Zerocash: Decentralized Anonymous Payments from Bitcoin

Eli Ben-Sasson , Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, Madars Virza

INTRODUCTION

The Bitcoin cryptocurrency efficiently solves the “Double Spending “ and “Trusted Third Party” problems [5]. However, it conducts transactions between public key addresses, and every node in the blockchain is aware of how much money every public key address holds. Thereby providing “Pseudo Anonymity” instead of complete Anonymity. Transaction graphs of an address can be analyzed, and with the help of some social engineering, a complete contact of the owner can be obtained.

Of course, multiple public key and private key pairs can be used by users. The application of this solution is very limited, as the user will still end up spending coins from one of his account to another, before he uses the other account to spend money. Earlier solutions to this privacy issue includes having Mixes (also called Laundries or Tumblers). These are entities who pool bitcoins from a number of people, shuffle them and return the same number of coins for a small fee. This is done to hide the true source and destination of the coins, and basically to modify the input chains. However, this has the disadvantages of delay, undue trust on the Mix party, etc.

Zerocoin, a predecessor to ZeroCash, was invented by I. Miers, C. Garman, M. Green, and A. D. Rubin [4]. Zerocoin effectively solved the Privacy issue of Bitcoin, by incorporating Zero-Knowledge Proofs in the Blockchain. In case of Zerocoin, a spender can burn one bitcoin, and create a zerocoin, publishing a zero-knowledge proof that he owned the bitcoin, and has the authority to spend the zerocoin. This prevents Transaction Graph analysis, as coins can no longer be linked to the previous transaction history, yet every node in the blockchain can securely verify that the transaction is valid, the coins are owned by the spender and the coins are not fake. Bitcoin used Digital signatures for this verification, which revealed a lot more information. Thus zero-coin can help a person to unlink a bitcoin from its origin. But it does not hide the amount or the receiver, as only one zerocoin can be created at a time from one bitcoin. So to wash 5 bitcoins as zerocoins, 5 transactions would be conducted. Also, Zerocoin was a decentralized Mix, in the sense that it consumes bitcoins and gives zerocoins, and not a standalone currency by itself.

One problem with Zerocoin was that it allowed only fixed denomination transaction. This is the problem that “Zerocash” solves. ZCash allows variable denomination currency, which can either be on the top of the bitcoin blockchain, or a standalone currency by itself. Also, Zerocoin proofs are very huge in size (40 Kbs) and requires 450 ms to verify which makes their practical use on the bitcoin blockchain difficult. ZCash solves this problem, by the use of “**Zero Knowledge Succinct Non-Interactive arguments of knowledge**” (zk-SNARKS).

ZK-SNARKS

Zk-SNARKS [2] can be understood as follows:

- **Zero-knowledge** - nothing is revealed beyond the truth of the statement
- **Succinct** indicates that the zero-knowledge proof is short and can be verified quickly.
- **Non-interactive -** verifier does not have to interact with the prover. Instead, the prover can publish (write) their proof in advance, and a verifier can ensure its correctness.
- **Argument –** computationally sound proofs (Prover is not infinitely powerful)
- **of Knowledge** means a proof of knowledge of some defined computation.

The formal definition states three algorithms :

- **KeyGen** is typically run by the verifier. It takes as input 1^k (k being the security parameter) and outputs some keypair, pk and vk.
- **P** (the proving algorithm) takes as input pk, a word x∈L and a NP-witness **w** for x, and outputs the proof **π**.
- **V** (the verifying algorithm) takes as input **vk**, x and π, and returns 0 or 1 depending on whether the verifier "accepts" the proof that x is in L.

The Verifier returns a 1, certifying that x∈L , but this does not certify that the prover must have not cheated by generating (x,π) in a non-standard way, without knowing the witness w. For this, a SNARK requires an additional property called “Extractability”, which needs to be satisfied as follows:

Extractability: for any polynomial-time prover P* running on input (pk,z)(where z is some auxiliary input) and producing a pair (x,π), there is a polynomial-time extractor E_{P^*} also running on input (pk,z) and producing w, such that the probability that (x,π) is accepted by the verifier and that w is not a NP-witness for x is negligible in k. [3]

Apart from Extractability, the zk-SNARKS also need to satisfy the Completeness, Succinctness, Soundness and Proof of Knowledge as well as Perfect Zero-Knowledge properties.

The ZCash construction uses Pinnochio zk-SNARKS. Pinocchio is a practical zk-SNARK that allows a prover to perform cryptographically verifiable computations with verification effort potentially less than performing the computation itself. [1, 6]

Generation of Public Parameters

For the zk-SNARKS that we understood above, some public parameters need to be generated at the beginning of the scheme in setup phase. These parameters are used to generate proofs and verify transactions.

A Multi-Party Computation Ceremony is used by the ZCash, which allows multiple independent parties to collaboratively construct the parameters. This protocol has the property that, in order to compromise the final parameters, all of the participants would have to be compromised or dishonest [7].

ZeroCash DAP Scheme

In this paper, the authors have provided the construction of a DAP (Decentralized Anonymous Payment) scheme and proved its security under specific assumptions.

The Construction of ZCash can be summarized into 6 steps as follows:

Step 1. User anonymity with fixed-value coins

A User u samples a random serial number sn and a trapdoor r, computes a coin commitment cm := $COMM_r(sn)$, and sets c := (r,sn, cm). A corresponding mint transaction tx_{Mint}, containing cm (but not sn or r), is sent to the ledger; tx_{Mint} is appended to the ledger only if u has paid 1BTC to a backing escrow pool. This means mint transactions resemble present-day "Certificates of Deposit".

CM_{List} denotes the list of all coin commitments. User u can spend the coin c with serial number sn, giving a zero-knowledge proof π that it appears in the CM_{List}. This redeeming of coins is done using Tx_{spend} transaction.

Privacy is achieved because only the serial number sn is revealed, no information about r is revealed, and finding a $COMM_x(sn)$ from the long CM_{List} is hard.

Double spending is not possible, because each serial number is unique. On spending a transaction, you reveal its serial number. If the same serial number gets revealed twice, the transaction will not be accepted into the ledger.

Step 2. Compressing the list of coin commitments

CM_{List} i.e. the list of all coin commitments was linear in step 1. In that case, for zero-knowledge proof verification, the time required to search the commitment from the list of coin commitments as well as the space is linear in size of the CMList. However, a much better representation would be the use of Merkle Trees. The authors chose Collision-Resistant Hash-based Merkle tree, as updating the root of Merkle Tree to account for insertion of new leaf node can be done in time and space proportional to tree depth.

The new NP statement for proof is "I know r such that $COMM_r(sn)$ appears as a leaf in a CRH-based Merkle tree whose root is rt."

Zerocash uses Merkle Trees of depth 64, thereby having 2^{64} possible coins.

Step 3. Extending coins for Direct Anonymous Payments

A) Adding addresses

In the previous steps, we solved the "Double Spending" Problem of a Payment scheme, due to the presence of unique serial numbers. However, we have not yet established the ownership of a coin. Hence, if user A mints a coin of serial number sn and gives it to B, he can still spend the same coin again before B

spends it. Moreover, even if he doesn't, he can come to know when B spends this coin, as he knew the serial number. This harms the anonymity of a payment scheme.

As a solution, we will first add the concept of address (a_{pk}, a_{sk}). Every coin that gets minted, will have the value of the new owner's public key a_{pk} inherent to it. Hence, only the user having a_{sk} will be able to spend this coin.

B) Fixed to variable Denomination

Our definition till now only includes fixed value payments. This means that in very transaction of this scheme, one bitcoin is burnt and one new coin is minted (similar to zerocoin). Hence , for a transaction worth 500 bitcoins, 500 new zcash transactions will have to be done which is not practical. Also, the fixed denomination till now only supports payments which are a multiple of 1 BTC and not any random value like 2.35 BTC. As a solution for this problem, we again introduce a new definition of coin c, with a value v. The value of the coin will be inherent in that coin. A coin of any value can now be minted using Tx_{MINT}.

For these new definitions, we use 3 Pseudorandom Functions, derived from the same PRF as follows :

$PRF_x^{addr}(.)$ $PRF_x^{sn}(.)$ $PRF_x^{pk}(.)$

Where $PRF_x^{sn}(.)$ is collision resistant.

For addresses, we select a random seed a_{sk} and sample $a_{pk} := PRF_{ask}^{addr}(0)$

Then we determine ρ, which is a secret value that determines the coin's serial number sn = $PRF_{ask}^{sn}(\rho)$

For incorporating value v, we define the coin as a commitment of a value v, and k where k is the commitment of a_{pk} and ρ. Two new randomness values r and s are used when creating each of these commitments.

The structure of this new coin looks like this :

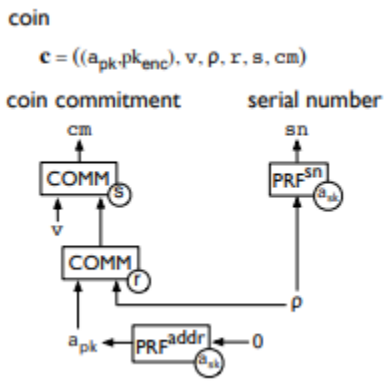


Fig 1: Coin Commitment in ZCash

With this new definition, we have solved both the problems mentioned above.

For Pour operation Tx_{POUR}(earlier called TX_{spent}), a set of input coins can be further broken down into 2 coins of value v1 and v2 respectively. The total value of output coins equals the total value of input coins respectively.

Suppose user A wants to split his coins and send to user B and C respectively. Then A can create two new coin commitments, by sampling ρ_B and ρ_C respectively, and using the public key addresses of users B and C to create the new coins. Thus, only B and C will be able to spend their respective new coins and not A. This resulting pour transaction gets appended to the Basecoin ledger. This Pour transaction reveals the serial number of the input coins, and it would be rejected if the serial number is already known (i.e. double spent).

We redefine the commitment scheme formally as follows :

> *"Given the Merkle-tree root* rt, *serial number* $\mathsf{sn}^{\mathsf{old}}$, *and coin commitments* $\mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}$, *I know coins* $\mathbf{c}^{\mathsf{old}}, \mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}}$, *and address secret key* $a_{\mathsf{sk}}^{\mathsf{old}}$ *such that:*
>
> - *The coins are well-formed: for* $\mathbf{c}^{\mathsf{old}}$ *it holds that* $k^{\mathsf{old}} = \mathsf{COMM}_{r^{\mathsf{old}}}(a_{\mathsf{pk}}^{\mathsf{old}} \| \rho^{\mathsf{old}})$ *and* $\mathsf{cm}^{\mathsf{old}} = \mathsf{COMM}_{s^{\mathsf{old}}}(v^{\mathsf{old}} \| k^{\mathsf{old}})$; *and similarly for* $\mathbf{c}_1^{\mathsf{new}}$ *and* $\mathbf{c}_2^{\mathsf{new}}$.
> - *The address secret key matches the public key:* $a_{\mathsf{pk}}^{\mathsf{old}} = \mathsf{PRF}_{a_{\mathsf{sk}}^{\mathsf{old}}}^{\mathsf{addr}}(0)$.
> - *The serial number is computed correctly:* $\mathsf{sn}^{\mathsf{old}} := \mathsf{PRF}_{a_{\mathsf{sk}}^{\mathsf{old}}}^{\mathsf{sn}}(\rho^{\mathsf{old}})$.
> - *The coin commitment* $\mathsf{cm}^{\mathsf{old}}$ *appears as a leaf of a Merkle-tree with root* rt.
> - *The values add up:* $v_1^{\mathsf{new}} + v_2^{\mathsf{new}} = v^{\mathsf{old}}$."

Step 4: Sending coins

For spending a coin, along with the secret key, knowledge of all secret parameters within a coin are required. This includes randomness values r, s, ρ. In our example, A could have sent a secret message to B and C telling those secret parameters, but that requires extra setup. Hence we modify our payment scheme even further. Each user now has a key pair ($addr_{pk}$, $addr_{sk}$), where $addr_{pk}$ = (a_{pk}, pk_{enc}) and $addr_{sk}$ = (a_{sk},sk_{enc}). The values (a_{pk}, a_{sk}) are generated as before. In addition, (pk_{enc},sk_{enc}) is a key pair for a key-private encryption scheme.

The user A now encrypts the message (v,r,s, ρ) with B's secret key. B can find and decrypt this message by scanning the pour transactions on the public ledger. This scheme is secure because only B has the secret key to open this cipher text. A can do the same for another user C.

Step 5 : Public Outputs

Till now, all transactions took the new currency coin into account. But in order for this scheme to work in the Bitcoin Blockchain, a user might want to redeem his coin back in bitcoins. To incorporate this, the pour operation is modified to add a value v_{pub} in its output value.

The new balance equation is : $\text{“}v_1^{\text{new}} + v_2^{\text{new}} + v_{\text{pub}} = v^{\text{old}}\text{”}$.

There is a string info field in which the target of the public output is specified. This string info can be used for storing refund information etc. This V_{pub} value is optional, and can be set to 0.

Step 6: Non-Malleability

An adversary can perform a Malleability attack here, by modifying information in the public string field of TX_{POUR} and retargeting the funds. To prevent this, digital signatures are used.

CONSTRUCTION

As the above construction relies on the use of zk-SNARKS, a one-time Public Parameter set-up is required. This means a trust-based set-up is required. If this setup is corrupted by a malicious party, the soundness property of ZK is affected, however, Anonymity still holds true.

Zk-SNARK is the heaviest component of the Zerocash system, and hence it is instantiated properly. The zk-SNARK is used to Prove and Verify “POUR” statement of NP. A small circuit C_{POUR} is used to prove the NP statement of POUR.

The Details of the DAP scheme can be summarized as follows:

A) Ingredients of the DAP scheme based on SHA-256:

1. Statistically-Hiding Commitment schemes
2. PRFs
3. Collision Resistant Hash Functions
4. One-Time strongly unforgeable digital signatures
5. Key-Private Public Key Encryption

B) The data-structures used in this scheme include:

- Basecoin-ledger
- Public parameters
- Addresses
- Coins
- New Transactions Types: Mint Transactions and Pour Transactions
- Commitments of minted coins and serial numbers of spent coins
- Merkle tree over commitments

C) Zk-SNARKs for NP-statement of POUR

D) Algorithms

The Algorithms used include the following :

1. System setup
2. Creating payment addresses
3. Minting coins
4. Pouring coins
5. Verifying Transactions
6. Receiving Coins

In this construction, Prover run-time is of few minutes and Verifier run-time of few milliseconds. The logic of each of these algorithms can be found in the Fig 2.

Properties of this system are :

- **Completeness**
  The Completeness of a DAP scheme requires the ability to spent unspent coins. This means that if c1 and c2 are two valid coins who commitments appear in the ledger L, and their serial numbers have not yet been revealed, the c1 and c2 can be spent using POUR statement.

- **Security** : The security of a DAP scheme is characterized by the below 3 properties:

  1) **Ledger indistinguishability** : This means that nothing is revealed besides public information from the ledger, even by chosen-transaction adversary.

  2) **Transaction non-malleability** : An adversary cannot manipulate transactions en-route to the ledger

  3) **Balance** : No user can own more money than received or minted by them.

The paper defines DAP schemes in its construction. The authors have also discussed the concrete instantiation in Zerocash. Zerocash can either be used as a stand-alone currency (fork of bitcoin) called altcoins, or it can be integrated into existing ledger-based currencies.

The paper also provides microbenchmarks for the prototype implementation, as well as results based on full-network simulations.

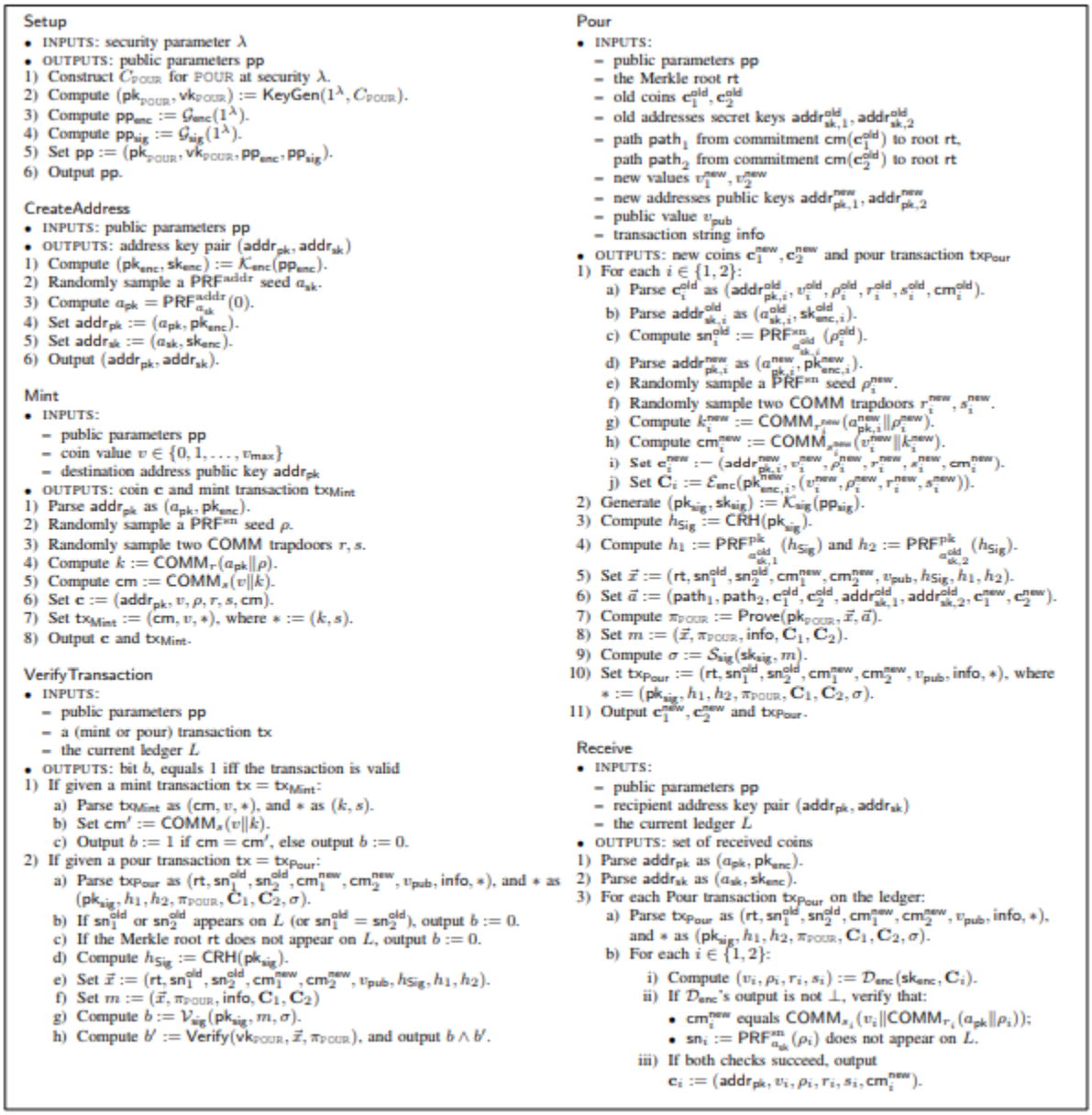
Implementation

A) Instantiating the Building Blocks

B) Instantiating the NP statement of POUR

C) Instantiating Sig (A non-malleable variant of Elliptic-Curve Digital Signature Algorithm -ECDSA is used here) [9]

D) Instantiating Enc (Elliptic Curve Integrated Encryption Scheme- ECIES is used) [10 , 11]



Setup
- INPUTS: security parameter λ
- OUTPUTS: public parameters pp

1) Construct C_{POUR} for POUR at security λ.
2) Compute $(\mathsf{pk}_{\text{POUR}}, \mathsf{vk}_{\text{POUR}}) := \mathsf{KeyGen}(1^\lambda, C_{\text{POUR}})$.
3) Compute $\mathsf{pp}_{\text{enc}} := \mathcal{G}_{\text{enc}}(1^\lambda)$.
4) Compute $\mathsf{pp}_{\text{sig}} := \mathcal{G}_{\text{sig}}(1^\lambda)$.
5) Set $\mathsf{pp} := (\mathsf{pk}_{\text{POUR}}, \mathsf{vk}_{\text{POUR}}, \mathsf{pp}_{\text{enc}}, \mathsf{pp}_{\text{sig}})$.
6) Output pp.

CreateAddress
- INPUTS: public parameters pp
- OUTPUTS: address key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$

1) Compute $(\mathsf{pk}_{\text{enc}}, \mathsf{sk}_{\text{enc}}) := \mathcal{K}_{\text{enc}}(\mathsf{pp}_{\text{enc}})$.
2) Randomly sample a $\mathsf{PRF}^{\text{addr}}$ seed a_{sk}.
3) Compute $a_{\mathsf{pk}} = \mathsf{PRF}^{\text{addr}}_{a_{\mathsf{sk}}}(0)$.
4) Set $\mathsf{addr}_{\mathsf{pk}} := (a_{\mathsf{pk}}, \mathsf{pk}_{\text{enc}})$.
5) Set $\mathsf{addr}_{\mathsf{sk}} := (a_{\mathsf{sk}}, \mathsf{sk}_{\text{enc}})$.
6) Output $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$.

Mint
- INPUTS:
  - public parameters pp
  - coin value $v \in \{0, 1, \ldots, v_{\max}\}$
  - destination address public key $\mathsf{addr}_{\mathsf{pk}}$
- OUTPUTS: coin c and mint transaction $\mathsf{tx}_{\text{Mint}}$

1) Parse $\mathsf{addr}_{\mathsf{pk}}$ as $(a_{\mathsf{pk}}, \mathsf{pk}_{\text{enc}})$.
2) Randomly sample a PRF^{sn} seed ρ.
3) Randomly sample two COMM trapdoors r, s.
4) Compute $k := \mathsf{COMM}_r(a_{\mathsf{pk}} \| \rho)$.
5) Compute $\mathsf{cm} := \mathsf{COMM}_s(v \| k)$.
6) Set $\mathbf{c} := (\mathsf{addr}_{\mathsf{pk}}, v, \rho, r, s, \mathsf{cm})$.
7) Set $\mathsf{tx}_{\text{Mint}} := (\mathsf{cm}, v, *)$, where $* := (k, s)$.
8) Output c and $\mathsf{tx}_{\text{Mint}}$.

VerifyTransaction
- INPUTS:
  - public parameters pp
  - a (mint or pour) transaction tx
  - the current ledger L
- OUTPUTS: bit b, equals 1 iff the transaction is valid

1) If given a mint transaction $\mathsf{tx} = \mathsf{tx}_{\text{Mint}}$:
   a) Parse $\mathsf{tx}_{\text{Mint}}$ as $(\mathsf{cm}, v, *)$, and $*$ as (k, s).
   b) Set $\mathsf{cm}' := \mathsf{COMM}_s(v \| k)$.
   c) Output $b := 1$ if $\mathsf{cm} = \mathsf{cm}'$, else output $b := 0$.
2) If given a pour transaction $\mathsf{tx} = \mathsf{tx}_{\text{Pour}}$:
   a) Parse $\mathsf{tx}_{\text{Pour}}$ as $(\mathsf{rt}, \mathsf{sn}_1^{\text{old}}, \mathsf{sn}_2^{\text{old}}, \mathsf{cm}_1^{\text{new}}, \mathsf{cm}_2^{\text{new}}, v_{\text{pub}}, \mathsf{info}, *)$, and $*$ as $(\mathsf{pk}_{\text{sig}}, h_1, h_2, \pi_{\text{POUR}}, \mathbf{C}_1, \mathbf{C}_2, \sigma)$.
   b) If $\mathsf{sn}_1^{\text{old}}$ or $\mathsf{sn}_2^{\text{old}}$ appears on L (or $\mathsf{sn}_1^{\text{old}} = \mathsf{sn}_2^{\text{old}}$), output $b := 0$.
   c) If the Merkle root rt does not appear on L, output $b := 0$.
   d) Compute $h_{\mathsf{Sig}} := \mathsf{CRH}(\mathsf{pk}_{\text{sig}})$.
   e) Set $\vec{x} := (\mathsf{rt}, \mathsf{sn}_1^{\text{old}}, \mathsf{sn}_2^{\text{old}}, \mathsf{cm}_1^{\text{new}}, \mathsf{cm}_2^{\text{new}}, v_{\text{pub}}, h_{\mathsf{Sig}}, h_1, h_2)$.
   f) Set $m := (\vec{x}, \pi_{\text{POUR}}, \mathsf{info}, \mathbf{C}_1, \mathbf{C}_2)$
   g) Compute $b := \mathcal{V}_{\text{sig}}(\mathsf{pk}_{\text{sig}}, m, \sigma)$.
   h) Compute $b' := \mathsf{Verify}(\mathsf{vk}_{\text{POUR}}, \vec{x}, \pi_{\text{POUR}})$, and output $b \wedge b'$.

Pour
- INPUTS:
  - public parameters pp
  - the Merkle root rt
  - old coins $\mathbf{c}_1^{\text{old}}, \mathbf{c}_2^{\text{old}}$
  - old addresses secret keys $\mathsf{addr}_{\mathsf{sk},1}^{\text{old}}, \mathsf{addr}_{\mathsf{sk},2}^{\text{old}}$
  - path path_1 from commitment $\mathsf{cm}(\mathbf{c}_1^{\text{old}})$ to root rt, path path_2 from commitment $\mathsf{cm}(\mathbf{c}_2^{\text{old}})$ to root rt
  - new values $v_1^{\text{new}}, v_2^{\text{new}}$
  - new addresses public keys $\mathsf{addr}_{\mathsf{pk},1}^{\text{new}}, \mathsf{addr}_{\mathsf{pk},2}^{\text{new}}$
  - public value v_{pub}
  - transaction string info
- OUTPUTS: new coins $\mathbf{c}_1^{\text{new}}, \mathbf{c}_2^{\text{new}}$ and pour transaction $\mathsf{tx}_{\text{Pour}}$

1) For each $i \in \{1, 2\}$:
   a) Parse $\mathbf{c}_i^{\text{old}}$ as $(\mathsf{addr}_{\mathsf{pk},i}^{\text{old}}, v_i^{\text{old}}, \rho_i^{\text{old}}, r_i^{\text{old}}, s_i^{\text{old}}, \mathsf{cm}_i^{\text{old}})$.
   b) Parse $\mathsf{addr}_{\mathsf{sk},i}^{\text{old}}$ as $(a_{\mathsf{sk},i}^{\text{old}}, \mathsf{sk}_{\text{enc},i}^{\text{old}})$.
   c) Compute $\mathsf{sn}_i^{\text{old}} := \mathsf{PRF}^{\text{sn}}_{a_{\mathsf{sk},i}^{\text{old}}}(\rho_i^{\text{old}})$.
   d) Parse $\mathsf{addr}_{\mathsf{pk},i}^{\text{new}}$ as $(a_{\mathsf{pk},i}^{\text{new}}, \mathsf{pk}_{\text{enc},i}^{\text{new}})$.
   e) Randomly sample a PRF^{sn} seed ρ_i^{new}.
   f) Randomly sample two COMM trapdoors $r_i^{\text{new}}, s_i^{\text{new}}$.
   g) Compute $k_i^{\text{new}} := \mathsf{COMM}_{r_i^{\text{new}}}(a_{\mathsf{pk},i}^{\text{new}} \| \rho_i^{\text{new}})$.
   h) Compute $\mathsf{cm}_i^{\text{new}} := \mathsf{COMM}_{s_i^{\text{new}}}(v_i^{\text{new}} \| k_i^{\text{new}})$.
   i) Set $\mathbf{c}_i^{\text{new}} := (\mathsf{addr}_{\mathsf{pk},i}^{\text{new}}, v_i^{\text{new}}, \rho_i^{\text{new}}, r_i^{\text{new}}, s_i^{\text{new}}, \mathsf{cm}_i^{\text{new}})$.
   j) Set $\mathbf{C}_i := \mathcal{E}_{\text{enc}}(\mathsf{pk}_{\text{enc},i}^{\text{new}}; (v_i^{\text{new}}, \rho_i^{\text{new}}, r_i^{\text{new}}, s_i^{\text{new}}))$.
2) Generate $(\mathsf{pk}_{\text{sig}}, \mathsf{sk}_{\text{sig}}) := \mathcal{K}_{\text{sig}}(\mathsf{pp}_{\text{sig}})$.
3) Compute $h_{\mathsf{Sig}} := \mathsf{CRH}(\mathsf{pk}_{\text{sig}})$.
4) Compute $h_1 := \mathsf{PRF}^{\text{pk}}_{a_{\mathsf{sk},1}^{\text{old}}}(h_{\mathsf{Sig}})$ and $h_2 := \mathsf{PRF}^{\text{pk}}_{a_{\mathsf{sk},2}^{\text{old}}}(h_{\mathsf{Sig}})$.
5) Set $\vec{x} := (\mathsf{rt}, \mathsf{sn}_1^{\text{old}}, \mathsf{sn}_2^{\text{old}}, \mathsf{cm}_1^{\text{new}}, \mathsf{cm}_2^{\text{new}}, v_{\text{pub}}, h_{\mathsf{Sig}}, h_1, h_2)$.
6) Set $\vec{a} := (\mathsf{path}_1, \mathsf{path}_2, \mathbf{c}_1^{\text{old}}, \mathbf{c}_2^{\text{old}}, \mathsf{addr}_{\mathsf{sk},1}^{\text{old}}, \mathsf{addr}_{\mathsf{sk},2}^{\text{old}}, \mathbf{c}_1^{\text{new}}, \mathbf{c}_2^{\text{new}})$.
7) Compute $\pi_{\text{POUR}} := \mathsf{Prove}(\mathsf{pk}_{\text{POUR}}, \vec{x}, \vec{a})$.
8) Set $m := (\vec{x}, \pi_{\text{POUR}}, \mathsf{info}, \mathbf{C}_1, \mathbf{C}_2)$.
9) Compute $\sigma := \mathcal{S}_{\text{sig}}(\mathsf{sk}_{\text{sig}}, m)$.
10) Set $\mathsf{tx}_{\text{Pour}} := (\mathsf{rt}, \mathsf{sn}_1^{\text{old}}, \mathsf{sn}_2^{\text{old}}, \mathsf{cm}_1^{\text{new}}, \mathsf{cm}_2^{\text{new}}, v_{\text{pub}}, \mathsf{info}, *)$, where $* := (\mathsf{pk}_{\text{sig}}, h_1, h_2, \pi_{\text{POUR}}, \mathbf{C}_1, \mathbf{C}_2, \sigma)$.
11) Output $\mathbf{c}_1^{\text{new}}, \mathbf{c}_2^{\text{new}}$ and $\mathsf{tx}_{\text{Pour}}$.

Receive
- INPUTS:
  - public parameters pp
  - recipient address key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$
  - the current ledger L
- OUTPUTS: set of received coins

1) Parse $\mathsf{addr}_{\mathsf{pk}}$ as $(a_{\mathsf{pk}}, \mathsf{pk}_{\text{enc}})$.
2) Parse $\mathsf{addr}_{\mathsf{sk}}$ as $(a_{\mathsf{sk}}, \mathsf{sk}_{\text{enc}})$.
3) For each Pour transaction $\mathsf{tx}_{\text{Pour}}$ on the ledger:
   a) Parse $\mathsf{tx}_{\text{Pour}}$ as $(\mathsf{rt}, \mathsf{sn}_1^{\text{old}}, \mathsf{sn}_2^{\text{old}}, \mathsf{cm}_1^{\text{new}}, \mathsf{cm}_2^{\text{new}}, v_{\text{pub}}, \mathsf{info}, *)$, and $*$ as $(\mathsf{pk}_{\text{sig}}, h_1, h_2, \pi_{\text{POUR}}, \mathbf{C}_1, \mathbf{C}_2, \sigma)$.
   b) For each $i \in \{1, 2\}$:
      i) Compute $(v_i, \rho_i, r_i, s_i) := \mathcal{D}_{\text{enc}}(\mathsf{sk}_{\text{enc}}, \mathbf{C}_i)$.
      ii) If $\mathcal{D}_{\text{enc}}$'s output is not $\perp$, verify that:
         - $\mathsf{cm}_i^{\text{new}}$ equals $\mathsf{COMM}_{s_i}(v_i \| \mathsf{COMM}_{r_i}(a_{\mathsf{pk}} \| \rho_i))$;
         - $\mathsf{sn}_i := \mathsf{PRF}^{\text{sn}}_{a_{\mathsf{sk}}}(\rho_i)$ does not appear on L.
      iii) If both checks succeed, output $\mathbf{c}_i := (\mathsf{addr}_{\mathsf{pk}}, v_i, \rho_i, r_i, s_i, \mathsf{cm}_i^{\text{new}})$.

Fig. 2: Construction of a DAP scheme using zk-SNARKs and other ingredients.

Performance Improvement

The authors ran a instance of this blockchain using Amazon EC2 resources. The simulated using different traffic mixes of ZCash and Bitcoin belonging to the range {0,25,59,75,100}%. This simulation was run with

1000 nodes ,and each node having about 32 peers on average. The size of this blockchain(1000 nodes) is one-third of the size of the bitcoin blockchain (3500 reachable nodes at a time). The results are represented in Fig 3 and Fig 4:

		Intel Core i7-2620M @ 2.70GHz 12GB of RAM	Intel Core i7-4770 @ 3.40GHz 16GB of RAM	
		1 thread	1 thread	8 threads
KeyGen	Time	7 min 48 s	5 min 17 s	4 min 11 s
	Proving key	896 MiB		
	Verification key	749 B		
Prove	Time	2 min 55 s	2 min 2 s	1 min 3 s
	Proof	288 B		
Verify	Time	8.5 ms	5.4 ms	

Fig. 3: Performance of our zk-SNARK for the NP statement POUR. ($N = 10, \sigma \leq 2.5\%$)

Intel Core i7-4770 @ 3.40GHz with 16GB of RAM (1 thread)		
Setup	Time	5 min 17 s
	pp	896 MiB
CreateAddress	Time	326.0 ms
	$addr_{pk}$	343 B
	$addr_{sk}$	319 B
Mint	Time	23 µs
	Coin c	463 B
	tx_{Mint}	72 B
Pour	Time	2 min 2.01 s
	tx_{Pour}	996 B[16]
VerifyTransaction	mint	8.3 µs
	pour (excludes L scan)	5.7 ms
Receive	Time (per pour tx)	1.6 ms

Fig. 4: Performance of Zerocash algorithms. ($N = 10, \sigma \leq 2.5\%$[17])

As compared to Zercoin, the performance of Zerocash provides the following improvement :

Area	Zerocoin	Zerocash	Improvement
Transaction Size	45 KB	288 B	97.7%
Spend Verification Time	450 ms	5.7 ms	98.6%

Table 1 : Performance Comparison of ZeroCoin and ZeroCash

Future directions

One of the future items mentioned in the paper was using Zk-SNARKS to let a user prove that he paid his due taxes on all transactions without revealing those transactions, their amounts, or even the amount of taxes paid.

Optimizations

In the extended version of this paper [12], several optimizations to the existing structures have been mentioned as follows :

1. **Everlasting Anonymity**
   This states that on the ZeroCash Ledger, transactions may persist virtually forever,hence users may wish to ensure the anonymity of their transactions also lasts forever, even if particular primitives are eventually broken (Quantum Computers, Cryptanalytic inventions etc.).
   Solutions for ensuring anonymity in face of such breakthrough attacks have been explained in the extended paper [12].

2. **Fast Block Propagation**
   This states that a node can validate proof-of-work of a block and other transactions right away and then immediately forward the block to its peers. The verification of ZCash Mint and Pour transactions can be done by a node after propagating, and before accepting the block. This would help reduce propagation delays.

3. **Batched Merkle Tree Updates**
   Each node in the Ledger stores a CM_{List} and SN_{List} in addition to the Ledger L. Instead of verifying whether the commitment exists in the entire CM_{List}, user can act when a new block gets added and the old coin commitments path can be used to compute the path of new coin commitments.
   With this, each user incurs an additional storage requirement of 2 KiB, but does not need to store the entire CM_{List} anymore.

4. **Scaling to 2^{64} serial numbers**
   In case of Bitcoin, nodes need to store only unspent transaction outputs, and this list gets reduced as transactions are spend. However for ZCash, nodes need to store the entire SN_{List} which grows with time. To solve this 3 steps are taken :

   a) A Merkle-Tree over SN_{List} is created to verify the membership of SN in SN_{List}. Leaves contain unspent serial numbers. If the serial number is unspent, a path would exist from the root of merkle tree to the serial number leaf, and this needs to be added to $Tx_{POUR.}$

   b) Instead of one Merkle-Tree, the SN_{List} is divided into sublists of Merkle Trees in chronological order. The user u needs to provide a path for each of these SN Merkle Trees in the Tx_{POUR}.

   c) To avoid costs of adding computing paths for each coin, timestamps are added.

Conclusion

While ZCash has improved significantly over Zerocoin, still there is a lot of room for development. A two minute time for generating zk-SNARK proof is a lot to expect from transacting users, as they can choose the traditional day methods like PayPal that confirm transactions in milliseconds. Even the Bitcoin offers a faster transaction time frame as it does not have the generation and verification of zk-proof criteria. Hence for this reason, we need to optimize the Proving algorithm.

References

[1] Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., & Virza, M. (2014). Zerocash: Decentralized Anonymous Payments from Bitcoin (extended version).

[2] Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., & Paneth, O. (2012). Erratum: Succinct Non-interactive Arguments via Linear Interactive Proofs. *TCC*.

[3] Gennaro, R., Gentry, C., Parno, B., & Raykova, M. (2012). Quadratic Span Programs and Succinct NIZKs without PCPs. *EUROCRYPT*.

[4] Miers, I., Garman, C., Green, M.D., & Rubin, A.D. (2013). Zerocoin: Anonymous Distributed E-Cash from Bitcoin. *2013 IEEE Symposium on Security and Privacy*, 397-411.

[5] Nakamoto, S. (2009). Bitcoin: A Peer-to-Peer Electronic Cash System. Cryptography Mailing list at https://metzdowd.com.

[6] Merkle, R.C. (1980). Protocols for Public Key Cryptosystems. *1980 IEEE Symposium on Security and Privacy*, 122-122.

[7] https://z.cash/technology/paramgen.html

[8] Bowe, S., Gabizon, A., & Green, M.D. (2017). A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK. *IACR Cryptology ePrint Archive, 2017*, 602.

[9] Wuille P. (2014), "Proposed BIP for dealing with malleability," Available at https://gist.github.com/sipa/8907691.

[10] Shoup, V. (2001). A Proposal for an ISO Standard for Public Key Encryption. *IACR Cryptology ePrint Archive, 2001*, 112.

[11] Certicom Research (2000) , "SEC 1: Elliptic curve cryptography," [Online]. Available: http://www.secg.org/collateral/sec1 final.pdf