Zcash Protocol Specification

Sean Bowe — Daira Hopwood — Taylor Hornby

January 28, 2016

1 Introduction

Zcash is an implementation of the *Decentralized Anonymous Payment* scheme **Zerocash** [2] with some adjustments to terminology, functionality and performance. It bridges the existing *transparent* payment scheme used by **Bitcoin** with a *confidential* payment scheme protected by zero-knowledge succinct non-interactive arguments of knowledge (*zk-SNARKs*).

2 Concepts

2.1 Integers and Endianness

All integers visible in **Zcash**-specific encodings are unsigned, have a fixed bit length, and are encoded as big-endian.

In bit layout diagrams, each box of the diagram represents a sequence of bits. If the content of the box is a byte sequence, it is implicitly converted to a sequence of bits using big endian order. The bit sequences are then concatenated in the order shown from left to right, and the result is converted to a sequence of bytes, again using big-endian order.

$\texttt{Leading}_k(x)$, where k is an integer and x is a bit sequence, returns the leading (initial) k bits of its input.

$\texttt{Trailing}_k(x)$, where k is an integer and x is a bit sequence, returns the trailing (final) k bits of its input.

2.2 Cryptographic Functions

CRH is a collision-resistant hash function. In **Zcash**, the *SHA-256 compression* function is used which takes a 512-bit block and produces a 256-bit hash. This is different from the *SHA-256* function, which hashes arbitrary-length strings.

PRF_x is a pseudo-random function seeded by x. Three *independent* PRF_x are needed in our scheme: $\mathsf{PRF}^{\mathsf{addr}}_x$, $\mathsf{PRF}^{\mathsf{sn}}_x$, and $\mathsf{PRF}^{\mathsf{pk}}_x$. It is required that $\mathsf{PRF}^{\mathsf{sn}}_x$ be collision-resistant across all x.

In **Zcash**, the *SHA-256 compression* function is used to construct all three of these functions. The bits `00`, `01` and `10` are included (respectively) within the blocks that are hashed, ensuring that the functions are independent.

$$\mathsf{a_{pk}} := \mathsf{PRF}^{\mathsf{addr}}_{\mathsf{a_{sk}}}(0) = \mathsf{CRH}\left(\boxed{256 \text{ bit } \mathsf{a_{sk}}}\boxed{0}\boxed{0}\boxed{0^{254}}\right)$$

$$\mathsf{sn} := \mathsf{PRF}^{\mathsf{sn}}_{\mathsf{a_{sk}}}(\rho) = \mathsf{CRH}\left(\boxed{256 \text{ bit } \mathsf{a_{sk}}}\boxed{0}\boxed{1}\boxed{\texttt{Trailing}_{254}(\rho)}\right)$$

$$\mathsf{h}_i := \mathsf{PRF}^{\mathsf{pk}}_{\mathsf{a_{sk}}}(i, \mathsf{h_{Sig}}) = \mathsf{CRH}\left(\boxed{256 \text{ bit } \mathsf{a_{sk}}}\boxed{1}\boxed{0}\boxed{i}\boxed{\texttt{Trailing}_{253}(\mathsf{h_{Sig}})}\right)$$

Daira: Should we instead define ρ to be 254 bits and $\mathsf{h_{Sig}}$ to be 253 bits?

2.3 Confidential Addresses and Private Keys

A key pair $(\mathsf{addr_{pk}}, \mathsf{addr_{sk}})$ is generated by users who wish to receive coins under this scheme. The public $\mathsf{addr_{pk}}$ is called a *confidential address* and is a tuple $(\mathsf{a_{pk}}, \mathsf{pk_{enc}})$ consisting of the public components of a *spend authority* key pair $(\mathsf{a_{pk}}, \mathsf{a_{sk}})$ and a *key-private encryption* key pair $(\mathsf{pk_{enc}}, \mathsf{sk_{enc}})$. The private $\mathsf{addr_{sk}}$ is called a *confidential private key* and is a tuple $(\mathsf{a_{sk}}, \mathsf{sk_{enc}})$ consisting of the respective *private* components of the aforementioned *spend authority* and *key-private encryption* key pairs.

Although users can accept payment from multiple parties with a single $\mathsf{addr_{pk}}$ without either party being aware, it is still recommended to generate a new address for each expected transaction to maximize privacy in the event that multiple sending parties are compromised or collude.

2.4 Coins

A *coin* (denoted $\mathbf{c}$) is a tuple $(\mathsf{a_{pk}}, \mathsf{v}, \rho, \mathsf{r})$ which represents that a value v is spendable by the recipient who holds the *spend authority* key pair $(\mathsf{a_{pk}}, \mathsf{a_{sk}})$ such that $\mathsf{a_{pk}} = \mathsf{PRF}^{\mathsf{addr}}_{\mathsf{a_{sk}}}(0)$. ρ and r are tokens randomly generated by the sender. Only a hash of these values is disclosed publicly, which allows these random tokens to blind the value and recipient *except* to those who possess these tokens.

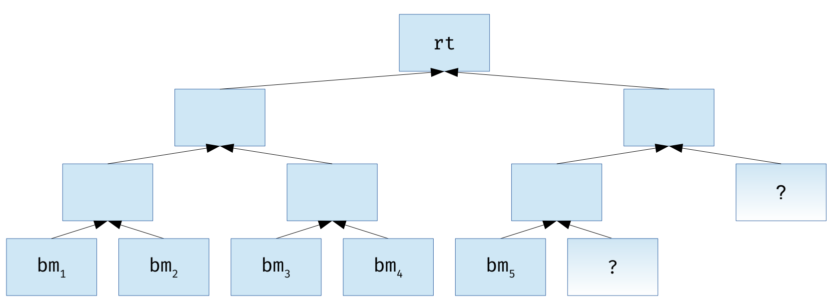
In-band secret distribution In order to transmit the secret v, ρ and r to the recipient (necessary for the recipient to later spend) *without* requiring an out-of-band communication channel, the *key-private encryption* public key $\mathsf{pk_{enc}}$ is used to encrypt these secrets to form a *transmitted coin ciphertext*. The recipient's possession of the associated $(\mathsf{addr_{pk}}, \mathsf{addr_{sk}})$ (which contains both $\mathsf{a_{pk}}$ and $\mathsf{sk_{enc}}$) is used to reconstruct the original *coin*.

Coin Commitments The underlying v and $\mathsf{a_{pk}}$ are blinded with ρ and r using the collision-resistant hash function CRH in a multi-layered process. The resulting hash $\mathsf{cm} = \mathtt{CoinCommitment}(\mathbf{c})$.

InternalH := CRH(	256 bit $\mathsf{a_{pk}}$	256 bit ρ	)
k := CRH(	384 bit r	$\mathtt{Leading}_{128}(\mathsf{InternalH})$	)
cm := CRH(	64 bit v · 192 bit padding	256 bit k	)

Serials A *serial number* (denoted sn) equals $\mathsf{PRF}^{\mathsf{sn}}_{\mathsf{a_{sk}}}(\rho)$. A *coin* is spent by proving knowledge of ρ and $\mathsf{a_{sk}}$ in zero knowledge while disclosing sn, allowing sn to be used to prevent double-spending.

2.5 Coin Commitment Tree



The *coin commitment tree* is an *incremental merkle tree* of depth d used to store *coin commitments* that *Pour transfers* produce. Just as the *unspent transaction output set* (UTXO) used in Bitcoin, it is used to express the existence of value and the capability to spend it. However, unlike the UTXO, it is *not* the job of this tree to protect against double-spending, as it is append-only.

Blocks in the blockchain are associated (by all nodes) with the root of this tree after all of its constituent *Pour descriptions*' *coin commitments* have been entered into the tree associated with the previous block.

2.6 Spent Serials Map

Transactions insert *serial numbers* into a *spent serial numbers map* which is maintained alongside the UTXO by all nodes.

Eli: a tx is just a string, so it doesn't insert anything. Rather, nodes process tx's and the "good" ones lead to the addition of serials to the spent serials map.

Transactions that attempt to insert a *serial number* into this map that already exists within it are invalid as they are attempting to double-spend.

Eli: After defining *transaction*, one should define what a *legal tx* is (this definition depends on a particular blockchain [view]) and only then can one talk about "attempts" of transactions, and insertions of serial numbers into the spent serials map.

2.7 The Blockchain

At a given point in time, the *blockchain view* of each *full node* consists of a sequence of one or more valid *blocks*. Each *block* consists of a sequence of one or more *transactions*. In a given node's *blockchain view*, *treestates* are chained in an obvious way:

- The input *treestate* of the first *block* is the empty *treestate*.
- The input *treestate* of the first *transaction* of a *block* is the final *treestate* of the immediately preceding *block*.
- The input *treestate* of each subsequent *transaction* in a *block* is the output *treestate* of the immediately preceding *transaction*.
- The final *treestate* of a *block* is the output *treestate* of its last *transaction*.

An *anchor* is a Merkle tree root of a *treestate*, and uniquely identifies that *treestate* given the assumed security properties of the Merkle tree's hash function.

Each *transaction* is associated with a sequence of *Pour descriptions*. TODO They also have a transparent value flow that interacts with the Pour $\mathsf{v}_{\mathsf{pub}}^{\mathsf{old}}$ and $\mathsf{v}_{\mathsf{pub}}^{\mathsf{new}}$. Inputs and outputs are associated with a value.

The total value of the outputs must not exceed the total value of the inputs.

The *anchor* of the first *Pour description* in a *transaction* must refer to some earlier *block*'s final *treestate*.

The *anchor* of each subsequent *Pour description* may refer either to some earlier *block*'s final *treestate*, or to the output *treestate* of the immediately preceding *Pour description*.

These conditions act as constraints on the blocks that a *full node* will accept into its *blockchain view*.

We rely on Bitcoin-style consensus for *full nodes* to eventually converge on their views of valid *blocks*, and therefore of the sequence of *treestates* in those *blocks*.

Value pool Transaction inputs insert value into a *value pool*, and transaction outputs remove value from this pool. The remaining value in the pool is available to miners as a fee.

3 Pour Transfers and Descriptions

A *Pour description* is data included in a *block* that describes a *Pour transfer*, i.e. a confidential value transfer. This kind of value transfer is the primary **Zerocash**-specific operation performed by transactions; it uses, but should not be confused with, the *POUR circuit* used for the *zk-SNARK* proof and verification.

A *Pour transfer* spends $\mathrm{N}^{\mathsf{old}}$ *coins* $\mathbf{c}_{1..\mathrm{N}^{\mathsf{old}}}^{\mathsf{old}}$ and creates $\mathrm{N}^{\mathsf{new}}$ *coins* $\mathbf{c}_{1..\mathrm{N}^{\mathsf{new}}}^{\mathsf{new}}$. **Zcash** transactions have an additional field `vpour`, which is a sequence of *Pour descriptions*.

Each *Pour description* consists of:

`vpub_old` which is a value $\mathsf{v}_{\mathsf{pub}}^{\mathsf{old}}$ that the *Pour transfer* removes from the value pool.

`vpub_new` which is a value $\mathsf{v}_{\mathsf{pub}}^{\mathsf{new}}$ that the *Pour transfer* inserts into the value pool.

`anchor` which is a merkle root rt of the *coin commitment tree* at some block height in the past, or the merkle root produced by a previous pour in this transaction. Sean: We need to be more specific here.

`scriptSig` which is a *script* that creates conditions for acceptance of a *Pour description* in a transaction. The `SHA256Compress` hash of this value is $\mathsf{h_{Sig}}$.

Daira: Why `SHA256Compress` and not *SHA-256*? The script is variable-length.

`scriptPubKey` which is a *script* used to satisfy the conditions of the `scriptSig`.

`serials` which is an $\mathrm{N}^{\mathsf{old}}$ size sequence of serials $\mathsf{sn}^{\mathsf{old}}_{1..\mathrm{N}^{\mathsf{old}}}$.

`commitments` which is a $\mathrm{N}^{\mathsf{new}}$ size sequence of *coin commitments* $\mathsf{cm}^{\mathsf{new}}_{1..\mathrm{N}^{\mathsf{new}}}$.

`ciphertexts` which is a $\mathrm{N}^{\mathsf{new}}$ size sequence each element of which is a *transmitted coin ciphertext*.

`vmacs` which is a $\mathrm{N}^{\mathsf{old}}$ size sequence of message authentication tags $\mathsf{h}_{1..\mathrm{N}^{\mathsf{old}}}$ that bind $\mathsf{h_{Sig}}$ to each $\mathsf{a_{sk}}$ of the *Pour description*.

`zkproof` which is the zero-knowledge proof $\pi_{\mathtt{POUR}}$.

Merkle root validity A *Pour description* is valid if rt is a Coin commitment tree root found in either the blockchain or a merkle root produced by inserting the Coin commitments of a previous *Pour description* in the transaction to the Coin commitment tree identified by that previous *Pour description*'s *anchor*.

Non-malleability A *Pour description* is valid if the script formed by appending `scriptPubKey` to `scriptSig` returns *true*. The `scriptSig` is cryptographically bound to $\pi_{\mathtt{POUR}}$.

Balance A *Pour transfer* can be seen, from the perspective of the transaction, as an input and an output simultaneously. $\mathsf{v}^{\mathsf{old}}_{\mathsf{pub}}$ takes value from the value pool and $\mathsf{v}^{\mathsf{new}}_{\mathsf{pub}}$ adds value to the value pool. As a result, $\mathsf{v}^{\mathsf{old}}_{\mathsf{pub}}$ is treated like an *output* value, whereas $\mathsf{v}^{\mathsf{new}}_{\mathsf{pub}}$ is treated like an *input* value.

Commitments and Serials A *transaction* that contains one or more *Pour descriptions*, when entered into the blockchain, appends to the *coin commitment tree* with all constituent *coin commitments*. All of the constituent *serial numbers* are also entered into the *spent serial numbers map* of the *blockchain view and mempool*. A *transaction* is not valid if it attempts to add a *serial number* to the *spent serial numbers map* that already exists in the map.

3.1 Pour Circuit and Proofs

In **Zcash**, $\mathrm{N}^{\mathsf{old}}$ and $\mathrm{N}^{\mathsf{new}}$ are both 2.

A valid instance of $\pi_{\mathtt{POUR}}$ assures that given a *primary input* $(\mathsf{rt}, \mathsf{sn}^{\mathsf{old}}_{1..\mathrm{N}^{\mathsf{old}}}, \mathsf{cm}^{\mathsf{new}}_{1..\mathrm{N}^{\mathsf{new}}}, \mathsf{v}^{\mathsf{old}}_{\mathsf{pub}}, \mathsf{v}^{\mathsf{new}}_{\mathsf{pub}}, \mathsf{h_{Sig}}, \mathsf{h}_{1..\mathrm{N}^{\mathsf{old}}})$, a witness of *auxiliary input* $(\mathsf{path}_{1..\mathrm{N}^{\mathsf{old}}}, \mathbf{c}^{\mathsf{old}}_{1..\mathrm{N}^{\mathsf{old}}}, \mathsf{a}^{\mathsf{old}}_{\mathsf{sk},1..\mathrm{N}^{\mathsf{old}}}, \mathbf{c}^{\mathsf{new}}_{1..\mathrm{N}^{\mathsf{new}}})$ exists, where:

for each $i \in \{1..\mathrm{N}^{\mathsf{old}}\}$: $\mathbf{c}^{\mathsf{old}}_i = (\mathsf{a}^{\mathsf{old}}_{\mathsf{pk},i}, \mathsf{v}^{\mathsf{old}}_i, \rho^{\mathsf{old}}_i, \mathsf{r}^{\mathsf{old}}_i)$

for each $i \in \{1..\mathrm{N}^{\mathsf{new}}\}$: $\mathbf{c}^{\mathsf{new}}_i = (\mathsf{a}^{\mathsf{new}}_{\mathsf{pk},i}, \mathsf{v}^{\mathsf{new}}_i, \rho^{\mathsf{new}}_i, \mathsf{r}^{\mathsf{new}}_i)$

The following conditions hold:

Merkle path validity for each $i \in \{1..\mathrm{N}^{\mathsf{old}}\} \mid \mathsf{v}^{\mathsf{old}}_i \neq 0$: path_i must be a valid path of depth d from $\mathtt{CoinCommitment}(\mathbf{c}^{\mathsf{old}}_i)$ to Coin commitment merkle tree root rt.

Balance $\mathsf{v}^{\mathsf{old}}_{\mathsf{pub}} + \sum_{i=1}^{\mathrm{N}^{\mathsf{old}}} \mathsf{v}^{\mathsf{old}}_i = \mathsf{v}^{\mathsf{new}}_{\mathsf{pub}} + \sum_{i=1}^{\mathrm{N}^{\mathsf{new}}} \mathsf{v}^{\mathsf{new}}_i$.

Serial integrity for each $i \in \{1..\mathrm{N}^{\mathsf{new}}\}$: $\mathsf{sn}^{\mathsf{old}}_i = \mathsf{PRF}^{\mathsf{sn}}_{\mathsf{a}^{\mathsf{old}}_{\mathsf{sk},i}}(\rho^{\mathsf{old}}_i)$.

Spend authority for each $i \in \{1..\mathrm{N}^{\mathsf{old}}\}$: $\mathsf{a}^{\mathsf{old}}_{\mathsf{pk},i} = \mathsf{PRF}^{\mathsf{addr}}_{\mathsf{a}^{\mathsf{old}}_{\mathsf{sk},i}}(0)$.

Non-malleability for each $i \in \{1..\mathrm{N}^{\mathsf{old}}\}$: $\mathsf{h}_i = \mathsf{PRF}^{\mathsf{pk}}_{\mathsf{a}^{\mathsf{old}}_{\mathsf{sk},i}}(i, \mathsf{h}_{\mathsf{Sig}})$

Commitment integrity for each $i \in \{1..\mathrm{N}^{\mathsf{new}}\}$: $\mathsf{cm}^{\mathsf{new}}_i = \mathtt{CoinCommitment}(\mathbf{c}^{\mathsf{new}}_i)$

4 Encoding Addresses, Private keys, Coins, and Pour descriptions

This section describes how **Zcash** encodes public addresses, private keys, coins, and *Pour descriptions*.

Addresses, keys, and coins, can be encoded as a byte string; this is called the *raw encoding*. This byte string can then be further encoded using Base58Check. The Base58Check layer is the same as for upstream **Bitcoin** addresses [1].

SHA-256 compression function outputs are always represented as strings of 32 bytes.

The language consisting of the following encoding possibilities is prefix-free.

4.1 Transparent Public Addresses

These are encoded in the same way as in **Bitcoin** [1].

4.2 Transparent Private Keys

These are encoded in the same way as in **Bitcoin** [1].

4.3 Confidential Public Addresses

A confidential address consists of a_{pk} and $\mathsf{pk}_{\mathsf{enc}}$. a_{pk} is a SHA-256 compression function output. $\mathsf{pk}_{\mathsf{enc}}$ is an encryption public key (currently ECIES, but this may change to Curve25519/crypto_box_seal), which represents an equivalence class of two points sharing an x coordinate on an elliptic curve.

4.3.1 Raw Encoding

The raw encoding of a confidential address consists of:

0x92	a_{pk} (32 bytes)	A 33-byte encoding of $\mathsf{pk}_{\mathsf{enc}}$

- A byte, **0x92**, indicating this version of the raw encoding of a **Zcash** public address.
- 32 bytes specifying a_{pk}.
- An encoding of $\mathsf{pk}_{\mathsf{enc}}$: The byte **0x01**, followed by 32 bytes representing the x coordinate of an elliptic curve point according to the FE2OSP primitive specified in section 5.5.4 of IEEE Std 1363-2000. [Non-normative note: Since the curve is over a prime field, this is just the 32-byte big-endian representation of the x coordinate. The overall encoding matches the EC2OSP-X primitive specified in section 5.5.6.3 of IEEE Std 1363a-2004. It does not matter which of the two points with the same x coordinate is used.]

Daira: check that this lead byte is distinct from other Bitcoin stuff, and produces 'z' as the Base58Check leading character.

Nathan: what about the network version byte?

Daira: add bibliographic references for the IEEE standards.

4.4 Confidential Address Secrets

A confidential address secret consists of a_{sk} and $\mathsf{sk}_{\mathsf{enc}}$. a_{sk} is a SHA-256 compression function output. $\mathsf{sk}_{\mathsf{enc}}$ is an encryption private key (currently ECIES), which is an integer.

4.4.1 Raw Encoding

The raw encoding of a confidential address secret consists of, in order:

0x93	$\mathsf{a_{sk}}$ (32 bytes)	$\mathsf{sk_{enc}}$ (32 bytes)

- A byte **0x93** indicating this version of the raw encoding of a **Zcash** private key.
- 32 bytes specifying $\mathsf{a_{sk}}$.
- 32 bytes specifying a big-endian encoding of $\mathsf{sk_{enc}}$.

Daira: check that this lead byte is distinct from other Bitcoin stuff, and produces 'z' as the Base58Check leading character.

Nathan: what about the network version byte?

4.5 Coins

Transmitted coins are stored on the blockchain in encrypted form, together with a *coin commitment* cm.

A *transmitted coin ciphertext* is an ECIES encryption of a *transmitted coin plaintext* to a *key-private encryption* key $\mathsf{pk_{enc}}$.

A *transmitted coin plaintext* consists of $(\mathsf{v}, \rho, \mathsf{r})$, where:

- v is a 64-bit unsigned integer representing the value of the *coin* in *zatoshi* (1 **ZEC** $= 10^8$ *zatoshi*).
- ρ is a 32-byte $\mathsf{PRF}^{\mathsf{sn}}_{\mathsf{a_{sk}}}$ seed.
- r is a 32-byte *COMM trapdoor*.

Note that the value s described as being part of a coin in the **Zerocash** paper is not encoded because it is fixed to zero.

4.6 Raw Encoding

The raw encoding of a *transmitted coin plaintext* consists of, in order:

0x00	v (8 bytes, big endian)	ρ (32 bytes)	r (32 bytes)

- A byte **0x00** indicating this version of the raw encoding of a *transmitted coin plaintext*.
- 8 bytes specifying a big-endian encoding of v.
- 32 bytes specifying ρ.
- 32 bytes specifying r.

5 Pours (within a transaction on the blockchain)

TBD.

6 Transactions

TBD.

7 References

[1] Base58Check encoding. `https://en.bitcoin.it/wiki/Base58Check_encoding`. Accessed: 2016-01-26.

[2] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland) 2014*, pages 459–474. IEEE, 2014.