Zerocash: Decentralized Anonymous Payments from Bitcoin*

Shahla Atapoor
Supervised by Janno Siim
University of Tartu, Estonia
shahla.atapoor@ut.ee
December 15, 2018

Abstract

Bitcoin is a widely used digital coin that allows user to have direct payments without any third party. Since all transactions are done between public addresses and are linked together by a public ledger, so one can track the payments of a particular address and violate privacy of the owner by long time monitoring. To deal with such privacy concerns, several solutions have been proposed, of which Zerocash [BSCG+14] is one of the most interesting ones. Zerocash (a.k.a. Zcash) is a digital coin that uses zero-knowledge Succinct ARguments of Knowledge (zk-SNARKs) to provide more anonymous transactions by hiding payment's origin, destination and the sum.

In this report, we review the construction of Zcash and explain step-by-step how the coin works and how it provides strong privacy for end-users. We will observe in details that Zcash is an instantiation of decentralized anonymous payment (DAP) schemes that is proposed in the seminal paper of Zerocash. Following the great performance of zk-SNARKs, transactions in Zcash are *succinct* (less than 1 kB) and verifying them takes around few milliseconds.

1 Introduction

During the last decade cryptographic currencies are become very well known and attract a lot of attentions. The most famous and valuable one is Bitcoin which introduced by Satoshi Nakamoto in 2008 [Nak08]. Bitcoin being a decentralized digital currency, has a little problem that its coins are digital, they are just numbers which one needs to store and the other one can copy the number and spend it several times. The brilliant solution of Satoshi Nakamato is broadcasting every transaction and store it in the Blockchain. The first implication is privacy, we know that one transaction is broadcast so it can be analyzed. There are several studies and papers that looking at the transactions' graph, can reveal a lot of information about the goal of transaction, amount of money and who does the transaction, which means that system leaks information. Bitcoin is pseudonymous in order to maintain privacy but users are mostly using one or just a few addresses so by doing some analyses it is easy to track them. On the other side if you are a merchants, once your competitor knows what your key is, they can track your cash flow, track your costumers and a lot of valuable business information may be leaked. One possible solution for hiding the sender of transaction is mixing the transactions by the trusted party, but that requires trust in a third party.

The users of Bitcoin need to send their coins to the third party, in order to use laundry for mixing the coins, and wash them up, and get them back. (They get different coins but with the same value). This approach still has some problems including, the time which is needed to have the coins completely mixed, must be large enough and there are some other problems like stealing or tracing the coins by mix. On the other hand, the legitimate users require following requirement: (1) The users want to have secure transactions. (2) The users want to have privacy without doing much effort. (3) Sometimes they may even not be aware of weakness of privacy.

Zerocoin which proposed by Mires et al. tackled the mentioned problems with very nice and sophisticated cryptographic approach. The users who wish to have their transaction history hidden can use the help of Mixes(laundries). Zerocoin Created an anonymous protocol that users do not need to trust to the third party, since there is a single party (third party) that uses cryptographic proofs to ensure that the mixing was done correctly. It does have some constrains in performance and functionality. Regarding performance, one problem is that, those proofs that are required to ensure the correct operation of the mix

*This report is prepared as partial fulfillment of the requirements for the course *Research Seminar in Cryptography* (MTAT.07.022) in Spring 2018 at Institute of Computer Science, University of Tartu.

are about 45 KB per transaction, (128-bit security) and they take about half a second to verify. Moreover these mixes can only work on the values of the same denomination, which means that one cannot split values and just can spend multiple coins of the same denominations. Here there is a leak of information that for how much you actually using a mix for. Each user has to take his coins, load them to laundry service and wash them clean and get them back, which these procedure needs explicit operations.

In the paper that we review in this report, Eli Ben-Sasson et al. proposed a decentralized anonymous payment scheme based on Zero knowledge Succinct Non-interactive Arguments of Knowledge(zk-SNARKs) and instantiate and implement the scheme as a digital currency. In this scheme there is an opportunity to have payments via users directly and private without leaking the information about the origin of transaction, the transmitted amount of money and the destination of transactions, which gives undeniable privacy to users.

In this report we aim to present a short review on Bitcoin and its weakness, and talk shortly about Zerocoin and its improvement compared to Bitcoin. Then mostly talk about a new DAP scheme based on ZK-SNARKs and instantiation of the scheme in order to get a new anonymous coin, Zcash.

The structure of report is as follows: Section 2 presents a high level explanation of the structure of the zk-SNARK that Zerocash uses to give zero knowledge proof for the statement. Then report follows the structure of DAP scheme in Section 3. Indeed, report explains all the steps forward from initial version of DAP scheme to the final version of it in the same section. Instantiation DAP scheme with Zerocash group and create the practical the Zerocash coin presented in Section 4. Finally we conclude the report in Section 5.

2 zk-SNARKs

Before going through the DAP scheme we will give a high level explanation of zk-SNARKs, later in the following sections we will talk with more details about them. The zk-SNARKs give very short proofs which verifier easily can verify. Assume given language $\mathcal{L}$ which is $\mathcal{NP}$ and a circuit $\mathcal{C}$ that is a nondeterministic circuit for language $\mathcal{L}$ with n multiplication gates. In order to prove and verify membership in $\mathcal{L}$ using a zk-SNARK we act as follows, When we input $\mathcal{C}$, a trusted third party does a one-time setup phase and outputs two public keys, one for proving (pk) and the other one for verifying (vk). The pk gives the ability that any (untrusted) prover can give non-interactive proof π which is zero knowledge and a proof of knowledge of witness w for $x \in \mathcal{L}$, relationship for instantiation of x with size n. There is possibility for all the verifiers to verify the proof π using public vk. Back to Succinctness definition, size of π is constant for the given security level and in order to verify the verifier takes linear time in the size of x (rather than linear in $|\mathcal{C}|$).

3 Decentralized anonymous payment schemes

In this section we introduce the DAP scheme, which guarantees anonymity of digital currencies. The DAP scheme uses Succinct Non-interactive Arguments of Knowledge (zk-SNARKs) [BSCG+14,Lip12,BCI+13, ABLZ17] under some cryptographic assumptions. For all the ledger based digital currency such as Bitcoin we can use the DAP scheme. As we know all the transactions are stored in the ledger which is held by all the users. The ledger can be only appended and there is no way to delete anything from the ledger by users. Recalling the information about Bitcoin [Nak08] and Zerocoin [MGGR13], we will give a high level view of the DAP scheme by separating into the 6 steps which introduced in the following subsections.

3.1 User anonymity with fixed-value coins

We start explaining the DAP scheme with using the same value for all coins, for example 1 Bitcoin. In DAP scheme we use the property of zk-SNARKs and a commitment scheme with the property of hiding (the specific value chosen cannot be known by the receiver at that time) and binding (The value chosen during the commit phase must be the only one that the sender can compute and that validates during the reveal phase). To commit a message m we use a randomness r and the function of commitment, defines, $cm := COMM_r(m)$ and it is possible to open c, just by revealing r and m.

In order to mint a new coin c, user needs to take a random serial number sn and a trapdoor r (randomness of commitment). Then he should compute $cm := COMM_r(sn)$ as a commitment for the coin

which is defined $c := (r, sn, cm)$. For each coin which is minted, there exist a mint transaction tx_{Mint} including cm without sn, which needs to be append to the ledger. There is a commitment list on the ledger which contains all the coin commitments. In the case that user wants to spend the coin, he should send the tx_{Spend} that contains sn of the coin and a proof that she knows r for one of the commitments in the commitment list. There is another list in the ledger which contains all the coins' serial number which are spent, and in the case of having tx_{Spend}, if the sn is not in the list the transaction is okay otherwise it will be discarded. This checking avoids double spending of each coin. Since the proof is zero knowledge, so no one can find information about r and finding a commitment in the list needs to invert $f(x) := COMM_x(sn)$ which is infeasible under some assumptions.

3.2 Compressing the list of coin commitments

By increasing the numbers of commitments, the naive commitment list grows linearly, the time and the space complexity are linear which is not very good. The way to avoid this problem in DAP scheme, the CMList is Collision Resistant Hash based (CRH) Merkle tree, so the complexity will decrease to logarithmic. The CRH is used in this construction to avoid having the same special representation of CMList. The root of tree is rt and all the commitments are arranged on the leaves. The user who wants to prove that he knows the witness r for a statement (a commitment in tree) have to prove he knows witness r such that there exist a commitment $COMM_r(sn)$ as a leaf of a CRH-based Merkle tree with root rt. In this new structure of CMList, the naive of the commitment list will grow exponentially comparing to the naive one. Note that this size can increase as much as the size that the given zk-SNARK implementation can support (produce a proof for that) (In Zerocash the authors use the tree of depth 64 which supports 2^{64} coins).

3.3 Extending coins for direct anonymous payments

In the previous subsection 3.2 we commit the serial number of a coin, it will create some problems later. Assume that user u_1 creates a coin c and sends the coin to user u_2. Spending this coin for u_2 is risky for the following issues:

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

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

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

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

Figure 1: Setup and create addresses [BSCG+14]

- the u_1 knows the serial number of coin and he can still spend the coin one more time and send it to another user (double spending).
- The other problem with spending coin with u_2 is that since u_1 knows serial number so, he can trace the coin and recognize when the coin is spent by u_2.

So, u_2 in order to protect his privacy, must spend the coin and try to mint new coin c'.

The other harmful issue with the current structurer of minting is that this structure does not support amounts which are not multiples of the fixed value of a coin for example the multiples of $1BTC$. So, when u_1 wants to send 100 Bitcoin (BTC) to u_2, he needs to have 100 transfers which is too much and even worse it leaks the information about the amount of the transactions. So, this simplified structure that we used in DAP payment scheme is not enough. Then we need to modify the commitment of coin by using collision-resistant pseudorandom functions such as $PRF^{addr}_x(.) := H(x||00||.)$, $PRF^{sn}_x(.) := H(x||01||.)$, and $PRF^{pk}_x(.) := H(x||10||.)$ with seed x to target payments and to derive serial numbers (See Fig.1). In Fig. 1, $\mathsf{KeyGen}(.)$, $\mathsf{g}_{\mathsf{sig}}(.)$, $\mathsf{K}_{\mathsf{enc}(.)}$ respectively show the key generation for pour transaction, key generation for one-time signature, and key generation for an encryption scheme.

To provide targets for payments, Zcash uses addresses, in the other words, each user needs to gen-

Mint

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

1) Parse $\mathsf{addr}_{\mathsf{pk}}$ as $(a_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}})$.
2) Randomly sample a $\mathsf{PRF}^{\mathsf{sn}}$ 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 $\mathsf{tx}_{\mathsf{Mint}} := (\mathsf{cm}, v, *)$, where $* := (k, s)$.
8) Output $\mathbf{c}$ and $\mathsf{tx}_{\mathsf{Mint}}$.

Figure 2: Mint a coin [BSCG+14]

erate key pair (a_{pk}, a_{sk}). The a_{pk} includes the coin c and the only user that can spend it, is the user who knows the a_{sk}. In order to generate key pair, one need to sample a random a_{sk} and define a_{pk} as, $apk := PRF^{addr}_{ask}(0)$. There is no limitation to number of address key pairs for users. In order to have better system functionality, Zcash designed a new minting. For minting a coin (see fig.2) c with value v, user needs to sample which is a randomness of serial number, and use that for determining the serial number for the coin as follows, $sn := PRF^{sn}_{a_{sk}}(\rho)$. After that the user commits to a tuple (a_{pk}, v, ρ) by computing $k := COMM_r(a_{pk} \parallel \rho)$ for a random r and $cm = COMM_s(v \parallel k)$ for a random s. So the new version of coin is $c := (a_{pk}, v, \rho, r, s, cm)$ and a mint transaction $tx_{Mint} := (v, k, s, cm)$. The way that Zcash construct the new coin, everyone by checking $cm := COMM_s(v \parallel k)$ can verify that cm in tx_{Mint} is a commitment of a coin of value v. Due to use of zk-SNARKs in construction of Zcash, no one can compromised the users privacy by learning the address key a_{pk}) or serial number by deriving from because these are hidden in k. Like as before, ledger only accepts a tx_{Mint} if user deposits the correct amount of v.

The operation for spending coin is named pour and it means that the value of some input coins will pour in the fresh output coins (see fig.3). Assume that the user with address key pair (a^{old}_{pk}, a^{old}_{sk}), wants to pour his coin $c^{old} = (a^{old}_{pk}, v^{old}, \rho^{old}, r^{old}, s^{old}, cm^{old})$ to the two new coins c^{new}_1 and c^{new}_2, with total value ${v_1}^{new} + {v_2}^{new} = v^{old}$, respectively targeted at address public keys $a^{new}_{pk,1}$ and $a^{new}_{pk,2}$. It is possible that the user pours the coins to the addresses which belong to himself or to some other users. The user u, for each $i \in \{1, 2\}$, proceeds as follows: (1) u samples serial number randomness ρ^{new}_i ; (2) u computes $k^{new}_i := COMM_{{r_i}^{new}}(a^{new}_{pk,i} \parallel \rho^{new}_i)$ for a random ${r_i}^{new}$; and (3) u computes $cm^{new}_i := COMM_{{s^{new}_i}}({v_i}^{new} \parallel {k_i}^{new})$ for a random s^{new}_i.

Now, the new coins $c^{new}_1 := (a^{new}_{pk,1}, v^{new}_1, \rho^{new}_1, r^{new}_1, s^{new}_1, cm^{new}_1)$ and $c^{new}_2 := (a^{new}_{pk,2}, v^{new}_2, \rho^{new}_2, r^{new}_2, s^{new}_2, cm^{new}_2)$ were produced. In the next really important step, user u needs to give a zk-SNARK proof π_{POUR} for the following NP statement, that Zcash named the statement, POUR:

Given the Merkle-tree root rt, serial number sn^{old}, and coin commitments cm^{new}_1, cm^{new}_2 , I know coins $c^{old}, c^{new}_1, c^{new}_2$, and address secret key a^{old}_{sk} such that:

- The coins are well-formed: for c^{old} it holds that $k^{old} = COMM_{r^{old}}(a^{old}_{pk} \parallel \rho^{old})$ and $cmold = COMM_{s^{old}}(v^{old} \parallel k^{old})$; and similarly for c^{new}_1 and c^{new}_2.
- The address secret key matches the public key : $a^{old}_{pk} = PRF^{addr}_{a^{old}_{sk}}(0)$.
- The serial number is computed correctly: $sn^{old} := PRF^{sn}_{a^{old}_{sk}}(\rho^{old})$.
- The coin commitment cm^{old} appears as a leaf of a Merkle-tree with root rt.
- The values add up: $v^{new}_1 + v^{new}_2 = v^{old}$.

Then pour transaction $tx_{Pour} := (rt, sn^{old}, cm^{new}_1, cm^{new}_2, _{POUR})$ is appended to the ledger (see fig.3). Moreover, the transaction with the serial number which appears in a previous transaction will be rejected, because it means that the coin with that serial number has been spent before, in other words, this checking avoids double spending. In the case that a user does not know the corresponding secret key $a^{new}_{sk,1}$ for public key $a^{new}_{pk,1}$, then he cannot spend c^{new}_1, because he cannot provide $a^{new}_{sk,1}$ as a part of the witness of a corresponding pour operation to produce proof π. Furthermore, when a user that knows $a^{new}_{sk,1}$ does spend c^{new}_1, the user u cannot track it, because he knows no information about its revealed serial number, which is $sn^{new}_1 := PRF^{sn}_{a^{new}_{sk,1}}(\rho^{new}_1)$.

The transaction pour tx_{pour} is zero-knowledge which means that it does not reveals anything about

the destination address keys, and the amount of coins which is transmitted. So this structure of payment is completely anonymous.

Generally, a user u may pour $N^{old} \geqslant 0$ coins into $N^{new} \geqslant 0$ coins. For simplicity we consider the case $N^{old} = N^{new} = 2$, without loss of generality. Indeed, for $N^{old} < 2$, the user can mint a coin with value 0 and then provide it as a null input, and for $N^{new} < 2$, the user can create (and discard) a new coin with value 0. For $N^{old} > 2$ or $N^{new} > 2$, the user can compose $logN^{old}+logN^{new}$ of the $2-input/2-output$ pours.

3.4 Sending and receiving coins

In this step we consider the way the first user u has to send the secret values in c_1^{new} to new user u_1. This procedure provides the ability for user u_1 who is the owner $a_{pk,1}^{new}$ to spend c_1^{new} if he wishes.

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

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

Figure 3: Pour algorithm [BSCG+14]

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

Figure 4: Verification algorithm [BSCG+14]

One way is that u can send a private message to u_1 containing the secret values but this requires a secure communication which needs other infrastructure and assumptions. The second way that Zcash proposes is by modifying the ledger by modifying the structure of the key pair. Modification is that 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 [BBDP01]. Then, u computes the ciphertext C_1 that is the encryption of the plaintext $(v_1^{new}, \rho_1^{new}, r_1^{new}, s_1^{new})$, under $pk_{enc,1}^{new}$ (which is part of u_1s address public key $addr_{enc,1}^{new}$), and includes C_1 in the pour transaction tx_{Pour}. The user u_1 can then find and decrypt this message (using his $sk_{enc,1}^{new}$) by scanning the pour transactions on the public ledger. Adding C_1 to tx_{Pour} does not leak any information about paid amounts and destination addresses, because of using the encryption scheme which has key-private property. (The user u does the same with c_1^{new} and includes a corresponding ciphertext C_2 in tx_{Pour}.)

In order to check that the mint and pour transactions are well-formed, there is need to run the verification algorithm. All the nodes that maintain the ledger can verify transactions. This algorithm takes some inputs including public parameters pp, a (mint or pour) transaction tx, the current ledger L and outputs bit b (bit b equals 1 iff the transaction is valid)(See fig.4).

After that there is need to run another algorithm named Receive Algorithm. This algorithm scans the ledger and retrieves unspent coins paid to a particu-

Receive

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

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

Figure 5: Receive Algorithm [BSCG+14]

lar user address by taking recipient address key pair $(addr_{pk}, addr_{sk})$ and the current ledger L as inputs and outputting set of (unspent and their serial numbers do not appear in the ledger) received coins when a user with the target addresses wishes to receive payments sent to their public address(See fig.5).

3.5 Public outputs

Up to here, Zcash constructs mint, merge, and split operations for users. Then Zcash modified the pour operation to add a public output in the case that as an example a user wants to convert it back to the base currency (Bitcoin). Zcash updated the balance equation in the NP statement POUR to $(v_1^{new} + v_2^{new} + v_{pub} = v_{old})$ and specified a $v_{pub} \geqslant 0$ for redeemed funds and added v_{pub} and an arbitrary string $info$ (which specifies the destination of these redeemed funds) to the pour transaction tx_{Pour}. If the user u set $v_{pub} = 0$ the public output will be optimal.

3.6 Non-malleability

In this section we will consider how to secure the protocol against the malleability attacks. As one can see that there is possibility for the attacker to attack a pour transaction tx_{Pour} by re-targeting the public output of the pour with modifying info but Zcash tackled this problem very cleverly. Zcash uses digital signatures in the new version of the NP statement POUR(See fig.3).

More precisely, during the pour operation, the user u (1) samples a key pair (pk_{sig}, sk_{sig}) for a one-time signature scheme; (2) computes $h_{Sig} := CRH(pk_{sig})$; (3) computes the two values $h_1 := PRF^{pk}_{a^{old}_{sk,1}}(h_{Sig})$ and $h_2 := PRF^{pk}_{a^{old}_{sk,2}}(h_{Sig})$, which act as MACs to tie h_{Sig} to both address secret keys; (4) modifies POUR to include the three values h_{Sig}, h_1, h_2 and prove that the latter two are computed correctly; and (5) uses sk_{sig} to sign every value associated with the pour operation, thus obtaining a signature σ, which is included, along with pk_{Sig}, in tx_{Pour}.

Since the $a^{old}_{sk,i}$ are secret, and with high probability h_{Sig} changes for each pour transaction, the values h_1, h_2 are unpredictable. Moreover, the signature on the NP statement (and other values) binds all of these together.

The construction that we discussed from different view and tried to solve the problems, summarized in part in (fig.6). Zcash finishes up by taking note of that, due to the zk-SNARK, Zcash development requires a one-time trusted setup phase to generate some public parameters. The trust influences soundness of the proofs, however in the case of corrupting the setup with a malicious party, anonymity will be kept.

4 Zerocash

Zcash instantiates the DAP scheme and provides an implementation of Zcash coin with 128 bits of security. The zk-SNARK component demonstrates all the constructions of DAP scheme, so Zcash intelligently implements the protocol with the instantiation which guarantees the efficiency of protocol in practice. To prove and verify a specific NP statement (pour), Zcash uses the zk-SNARK. While zk-SNARKs are asymptotically efficient, their concrete efficiency depends on the arithmetic circuit C that is used to decide the NP statement.

Zcash tries to find the best instantiations by designing a relatively-small arithmetic circuit C_{POUR} for verifying the NP proof π. Zerocash uses SHA256 to instantiate pseudorandom functions, commitment schemes, and collision-resistant hashing based functions. Zcash for first instantiation of circuit used a circuit that they designed and optimized by hand due to target for verifying SHA256 computations. After that due to verify the NP statement C_{POUR} Zerocash uses a circuit which can check all the checking in the statement.

Using a suitable zk-SNARK implementation for

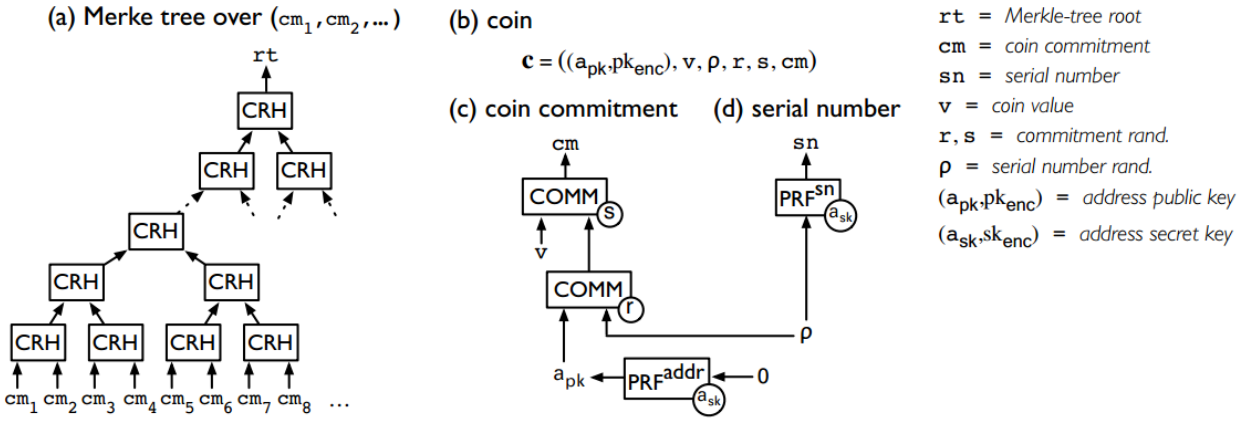

Figure 6: (a) Illustration of the CRH-based Merkle tree over the list CMList of coin commitments. (b) A coin c. (c) Illustration of the structure of a coin commitment cm. (d) Illustration of the structure of a coin serial number sn [BSCG+14].

arithmetic circuits [BSCTV13], beside the cleaver instantiation of parameters in Zcash results in prover that takes few minutes to prove and verifier takes few milliseconds to verify the proof. By these achievements for DAP it is possible to use the scheme for practical deployment. There is possibility to use Zerocash for Bitcoin or forks of it (commonly referred to as altcoins).

5 Conclusion

Beside very nice features that blockchain technology and its by-products such as digital currencies provide, guaranteeing strong privacy and security of end-users always has been one of main priorities for society and researchers of the area.

In this report we reviewed Zerocash [BSCG+14], which is one of the most known and popular privacy-preserving cryptocurrency that is proposed to guarantee users' privacy. In other words, Zerocash does not reveal any information about user identities, source or destination addresses, and the account of transferred or spent coins. We had an overview on the construction of the coin, and we saw that the designer of coin uses different cryptographic primitives (including PRFs, commitment schemes, hash functions, one-time signature schemes, public-key encryption schemes, and zk-SNARKs) to provide strong privacy and security for end-users. However, we observe that the main tool behind indistinguishability properties of the coin is a zk-SNARK that allows a user (a prover) to give very short (succinct) proofs in different transactions (operations) and more importantly these proofs can be verified by a low-power verifier very efficiently (in less than a second).

Acknowledgment: Thanks to Karim Baghery for his helpful discussions on reading the seminal paper.

References

[ABLZ17] Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. A subversion-resistant SNARK. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–33. Springer, 2017.

[BBDP01] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 566–582. Springer, 2001.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography*, pages 315–333. Springer, 2013.

[BSCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Practical decentralized anonymous e-cash from bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy. IEEE*, 2014.

[BSCTV13] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive arguments for a von neumann architecture. *IACR Cryptology ePrint Archive*, 2013:879, 2013.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *Theory of Cryptography Conference*, pages 169–189. Springer, 2012.

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.