements to match a committed note, and so NonZeroSpends(i+1) might not be a subset of Outputs(i) if we included them.

- [Orchard] To prove that NonZeroSpends(i+1) ⊆ Outputs(i), we need to first show that the witness value nk is determined by the note being spent in an Action. This depends on the fact that ivk is uniquely determined by (g_d, pk_d), and nk is determined by ivk due to binding of Commit^{ivk} (see next slide). If a note were to repeat in the witnesses of two non-zero-valued spends, it would have the same nullifier, computed deterministically from the spent note and nk. Therefore NonZeroSpends(i+1) is a set. Furthermore each non-zero-valued spent note must have a Merkle path from its commitment to some anchor for the final treestate of a transaction before index i. By induction over i (having proven that NonZeroSpends(i) ⊆ Outputs(i-1) ⊆ Outputs(i), and using properties of a Merkle tree over a collision-resistant hash) we infer that NonZeroSpends(i+1) ⊆ Outputs(i).

- The argument that PoolValueBalance(i+1) ≤ PoolValueBalance(i) + Issuance(i) is quite involved due to the use of value commitments, but the hard part of that is covered in § 4.13 & 4.14 of the protocol spec.

- (As we said on the previous slide) since Receive does not return a note without checking its commitment, we know by binding of NoteCommit^{Orchard} that if any note is returned, it must match the witnessed output note. We do not need to reason about the decryption here.

Security argument for Balance [Orchard]

- If there were more than one nullifier for a positioned note, we could forge or double-spend.

- A note commitment (either cm or cm_x)$^{!!}$ binds all fields (g_d, pk_d, v, ρ, ψ) of the note.

- The Action statement checks:

  - $pk_d = [ivk]\, g_d$

    - There is only one such ivk for given (g_d, pk_d), because ivk $\in [0, q_\mathbb{P}) \subseteq [0, r_\mathbb{P})$ and $g_d \neq \mathscr{O}_\mathbb{P}$ (the Pallas group has prime order $r_\mathbb{P}$).

  - $ivk = \text{Commit}^{ivk}_{rivk}(ak, nk)$

    - This binds ak and nk given ivk.

    - Commit^{ivk} is instantiated by SinsemillaShortCommit; binding depends on hardness of DL on Pallas (protocol spec § 5.4.8.4, Thms 5.4.3 & 5.4.4).

  - $nf = \text{DeriveNullifier}_{nk}(\rho, \psi, cm) = \text{Extract}_\mathbb{P}([(PRF_{nk}(\rho) + \psi) \bmod q_\mathbb{P}]\, \mathscr{K}^{Orchard} + cm)$

    - nf deterministically depends only on fields bound to the note.

- We need an anchor for the note commitment tree to be a position-binding vector commitment to its leaves.

  - It is a conventional fixed-depth Merkle tree; this is proven in [García 2005, Lemma 1] given that the tree's hash function SinsemillaHash is collision-resistant (Thm 5.4.3 and Thm 5.4.4), which relies on hardness of DL on Pallas.

  - The leaves of the tree are x-coordinates of note commitments, $cm_x = \text{Extract}^{\perp}_\mathbb{P}(cm)$. However the argument in Thm 5.4.3 extends to adding an independent randomness base, and so these short commitments are also binding (I should write this out explicitly in § 5.4.8.4).

  - The Uncommitted^{Orchard} value used to fill the tree should not match any actual short note commitment (Thm 5.4.6).

  - Collisions on nf cannot be used to violate Balance (they only affect Spendability, as we'll see next).

Security argument for Balance [Sapling]

- If there were more than one nullifier for a positioned[!!] note, we could forge or double-spend.

  - In Sapling it is possible to have two notes with the same commitment. But when a note is spent, we must witness both the commitment opening and the position, and the note commitment tree is position-binding.

- A note commitment (either cm or cm_u)[!!] binds all fields (g_d, pk_d, v) of the note.

- The Action statement checks:

  - $pk_d = [ivk]\ g_d$

    - There is only one such ivk for given (g_d, pk_d), because ivk $\in [0, 2^{251}) \subseteq [0, r_{\mathbb{J}})$ and g_d has order at least $r_{\mathbb{J}}$.

  - ivk = CRH^{ivk}(ak, nk)

    - This binds ak and nk given ivk.

    - CRH^{ivk} is instantiated by BLAKE2s-256 truncated to 251 bits, which can reasonably be assumed collision-resistant.

  - nf = $PRF^{nfSapling}_{nk\star}(\rho\star)$ for $\rho\star = repr_{\mathbb{J}}$(MixingPedersenHash(cm, pos))

    - nf deterministically depends only on fields bound to the positioned note.

- We need an anchor for the note commitment tree to be a position-binding vector commitment to its leaves.

  - It is a conventional fixed-depth Merkle tree; this is proven in [García 2005, Lemma 1] given that the tree's hash function PedersenHash is collision-resistant (§ 5.4.1.7 and Thm 5.4.1), which relies on hardness of DL on Jubjub.

  - The leaves of the tree are u-coordinates of note commitments, $cm_u = Extract_{\mathbb{J}^{(r)}}$(cm). However $Extract_{\mathbb{J}^{(r)}}$ is injective (Thm 5.4.8), and so these short commitments are also binding (I should say so explicitly in § 5.4.8.2).

  - The Uncommitted[Sapling] value used to fill the tree should not match any actual short note commitment (Thm 5.4.5).

  - Collisions on nf cannot be used to violate Balance (they only affect Spendability, as we'll see next).

Security definition for Spendability

- [Spendability] Informally: given any correctly balanced (nonempty set of spendable[!!] notes N with corresponding spending keys K, set of valid output note plaintexts P, valid anchor A, and valid expiration height H), an honest holder of K can construct a transaction Tx that will successfully spend N to P if mined after[!!] A and no later[!!] than height H, even if Tx is revealed and then arbitrary other transactions (constructed without using the honest user to double-spend any note in N) appear on the ledger after A and before Tx.
  - A failure of this property that depends on transactions appearing after A is called a "roadblock attack". A failure due to distinct notes having the same nullifier is called a "Faerie Gold attack".

- **Security game:** The adversary prepares a chain state $Chain$ with final anchor A using the oracle. It is allowed to call CreateAccountHandle, DeriveAddress, CreateHonestTransaction, CreateRawTransaction, GetEncoding, RevealIncomingViewingKey, and RevealFullViewingKey. It controls what the honest user does and knows everything except the honest user's spending keys.

  The adversary then picks (a nonempty subset of output notes N in $Chain$ with account handles, that are each receivable with an IVK or FVK derived from the corresponding account and not honestly spent in $Chain;$ a valid set of note plaintexts $P;$ and a valid expiry height H); such that N and P are correctly balanced. It calls CreateHonestTransaction with (some state extending $Chain$[!!], A, N, P, H). It wins if CreateHonestTransaction fails. Otherwise let Tx be the resulting handle; in that case the adversary wins if it can find a state extending $Chain$ and no later[!!] than height H in which (notes in N are still not honestly spent) and (Tx is invalid or its Effects are not to spend N with outputs P).

  - This security property ensures that a note receivable by an account will continue to be spendable from that account, until it is honestly spent. The converse does not hold; a spendable note may not be receivable because its note ciphertext is incorrect. By design, the Output/Action circuits do not check receivability.
  - Together with Balance, this proves Spend authorization (because an adversary that breaks Spend authorization could double-spend from N, making Tx invalid).
  - This strengthens "Completeness" in [BCGGMTV2014], which only requires Tx to be valid when mined immediately and so doesn't model roadblock attacks. Also, the security game allows picking N from the set of notes that have not been *honestly* spent, allowing that the adversary may be able to spend them via some vulnerability. (The Completeness definition assumed that if a note's nullifier had appeared on the ledger then it was honestly spent, which was the flaw that allowed Completeness to hold for Zerocash despite it being vulnerable to Faerie Gold attacks.)
  - Notice that the adversary is allowed to ask the oracle to spend notes with the same hidden spending key as a note in N, in both phases of the attack (before and after knowing Tx).

Security argument for Spendability [Orchard]

- The meat of the argument is to show, given that the adversary does not know the spending keys for N and that they are not allowed to spend those notes via CreateHonestTransaction, that the validity and Effects of Tx only depend on the state of $Chain$, and will not be affected by subsequent chain state (provided that Tx is mined no later than height H).

- There could be various reasons why Tx would not be valid depending on the detail of consensus rules, but the main one we'll consider here is that nullifiers of notes in N could be revealed by intermediate transactions.

- Recall that nf = $\text{DeriveNullifier}_{nk}(\rho, \psi, cm) = \text{Extract}_{\mathbb{P}}([(\text{PRF}_{nk}(\rho) + \psi) \bmod q_{\mathbb{P}}]\, \mathcal{K}^{\text{Orchard}} + cm)$.
  We need to show that two committed notes cannot have the same nullifier, in order to rule out Faerie Gold attacks.

- $\mathcal{K}^{\text{Orchard}}$ is independent of any of the bases used in NoteCommit$^{\text{Orchard}}$. Therefore, DeriveNullifier acts as a collision-resistant Pedersen hash that is binding on the note fields (by an argument we also used for Balance), and therefore there cannot be two *distinct* notes with the same nullifier, assuming hardness of DL on Pallas.

- This still leaves the possibility that *identical* notes could occur at different positions, since the position is not bound by the note commitment. However, a unique nullifier nf$^{\text{old}}$ from the spend side of an Action is enforced to be used for the ρ field of the note on the output side, which is therefore unique (this holds even if the spent note was a dummy).

- In order to be able to argue that the adversary cannot spend a note itself to prevent the honest transaction Tx from spending it, we also need unforgeability of RedPallas and correct use of randomization for spending keys (ak).

- "A note that is receivable by an account's IVK or FVK must be spendable from that account until it is honestly spent" follows by construction from the definitions of "receivable", "spendable", and "spent", and properties from the argument used for Balance.
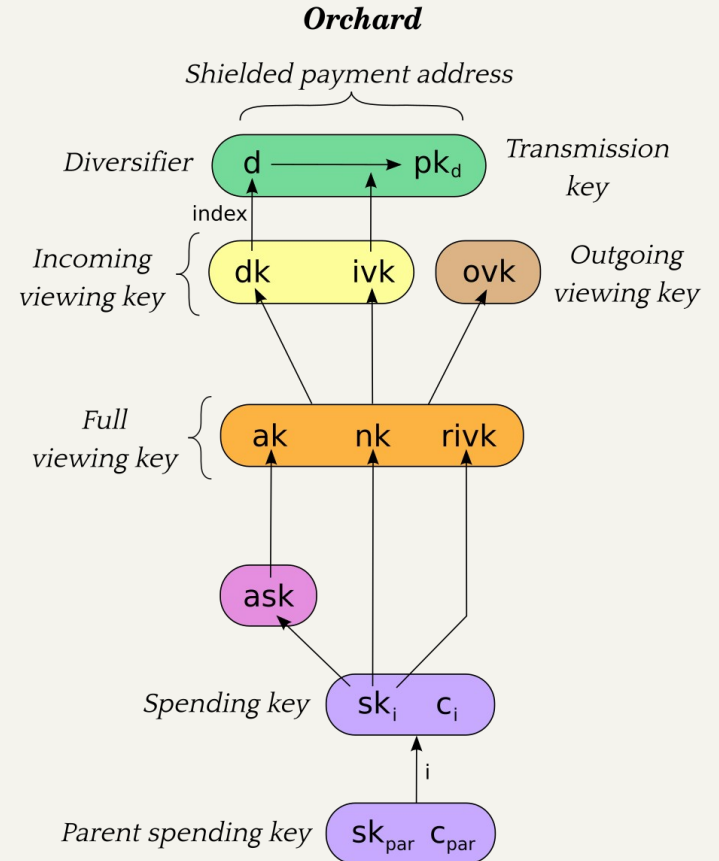
Security argument for Spendability [Sapling]

- As in the case of Orchard, there could be various reasons why *Tx* would not be valid depending on the detail of consensus rules, but the one we'll consider here is that nullifiers of notes in *N* could be revealed by intermediate transactions.

- Recall that $nf = PRF^{nfSapling}_{nk\star}(\rho\star) = BLAKE2s\text{-}256($ "`Zcash_nf`" $, LEBS2OSP_{256}(nk\star) \parallel LEBS2OSP_{256}(\rho\star))$

- where $\rho\star = repr_{\mathbb{J}}(\rho)$ and $\rho = MixingPedersenHash(cm, pos) = cm + [pos]\,\mathscr{J}^{Sapling}$ for note commitment cm at position pos.
  We need to show that two distinct *positioned* notes cannot have the same nullifier, in order to rule out Faerie Gold attacks.

- The security requirement for MixingPedersenHash says that its application to $cm = WindowedPedersenCommit_r(M)$ and x is collision-resistant on (r, M, x). The base $\mathscr{J}^{Sapling}$ is independent of any of the bases used in WindowedPedersenCommit, and all of these are in the large prime-order subgroup of Jubjub, so by the argument in [BGG1995, Appendix A], this security requirement holds assuming hardness of DL on that subgroup.

- Since there can be only one note at a given position in a valid chain, distinct positioned notes necessarily have different pos, and so different ρ (and different $\rho\star$, since $LEBS2OSP_{256}$ is injective), under the same assumption.

- Assuming also that BLAKE2s-256 is collision-resistant for a given personalization, this implies that there cannot be two distinct positioned notes with the same nullifier.

- In order to be able to argue that the adversary cannot spend a note itself to prevent the honest transaction *Tx* from spending it, we also need unforgeability of RedJubjub and correct use of randomization for spending keys (ak).

- "A note that is receivable by an account's IVK or FVK must be spendable from that account until it is honestly spent" follows by construction from the definitions of "receivable", "spendable", and "spent", and properties from the argument used for Balance.

View consistency

- "Information that is successfully decrypted using a viewing key is consistent with that used in other security definitions."

- Sorry, the detail on this got cut due to lack of time.

- [Sapling] See ZIP 310 for what it is supposed to cover.
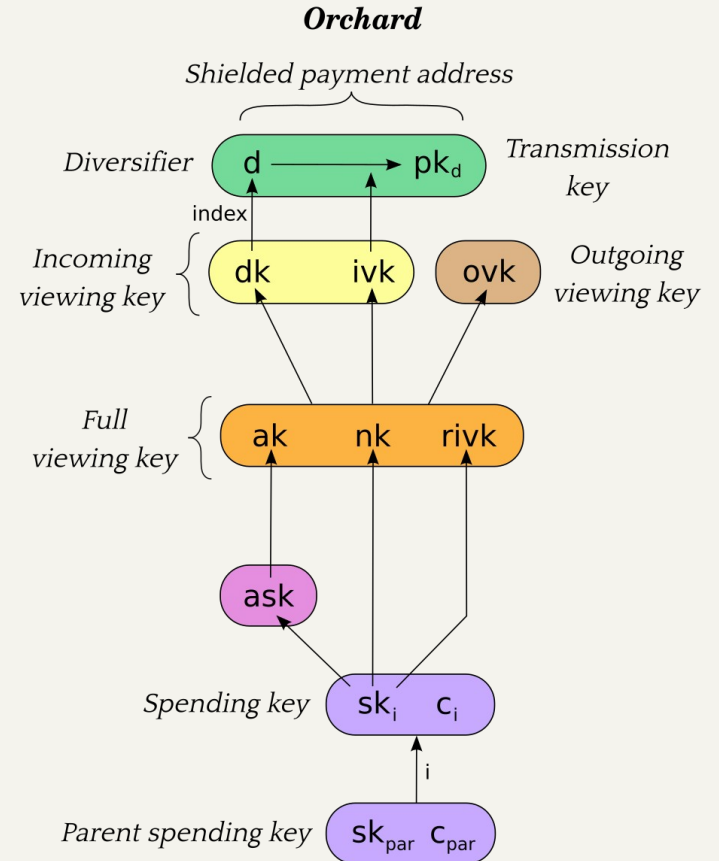
- [Orchard] See zcash/zips#606 (no detail yet).

Key separations [Orchard]

- [(d, pk$_d$, index), ...] $\nrightarrow$ dk, ovk

- (dk, ivk, ovk) $\nrightarrow$ ak, nk, rivk

  - We also have (dk, ivk) $\nrightarrow$ ovk and ovk $\nrightarrow$ dk, ivk

- [(d, pk$_d$, index), ...] $\nrightarrow$ ivk

- ak $\nrightarrow$ ask

- (ask, nk, [Orchard] rivk) $\nrightarrow$ sk, c$_i$

- ZIP 32 Child $\nrightarrow$ Parent or Sibling

- Key entropy preservation

Orchard

Shielded payment address

Diversifier — d → pk$_d$ — Transmission key

Incoming viewing key { dk   ivk }   ovk — Outgoing viewing key

index

Full viewing key { ak   nk   rivk }

ask

Spending key — sk$_i$ c$_i$
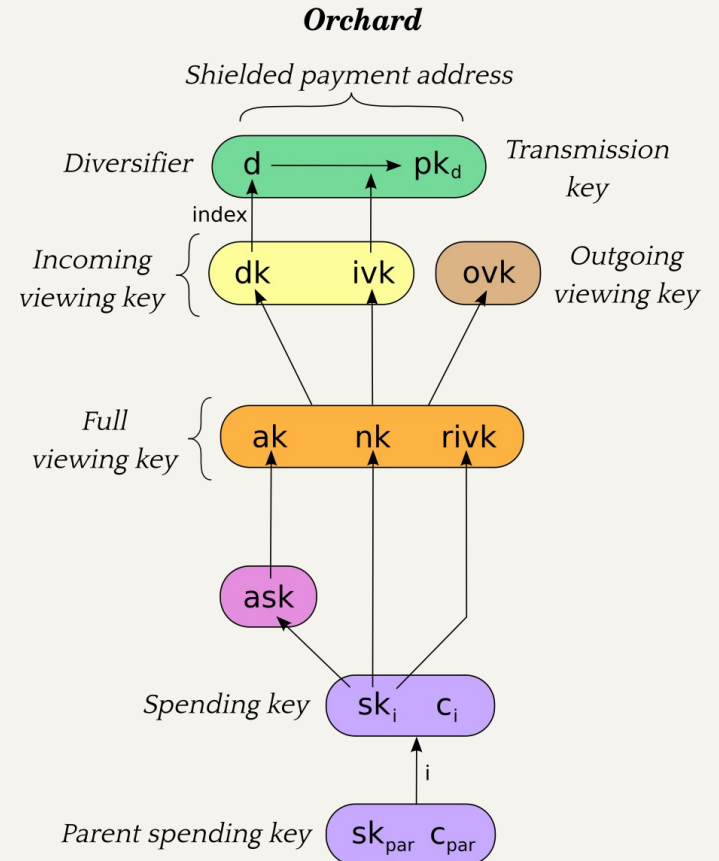
i

Parent spending key — sk$_{par}$ c$_{par}$

Key separations [Orchard]

- [(d, pk$_d$, index), ...] ↛ dk, ovk depends on
  - known-plaintext key recovery security of FF1-AES;
    - 88-bit Feistel cipher based on AES
    - [NIST2016] Appendix A and [BRS2010]
  - ivk ↛ ak, nk, rivk;
    - Notice (d, pk$_d$) ↛ ivk is not post-quantum, but ivk ↛ ak, nk, rivk is (next slide).
  - PRF^{expand} entropy preservation (so that dk and ivk have high entropy);
  - ak, nk, rivk entropy.

Orchard

Shielded payment address

Diversifier d → pk$_d$ *Transmission key*

index

Incoming viewing key { dk   ivk   ovk } *Outgoing viewing key*

Full viewing key { ak   nk   rivk }

ask

Spending key sk$_i$ c$_i$

i

Parent spending key sk$_{par}$ c$_{par}$

Key separations [Orchard]

- (dk, ivk, ovk) ↛ ak, nk, rivk depends on

  - an ad-hoc (but reasonable) assumption on BLAKE2b-512;

    - Adversary breaks DL on ivk to get ⟨encode(ak, nk), **b**⟩ + rivk where **b** is the vector of discrete logarithms of Sinsemilla bases wrt the randomness base $\mathscr{R}$.

    - ak, nk, and rivk are each ~254 bits hence resistant to Grover search. So if BLAKE2b-512 is a good hash function we should expect that given (dk, ovk, ⟨encode(ak, nk), **b**⟩ + rivk) where dk ∥ ovk = PRF^{expand}_{rivk}(ak, nk), against a quantum adversary (ak, nk, rivk) appear computationally random.

  - ak, nk, rivk entropy.

- (dk, ivk, ovk) ↛ ak, nk, rivk depends on

  - the same assumption on BLAKE2b-512 classically, or a weaker assumption + hardness of DL;

  - ak, nk, rivk entropy.

Orchard

Shielded payment address

Diversifier — d → pk_d — Transmission key

index

Incoming viewing key — dk   ivk   ovk — Outgoing viewing key

Full viewing key — ak   nk   rivk

ask

Spending key — sk_i c_i
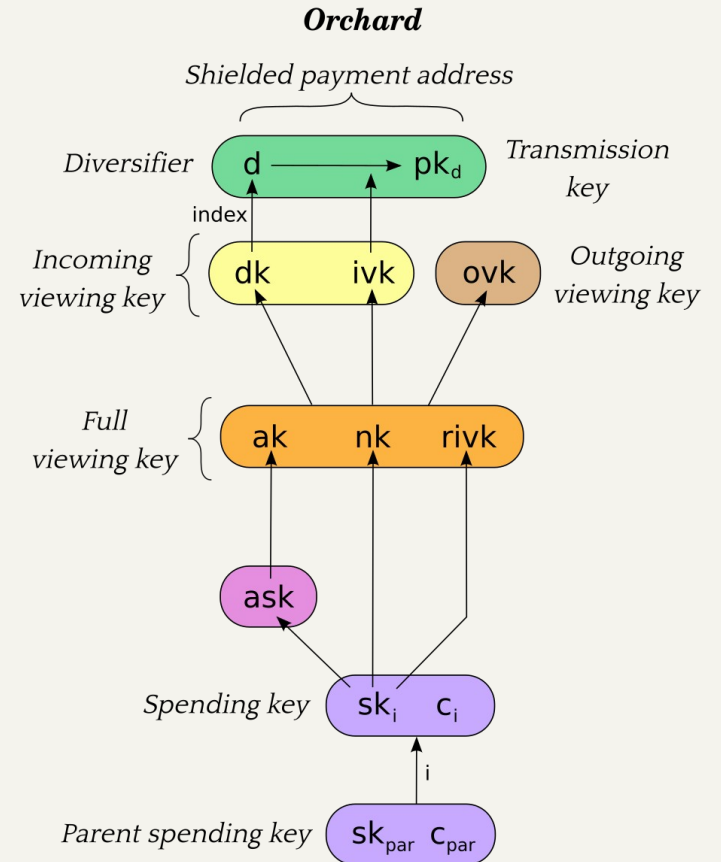
i

Parent spending key — sk_{par} c_{par}

Key separations [Orchard]

- $[(d, pk_d, index), ...] \nrightarrow ivk$ depends on

  - known-plaintext key recovery security of FF1-AES;

  - hardness of DL;

  - PRF^{expand} entropy preservation;

  - ak, nk, rivk entropy.

- $ak \nrightarrow ask$ depends on

  - hardness of DL;

  - PRF^{expand} entropy preservation;

  - sk_i, c_i entropy.
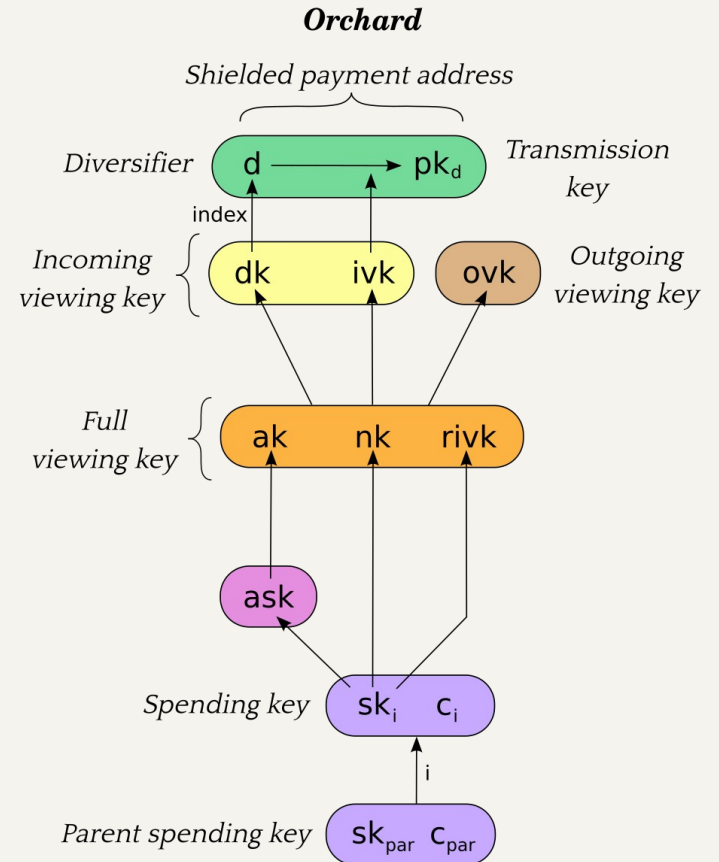
Key separations [Orchard]

- $(ask, nk, rivk) \nrightarrow sk, c_i$ depends on
  - PRF^{expand} as a KDF;
  - sk, c_i entropy.
- ZIP 32 Child $\nrightarrow$ Parent or Sibling
  - Omitted due to lack of time.

Orchard

Shielded payment address

Diversifier — d → pk_d — *Transmission key*

index

Incoming viewing key { dk   ivk }   ovk   *Outgoing viewing key*

Full viewing key { ak   nk   rivk }

ask

Spending key sk_i c_i

i

Parent spending key sk_{par} c_{par}

Key entropy preservation [Orchard]

- $(sk_i, c_i) \rightarrow$ ask, nk, rivk

  - No significant entropy loss at the desired security level because (sk_i, c_i) are 512 random bits.

- (ak, nk, rivk) $\rightarrow$ dk, ivk, ovk

  - 1-bit entropy loss for ivk due to $Commit^{ivk}$ outputting a Pallas *x*-coordinate. There is no known way to exploit this even slightly, relative to 126-bit security against Pollard rho attacks. If a Pollard kangaroo-style algorithm were applicable, it could at most reduce security by half a bit.

  - No significant entropy loss for dk or ovk.

- (dk, ivk) $\rightarrow$ (d, pk_d)

  - Modelling FF1-AES as a PRP, d has 88 bits of entropy for a *given* dk. However, even for a quantum adversary this is not searchable when dk (which is 256 random bits) is unknown.

- ZIP 32 $(sk_{par}, c_{par}) \rightarrow (sk_i, c_i)$

  - For Sapling there was a potential bottleneck in $ToScalar^{Sapling} \circ PRF^{expand}$ but it is negligible (see sapling_entropy.py). In any case it is fixed for Orchard.

Privacy

- Zcash transactions leak the following metadata:
    - Transaction version (v4 or v5, as of NU5)
    - Lock time (rarely used; may be a distinguisher if it is)
    - Expiry height and anchor position
        - These gives information about what block height the creator was synced to, and some policy information.
    - Transparent inputs and outputs
    - Public value balances
        - Together with the transparent inputs and outputs, these determine the fee and the amount being transferred between pools.
        - The fee is dependent on policy, but for a given policy we attempt to make it only depend on other already-leaked metadata.
    - Numbers of JoinSplits, Spends, Outputs, and Actions
        - This could correlate with other information under certain circumstances. For example in a "dusting attack", the adversary sends a victim lots of small-valued notes (or notes with strategically chosen values), increasing the probability that the victim's transactions will have a larger number of spends than other transactions (or an identifiable number of spends). There are note management strategies that can mitigate this, but they have not been implemented yet.
    - Which spends of given notes are repeated across transactions
        - This may happen because a previous transaction expired and the user is trying to spend some of the same notes.
    - Whether the transaction is coinbase
    - For coinbase transactions, the amounts / destination addresses / memos of shielded outputs
    - Orchard flags (enableSpends/enableOutputs)
        - Under normal circumstances these only depend on whether the transaction is coinbase.

Privacy

- Overall, Zcash – like most other deployed cryptocurrencies – is not designed to resist attacks using quantum computers.

- However, it has a limited form of resistance to quantum attacks against the privacy of shielded transactions in cases where the destination address is treated as a shared secret between sender and recipient.

- Some interesting privacy properties are:

  - Note Privacy Without Address: if I do not have addresses for some subset of users, can I gain information other than leaked metadata about transactions sent to those addresses (other than that the destination addresses are in the subset), only from the public ledger?

  - Note Privacy: can I gain information other than leaked metadata, only from the public ledger?

  - 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?

- We assume that an honest transaction creator has a good RNG.

- We do not use the indistinguishability-based definition from the Zerocash paper. Instead we enumerate leaked metadata (as on the previous page) and then consider other potential sources of information leakage individually. If you look at the structure of the Ledger Indistinguishability proof (Appendix D.1), the meat of it is to show that such sources are jointly indistinguishable from a uniform distribution on some fixed sets. Our approach achieves the same effect but is IMHO clearer and more modular.

- We do not use the oracle defined earlier, because it assumes knowledge soundness, and we want to analyse privacy properties without that assumption.

Privacy

- Components of a fully shielded Sapling or Orchard transaction that could in principle leak additional information:

- Value commitments for spent and output notes, and binding signatures

  - The value commitments are perfectly blinded on their own, but we need to consider whether information could be jointly leaked by the value commitments and binding signature (because a DL-breaking adversary can undo the scalar multiplication $[r]\,\mathscr{P}_{\mathbb{G}}$ in RedDSA and so find the secret key of the binding signature).

  - A careful analysis shows that no more information is leaked against a DL-breaking (or quantum) adversary than that the transaction balances.

- Other spend-related fields

  - Proofs: Groth16 and Halo 2 are perfectly zero knowledge

  - Spend auth signature and its public key (rk)

    - The private key rsk is perfectly blinded, given that the proof perfectly hides the randomizer α.

    - The signature is on a message that is public (the transaction hash) with a key that is public. A DL-breaking adversary can find rsk but that is fine, since rsk is per-signature. (This is elegant; the randomization gives us post-quantum key privacy even when using a DL-based signature algorithm.)

Privacy

- Other spend-related fields (continued)
  - Nullifier of the spent note [Orchard]
    - nf = DeriveNullifier$_{nk}$(ρ, ψ, cm) = Extract$_{\mathbb{P}}$([(PRF$_{nk}$(ρ) + ψ) mod $q_{\mathbb{P}}$] $\mathcal{K}^{Orchard}$ + cm).
    - The distribution of (PRF$_{nk}$(ρ) + ψ) mod $q_{\mathbb{P}}$ takes a slightly smaller range than [0, $r_{\mathbb{P}}$), but negligibly so compared to $r_{\mathbb{P}}$. (Thanks, Hasse-Weil bound!)
    - This construction is designed so that if either Poseidon is a secure PRF (which is a reasonable post-quantum assumption), or Poseidon is non-trivial and assuming DDH on Pallas, nf will look like a randomly distributed Pallas x-coordinate.
    - Therefore, only a holder of the nk element of a full viewing key for a note can determine when it is spent.
- Other output-related fields
  - Note commitments of output notes
    - These are perfectly blinded.

Privacy

- Note ciphertexts and Output ciphertexts (including epk)

  - Against a Standard adversary, these use an ephemeral Diffie-Hellman-based encryption with ChaCha20-Poly1305, which should be secure.

  - Given the destination address, the ephemeral Diffie-Hellman-based encryption can be broken by a DL-breaking adversary.

  - If we are a DL-breaking (or quantum) adversary that knows (g_d, pk_d) for some d, then we can find ivk and can decrypt any associated note plaintext (just as if had learned ivk legitimately).

  - On the other hand, suppose we are a DL-breaking (or quantum) adversary that does not know (g_d, pk_d) for *any* d. In that case, for each note ciphertext we are given epk = [esk] g_d which is a uniform-random group element (because esk is uniform on the scalar field), and a ChaCha20-Poly1305 encryption of the note plaintext with KDF(sharedSecret, epk) where sharedSecret = [ivk] epk.

    - It does not matter that sharedSecret is also equal to [esk] pk_d. No advantage is gained by g_d being in an 88-bit subspace.

  - $KDF^{Orchard}$ is computed as a BLAKE2b-256 hash of sharedSecret and epk. Under a reasonable one-wayness assumption on $KDF^{Orchard}$ as a seeded hash, we do not get sufficient information about the shared secret itself to perform any DL attack.

  - Output ciphertexts use only symmetric encryption, and ChaCha20-Poly1305 can reasonably be assumed to be a post-quantum secure encryption algorithm.

Privacy

- Note that in order for it to be reasonable to assume that a quantum adversary might not have access to a substantial proportion of user addresses, it is necessary that they cannot get from the ivk of a known address to other elements in the key structure, i.e. {ak, nk, ovk, rivk, dk}. We covered this in the slides on key separations.

- Therefore, a DL-breaking address holder can only see incoming transfers to that address (i.e. what a legitimate holder of ivk would see), not outgoing ones, and not including the addresses of the other senders provided they are not included in memos.

- If aiming for post-quantum privacy, you should not include addresses in memos. This is not an onerous additional restriction if you need to keep addresses secret anyway.

Diversified address unlinkability

- **Security game**: The adversary is given an oracle that can obtain diversified addresses for honest users' accounts (i.e. it is allowed to calls CreateAccountHandle and DeriveAddress). It is then challenged with a new address from one of the calls to CreateAccountHandle at random, and wins if it can tell which one.

  - Its may be sufficient to restrict to two calls to CreateAccountHandle, since allowing more is not likely to change the possibility of attack (although it may affect its cost).

- There is a security argument for a more restricted version of this property in § 5.4.1.6 of the protocol spec, but it is slightly flawed and handwavy:

  - The security game described in the spec does not allow the adversary to obtain multiple addresses for the same account. That would be okay if diversified addresses were re-randomizable, but they intentionally are not. (This was to address a potential attack that involves sending the note ciphertext with a different g_d than the one in the note and then observing whether decryption succeeds.)

  - The spec describes a reduction from its security game to key privacy of ElGamal encryption, but this depends on the fact that g_d is chosen uniformly. In practice it is not; it is an output of DiversifyHash on an 88-bit value that is an output of FF1-AES. This looks solvable: to fix it we would need to rely on FF1-AES being a PRP.

- It might be more realistic to give the adversary more capabilities. However, access to CreateRawTransaction and RevealReceivedOutputs is too strong and makes the security property unachievable.

- Further detail cut for lack of time.

Cryptographic building blocks

	Sprout	Sapling	Orchard
Application curve	—	Jubjub	Pallas
Note encryption	DHAES / Curve25519 ChaCha20-Poly1305	DHAES / Jubjub ChaCha20-Poly1305	DHAES / Pallas ChaCha20-Poly1305
Signatures	EdDSA / Ed25519	RedDSA / Jubjub	RedDSA / Pallas
Proof system	Groth16 / BLS12-381 since Sapling activation	Groth16 / BLS12-381	Halo 2 / Vesta
Arithmetization	R1CS	R1CS	PLONKish
Merkle hash	SHA256Compress	Bowe–Hopwood	Sinsemilla
Note commitments	SHA-256	Bowe–Hopwood	Sinsemilla
Value commitments	—	Pedersen	Pedersen
PRFs	SHA256Compress	BLAKE2s, BLAKE2b	Poseidon, BLAKE2b
PRP	—	FF1-AES	FF1-AES

Cryptographic strength

- Zcash is estimated to achieve a security level against classical (non-quantum) attacks of 125 bits — i.e. we believe an attacker would need to do at least approximately 2^{125} work to break any intended security property after the Sapling upgrade.
  - For this estimate, we take as the unit of work a hash function evaluation or a scalar field operation in a field of around 256 bits.
- This depends on the following estimated strengths for elliptic curves:
  - secp256k1: Pollard rho security 127.0 bits
  - Curve25519: Pollard rho security 125.8 bits
  - [Used only for Sprout before Sapling activation] BN-254 ($\mathbb{G}_1$ and $\mathbb{G}_2$ curves): Pollard rho security 126.6 bits
    - Cheon attack security $2^{124.6}$ scalar field operations (estimate from here)
    - SNTFS (Special Tower Number Field Sieve) security approximately 102 bits (estimate from Table 8 of [GS2021]).
  - BLS12-381 ($\mathbb{S}_1$ and $\mathbb{S}_2$ curves): Pollard rho security 127.3 bits
    - Cheon attack security $2^{125.2}$ scalar field operations (estimate from here)
    - STNFS security approximately 126 bits (estimated using code by Guillevic and Singh, consistent with [GS2021])
  - Jubjub: Pollard rho security 125.8 bits
  - Pallas and Vesta: Pollard rho security 126.0 bits
- Generally, components that use elliptic curves have proven security reducible to the discrete log problem on those curves, *or* have been analysed in the generic group model.

Cryptographic strength

- Note encryption:

  - The security of Diffie–Hellman-based encryption on the Curve25519, Jubjub, or Pallas elliptic curves should be equal to the corresponding Pollard rho estimates for those curves.

  - Because the ranges of scalars used for note encryption are slightly smaller than the full scalar ranges, Pollard kangaroo attacks are also possible, but they do not improve on Pollard rho:

    - For Sprout, the scalar range is 253 bits which gives an expected cost of 2^{127} group operations (from [Teske2003]) for Pollard kangaroo on Curve25519, compared to $2^{125.8}$ for Pollard rho.

    - For Sapling, the scalar range is 251 bits which gives an expected cost of 2^{126} group operations for Pollard kangaroo on Jubjub, compared to $2^{125.8}$ for Pollard rho.

    - For Orchard the scalar range is nearly the full range, so Pollard kangaroo attacks are not applicable.

  - IETF_CHACHA20_POLY1305: authenticated encryption with 256-bit keys (security > 128 bits).

Cryptographic strength

- Sprout and Sapling shielded protocols

  - The best known *cryptanalytic* attack against Sprout (after Sapling activation) and Sapling is a Cheon attack on BLS12-381, which computes discrete logarithms taking into account information provided by the Sapling parameters. This gives an expected security level of 125.2 bits in terms of scalar field operations.

  - There are also hash function collision attacks against Sprout on SHA-256 truncated to 253 bits, and against Sapling on BLAKE2s truncated to 251 bits. These would constrain the expected security level to 126.5 bits for Sprout and 125.5 bits for Sapling.

  - After the Sapling upgrade, the proof system used for both the Sprout and Sapling shielded protocols is Groth16; a relatively well-understood zk-SNARK that is used by many projects and that has two independent security proofs [BGM2017] [Maller2018]. This proof system is dependent on the Sapling MPC trusted setup. Groth16 has unconditional zero knowledge (i.e. not dependent on the security of the BLS12-381 curve).

- Orchard shielded protocol

  - The best known *cryptanalytic* attack against Orchard is a Pollard rho attack against the Pallas or Vesta curves, giving an expected security level of 126.0 bits.

  - The proof system used for the Orchard shielded protocol is Halo 2. This is a relatively new system, but has a security proof that has undergone a third-party audit (not yet published). Deployment of Orchard on the Zcash mainnet was conditional on that audit finding no significant vulnerabilities. The soundness of Halo 2 does not require a trusted setup. Like Groth16, Halo 2 has unconditional zero knowledge (not dependent on the security of the Pallas or Vesta curves).

Orchard Action Circuit
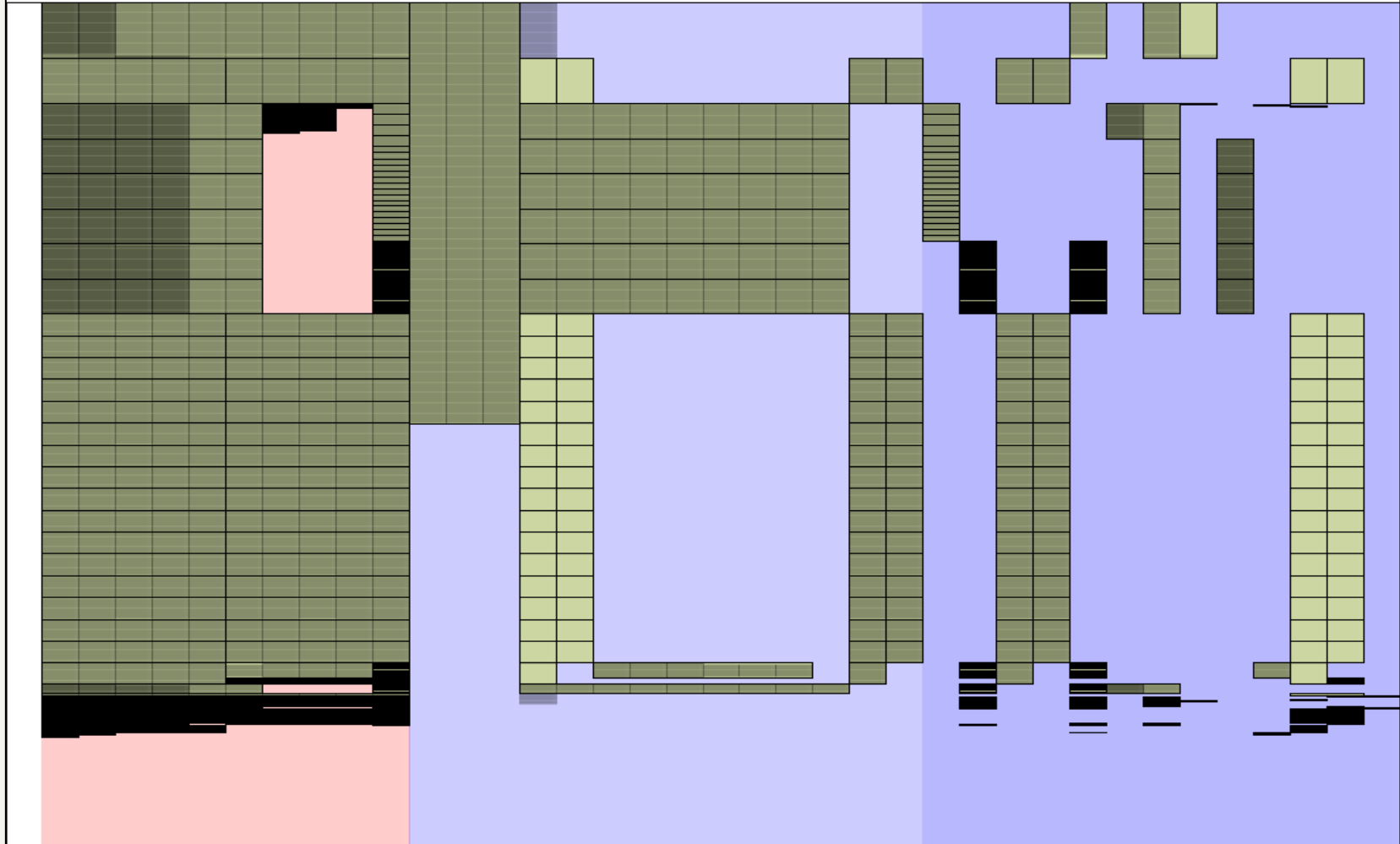
Security assumptions

	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 $\{PRF_{nk}(x): nk \in \mathbb{F}\}$ for random $x \in \mathbb{F}$ gives an adequate scalar distribution for DDH_{Pallas}.
‡ Statistical distance $<2^{-167.8}$ from perfect.

Questions?

Daira-Emma Hopwood (ze/hir)

<daira-emma@electriccoin.co>
@daira.bsky.social

Slides at https://github.com/daira/zcash-security