Lesson 11

This Week

ZCash / Bulletproofs / Monero
Aztec / SONIC / PLONK etc.
Data privacy / proof of computation
STARK theory / Standards / Circuit libraries / ZKP Languages

Starknet Update

See https://swagtimus.substack.com/p/starknet-roundup-7?s=w

1. StarkEx has more transactions than ETH mainnet
   Weekly count: 11.6M vs. 8.5M.

2. [Starknet Scaffold] (https://github.com/tarrencev/starknet-scaffold)

3. Starknet Hackathon

4. Modular Contract approach
   Based on :
   Extensibility Pattern

   "One of the best approaches to minimize introducing bugs is to reuse existing, battle-tested code, a.k.a. using libraries. But code reutilization in StarkNet's smart contracts is not easy:

   - Cairo has no explicit smart contract extension mechanisms such as inheritance or composability
   - Using imports for modularity can result in clashes (more so given that arguments are not part of the selector), and lack of overrides or aliasing leaves no way to resolve them
   - Any `@external` function defined in an imported module will be automatically re-exposed by the importer (i.e. the smart contract)

   The idea is to have two types of Cairo modules: libraries and contracts.
   Libraries define reusable logic and storage variables which can then be extended and exposed by contracts.
   Contracts can be deployed, libraries cannot.

Open Zeppelin Cairo Contracts

- Account
- ERC20
- ERC721
- Contract extensibility pattern
- Proxies and upgrades

- Utilities

Curve 25519 security level

In general
If the curve order has a large prime factor around 2^{2k} then the best known attacks against it take $O(2^k)$ time (where the O-notation conceals a constant which is a bit larger than 1, but is constant for the various types of curves), and so by convention, we say that such a curve as "k-bits of strength".
For Curve 25519, the curve order has a prime factor circa 2^252; by the above standard metric, this yields a security of 126 bits.

Spec
Stack Exchange question
See adversarial advantage
Also see this paper

ZCash

We will look at 3 approaches to adding privacy or confidentiality to financial assets

- ZCash
- Monero
- Aztec

Background

https://github.com/ethereum/EIPs/issues/1724
This EIP defines the standard interface and behaviours of a confidential token contract, where ownership values and the values of transfers are encrypted.

```
interface zkERC20 {
   event CreateConfidentialNote(address indexed _owner, bytes _metadata);
   event DestroyConfidentialNote(address indexed _owner, bytes32 _noteHash);

   function cryptographyEngine() external view returns (address);
   function confidentialIsApproved(address _spender, bytes32 _noteHash) external
view returns (bool);
   function confidentialTotalSupply() external view returns (uint256);
   function publicToken() external view returns (address);
   function supportsProof(uint16 _proofId) external view returns (bool);
   function scalingFactor() external view returns (uint256);

   function confidentialApprove(bytes32 _noteHash, address _spender, bool _status,
bytes _signature) public;
   function confidentialTransfer(bytes _proofData) public;
   function confidentialTransferFrom(uint16 _proofId, bytes _proofOutput) public;
}
```

compare this to the ERC20 interface

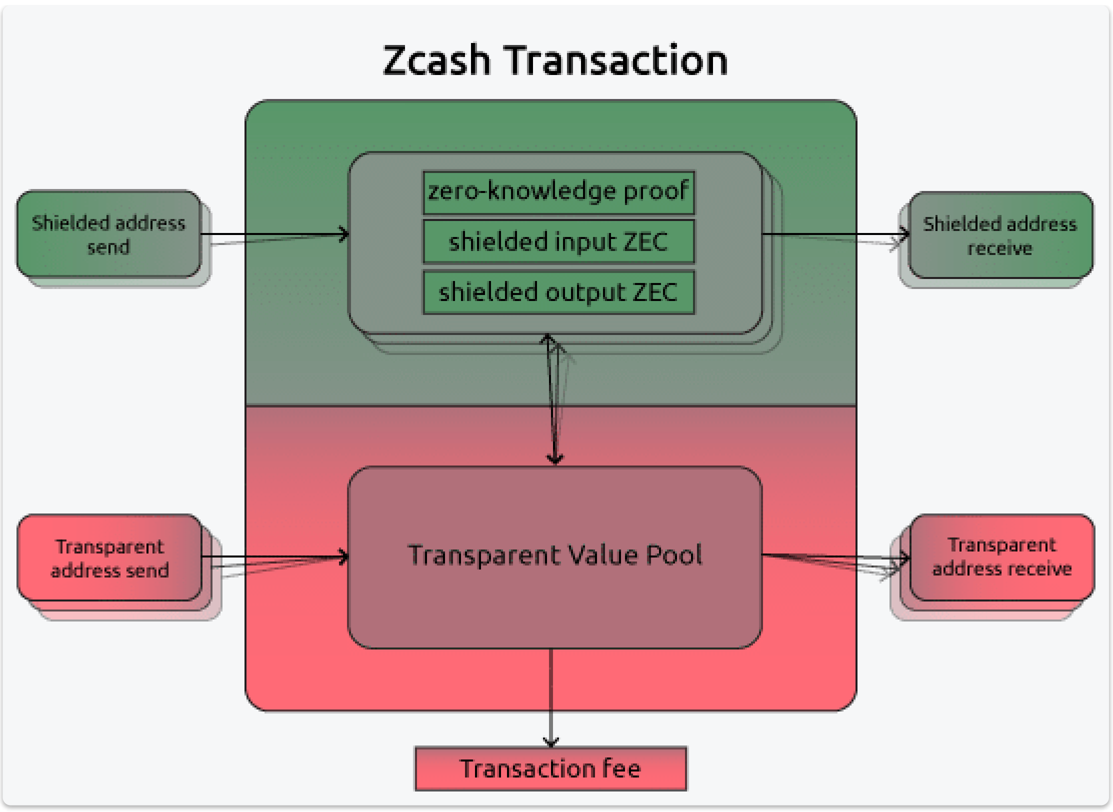
```
interface IERC20 {

function totalSupply() external view returns (uint256);
function balanceOf(address who) external view returns (uint256);
function allowance(address owner, address spender)
external view returns (uint256);

function transfer(address to, uint256 value) external returns (bool);
function approve(address spender, uint256 value)
external returns (bool);

function transferFrom(address from, address to, uint256 value)
external returns (bool);

event Transfer(
address indexed from,
address indexed to,
uint256 value
);

event Approval(
address indexed owner,
address indexed spender,
uint256 value
);

}
```

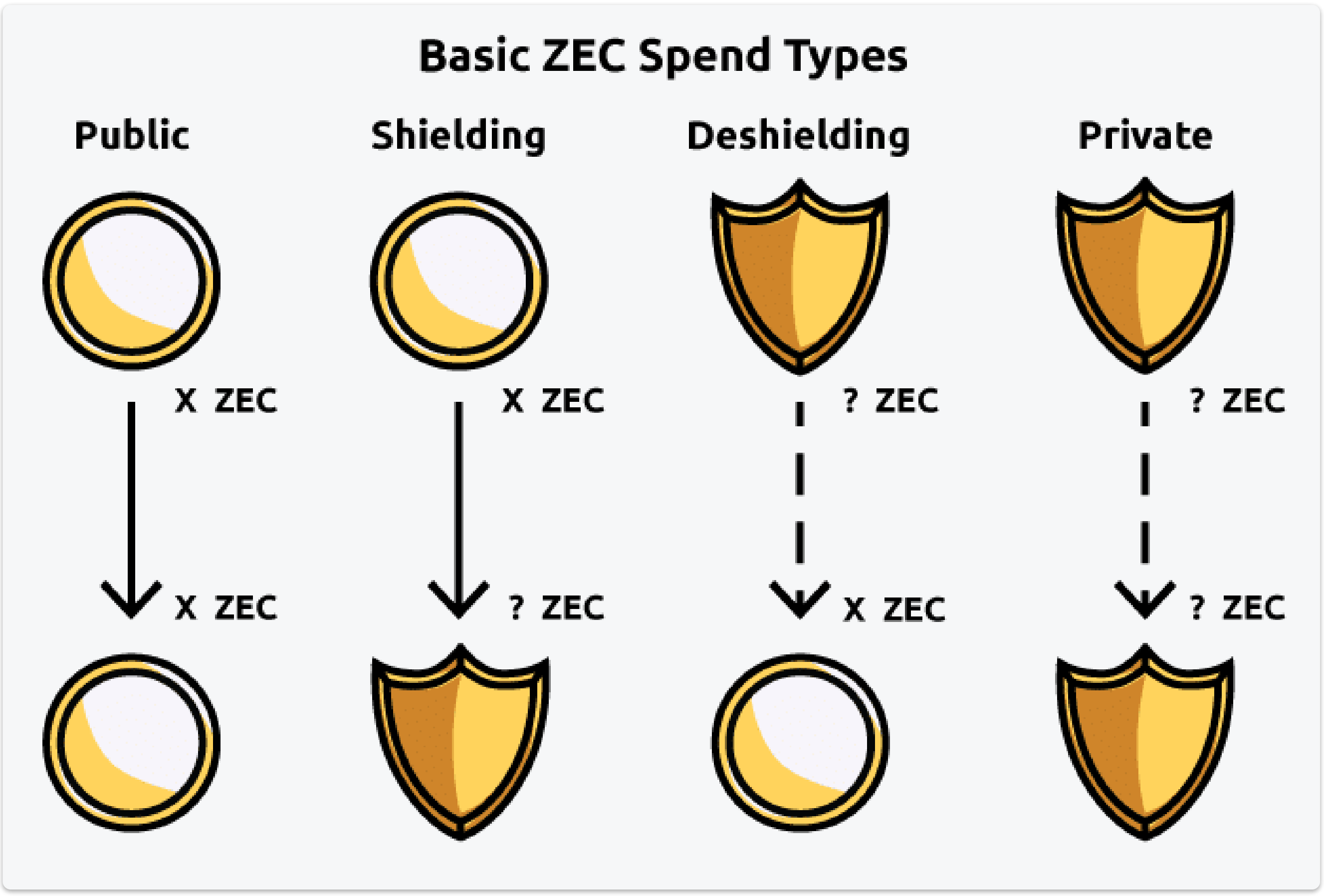
Zcash specification

"Zcash is an implementation of the Decentralized Anonymous Payment scheme Zerocash, with security fixes and improvements to performance and functionality. It bridges the existing transparent payment scheme used by Bitcoin with a shielded payment scheme secured by zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs). It attempted to address the problem of mining centralization by use of the Equihash memory-hard proof-of-work algorithm."

Overview

Value in Zcash is either transparent or shielded. Transfers of transparent value work essentially as in Bitcoin and have the same privacy properties.
(See Block Explorer))





Examples

PUBLIC

https://blockchair.com/zcash/transaction/e10d50d127c55db1714af2b420d146c474c2787f95e0cabb5fe8a3879562b770

SHIELDING

https://blockchair.com/zcash/transaction/de1d78ee310ba2c9fbeb13881f431fdc8b55aa5f79e61dd3c294177401878f11

DE-SHIELDING

https://blockchair.com/zcash/transaction/218f5cee4aa8d3469ea0e4cd49fc0b5b60e3d53fcff9392384a2c137fc48ea45

PRIVATE

https://blockchair.com/zcash/transaction/35f6674a1691f21aff6a3819467dbba82aaebf061d50c6ac55f39fbeae73b9a6

ZCash Addresses

Addresses which start with "t" behave similarly to Bitcoin, exposing addresses and balances on the blockchain and we refer to these as "transparent addresses".
Addresses which start with "z" or "zc" or "zs" include the privacy enhancements provided by zk proofs and we refer to these as "shielded addresses".
It is possible to send ZEC between these two address types.

The commitment / nullifier idea

1. Commitments
   A commitment scheme is defined by algorithms *Commit* and *Open*
   Given a message m and randomness r, compute as output a value c

```
c = Commit(m,r).
```

A commitment scheme has 2 properties:
1. Binding. Given a commitment c, it is hard to compute a different pair of message and randomness whose commitment is c. This property guarantees that there is no ambiguity in the commitment scheme, and thus after c is published it is hard to open it to a different value.
2. Hiding. It is hard to compute any information about m given c.

In ZCash Pedersen hashes are used to create the commitments using generator points on an elliptic curve
Given a value v which we want to commit to, and some random number r

commitment c = $v * G_v + r * G_r$
Where G_v and G_r are generator points on an elliptic curve.

2. Nullifiers
   Nullifiers are used to signal that a note has been spent. Each note can deterministically produce a unique nullifier.
   When spending a note,
   1. The nullifier set is checked to ascertain whether that note has already been spent.
   2. If no nullifier exists for that note, the note can be spent
   3. Once the note has been spent its nullifier is calculated and added to the nullifier set

Note that the nullifier is unlinkable, knowledge of a nullifier does not give knowledge of the note that produced it.

Shielded value is carried by notes ,which specify an amount and (indirectly) a shielded payment address, which is a destination to which notes can be sent.
As in Bitcoin, this is associated with a private key that can be used to spend notes sent to the address; in Zcash this is called a spending key.

To each note there is cryptographically associated a note commitment . Once the transaction creating a note has been mined, the note is associated with a fixed note position in a tree of note commitments, and with a nullifier unique to that note.

Computing the nullifier requires the associated private spending key.
It is infeasible to correlate the note commitment or note position with the corresponding nullifier without knowledge of at least this key.

An unspent valid note, at a given point on the block chain, is one for which the note commitment has been publically revealed on the block chain prior to that point, but the nullifier has not.

For each shielded input ,
• there is a revealed value commitment to the same value as the input note
• if the value is nonzero, some revealed note commitment exists for this note;
• the prover knew the proof authorizing key of the note;
• the nullifier and note commitment are computed correctly.

and for each shielded output ,

• there is a revealed value commitment to the same value as the output note
• the note commitment is computed correctly;
• it is infeasible to cause the nullifier of the output note to collide with the nullifier of any other note.

Outside the zk-SNARK , it is checked that the nullifiers for the input notes had not already been revealed (i.e. they had not already been spent).

A shielded payment address includes a transmission key for a "key-private" asymmetric encryption scheme.
Key-private means that ciphertexts do not reveal information about which key they were encrypted to, except to a holder of the corresponding private key, which in this context is called the receiving key. This facility is used to communicate encrypted output notes on the block chain to their intended recipient, who can use the receiving key to scan the block chain for notes addressed to them and then decrypt those notes.

Transactions

Each transaction has a list of Spend and Output Descriptions
Output Descriptions create new notes
Only sender's outgoing view key and recipient's incoming view key can decrypt
Only the recipient can spend
Spend Descriptions spend existing notes , the spender proves in zero knowledge that

- The note exists
- The spender owns the note
- The note has not been spent before, by computing a nullifier unique to that note and checking this against the nullifier set.

Which note was used is not revealed
Who the sender, recipient, or the amount is not revealed
The nullifier is unique to each note, and is revealed when spent

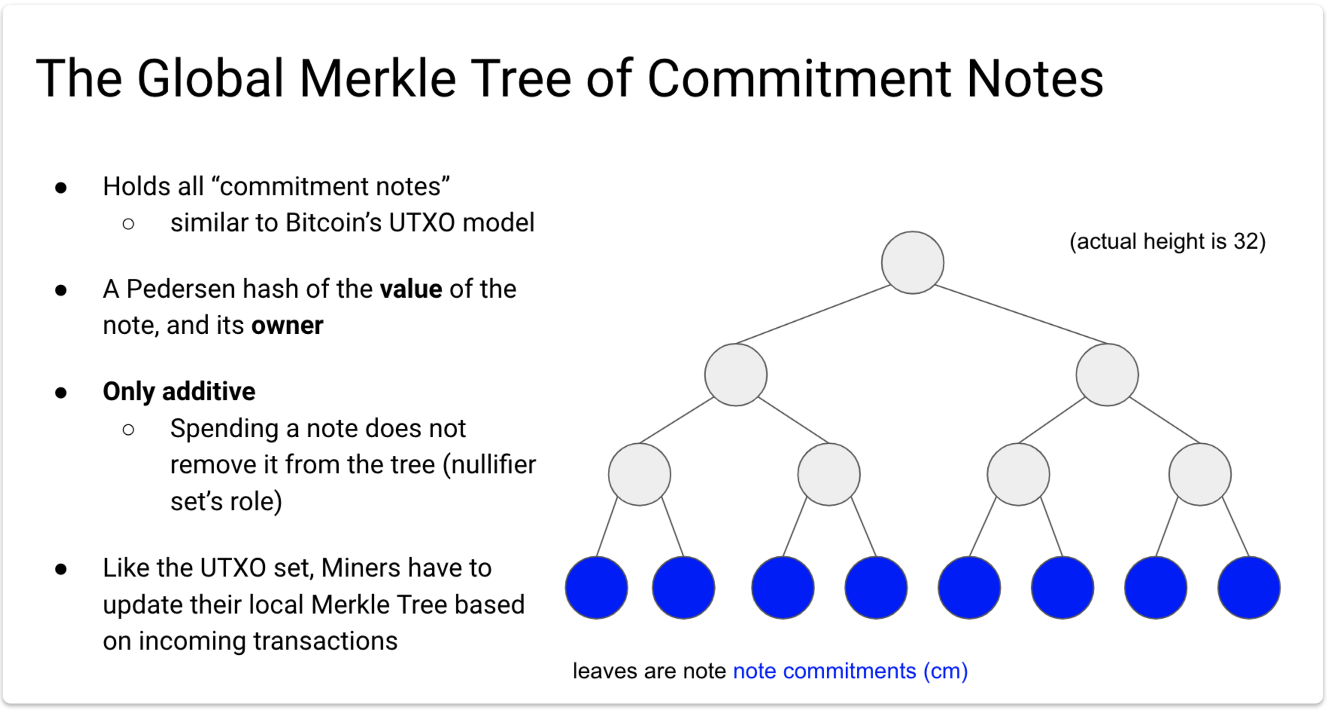
Balancing is also done to check the transaction, achieved with pedersen hashes and blinding

The basis of the privacy properties of Zcash is that when a note is spent, the spender only proves that some commitment for it had been revealed, without revealing which one.
This implies that a spent note cannot be linked to the transaction in which it was created. That is, from an adversary’s point of view the set of possibilities for a given note input to a transaction —its note traceability set — includes all previous notes that the adversary does not control or know to have been spent

The zk-SNARK circuit has a fixed size, and the maximum number of inputs and outputs per JoinSplit must be fixed in order to achieve that. 2 inputs and two outputs are the minimum needed in order to be able to join and split shielded notes (hence the name "JoinSplit"). This is the same as in the Pour proofs of the original Zerocash design.

SPENDING A NOTE

Details from
https://docs.google.com/presentation/d/1qsOtMLiBVhVMbeB_R0heTSMRsKnhuOKfhACFiXKM-J0/edit#slide=id.g5b8b5aa9ee_1_388



The Global Merkle Tree of Commitment Notes

- Holds all “commitment notes”
  - similar to Bitcoin’s UTXO model
- A Pedersen hash of the **value** of the note, and its **owner**
- **Only additive**
  - Spending a note does not remove it from the tree (nullifier set’s role)
- Like the UTXO set, Miners have to update their local Merkle Tree based on incoming transactions

(actual height is 32)

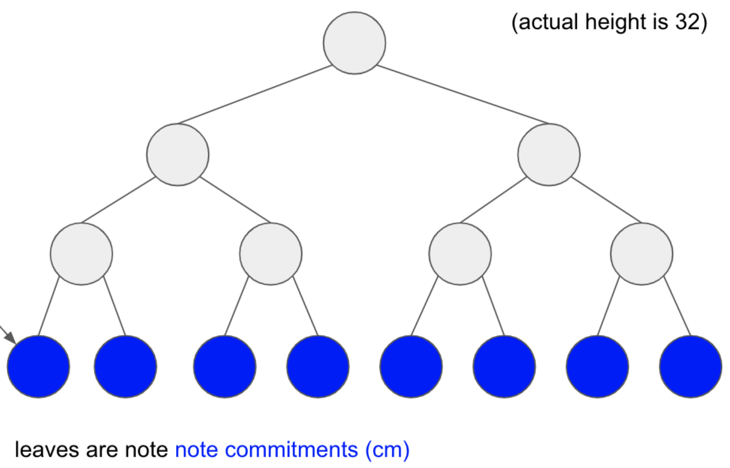
leaves are note note commitments (cm)

Output Description

Creates a new note

Shielded Transaction:
– Spend Description(s)
– Output Description(s)
has a note commitment (cm):
– Binding signature
...

note commitment (cm) is a “hash” of:
• (pk, d) – the transmission key & diversifier of the recipient’s address (more on that later)
• plaintext value of the note



When a transaction to spend a note is received, it needs to be validated, requiring the following checks:

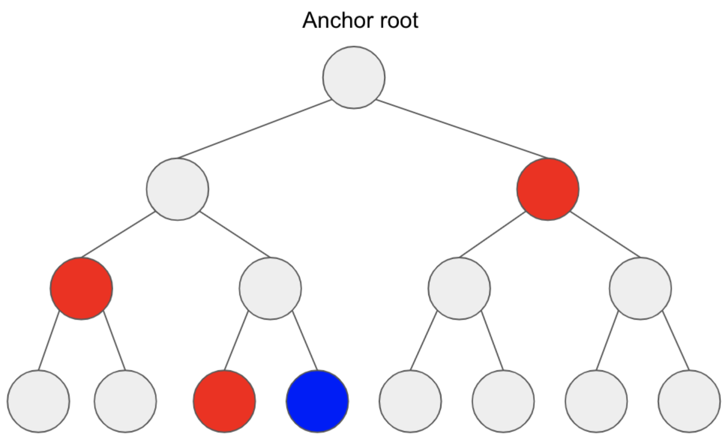
1. The note exists in the merkle tree
2. The transaction initiator is the owner of the note.
3. The note hasn't been spent.

Spend Description

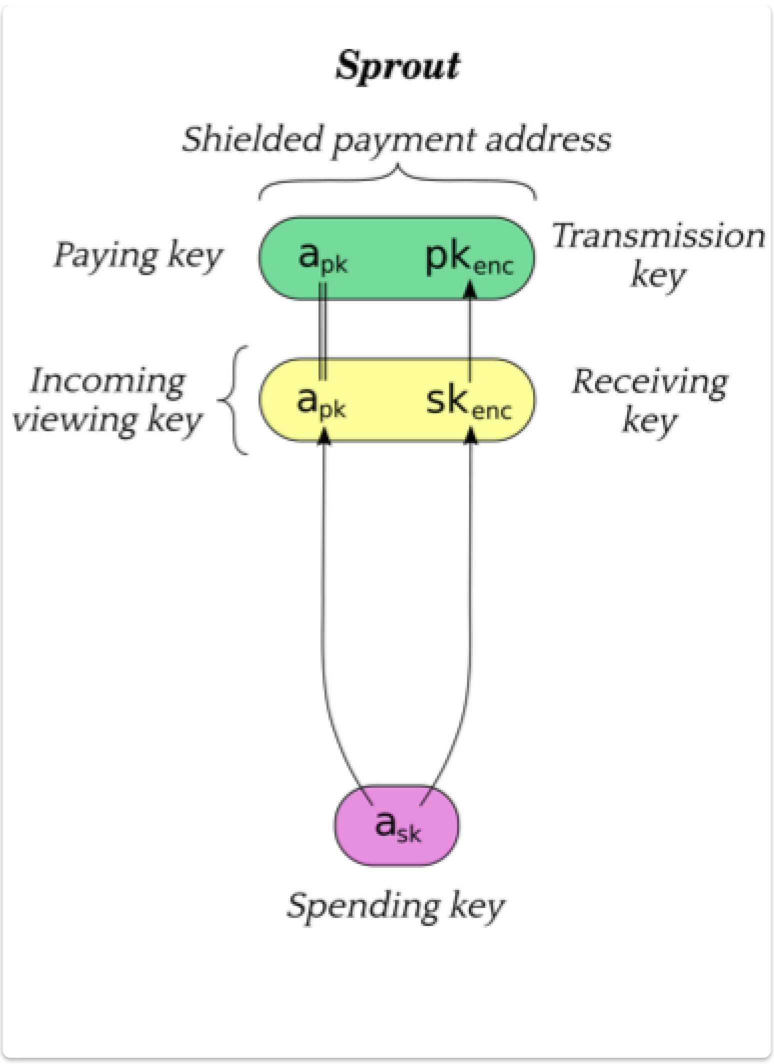
Spends a note

The **spender proves** in zero-knowledge that:

- **The note exists**
  - Proof uses an “anchor root” of the commitment note Merkle tree the Miner has seen before
- **The spender owns the note**
- **The note has not been used before**
  - Computes and reveals a *nullifier* that is unique (but not linkable) to the note being spent



The nullifiers are necessary to prevent double-spending: each note on the block chain only has one valid nullifier, and so attempting to spend a note twice would reveal the nullifier twice, which would cause the second transaction to be rejected.



Spend Description

The Spend description commits to a value commitment and all necessary public parameters needed to verify the proof constructed on private parameters to validate note ownership and spendability.
It consists of

```
cv : value commitment
rt : root anchor — which Merkle root is used for the proof
nf : nullifier for the note
rk : randomized public key for the authorization key
sig : sign hash of the spend description using private  / public key
proof : zk-SNARK proof that given these public values, there are private values to
validate all this
```

The ZKProof

What is needed in the proof ?

1. Private inputs

   merkle path : the Merkle path to the commitment note being spent
   position : the position of the commitment note (e.g. index)
   gd : the diversifier of the public address owning this note

pkd : the transmission key of the owner
v : value of the note
rcv : randomness used in the Pedersen Hash for value commitment
cm : note commitment of the note being spent
rcm : the randomness used in the Pedersen Hash for note commitment
α : alpha used to hide the authorization key that signs the spend
ak : the owner's authorization key (that was randomized)
nsk : the proof authorization key used for the nullifier

2. Public inputs
rt : root anchor that was used for the Merkle path in the proof
cv : Pedersen Hash (commitment) of the value
nf : nullifier to spend the note
rk : the randomized authorization public key

References

Attacking ZCash for fun or profit

Sapling Shielded Transactions
ZCash Protocol.
Elliptic Curve in Sapling

Cryptography used

For hash functions, ZCash improved on the Pedersen hash function creating the Bowe-Hopwood Pedersen Hash

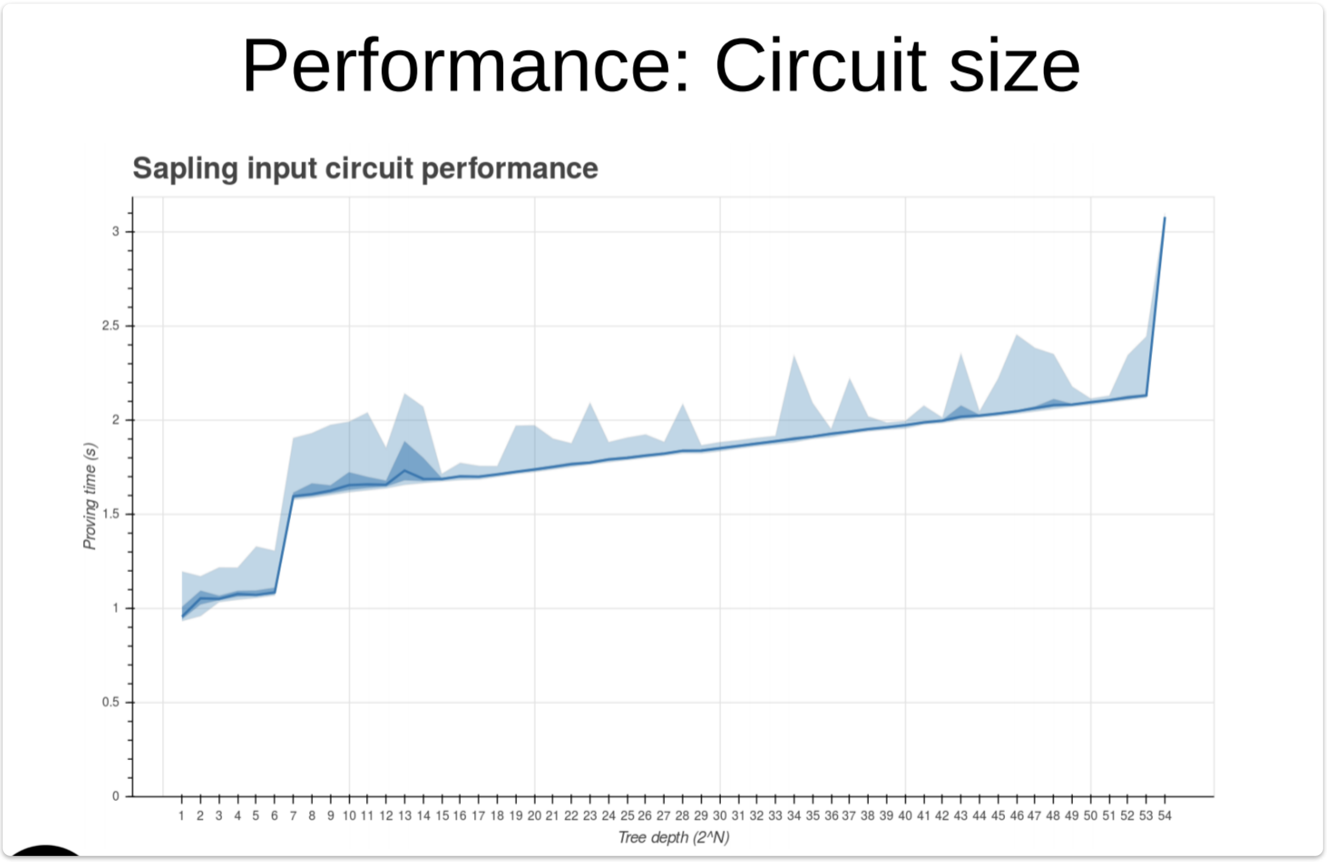
Originally ZCash used BLS12-381 for its elliptic curve as optimal for for zkSNARKS with a security bit level of 128 which had an embedded twisted Edwards curve (named Jubjub).
The latest version, Orchard, is using two elliptic curves, Pallas and Vesta, that form a cycle: the base field of each is the scalar field of the other.
In Orchard, we use Vesta for the proof system (playing a similar rôle to BLS12-381 in Sapling), and Pallas for the application circuit (similar to Jubjub in Sapling).

Interaction is in Rust via the Bellman library.

Zcash uses four signature schemes:
• one used for signatures that can b evalidated by script operations such as OP_CHECKSIG and OP_CHECKMULTISIG as in Bitcoin;
• one called JoinSplitSig which is used to sign transactions that contain at least one JoinSplit description
• one called SpendAuthSig which is used to sign authorizations of Spend transfers
• one called BindingSig. A Sapling binding signature is used to enforce balance of Spend transfers and Output transfers, and to prevent their replay across transactions.
The signature scheme used in script operations is instantiated by ECDSA on the secp256k1 curve. JoinSplitSig is instantiated by Ed25519. SpendAuthSig and BindingSig are instantiated by RedDSA; on the Jubjub curve in Sapling, and on the Pallas curve in Orchard.

Performance

In Sapling the proving time has been reduced to an average of 2.3s
Each proof requires 3 field elements, and 3 pairing checks.
The Merkle tree is based on Bowe-Hopwood Pedersen hashes, with depth 32, 1369 constraints per level, giving a total of 43,808 constraints.



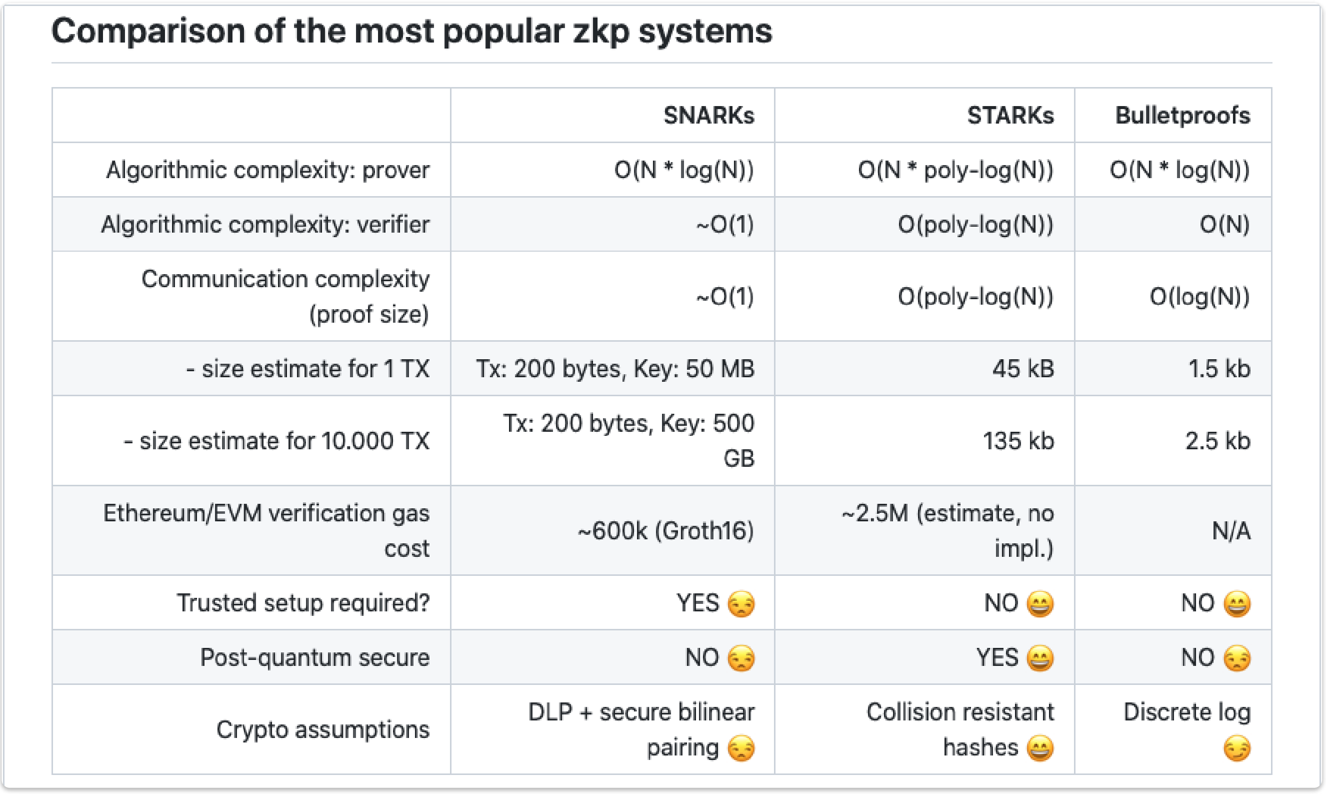
Performance is not linear in tree depth / circuit size.

BLOCK EXPLORER

Block Explorer

Bulletproofs

See paper : Bulletproofs



Comparison of the most popular zkp systems

	SNARKs	STARKs	Bulletproofs
Algorithmic complexity: prover	O(N * log(N))	O(N * poly-log(N))	O(N * log(N))
Algorithmic complexity: verifier	~O(1)	O(poly-log(N))	O(N)
Communication complexity (proof size)	~O(1)	O(poly-log(N))	O(log(N))
- size estimate for 1 TX	Tx: 200 bytes, Key: 50 MB	45 kB	1.5 kb
- size estimate for 10.000 TX	Tx: 200 bytes, Key: 500 GB	135 kb	2.5 kb
Ethereum/EVM verification gas cost	~600k (Groth16)	~2.5M (estimate, no impl.)	N/A
Trusted setup required?	YES 😒	NO 😄	NO 😄
Post-quantum secure	NO 😒	YES 😄	NO 😒
Crypto assumptions	DLP + secure bilinear pairing 😒	Collision resistant hashes 😄	Discrete log 😏

- Bullet proofs have short proofs without a trusted setup.
- Their underlying cryotographic assumption is the discrete log problem.
- They are made non interactive using the Fiat-Shamir heuristic.
- One of the paper's authors referred to them as
  “Short like a bullet with bulletproof security assumptions”
- They were designed to provide confidential transactions for cryptocurrencies.
- They support proof aggregation, so that proving that m transaction values are valid, adds only $O(log(m))$ additional elements to the size of a single proof.
- Pederson commitments are used for the inputs
- They do not require pairings and work with any elliptic curve with a reasonably large subgroup size
- The verifier cost scales linearly with the computation size.

Bulletproofs were based on ideas from Groth16 SNARKS, but changed various aspects.

- They give a more compact version of the inner product argument of knowledge
- Allow construction of a compact rangeproof using such an argument of knowledge
- They generalize this idea to general arithmetic circuits.

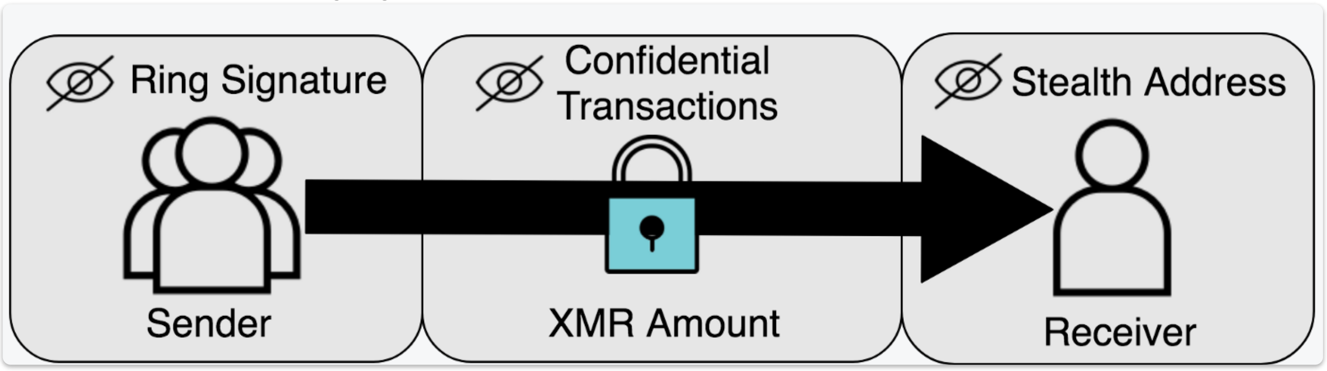
Some use cases for bulletproofs

- Range proofs
- Merkle proofs (accumulators)
- Proof of Solvency

Monero

Initial approach

Initially Monero used ring signatures



This approach had scalability and security issues.
They later moved to using bulletproofs

Overview

Committments to inputs are used to shield the input details
We need to show that the sum of inputs and outputs add up, but there is a potential problem with overflow since we work on a finite field.
This is solved with range proofs to show that the values are in the correct range, without revealing the values.

Monero comment on their move to bulletpoofs :
"With our current range proofs, the transaction is around 13.2 kB in size. If I used single-output bulletproofs, the transaction reduces in size to only around 2.5 kB
This is, approximately, an 80% reduction in transaction size, which then translates to an 80% reduction in fees as well.
The space savings are even better with multiple-output proofs. This represents a significant decrease in transaction sizes. Further, our initial testing shows that the time to verify a bulletproof is lower than for the existing range proofs, meaning speedier blockchain validation. "