Zcash Protocol Specification

Sean Bowe — Daira Hopwood

December 14, 2015

1 Introduction

Zcash is an implementation of the *decentralized anonymous payment* (DAP) scheme **Zerocash** with minor adjustments to terminology, functionality and performance. It bridges the existing value transfer scheme used by Bitcoin with an anonymous payment scheme protected by zero-knowledge succinct non-interactive arguments of knowledge (**zk-SNARK**s).

2 Concepts

2.1 Endianness

All numerical objects in Zcash are big endian.

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.

$\mathbf{PRF_x}$ is a pseudo-random function seeded by x. Three *independent* $\mathbf{PRF_x}$ are needed in our scheme: $\mathbf{PRF^{addr}_x}$, $\mathbf{PRF^{sn}_x}$, and $\mathbf{PRF^{pk_i}_x}$. It is required that $\mathbf{PRF^{sn}_x}$ be collision-resistant in order to prevent a double-spending attack. In **Zcash** , the *SHA-256 compression* function is used to seed 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.

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

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

$$h_i = \mathbf{PRF^{pk_i}_{\mathsf{a_{sk}}}}(\mathsf{h_{Sig}}) = \mathbf{CRH}\left(\boxed{\text{256 bit } \mathsf{a_{sk}}}\boxed{1}\boxed{0}\boxed{\text{i}}\boxed{\text{253 bit truncated } \mathsf{h_{Sig}}} \right)$$

2.3 Confidential Address Keypair

A keypair $(\mathsf{addr_{pk}}, \mathsf{addr_{sk}})$ is generated by a user any time they wish to receive value from another in the system. The public $\mathsf{addr_{pk}}$ is called a *protected address* and is a tuple $(\mathsf{a_{pk}}, \mathsf{pk_{enc}})$ which are the public components of a *spend authority* keypair $(\mathsf{a_{pk}}, \mathsf{a_{sk}})$ and a *key-private encryption* keypair $(\mathsf{pk_{enc}}, \mathsf{sk_{enc}})$. The private $\mathsf{addr_{sk}}$ is called a *protected address secret* and is a tuple $(\mathsf{a_{sk}}, \mathsf{sk_{enc}})$ which are the respective *private* components of the aforementioned *spend authority* and *key-private encryption* keypairs.

2.4 Buckets

A bucket (denoted b) is a tuple $(\mathsf{v}, \mathsf{a}_{\mathsf{pk}_i}, \mathsf{r}, \rho)$ which represents that a value v is spendable by the recipient who holds the *spend authority* keypair $(\mathsf{a_{pk}}, \mathsf{a_{sk}})$. r and ρ are randomly generated tokens which are used to blind the value and recipient *except* to those who possess these tokens.

In-band secret distribution In order to send the secret v, r and ρ 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 an *encrypted bucket*. 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 bucket.

Bucket commitments The underlying v and $\mathsf{a_{pk}}$ are blinded with r and ρ using the collision-resistant hash function **CRH** in a multi-layered process. The resulting hash $\mathtt{bm}$ is called a *bucket commitment*.

$$\mathsf{InternalH} := \mathbf{CRH}\left(\boxed{\text{256 bit } \mathsf{a_{pk}}}\boxed{\text{256 bit } \rho}\right)$$

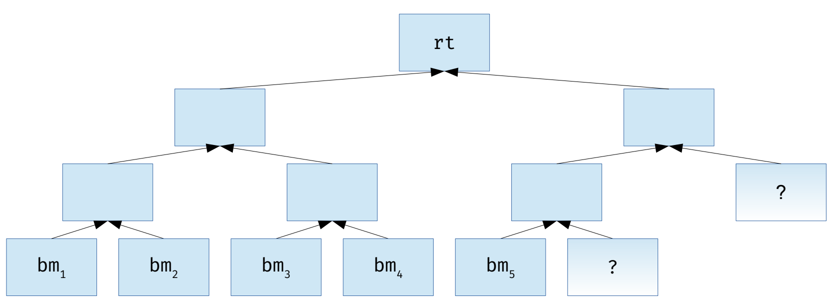
$$\mathsf{k} := \mathbf{CRH}\left(\boxed{\text{256 bit } \mathsf{r}}\boxed{\text{256 bit } \mathsf{InternalH}}\right)$$

$$\mathtt{bm} := \mathbf{CRH}\left(\boxed{\text{64 bit } \mathsf{v}}\boxed{\text{192 bit padding}}\boxed{\text{256 bit } \mathsf{k}}\right)$$

We say that the bucket commitment of a bucket $\mathsf{b} = \mathtt{BucketCommitment}(\mathsf{b})$.

Serials A serial sn is produced by $\mathbf{PRF}^{\mathbf{sn}}_{\mathsf{a_{sk}}}(\rho)$. Part of the process of spending a bucket is disclosing this serial without disclosing either ρ or $\mathsf{a_{sk}}$. This allows it to be used to prevent double-spending.

2.5 Bucket Commitment Tree



The bucket commitment tree is an *incremental merkle tree* of depth d used to store bucket commitments that transactions produce. Just as the *unspent transaction output set* (UTXO) used in Bitcoin proper, 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 transactions' bucket commitments have been entered into the tree associated with the previous block.

2.6 Spent Serials Map

Transactions insert serials into a *spent serials map* which is maintained alongside the UTXO by all nodes. Transactions that attempt to insert a serial into this map that already exists within it are invalid as they are attempting to double-spend.

2.7 Bitcoin Transactions

Bitcoin transactions consist of a vector of inputs (`vin`) and a vector of outputs (`vout`). Inputs and outputs are associated with a value. The total value of the outputs must not exceed the total value of the inputs.

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

`Pour`s are the primary operations performed by transactions that interact with our scheme. In principle, it is the action of spending N_{Old} buckets $\mathsf{b}^{\mathsf{old}}$ and creating N_{New} buckets $\mathsf{b}^{\mathsf{new}}$. **Zcash** transactions have an additional field `vpour`, which is a vector of `Pour`s. Each `Pour` consists of:

`vpub_old` which is a value $\mathsf{vpub}_{\mathsf{old}}$ that the pour removes from the value pool.

`vpub_new` which is a value $\mathsf{vpub}_{\mathsf{new}}$ that the pour inserts into the value pool.

`anchor` which is a merkle root rt of the bucket commitment tree at some block height in the past, or the merkle root produced by a previous pour in this transaction. **(TODO: clarify this)**

`scriptSig` which is a Bitcoin script which creates conditions for acceptance of a `Pour` in a transaction. The `SHA256Compress` hash of this value is $\mathsf{h}_{\mathsf{Sig}}$.

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

`serials` which is an N_{Old} size vector of serials $\mathsf{sn}_1^{old}, \mathsf{sn}_2^{old}, ..., \mathsf{sn}_{N_{Old}}^{old}$.

`commitments` which is a N_{New} size vector of bucket commitments $\mathsf{bm}_1^{new}, \mathsf{bm}_2^{new}, ..., \mathsf{bm}_{N_{New}}^{new}$.

`encrypted_buckets` which is a N_{New} size vector of encrypted buckets.

`vmacs` which is a N_{Old} size vector of message authentication codes h which bind $\mathsf{h}_{\mathsf{Sig}}$ to each a_{sk} of the `Pour`.

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

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

Non-malleability A `Pour` 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` can be seen, from the perspective of the transaction, as an input and an output simultaneously. `vpub_old` takes value from the value pool and `vpub_new` adds value to the value pool. As a result, `vpub_old` is treated like an *output* value, whereas `vpub_new` is treated like an *input* value.

Commitments and Serials Transactions which contain `Pour`s, when entered into the blockchain, append to the bucket commitment tree with all constituent bucket commitments. All of the constituent serials are also entered into the spent serials map of the blockchain *and* mempool. Transactions are not valid if they attempt to add a serial to the spent serials map that already exists.

3.1 $\pi_{\texttt{POUR}}$

In **Zcash** , N_{Old} and N_{New} are both 2.

A valid instance of $\pi_{\texttt{POUR}}$ assures that given a *primary input* (rt, sn_1^{old}, sn_2^{old}, bm_1^{new}, bm_2^{new}, $\mathsf{vpub}_{\mathsf{old}}$, $\mathsf{vpub}_{\mathsf{new}}$, $\mathsf{h}_{\mathsf{Sig}}$, h_1, h_2), a witness of *auxiliary input* (path_1, path_2, $\mathsf{b}_1^{\mathsf{old}}$, $\mathsf{b}_2^{\mathsf{old}}$, ${\mathsf{a}_{\mathsf{sk}}}_1^{old}$, ${\mathsf{a}_{\mathsf{sk}}}_2^{old}$, $\mathsf{b}_1^{\mathsf{new}}$, $\mathsf{b}_2^{\mathsf{new}}$) exists, where:

for each $i \in \{1,2\}$: $\mathsf{b}_{\mathsf{i}}^{\mathsf{old}} = (\mathsf{v}_{\mathsf{i}}^{\mathsf{old}}, {\mathsf{a}_{\mathsf{pk}}}_i^{old}, \mathsf{r}_i^{old}, \rho_i^{old})$

for each $i \in \{1,2\}$: $\mathsf{b}_{\mathsf{i}}^{\mathsf{new}} = (\mathsf{v}_{\mathsf{i}}^{\mathsf{new}}, {\mathsf{a}_{\mathsf{pk}}}_i^{new}, \mathsf{r}_i^{new}, \rho_i^{new})$.

The following conditions hold:

Merkle path validity for each $i \in \{1,2\} \mid \mathsf{v}_{\mathsf{i}}^{\mathsf{old}} \neq 0$: $\mathsf{path}_{\mathsf{i}}$ must be a valid path of depth d from $\texttt{BucketCommitment}(\mathsf{b}_{\mathsf{i}}^{\mathsf{old}})$ to bucket commitment merkle tree root rt.

Balance $\mathsf{vpub}_{\mathsf{old}} + \mathsf{v}_1^{\mathsf{old}} + \mathsf{v}_2^{\mathsf{old}} = \mathsf{vpub}_{\mathsf{new}} + \mathsf{v}_1^{\mathsf{new}} + \mathsf{v}_2^{\mathsf{new}}$.

Serial integrity for each $i \in \{1,2\}$: $\mathbf{PRF}^{\mathbf{sn}}_{{\mathsf{a}_{\mathsf{sk}}}_{\mathbf{i}}^{\mathbf{old}}}(\rho_i^{old}) = \mathsf{sn}_i^{old}$.

Spend authority for each $i \in \{1,2\}$: ${\mathsf{a}_{\mathsf{pk}}}_i^{old} = \mathbf{PRF}^{\mathbf{addr}}_{{\mathsf{a}_{\mathsf{sk}}}_{\mathbf{i}}^{\mathbf{old}}}(0)$.

Non-malleability for each $i \in \{1,2\}$: $h_i = \mathbf{PRF}^{\mathbf{pk}_{\mathbf{i-1}}}_{{\mathsf{a}_{\mathsf{sk}}}_{\mathbf{i}}^{\mathbf{old}}}(\mathsf{h}_{\mathsf{Sig}})$

Commitment integrity for each $i \in \{1,2\}$: $\mathsf{bm}_i^{new} = \texttt{BucketCommitment}(\mathsf{b}_{\mathsf{i}}^{\mathsf{new}})$