Z-Channel: Scalable and Efficient Scheme in Zerocash

Yuncong Zhang* shjdzhangyuncong@sjtu.edu.cn
Yu Long* longyu@sjtu.edu.cn
Zhen Liu* liuzhen@sjtu.edu.cn
Zhiqiang Liu* liu-zq@cs.sjtu.edu.cn
Dawu Gu* dwgu@sjtu.edu.cn

*Shanghai Jiao Tong University

***Abstract*—Decentralized ledger-based cryptocurrencies such as Bitcoin provide a means to construct payment systems without requiring a trusted bank, yet the anonymity of Bitcoin is proved to be far from satisfactory. Zerocash is the first full-fledged anonymous digital currency based on the blockchain technology, using zk-SNARK as the zero-knowledge module for the privacy protection. Zerocash solves the privacy problem but still suffers two major problems: insufficient scalability and latency in making a payment. Meanwhile, micropayment channel proves to be a nice solution to these issues in blockchain-based digital currencies. In this paper, we present Z-Channel, the construction of micropayment system on Zerocash, which effectively solves the scalability and instant payment problems in Zerocash. Z-Channel relies on multisignature and lock time functionalities which are not provided by Zerocash. We manage to improve the Zerocash scheme to support these functionalities without compromising the privacy guaranteed by Zerocash. Finally, the simulation results demonstrate that Z-Channel significantly improves the scalability and reduces the average confirmation time for the payments conducted in Zerocash.**

Keywords*—*Cryptocurrency, Zerocash, Scalability, Privacy, Instant payment

I. Introduction

Decentralized ledger-based cryptocurrencies such as Bitcoin [21] provide a means to construct payment systems without requiring a trusted bank. Following Bitcoin, many digital currencies have been devised trying to improve Bitcoin with respect to its functionalities [7], [15], [8], [17], consensus schemes [26], [15], scalability and efficiency [28], [7], and privacy [27], [16], etc.

Privacy protection is one of the features of ledger-based digital currency that attract the most attention [4]. Bitcoin has been thoroughly analyzed and its privacy protection is proved to be easily compromised [23]. By analyzing the transaction graph, values and dates in the ledger one can possibly link Bitcoin addresses with real world identity. To break such linkability in Bitcoin, one can store his Bitcoin into a *mix*, which is a trusted central party which mixes Bitcoins from different users and gives different coins back to them after sufficient amount of coins are mixed together. However, the delay in redeeming the coins and the trust to a central party is unacceptable to some users with strong motivation to hide information. A remedy is to implement a decentralized mix. To accomplish this, protocols have been designed such as TumbleBit [12], CoinSwap [19], CoinParty [29] and CoinShuffle[24] which is based on the work of CoinJoin [18]. Additionally, many altcoins have been developed, including Zerocoin [20], BlindCoin [27] and its predecessor Mixcoin [5] and Pinocchio coin [6], etc. These solutions, however, suffer from the following drawbacks: 1) Insufficient performance. Most of them require one or more rounds of interaction between many parties. 2) Lack of functionality. They simply present a way for users to “wash” their coins from time to time, but everyday transactions are still conducted without privacy.

Compared to the mix-based solutions, Zerocash [25] is the first full-fledged privacy preserving ledger-based digital currency, which completely conceals the user identity and amount of payment in each and every transaction. The construction of Zerocash uses zero-knowledge proof, specifically zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs) [2], [9].

Despite all the advantages in privacy protection, the design of Zerocash does not address the scalability and efficiency problems which exist in almost all the ledger-based digital currencies. In fact, the transaction size of Zerocash is larger than that of Bitcoin, and the time to verify zk-SNARK proof is longer than verifying a Bitcoin transaction, which makes the scalability problem in Zerocash even worse than in Bitcoin.

For other ledger-based digital currencies, there have been works trying to solve the scalability and efficiency issues. Changing the blocksize [1] is a straightforward way to improve the scalability, though it compromises the efficiency with higher network latency and longer verification time. The block merging technique proposed in MimbleWimble [14] requires a special structure for the blocks and transactions, sacrificing a majority of the functionalities of the digital currency. Micropayment channel [22] proves to be the most promising in solving both of the scalability and efficiency problems effectively. By transactions conducted securely off-chain, micropayment channel is likely to enable Bitcoin or similar altcoins to support billions of users. Despite the dramatic improvement in scalability and payment speed micropayment channels is promising, no work has been proposed to construct a micropayment system on Zerocash [1].

[1]The work of BOLT (Blind Off-chain Lightweight Transactions) [11] mentions Zerocash, claiming that if a BOLT is built on Zerocash, it would provide better channel privacy than built on other currencies. However, BOLT focuses on solving the linkability issue in channels, while the concrete construction of BOLT over Zerocash is not specified in their work.

A. *Our contribution*

In this work we address the above problems by the following contributions:

We develop a micropayment scheme over Zerocash, which we call **Z-Channel**. Compared with Zerocash, Z-Channel significantly enhances the scalability, allowing a great number of users to perform high-frequency transactions off-chain in day-to-day routine, and the payment is made nearly instantly. Meanwhile, the Z-Channels are established and terminated with strong privacy guarantee.

To implement Z-Channel on Zerocash, we improve the Distributed Anonymous Payment (DAP) scheme for Zerocash and propose a new scheme called **DAP Plus**. DAP Plus provides the multisignature and lock time features lacked in Zerocash, which are needed by Z-Channel and many other designs of micropayment channel. We prove that DAP Plus scheme is privacy preserving, in the sense that a transaction does not leak any information about its issuers, its input or output. We give the formal definition of the security of DAP Plus scheme based on the original DAP scheme. We prove that DAP Plus scheme is secure under this definition. In addition, we find a weakness in the security model proposed in [25] for the original DAP scheme, which we fix in our scheme.

Moreover, we implement the zk-SNARK circuit for the NP statement in DAP Plus scheme based on the code of ZCash, and benchmark the time consumption of conducting zero-knowledge proof. Our experiment also proves the efficiency of our construction of Z-Channel. Finally, the simulation results demonstrate that Z-Channel significantly improves the scalability and reduces the average confirmation time for the payments conducted in Zerocash.

B. *Paper organization*

The remainder of the paper is organized as follows. Section II introduces the preliminaries needed for our work. Section III presents DAP Plus scheme which improves the Zerocash scheme by embedding the multisignature and lock time mechanisms. In Section IV, we present the construction of Z-Channel based on our newly proposed scheme. Section V gives the security analysis for the DAP Plus scheme and the Z-Channel protocols. Section VI analyzes the performance of Z-Channel. Finally, Section VII concludes this paper.

II. Preliminaries

A. *Background on zk-SNARKs*

Zerocash relies on **non-interactive zero-knowledge proof** to avoid identifying users by their signature verification public keys. The zero-knowledge proving scheme adopted by Zerocash is zk-SNARK [9]. Suppose Alice has an NP problem instance x and its witness w. She wants to prove to Bob that x is a valid instance, without revealing w to Bob. Using zk-SNARK, she inputs x and w to generate a proof π, and sends π instead of w to Bob. Bob then inputs x and π to zk-SNARK and is convinced that π is a valid proof of x.

Let C denote a circuit verifying an NP language $\mathcal{L}_C$ which takes as input an instance x and witness w, and outputs b indicating if w is a valid witness for x.

A zk-SNARK (Succinct Non-interactive ARguments of Knowledge) is a triple of algorithms $(\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$.

The algorithm KeyGen takes C as input and outputs a *proving key* pk and a *verification key* vk.

The algorithm Prove takes as input an instance of the NP problem x and a witness w, as well as pk, and generates a non-interactive proof π for the statement $x \in \mathcal{L}_C$.

The algorithm Verify takes as input the instance x and the proof π, as well as vk, and outputs b indicating if he is convinced that $x \in \mathcal{L}_C$.

A zk-SNARK is *correct* if the honest prover can convince the verifier. It has the quality of *proof-of-knowledge* if the verifier accepting a proof implies the prover knowing the witness. It has the quality of *perfect zero-knowledge* if there exists a simulator which can always generate the same results for any instance $x \in \mathcal{L}_C$ without knowing witness w.

The work of Zerocash is based on the zk-SNARK implementation proposed in [3].

B. *Zerocash DAP Scheme*

The *decentralized anonymous payment scheme* (DAP scheme) is a full-fledged anonymous mechanism based on ledger based currency system. The DAP scheme can be built on the top of any ledger-based digital currency, which we refer to as *basecoin*. The DAP scheme modifies basecoin by introducing a new type of address called *shielded address* and two types of transactions *mint transaction* and *pour transaction*.

A DAP scheme is a tuple of six algorithms $(\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{Verify}, \mathsf{Receive})$.

The algorithm Setup takes as input a security parameter λ and outputs public parameters pp.

The algorithm $\mathsf{CreateAddress}$ outputs a newly generated shielded address/key pair $(\mathsf{addr_{pk}}, \mathsf{addr_{sk}})$.

The algorithm Mint takes as input a value v and the destination address $\mathsf{addr_{pk}}$, and outputs a coin $\mathbf{c}$ and a mint transaction $\mathsf{tx_{Mint}}$. A mint transaction is much alike a usual transaction, replacing the *pay to public key hash* with *pay to commitment*, i.e. taking a commitment cm to the coin as the output. The transaction is deemed valid if it is valid as a basecoin transaction while the commitment is correctly computed.

The algorithm Pour takes as input two input coins, secrets for the input coins, two destination addresses and other information, and outputs two new coins and a pour transaction $\mathsf{tx_{Pour}}$. A zero-knowledge proof π_{POUR} is appended to $\mathsf{tx_{Pour}}$ to prove the validity of this transaction, i.e. the validity of the input coins and the balance of this transaction, etc. The transaction reveals the unique serial numbers of the input coins to prevent double spending. To prove the existence of the input coins on the ledger, all the commitments on the ledger are maintained in a Merkle-tree, and Pour algorithm additionally takes as input a Merkle root rt in the Merkle-tree history, and the paths from the commitments to rt.

The algorithm Verify takes as input the public parameters pp and a transaction $\mathsf{tx_{Mint}}$ or $\mathsf{tx_{Pour}}$ as well as a ledger, and outputs a bit b indicating if this transaction is valid to be appended on the ledger.

The algorithm Receive takes as input a shielded address and its key $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ as well as a ledger, and outputs all unspent coins paid to the given address on the ledger.

Apart from the KeyGen and Pour algorithms, the other algorithms only take milliseconds to execute, while KeyGen usually takes around five minutes, and Pour takes one or two minutes.

C. Micropayment Channel

Micropayment channel [22] allows two parties to make payments to each other without publishing transactions on the ledger, tremendously reducing the time for confirming a payment. A micropayment channel scheme basically consists of three protocols: establish channel, update channel, and close channel.

The establish and closing of a channel involves interaction with the ledger. They are comparably slow but conducted only once in the lifetime of a channel. Meanwhile, the update procedure is executed each time a payment is made, and it can be executed with high frequency.

Usually, to establish a channel each of the involving parties have to freeze some of their currency in the channel. When the channel is shutdown, the frozen coins are distributed back to the parties. The update of a channel modifies the state of the channel, which usually is the way to distribute the frozen currency. The parties make payment to each other by renegotiating the distribution. The update protocol is executed off-chain, thus can be conducted frequently and with low latency.

D. Distributed Signature Generation Scheme

The naive implementation of multisignature scheme in Bitcoin, i.e. counting the number of signatures, reveals some information which, though not much of an issue in Bitcoin, completely compromising the privacy if adopted in Zerocash. We implement the multisignature feature in an alternative way, namely the **distributed signature generation scheme** [10]. Specifically, we require the scheme to support the following operations:

1) **Distributed key generation**. Multiple parties cooperate to generate a pair of public/private keys pk and sk. After the protocol is done, pk is known by all the parties, while sk is invisible to every one. Each party holds a share sk_i of the private key.
2) **Distributed signature generation**. Given a message M, the parties holding the pieces sk_i of the private key cooperate to generate a signature σ on M. Specifically, each party generates a share σ_i of the signature alone and broadcasts it to other parties. Anyone obtaining all the shares of the signature can recover the complete signature σ. This signature can be verified by pk and is indistinguishable from the signatures directly signed by sk.

III. DAP Plus: Improved Decentralized Anonymous Payment Scheme

Our construction of Z-Channel relies on two functionalities: multisignature and lock time. However, they are not provided by the original Zerocash scheme, i.e. DAP scheme. To solve this issue, we present DAP Plus, which is an improvement to the DAP scheme, with support to multisignature and lock time features.

A. Main Idea of DAP Plus Scheme

In this subsection, we present the improvements of DAP+ compared to the original DAP scheme. For convenience, we assume that the involved parties are Alice and Bob, and Alice is trying to send a coin to Bob.

Commit a public key lock in the coin. In Zerocash, a coin consists of a commitment cm and some secret information necessary for spending this coin. The commitment involves the following information: the destination address a_{pk} (which is part of the shielded address $\mathsf{addr}_{\mathsf{pk}}$, called receiving address) owned by Bob, the value v and a random string ρ which is used by Bob to compute sequence number sn which is the unique identifier of the coin. To spend the coin, Bob has to reveal the sequence number sn, so that he cannot spend the coin again. Other information are kept secret, and Bob generates a zero-knowledge proof π_{POUR} to prove that the revealed sn is valid.

We modify this by requiring Alice to commit a *public key lock* pkL into the coin commitment cm. pkL is a properly encoded public key, which is *required* to be generated by a *distributed signature generation scheme*. Since zk-SNARK only supports fixed length input [3], in order to allow pkL to be of arbitrary length, we commit the hash of pkL, denoted by pkH instead of the original public key. To validate the transaction, for each input coin, Alice appends to the transaction a signature σ which is verified by pkL. We denote the part of the transaction that is protected by this signature by a function $\mathsf{ToBeSigned}()$, and leave it to be determined by the application that builds on top of DAP+ scheme.

To allow other parties to verify the signature, pkL should be disclosed as the coin is spent. Note that the anonymity of Bob against Alice is thus compromised, since Alice would immediately perceive when Bob spends the coin, by identifying the public key lock published in the transaction. To solve this problem, we require that Alice does not know pkL nor its hash pkH, but a commitment pkcm which is generated by Bob with trapdoor a_{sk} (which is the part of $\mathsf{addr}_{\mathsf{sk}}$ corresponding to a_{pk}, called spending key) with input pkH. Therefore, before Alice sends a coin to Bob, Bob needs to generate a fresh pkcm randomly and sends to Alice together with Bob's shielded address $\mathsf{addr}_{\mathsf{pk}}$. When Bob spends the coin, he proves that pkH is committed into the coin with his knowledge of a_{sk}. Alice cannot perceive that the revealed pkL is related to the pkcm she put into the coin previously.

Commit a lock time lock in coin. Next, we commit a lock time tL into the coin. To avoid the clock synchronizing issue, we use the block height as the clock. For simplicity, we denote the height of the block containing a coin commitment cm by $\mathsf{BH}(\mathsf{cm})$. We then require that Alice appends a *minimum block height* MBH in the pour transaction. A transaction is considered invalid if its MBH is larger than the height of the block containing it, thus cannot get on the ledger until the block height reaches MBH. For each input coin, Alice should prove that $\mathsf{BH}(\mathsf{cm}) + \mathsf{tL} < \mathsf{MBH}$ in zero-knowledge.

There is, however, a tricky issue about $\mathsf{BH}(\mathsf{cm})$, since it is somehow independent from cm, i.e. there is no computational relationship between them. Therefore, it is hard to prove in zero-knowledge that Alice has input the correct $\mathsf{BH}(\mathsf{cm})$ as a secret input to the zk-SNARK prover. In the meantime, $\mathsf{BH}(\mathsf{cm})$ cannot be disclosed, as this would definitely compromise the privacy of Alice.

We solve this issue by noting that Alice does not have to prove that $\mathsf{BH}(\mathsf{cm}) + \mathsf{tL} < \mathsf{MBH}$, but $\mathsf{BH}(?) + \mathsf{tL} < \mathsf{MBH}$ where $\mathsf{BH}(?)$ is the block height of something that is guaranteed to be later than cm on the ledger, and is safe to be disclosed. The best candidate for this is the Merkle-tree root rt, which is used to prove the existence of the input coin commitment. Each time when a new coin commitment is appended on a ledger, the root is updated to a new one, thus there is a one-to-one correspondence between the list of commitments and the history of roots. We then naturally define the block height of a Merkle-root rt as that of the corresponding commitment and denote it by $\mathsf{BH}(\mathsf{rt})$.

Logical relationship between public key lock and lock time. If a coin commits a public key lock pkL and lock time tL, we say the coin is *locked* by pkL with tL blocks. If tL is set to the maximum lock time MLT, then we say the coin is locked by pkL forever. We denote a pair of public key commitment and lock time by $\mathsf{lock} = (\mathsf{pkcm}, \mathsf{tL})$, and a pair of public key lock and signature by $\mathsf{unlock} = (\mathsf{pkL}, \sigma)$. We say unlock *unlocks* a lock if pkL is a correct opening of pkcm and the contained signature is valid.

We decide to take the "OR" relationship between the public key lock and the lock time. That is to say, the transaction is valid either when the lock time expires or a valid unlock is provided. To say it in another way, a coin is locked by tL blocks unless overridden by the signature.

We accomplish this by adding a *overriding* boolean flag ovd as a public input to zk-SNARK, which is true if and only if a valid unlock is appended in the transaction. Then, Alice only has to prove in zero-knowledge that $\mathsf{ovd}||(\mathsf{BH}(\mathsf{cm})+\mathsf{tL} < \mathsf{MBH})$ is true, where $||$ means logical OR.

Note that this logic can be easily modified, without modifying the zk-SNARK part of the scheme. For example, by always setting ovd to false and requiring a valid unlock, the logic between the locks then becomes "AND". Similarly, always setting ovd to true totally neglects the lock time. We will use a slightly modified version of logic in Z-Channel, but for simplicity, we only describe constructing with basic logic in this section.

B. Construction of DAP Plus Scheme

Apart from the improvements mentioned in the previous subsection, the definition and construction of the algorithms in the DAP+ scheme are similar to the original DAP scheme in [25]. We present the full definitions here for completeness (the modification is shown in bold font).

A *DAP Plus scheme*, or DAP+ scheme, is a tuple of polynomial-time algorithms (Setup, $\mathsf{CreateAddress}$, $\mathsf{CreatePKCM}$, $\mathsf{MintPlus}$, $\mathsf{PourPlus}$, $\mathsf{VerifyPlus}$, $\mathsf{ReceivePlus}$).

We first present the cryptographic building blocks.

- Keyed pseudorandom functions $\mathsf{PRF}^{\mathsf{addr}}$ for generating addresses, $\mathsf{PRF}^{\mathsf{sn}}$ for serial numbers and $\mathsf{PRF}^{\mathsf{pk}}$ for binding public keys with addresses.
- Information hiding trapdoor commitment COMM.
- Fixed-input-length collision resistant hash function CRH and flexible-input-length hash function Hash.
- Zero-knowledge module zk-SNARK (KeyGen, Prove, Verify), where KeyGen generates a pair of proving key $\mathsf{pk}_{\mathsf{POUR}}$ and verification key $\mathsf{vk}_{\mathsf{POUR}}$, Prove generates a zero-knowledge proof π_{POUR} for an NP statement and Verify checks if a zero-knowledge proof is correct.
- Public signature scheme $(\mathcal{G}_{\mathsf{sig}}, \mathcal{K}_{\mathsf{sig}}, \mathcal{S}_{\mathsf{sig}}, \mathcal{V}_{\mathsf{sig}})$, where $\mathcal{G}_{\mathsf{sig}}$ is for generating global public parameter $\mathsf{pp}_{\mathsf{sig}}$, $\mathcal{K}_{\mathsf{sig}}$ is the key generation algorithm, $\mathcal{S}_{\mathsf{sig}}$ is the signing algorithm and $\mathcal{V}_{\mathsf{sig}}$ is the verification algorithm.
- Distributed public signature scheme $(\mathcal{G}_{\mathsf{dst}}, \mathcal{K}_{\mathsf{dst}}, \mathcal{S}_{\mathsf{dst}}, \mathcal{V}_{\mathsf{dst}})$ is defined similar to above, but the algorithms can be executed distributedly by more than one parties.
- Public encryption scheme $(\mathcal{G}_{\mathsf{enc}}, \mathcal{K}_{\mathsf{enc}}, \mathcal{E}_{\mathsf{enc}}, \mathcal{D}_{\mathsf{enc}})$, where $\mathcal{G}_{\mathsf{enc}}$ is for public parameter generation, $\mathcal{K}_{\mathsf{enc}}$ is the key generation algorithm, $\mathcal{E}_{\mathsf{enc}}$ is the encryption algorithm and $\mathcal{D}_{\mathsf{enc}}$ is the decryption algorithm.

We then present the detailed description of the algorithms. For simplicity, we use subscript $1..2$ to represent a pair each with subscript 1 and 2. For example, $\mathbf{c}_{1..2}^{\mathsf{old}}$ represents $\mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}$.

System setup. The algorithm Setup generates a set of public parameters. It is executed by a trusted party only once at the startup of the ledger, and made public to all parties. Afterwards, no trusted party is needed.

- *Input:* security parameter λ
- *Output:* public parameters pp

To generate the public parameters, first invoke KeyGen algorithm to generate $(\mathsf{pk}_{\mathsf{POUR}}, \mathsf{vk}_{\mathsf{POUR}})$, then invoke algorithms $\mathcal{G}_{\mathsf{sig}}$, $\mathcal{G}_{\mathsf{enc}}$ and $\mathcal{G}_{\mathsf{dst}}$ to obtain the public parameters for the public signature schemes and the public encryption scheme.

Create address. The algorithm $\mathsf{CreateAddress}$ generates a new pair of shielded address/key pair. Each user may execute $\mathsf{CreateAddress}$ algorithm arbitrary number of times. The shielded address $\mathsf{addr}_{\mathsf{pk}}$ is used by other parties to send him coins.

- *Input:* public parameters pp
- *Output:* shielded address/key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$

To generate the key pair, first sample a random string a_{sk} and compute $a_{\mathsf{pk}} = \mathsf{PRF}_{a_{\mathsf{sk}}}^{\mathsf{addr}}(0)$. Then, invoke $\mathcal{K}_{\mathsf{enc}}$ algorithm to generate a pair of public/private key pairs $(\mathsf{pk}_{\mathsf{enc}}, \mathsf{sk}_{\mathsf{enc}})$. Finally, output $\mathsf{addr}_{\mathsf{pk}} = (a_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}})$ and $\mathsf{addr}_{\mathsf{sk}} = (a_{\mathsf{sk}}, \mathsf{sk}_{\mathsf{enc}})$.

Create public key commitment. The algorithm $\mathsf{CreatePKCM}$ generates a commitment for a public key lock pkL. For complete anonymity, each time Alice tries to generate a coin (with $\mathsf{MintPlus}$ or $\mathsf{PourPlus}$ algorithm introduced later) for Bob, Bob invokes $\mathsf{CreatePKCM}$ algorithm to generate a fresh public key commitment pkcm and sends the pkcm to Alice.

- *Input:*
  - public parameters pp
  - address key $\mathsf{addr_{sk}}$
- *Output:*
  - a pair of public/private keys
  - tuple $(\mathsf{pkL}, \mathsf{pkcm})$

To generate pkcm, invoke $\mathcal{K}_{\mathsf{dst}}$ algorithm to generate and output a pair of public/private keys $\mathsf{pk_{dst}}, \mathsf{sk_{dst}}$. Set $\mathsf{pkL} = \mathsf{pk_{dst}}$ and compute $\mathsf{pkH} := \mathsf{Hash}(\mathsf{pkL})$. Parse $\mathsf{addr_{sk}}$ as $(a_{\mathsf{sk}}, \mathsf{sk_{enc}})$, compute $\mathsf{pkcm} := \mathsf{COMM}_{a_{\mathsf{sk}}}(\mathsf{pkH})$. Output the tuple $(\mathsf{pkL}, \mathsf{pkcm})$.

For privacy, each generated pkcm must be used only once. It is recommended that a user stores the output tuples $(\mathsf{pkL}, \mathsf{pkcm})$ in a table PKCM. When receiving a coin from the ledger (as described in ReceivePlus algorithm), check that the pkcm is in table PKCM, and delete it from the table after the coin using this pkcm is spent.

Mint coin. The MintPlus algorithm generates a coin and a mint transaction.

- *Input:*
  - public parameter pp
  - coin value v
  - destination address $\mathsf{addr_{pk}}$
  - **a lock** lock
- *Output:*
  - coin $\mathbf{c}$
  - mint transaction $\mathsf{tx_{Mint}}$

The Mint algorithm in Zerocash is invoked to generate a Mint transaction which spends unspent output in basecoin and outputs a coin commitment. MintPlus modifies the original algorithm, by additionally committing a public key commitment pkcm and a lock time tL. The other parts of the algorithm are left unmodified. The details of the MintPlus algorithm are presented in Alg.1.

Algorithm 1: MintPlus Algorithm

```
Parse $\mathsf{addr_{pk}}$ as $(a_{\mathsf{pk}}, \mathsf{pk_{enc}})$;
Randomly sample a $\mathsf{PRF^{sn}}$ seed $\rho$;
Randomly sample the $\mathsf{COMM}$ trapdoors $r, s$;
**Compute** $m := \mathsf{COMM}_r(a_{\mathsf{pk}}, \rho, \mathsf{lock})$;
Compute $\mathsf{cm} := \mathsf{COMM}_s(v, m)$;
**Set** $\mathbf{n} := (v, \rho, r, s, \mathsf{lock})$;
**Set** $\mathbf{c} := (a_{\mathsf{pk}}, \mathsf{cm}, \mathbf{n})$;
Set $\mathsf{tx_{Mint}} := (\mathsf{cm}, v, m, s)$;
Output $\mathbf{c}$ and $\mathsf{tx_{Mint}}$.
```

Pour algorithm. The PourPlus algorithm transfers values from two input coins into two new coins, and optionally transfer part of the input value back to the basecoin. Pouring allows parties to subdivide coins, merge coins or transfer ownership. PourPlus generates two coins and a pour transaction.

The inputs to the PourPlus algorithm can be roughly categorized into two groups. One group consists of the witnesses for validating the input coins. Specifically, we define a CoinWitness to be an assemble of the following information:

- A coin $\mathbf{c}$ and the address key $\mathsf{addr_{sk}}$ bound to it; and
- witnesses for existence of $\mathbf{c}$ on the ledger, i.e. a Merkle root rt and path path; and
- locks introduced in DAP+, i.e. $\mathsf{pkL}, \mathsf{sk}$ where sk is private key of pkL and $\mathsf{COMM}_{a_{\mathsf{sk}}}(\mathsf{pkL})$ is contained in $\mathbf{c}$.

Another group of inputs consists of the specifications for generating the new coins. In fact, the specifications for each coin are exactly the same to the inputs of the MintPlus algorithm. We define MintSpec to be a tuple $(\mathsf{addr_{pk}}, v, \mathsf{lock})$.

The inputs and outputs of PourPlus algorithm are summarized as follows:

- *Input:*
  - public parameter pp
  - public value v_{pub}
  - **minimum block height** MBH
  - **old coin witnesses** $\{\mathsf{CoinWitness}_i = (\mathbf{c}_i^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},i}^{\mathsf{old}}, \mathsf{rt}_i, \mathsf{path}_i, \mathsf{pkL}_i^{\mathsf{old}}, \mathsf{sk}_i)\}_{i=1}^2$
  - **new coin specifications** $\{\mathsf{MintSpec}_i = (\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{new}}, v_i^{\mathsf{new}}, \mathsf{lock}_i^{\mathsf{new}})\}_{i=1}^2$
- *Output:*
  - coins $\mathbf{c}_1$, $\mathbf{c}_2$
  - pour transaction $\mathsf{tx_{Pour}}$

PourPlus algorithm modifies the original Pour algorithm, by publishing the public key lock $\mathsf{pkL}_i^{\mathsf{old}}$ previously committed in each input coin. For each $\mathsf{pkL}_i^{\mathsf{old}}$ append the corresponding signature if needed. The details of the PourPlus algorithm are presented in Alg.2.

Verify Transaction Algorithm. The VerifyPlus algorithm outputs a bit b indicating if a given transaction is valid on a ledger.

- *Input:*
  - public parameters pp
  - mint/pour transaction tx
  - ledger L
- *Output:* bit b indicating if the transaction is valid

VerifyPlus modifies the original Verify algorithm, by additionally verifying the signatures of the public key locks if needed. The public inputs to the zk-SNARK module are also changed accordingly. The details of VerifyPlus algorithm are presented in Alg.3.

Receive Algorithm. The ReceivePlus algorithm scans the ledger and outputs coins on the ledger belonging to a given shielded address.

- *Input:*
  - public parameters pp
  - recipient shielded address/key pair $(\mathsf{addr_{pk}}, \mathsf{addr_{sk}})$
  - **public key commitment set** PKCM
  - ledger L
- *Output:* set of received coins

ReceivePlus modifies the original Receive algorithm, by additionally checking that the public key commitment pkcm is

Algorithm 2: PourPlus Algorithm

```
Algorithm PourPlus()
    MintNewcoin();
    PubkeyMac();
    ZKprove();
    PreventForgery();
    Unlock();
    Output();
Procedure MintNewcoin()
    for $i \in \{1,2\}$ do
        Compute $\mathbf{c}_i^{\mathsf{new}} := \mathsf{MintPlus}(\mathsf{MintSpec}_i)$;
        Parse $\mathbf{c}_i^{\mathsf{new}}$ as $(a_{\mathsf{pk},i}^{\mathsf{new}}, \mathsf{cm}_i^{\mathsf{new}}, \mathbf{n}_i^{\mathsf{new}})$;
        Set $\mathbf{C}_i := \mathcal{E}_{\mathsf{enc}}(\mathsf{pk}_{\mathsf{enc},i}^{\mathsf{new}}, \mathbf{n}_i^{\mathsf{new}})$;
    end
Procedure PubkeyMac()
    Generate $(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}) := \mathcal{K}_{\mathsf{sig}}(\mathsf{pp}_{\mathsf{sig}})$;
    Compute $h_{\mathsf{sig}} := \mathsf{CRH}(\mathsf{pk}_{\mathsf{sig}})$;
    for $i \in \{1,2\}$ do
        Parse $\mathsf{addr}_{\mathsf{sk},i}^{\mathsf{old}}$ as $(a_{\mathsf{sk},i}^{\mathsf{old}}, \mathsf{sk}_{\mathsf{enc},i}^{\mathsf{old}})$;
        Compute $h_i := \mathsf{PRF}_{a_{\mathsf{sk},i}^{\mathsf{old}}}^{\mathsf{pk}}((i-1)\|h_{\mathsf{sig}})$;
    end
Procedure ZKprove()
    for $i \in \{1,2\}$ do
        Parse $\mathbf{c}_i^{\mathsf{old}}$ as $(a_{\mathsf{pk},i}^{\mathsf{old}}, \mathsf{cm}_i^{\mathsf{old}}, \mathbf{n}_i^{\mathsf{old}})$;
        Parse $\mathbf{n}_i^{\mathsf{old}}$ as $(v_i^{\mathsf{old}}, \rho_i^{\mathsf{old}}, r_i^{\mathsf{old}}, s_i^{\mathsf{old}}, \mathsf{lock}_i^{\mathsf{old}})$;
        Compute $\mathsf{sn}_i^{\mathsf{old}} := \mathsf{PRF}_{a_{\mathsf{sk},i}^{\mathsf{old}}}^{\mathsf{sn}}(\rho_i^{\mathsf{old}})$;
        **Compute** $\mathsf{pkH}_i^{\mathsf{old}} := \mathsf{Hash}(\mathsf{pkL}_i^{\mathsf{old}})$;
        **Compute** $\mathsf{ovd}_i := \mathsf{BH}(\mathsf{rt}_i) + \mathsf{tL}_i^{\mathsf{old}} \geq \mathsf{MBH}$;
    end
    Set $\vec{x} := (\mathsf{rt}_{1..2}, \mathsf{sn}_{1..2}^{\mathsf{old}}, \mathsf{pkH}_{1..2}^{\mathsf{old}}, \mathsf{cm}_{1..2}^{\mathsf{new}}, v_{\mathsf{pub}}, h_{\mathsf{sig}},$
      $h_{1..2}, \mathsf{MBH}, \mathsf{ovd}_{1..2})$;
    Set $\vec{a} := (\mathsf{path}_{1..2}, a_{\mathsf{sk},1..2}^{\mathsf{old}}, \mathbf{c}_{1..2}^{\mathsf{old}}, \mathbf{c}_{1..2}^{\mathsf{new}})$;
    Compute $\pi_{\mathsf{POUR}} := \mathsf{Prove}(\mathsf{pk}_{\mathsf{POUR}}, \vec{x}, \vec{a})$;
Procedure PreventForgery()
    Set $M := (\vec{x}, \pi_{\mathsf{POUR}}, \mathsf{MBH}, \mathbf{C}_{1..2}, \mathsf{pkL}_{1..2}^{\mathsf{old}})$;
    Compute $\sigma := \mathcal{S}_{\mathsf{sig}}(\mathsf{sk}_{\mathsf{sig}}, M)$;
Procedure Unlock()
    **Set** $\mathsf{msg} := \mathsf{ToBeSigned}()$;
    for $i \in \{1,2\}$ do
        if $\mathsf{ovd}_i$ then
            **Compute**[2] $\sigma_i = \mathcal{S}_{\mathsf{dst}}(\mathsf{sk}_i, \mathsf{msg})$;
        else
            **Set** $\sigma_i = \perp$;
        end
        **Set** $\mathsf{unlock}_i = (\mathsf{pkL}_i^{\mathsf{old}}, \sigma_i)$;
    end
Procedure Output()
    Set $\mathsf{tx}_{\mathsf{Pour}} := (\mathsf{rt}_{1..2}, \mathsf{sn}_{1..2}^{\mathsf{old}}, \mathsf{cm}_{1..2}^{\mathsf{new}}, v_{\mathsf{pub}}, \mathsf{MBH}, *)$,
      where $* := (\mathsf{pk}_{\mathsf{sig}}, h_{1..2}, \pi_{\mathsf{POUR}}, \mathbf{C}_{1..2}, \sigma,$
      $\mathsf{unlock}_{1..2})$;
    Output $\mathbf{c}_{1..2}^{\mathsf{new}}, \mathsf{tx}_{\mathsf{Pour}}$;
```

previously generated by CreatePKCM and never used before. The details of the ReceivePlus algorithm are presented in Alg.4.

The NP Statement. Finally, we modify the NP statement POUR for the zk-SNARK module to add a claim that the public key lock pkL and the lock times tL have been correctly committed, and that the lock times are either expired or overridden. Following is the detail of the modified NP statement POUR for the zero-knowledge proof.

Algorithm 3: VerifyPlus Algorithm

```
if tx is of type $\mathsf{tx}_{\mathsf{Mint}}$ then
    Parse $\mathsf{tx}_{\mathsf{Mint}}$ as $(\mathsf{cm}, v, m, s)$;
    Set $\mathsf{cm}' := \mathsf{COMM}_s(v, m)$;
    Output $b := 1$ if $\mathsf{cm} = \mathsf{cm}'$, else output $b := 0$.
else
    Parse $\mathsf{tx}_{\mathsf{Pour}}$ as $(\mathsf{rt}_{1..2}, \mathsf{sn}_{1..2}^{\mathsf{old}}, \mathsf{cm}_{1..2}^{\mathsf{new}}, v_{\mathsf{pub}}, \mathsf{MBH}, *)$
      and $*$ as $(\mathsf{pk}_{\mathsf{sig}}, h_{1..2}, \pi_{\mathsf{POUR}}, \mathbf{C}_{1..2}, \sigma, \mathsf{unlock}_{1..2})$;
    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 $b := 0$ and exit;
    If the Merkle tree root $\mathsf{rt}_1$ or $\mathsf{rt}_2$ does not appear on
      $L$, output $b := 0$ and exit;
    Compute $h_{\mathsf{sig}} := \mathsf{CRH}(\mathsf{pk}_{\mathsf{sig}})$;
    for $i \in \{1,2\}$ do
        **Parse** $\mathsf{unlock}_i$ **as** $(\mathsf{pkL}_i, \sigma_i)$;
        **Compute** $\mathsf{pkH}_i^{\mathsf{old}} := \mathsf{Hash}(\mathsf{pkL}_i)$;
        **Set** $\mathsf{ovd}_i := (\sigma_i \neq \perp)$;
        **if** $\mathsf{ovd}_i$ ***and*** $\mathcal{V}_{\mathsf{dst}}(\mathsf{pkL}_i, \mathsf{msg}, \sigma_i) = 0$ **then**
            **output** $b := 0$ **and exit**
        **end**
    end
    Set $\vec{x} := (\mathsf{rt}_{1..2}, \mathsf{sn}_{1..2}^{\mathsf{old}}, \mathsf{pkH}_{1..2}^{\mathsf{old}}, \mathsf{cm}_{1..2}^{\mathsf{new}}, v_{\mathsf{pub}}, h_{\mathsf{sig}},$
      $h_{1..2}, \mathsf{MBH}, \mathsf{ovd}_{1..2})$;
    Set $M := (\vec{x}, \pi_{\mathsf{POUR}}, \mathsf{MBH}, \mathbf{C}_{1..2}, \mathsf{pkL}_1^{\mathsf{old}}, \mathsf{pkL}_2^{\mathsf{old}})$;
    If $\mathcal{V}_{\mathsf{sig}}(\mathsf{pk}_{\mathsf{sig}}, M, \sigma) = 0$ output $b := 0$ and exit;
    If $\mathsf{Verify}(\mathsf{vk}_{\mathsf{POUR}}, \vec{x}, \pi_{\mathsf{POUR}}) = 0$ output $b := 0$ and
      exit;
    **Set** $\mathsf{msg} := \mathsf{ToBeSigned}()$;
    Output $b := 1$;
end
```

Algorithm 4: ReceivePlus Algorithm

```
Parse $\mathsf{addr}_{\mathsf{pk}}$ as $(a_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}})$, $\mathsf{addr}_{\mathsf{sk}}$ as $(a_{\mathsf{sk}}, \mathsf{sk}_{\mathsf{enc}})$;
for each Pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ on $L$ do
    Parse $\mathsf{tx}_{\mathsf{Pour}}$ as $(\mathsf{rt}_{1..2}, \mathsf{sn}_{1..2}, \mathsf{cm}_{1..2}, v_{\mathsf{pub}}, \mathsf{MBH}, *)$;
    for each $i \in \{1,2\}$ do
        Compute $(v, \rho, r, s, \mathsf{lock}) := \mathcal{D}_{\mathsf{enc}}(\mathsf{sk}_{\mathsf{enc}}, \mathbf{C}_i)$;
        if $\mathcal{D}_{\mathsf{enc}}$ does not output $\perp$ then
            Verify that
              $\mathsf{cm}_i = \mathsf{COMM}_s(v, \mathsf{COMM}_r(a_{\mathsf{pk}}, \rho, \mathsf{lock}))$;
            **Parse** $\mathsf{lock}$ **as** $(\mathsf{pkcm}, \mathsf{tL})$;
            **Check that** $\mathsf{pkcm}$ **is in** PKCM **and never
              appears in other coins, if so, output**
              $\mathbf{c} := (a_{\mathsf{pk}}, \mathsf{cm}_i, \mathbf{n})$ where
              $\mathbf{n} = (v, \rho, r, s, \mathsf{lock})$;
        end
    end
end
```

Given

$$\vec{x} = (\mathsf{rt}_{1..2}, \mathsf{sn}_{1..2}^{\mathsf{old}}, \mathsf{pkH}_{1..2}^{\mathsf{old}}, \mathsf{cm}_{1..2}^{\mathsf{new}}, v_{\mathsf{pub}}, h_{\mathsf{sig}}, h_{1..2}, \mathsf{MBH}, \mathsf{ovd}_{1..2}),$$

[2]This procedure may be executed distributedly, where the input sk_i is shared by more than one parties, and σ_i is synthesized from the shared signatures.

where $\mathsf{pkH}_i^{\mathsf{old}} = \mathsf{Hash}(\mathsf{pkL}_i^{\mathsf{old}})$, for $i \in \{1, 2\}$, I know

$$\vec{a} = (\mathsf{path}_{1..2}, a_{\mathsf{sk},1..2}^{\mathsf{old}}, \mathbf{c}_{1..2}^{\mathsf{old}}, \mathbf{c}_{1..2}^{\mathsf{new}}),$$

such that:

- For each $i \in \{1, 2\}$:
  - The path_i is a valid authentication path for leaf $\mathsf{cm}_i^{\mathsf{old}}$ with respect to root rt_i, in a CRH-based Merkle tree.
  - The private key $a_{\mathsf{sk},i}^{\mathsf{old}}$ matches the public address of $a_{\mathsf{pk},i}^{\mathsf{old}}$.
  - The serial number $\mathsf{sn}_i^{\mathsf{old}}$ is computed correctly, i.e. $\mathsf{sn}_i^{\mathsf{old}} = \mathsf{PRF}^{\mathsf{sn}}_{a_{\mathsf{sk},i}^{\mathsf{old}}}(\rho_i^{\mathsf{old}})$.
  - **The coin $\mathbf{c}_i^{\mathsf{old}}$ is well formed, i.e.** $\mathsf{cm}_i^{\mathsf{old}} = \mathsf{COMM}_{s_i^{\mathsf{old}}}(v_i^{\mathsf{old}}, \mathsf{COMM}_{r_i^{\mathsf{old}}}(a_{\mathsf{pk},i}^{\mathsf{old}}, \rho_i^{\mathsf{old}}, \mathsf{COMM}_{a_{\mathsf{sk},i}^{\mathsf{old}}}(\mathsf{pkH}_i^{\mathsf{old}}), \mathsf{tL}_i^{\mathsf{old}}))$.
  - **The coin $\mathbf{c}_i^{\mathsf{new}}$ is well formed, i.e.** $\mathsf{cm}_i^{\mathsf{new}} = \mathsf{COMM}_{s_i^{\mathsf{new}}}(v_i^{\mathsf{new}}, \mathsf{COMM}_{r_i^{\mathsf{new}}}(a_{\mathsf{pk},i}^{\mathsf{new}}, \rho_i^{\mathsf{new}}, \mathsf{lock}_i^{\mathsf{new}}))$.
  - The address secret key ties h_{sig} to h_i, i.e. $h_i = \mathsf{PRF}^{\mathsf{pk}}_{a_{\mathsf{sk},i}^{\mathsf{old}}}((i-1)\|h_{\mathsf{sig}})$.
  - **The lock time expires or is overridden, i.e.** $\mathsf{ovd}_i || (\mathsf{BH}(\mathsf{rt}_i) + \mathsf{tL}_i^{\mathsf{old}}[k_i] < \mathsf{MBH})$.
- Balance is preserved: $v_1^{\mathsf{new}} + v_2^{\mathsf{new}} + v_{\mathsf{pub}} = v_1^{\mathsf{old}} + v_2^{\mathsf{old}}$

IV. Z-Channel

We now implement the micropayment system over Zerocash, which we call *Z-Channel*. Before we give the detailed definition and constructions, we first present an overview of how Z-Channel works.

A Z-Channel $\mathcal{ZC}$ is established for Alice and Bob on a ledger L, if the ledger confirms a coin $\mathbf{c}^{\mathsf{shr}}$, which is locked forever by a public key $\mathsf{pk}_{AB}^{\mathsf{shr}}$ shared by Alice and Bob. To close the channel, Alice and Bob have to cooperate to sign a transaction $\mathsf{tx}^{\mathsf{cls}}$ which takes $\mathbf{c}^{\mathsf{shr}}$ as input and outputs two coins for Alice and Bob separately, i.e. locked by Alice's and Bob's own public keys respectively.

However, either Alice or Bob can be malicious and uncooperative, in which case $\mathbf{c}^{\mathsf{shr}}$ may be locked forever. To avoid that, Alice and Bob each possesses a *note* $\mathbf{n}$ signed by $\mathsf{sk}_{AB}^{\mathsf{shr}}$. Whenever needed, each party can close the channel on his/her own, by creating $\mathsf{tx}^{\mathsf{cls}}$ with the data of $\mathbf{n}$. Furthermore, $\mathsf{tx}^{\mathsf{cls}}$ can only spend $\mathbf{c}^{\mathsf{shr}}$ in a way that is specified by $\mathbf{n}$, for example, to give Alice a coin of denomination v_A and Bob v_B.

When Alice pays Bob through $\mathcal{ZC}$, say coin of denomination v, they need to update their notes to instead give Alice coin of denomination $v'_A = v_A - v$ and Bob $v'_B = v_B + v$. They accomplish this by signing a new note by cooperation. We give each note a sequence number started from 0, and increment it with each update.

It is hard to guarantee that Alice will delete the obsoleted notes and never use them to generate $\mathsf{tx}^{\mathsf{cls}}$. However, we can design the protocol to guarantee that once she does so, she will suffer great loss if Bob finds out in time. To accomplish this, we construct the notes so that they are *revocable*, and when a note of higher sequence number is signed, they immediately sign a *revocation* on the obsoleted note.

A. Main Idea of Z-Channel

Above is the working scenario of a Z-Channel. Next we present the main ideas about how to fulfill this scenario.

Definition of note. A note should contain enough data for a party to construct a complete closing transaction, and be exact enough in specifying how to spend the shared coin. To accomplish this, we first specify the $\mathsf{ToBeSigned}()$ function in DAP+ scheme to contain the following: serial numbers $\mathsf{sn}_{1..2}^{\mathsf{old}}$ of the input coins, and output coin commitments $\mathsf{cm}_{1..2}^{\mathsf{new}}$. Then we define a note to be a tuple $(\mathsf{sn}^{\mathsf{shr}}, \mathsf{sn}^{\mathsf{dmy}}, \mathsf{cm}_A^{\mathsf{cls}}, \mathsf{cm}_B^{\mathsf{cls}})$, where $\mathsf{sn}^{\mathsf{shr}}$ is serial number of share coin, $\mathsf{sn}^{\mathsf{dmy}}$ is that of a dummy coin, and $\mathsf{cm}_A^{\mathsf{cls}}, \mathsf{cm}_B^{\mathsf{cls}}$ are coin commitments for Alice and Bob respectively. To generate $\mathsf{tx}^{\mathsf{cls}}$ from a signed note, one may execute the $\mathsf{PourPlus}$ algorithm, and in the $\mathsf{unlock}()$ procedure directly set the note signature as the result.

Before generating the note, Alice and Bob need to negotiate the random strings for the new coins. We will discuss the details later.

Revocation. To revoke a note, the basic idea is to give the other party permission to spend both of the output coins of a revoked note. For example, the revocation signed by Alice for Bob allows Bob to generate a revocation transaction $\mathsf{tx}^{\mathsf{rev}}$ to spend Alice's coin output by $\mathsf{tx}^{\mathsf{cls}}$.

However, there is still a chance for Alice to publish the outdated $\mathsf{tx}^{\mathsf{cls}}$ and take her own coin. Even worse, with Bob's revocation she may even take Bob's coin away. To solve this problem, Alice and Bob have to possess different versions of note, and in Alice's version of note, Alice's coin should be time-locked to give Bob time to publish the revocation transaction.

We accomplish this by instead locking Alice's coin in Alice's version of $\mathsf{tx}^{\mathsf{cls}}$ by a shared public key $\mathsf{pk}_{AB}^{\mathsf{cls}}$ with lock time T, and same for Bob's coin in Bob's version. The revocation signed by Alice then gives Bob permission to override the lock time. Meanwhile, neither of Alice or Bob can take the other's coin by publishing his/her own version of $\mathsf{tx}^{\mathsf{cls}}$.

Now, Alice has to wait for T blocks even as she publishes the most updated closing transaction. A problem arises that after T blocks both Alice and Bob have the ability to take away the coin that should belong to Alice. To solve this problem, we modify the DAP+ scheme sightly, by requiring that even when the lock time expires, a signature is still needed, which is signed on a message sightly different from $\mathsf{ToBeSigned}()$ while still keeping the information, for example, the message obtained by appending a sequence of zeros to $\mathsf{ToBeSigned}()$. We call this is a "weak" signature in the sense that it does not override the lock time. The modifications to the $\mathsf{PourPlus}$ and $\mathsf{VerifyPlus}$ algorithms are straightforward, i.e. simply adding the processes of generating and verifying the weak signature when ovd is false.

We then modify the update procedure to let Alice "weakly sign" another note $(\mathsf{sn}_A^{\mathsf{cls}}, \mathsf{sn}^{\mathsf{dmy}}, \mathsf{cm}_B^{\mathsf{rdm}}, \mathsf{cm}^{\mathsf{dmy}})$ for Bob with her secret share of $\mathsf{sk}_B^{\mathsf{cls},A}$, where $\mathsf{cm}_B^{\mathsf{rdm}}$ is the *redeem coin* locked by Bob's own public key. Similar modification applies to Bob. Now, to close a channel, Alice has to first generate and publish her version of $\mathsf{tx}^{\mathsf{cls}}$, then wait for T blocks before publishing $\mathsf{tx}^{\mathsf{rdm}}$.

Since their notes are different, when we say "Alice signs a note for Bob" we actually mean Alice generates a share of signature and sends to Bob, then Bob completes the signature. In this way, Alice does not know the complete signatures of Bob's notes.

Establish the channel. We now talk about how does $\mathbf{c}^{\mathsf{shr}}$ gets on the ledger in the first place. To generate this coin, Alice and Bob each spends his/her own money to generate a *fund coin* $\mathbf{c}_A^{\mathsf{fund}}$ and $\mathbf{c}_B^{\mathsf{fund}}$ of appropriate denomination, locked forever by Alice's and Bob's own public keys respectively. They then generates $\mathsf{tx}^{\mathsf{shr}}$ which takes the fund coins as inputs and outputs $\mathbf{c}^{\mathsf{shr}}$. To prevent $\mathbf{c}^{\mathsf{shr}}$ from locking their coins forever, they sign the notes of sequence number 0 for each other before they sign $\mathsf{tx}^{\mathsf{shr}}$.

Consensus on the random strings. We mentioned that with the data of a note, Alice should be able to generate the complete transaction. However, she is unable to do so without knowing all the random strings used to generate the serial numbers and commitments in the note.

Instead of letting Alice and Bob negotiate all these data when generating each note, we save all these communication costs by letting them agree on a secret seed which is used to generate everything with a pseudorandom function $\mathsf{PRF}_{\mathcal{ZC}}$. We give each random string a unique identifier, which is used as the input to the pseudorandom function. Meanwhile, all their public keys to be used in the channel are negotiated in the establishment phase. In this way, Alice and Bob automatically agree on all data about each transaction except the denomination of coins, and the only interactions needed in updating a channel are for distributed generation of signatures.

We complete this subsection by presenting all the transactions and coins confirmed on the ledger, when a Z-Channel is closed peacefully by Alice, in Fig.1

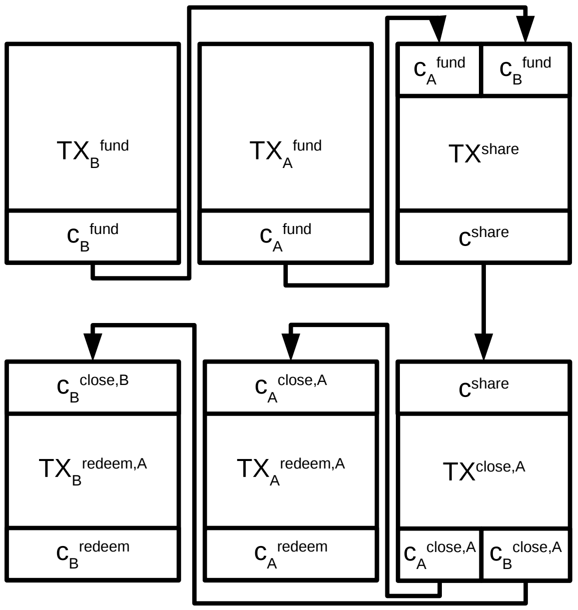


Fig. 1: Transactions and coins on ledger in a closed Z-Channel. In this example Alice's version of closing transaction is published. If the closing transaction is outdated, the graph is the same except that the redeem transaction of Alice is replaced by a revocation transaction of Bob.

$\mathbf{c}$	v	a_{sk}	ρ	r	s	pkL	tL
$\mathbf{c}_A^{\mathsf{fund}}$	v_A	a_{sk}	ρ_A^{fund}	r_A^{fund}	s_A^{fund}	$\mathsf{pk}_A^{\mathsf{fund}}$	MLT
$\mathbf{c}_B^{\mathsf{fund}}$	v_B	a_{sk}	ρ_B^{fund}	r_B^{fund}	s_B^{fund}	$\mathsf{pk}_B^{\mathsf{fund}}$	MLT
$\mathbf{c}^{\mathsf{shr}}$	$v_A + v_B$	a_{sk}	ρ^{shr}	r^{shr}	s^{shr}	$\mathsf{pk}_{AB}^{\mathsf{shr}}$	MLT
$\mathbf{c}_{A,i}^{\mathsf{cls},A}$		a_{sk}	$\rho_{A,i}^{\mathsf{cls},A}$	$r_{A,i}^{\mathsf{cls},A}$	$s_{A,i}^{\mathsf{cls},A}$	$\mathsf{pk}_{AB}^{\mathsf{cls}}$	T
$\mathbf{c}_{B,i}^{\mathsf{cls},A}$		a_{sk}	$\rho_{B,i}^{\mathsf{cls},A}$	$r_{B,i}^{\mathsf{cls},A}$	$s_{B,i}^{\mathsf{cls},A}$	$\mathsf{pk}_A^{\mathsf{cls}}$	MLT
$\mathbf{c}_{B,i}^{\mathsf{cls},B}$		a_{sk}	$\rho_{B,i}^{\mathsf{cls},B}$	$r_{B,i}^{\mathsf{cls},B}$	$s_{B,i}^{\mathsf{cls},B}$	$\mathsf{pk}_{AB}^{\mathsf{cls}}$	T
$\mathbf{c}_{A,i}^{\mathsf{cls},B}$		a_{sk}	$\rho_{A,i}^{\mathsf{cls},B}$	$r_{A,i}^{\mathsf{cls},B}$	$s_{A,i}^{\mathsf{cls},B}$	$\mathsf{pk}_B^{\mathsf{cls}}$	MLT
$\mathbf{c}_A^{\mathsf{rdm}}$		a_{sk}	ρ_A^{rdm}	r_A^{rdm}	s_A^{rdm}	$\mathsf{pk}_A^{\mathsf{rdm}}$	MLT
$\mathbf{c}_B^{\mathsf{rdm}}$		a_{sk}	ρ_B^{rdm}	r_B^{rdm}	s_B^{rdm}	$\mathsf{pk}_B^{\mathsf{rdm}}$	MLT
$\mathbf{c}_A^{\mathsf{rev}}$		a_{sk}	ρ_A^{rev}	r_A^{rev}	s_A^{rev}	$\mathsf{pk}_A^{\mathsf{rev}}$	MLT
$\mathbf{c}_B^{\mathsf{rev}}$		a_{sk}	ρ_B^{rev}	r_B^{rev}	s_B^{rev}	$\mathsf{pk}_B^{\mathsf{rev}}$	MLT

TABLE I: Coins specifications in Z-Channel

B. Construction of Z-Channel Protocol

A Z-Channel Protocol ZCP is a tuple of subprotocols ($\mathsf{Establish}$, Update, Close). Before we give concrete construction of these subprotocols, we first present several algorithms needed.

Generate coin from seed. Table.I summarizes all the data determined in the establishment of Z-Channel needed by each coin. We use A and B in superscript to differentiate the closing transactions of Alice or Bob's version. We use subscript i to denote the sequence number of the closing transactions. Note that there is only one a_{sk} for all the coins, i.e. all the coins are bound to the same receiving address.

All the values in the a_{sk}, ρ, r and s columns of the table can be generated by $\mathsf{PRF}_{\mathcal{ZC}}$ with the seed. The locks pkL and tL cannot be computed by $\mathsf{PRF}_{\mathcal{ZC}}$, but they can be stored and looked up by their identifiers once generated. The denominations of the coins in closing transactions after the first update can be determined when they decide to make the update. Therefore, we can always assume that when they need to generate a coin, they already have all the needed data. Denote the procedure of computing the data of a coin by $\mathsf{GetCoin}()$.

Sign and verify notes. When Alice and Bob agree on denominations v_A and v_B for notes of sequence number i, Alice and Bob each invokes $\mathsf{SignNote}$ to generate shares of signature for each other. The inputs are v_A, v_B, and sequence number i. The version for Bob is presented in Alg.5. The version for Alice is obtained by swapping the A and B labels.

On receiving the signature shares, Alice completes the signature and verifies it, as is shown in Alg.6. Bob's version simply swaps the A and B subscripts.

Next, we present the construction of the subprotocols. Note that in Alg.7 and Alg.8 we divide (by horizontal rule) the

Algorithm 5: SignNote Algorithm

```
Lookup for secret keys $\mathsf{sk}^{\mathsf{shr}}$, $\mathsf{sk}^{\mathsf{cls}}$;
Compute $\mathsf{cm}^{\mathsf{shr}}, \mathsf{sn}^{\mathsf{shr}} := \mathsf{GetCoin}()$;
Compute $\mathsf{cm}^{\mathsf{cls},A}_{A,i}, \mathsf{sn}^{\mathsf{cls},A}_{A,i} := \mathsf{GetCoin}()$;
Compute $\mathsf{cm}^{\mathsf{cls},A}_{B,i}, \mathsf{sn}^{\mathsf{cls},A}_{B,i} := \mathsf{GetCoin}()$;
Let $\mathbf{n}_1 := (\mathsf{sn}^{\mathsf{shr}}, \mathsf{sn}^{\mathsf{dmy}}, \mathsf{cm}^{\mathsf{cls},A}_{A,i}, \mathsf{cm}^{\mathsf{cls},A}_{A,i})$;
Let $\mathbf{n}_2 := (\mathsf{sn}^{\mathsf{cls},A}_{A,i}, \mathsf{sn}^{\mathsf{dmy}}, \mathsf{cm}^{\mathsf{rdm}}_{A}, \mathsf{cm}^{\mathsf{dmy}})$;
Compute $\sigma_1 := \mathcal{S}_{\mathsf{dst}}(\mathsf{sk}^{\mathsf{shr}}, \mathbf{n}_1)$;
Compute $\sigma_2 := \mathcal{S}_{\mathsf{dst}}(\mathsf{sk}^{\mathsf{cls}}, \mathbf{n}_2)$;
Output $\sigma_1, \sigma_2$;
```

Algorithm 6: CompleteSign Algorithm

```
Lookup for keys $\mathsf{sk}^{\mathsf{shr}}$, $\mathsf{sk}^{\mathsf{cls}}$, $\mathsf{pk}^{\mathsf{shr}}$, $\mathsf{pk}^{\mathsf{cls}}$;
Compute $\mathbf{n}_1$ and $\mathbf{n}_2$ as in Alg.5;
Compute $\sigma'_1 := \mathcal{S}_{\mathsf{dst}}(\mathsf{sk}^{\mathsf{shr}}, \mathbf{n}_1)$;
Compute $\sigma'_2 := \mathcal{S}_{\mathsf{dst}}(\mathsf{sk}^{\mathsf{cls}}, \mathbf{n}_2)$;
Compute $\sigma^{\mathsf{cls}}$ by combining $\sigma_1$ and $\sigma'_1$;
Compute $\sigma^{\mathsf{rdm}}$ by combining $\sigma_2$ and $\sigma'_2$;
Compute $b_1 := \mathcal{V}_{\mathsf{dst}}(\mathsf{pk}^{\mathsf{shr}}, \mathbf{n}_1, \sigma^{\mathsf{cls}})$;
Compute $b_2 := \mathcal{V}_{\mathsf{dst}}(\mathsf{pk}^{\mathsf{cls}}, \mathbf{n}_2, \sigma^{\mathsf{rdm}})$;
if $b_1 = 1$ and $b_2 = 1$ then
    Output $\sigma^{\mathsf{cls}}, \sigma^{\mathsf{rdm}}$;
else
    Output $\perp$;
end
```

procedures into groups. In each group the procedures can be executed simultaneously, i.e. regardless of the presented order, while different groups have to be finished in sequence. For clarity, we omit the description of sending data to the other party, or checking the correctness, etc. In each group, if they fall into dispute, or one party finds another not behaving in the expected way, he/she can immediately abort the protocol. (When the channel is established, to abort means executing the Close protocol.)

Establish the channel. In the establishment phase, Alice and Bob agree on a random seed seed which completely determines all the random values used to generate the coins. They also need to generate and exchange all the public keys needed in the channel. Therefore, all the coins are completely determined in the establishment phase, except their denominations. After that, they generate fund coins and the share coin as described in the previous subsection. This protocol is formalized in Alg.7.

Update the state of channel. To update the channel, Alice and Bob sign new notes for each other. After that, they sign revocations for each other to revoke the old version of notes. This protocol is formalized in Alg.8.

Close the channel. Without loss of generality, assume Alice is the party that actively closes the channel. Alice publishes closing transaction specified by her most updated note. Then they generate redeeming transactions to take away their coins. Alice has to wait for T blocks before publishing the redeeming transaction, while Bob can make it immediately. This protocol is formalized in Alg.9.

Algorithm 7: Establish Protocol

```
Alice and Bob agree on seed, $v_A$ and $v_B$;
Alice and Bob invoke distributed version of $\mathcal{K}_{\mathsf{dst}}$
  twice to generate $\mathsf{pk}^{\mathsf{shr}}_{AB}$ and $\mathsf{pk}^{\mathsf{cls}}_{AB}$;
--------------------------------------------
Alice generates $\mathsf{pk}^{\mathsf{fund}}_A, \mathsf{pk}^{\mathsf{cls}}_A, \mathsf{pk}^{\mathsf{rdm}}_A, \mathsf{pk}^{\mathsf{rev}}_A$;
Bob generates $\mathsf{pk}^{\mathsf{fund}}_B, \mathsf{pk}^{\mathsf{cls}}_B, \mathsf{pk}^{\mathsf{rdm}}_B, \mathsf{pk}^{\mathsf{rev}}_B$;
--------------------------------------------
Alice computes $\mathsf{SignNote}(v_B, v_A, 0)$;
Bob computes $\mathsf{SignNote}(v_A, v_B, 0)$;
--------------------------------------------
Alice signs $(\mathsf{sn}^{\mathsf{fund}}_A, \mathsf{sn}^{\mathsf{fund}}_B, \mathsf{cm}^{\mathsf{shr}}, \mathsf{cm}^{\mathsf{dmy}})$;
Bob signs $(\mathsf{sn}^{\mathsf{fund}}_A, \mathsf{sn}^{\mathsf{fund}}_B, \mathsf{cm}^{\mathsf{shr}}, \mathsf{cm}^{\mathsf{dmy}})$;
--------------------------------------------
Alice confirms on ledger $\mathbf{c}^{\mathsf{fund}}_A := \mathsf{GetCoin}()$
Bob confirms on ledger $\mathbf{c}^{\mathsf{fund}}_B := \mathsf{GetCoin}()$
--------------------------------------------
Alice/Bob confirms on ledger $\mathbf{c}^{\mathsf{shr}}$;
```

Algorithm 8: Update Protocol

```
Alice and Bob agree on $v_{A,i}$ and $v_{B,i}$;
--------------------------------------------
Alice computes $\mathsf{SignNote}(v_{B,i}, v_{A,i}, i)$;
Bob computes $\mathsf{SignNote}(v_{A,i}, v_{B,i}, i)$;
--------------------------------------------
Alice signs $(\mathsf{sn}^{\mathsf{cls},A}_{A,i}, \mathsf{sn}^{\mathsf{dmy}}, \mathsf{cm}^{\mathsf{rev}}_B, \mathsf{cm}^{\mathsf{dmy}})$;
Bob signs $(\mathsf{sn}^{\mathsf{cls},B}_{B,i}, \mathsf{sn}^{\mathsf{dmy}}, \mathsf{cm}^{\mathsf{rev}}_A, \mathsf{cm}^{\mathsf{dmy}})$;
```

Algorithm 9: Close Protocol

```
Alice confirms on ledger $\mathbf{c}^{\mathsf{cls},A}$;
Bob confirms on ledger $\mathbf{c}^{\mathsf{rdm}}_B$;
Alice waits $T$ blocks and confirms on ledger $\mathbf{c}^{\mathsf{rdm}}_A$;
```

V. Completeness and Security of Z-Channel Protocol

We define the completeness and security of Z-Channel protocol in Definition 1 and 2.

Definition 1: A Z-Channel Protocol ZCP is said to be *complete* if the transactions made in Z-Channel transfer correct value of currencies between the parties when it is closed.

Definition 2: A Z-Channel Protocol ZCP is said to be secure if it satisfies the properties of *currency security* and *channel privacy*.

The completeness property follows directly from the design of the Z-Channel protocol. The analysis is omitted, since the description of the protocol is self-explanatory. We claim that the completeness of DAP+ scheme implies the completeness of Z-Channel. The completeness of DAP+ is discusses later in subsection V-C. The definitions of the two properties *currency security* and *channel privacy* are presented in subsection V-A and V-B.

Our main theorem claims that the construction of Z-Channel Protocol in section IV is secure.

Theorem 5.1: The protocols presented in Alg.7, Alg.8 and Alg.9 form a secure Z-Channel Protocol.

Proof: By Lemma 5.2, the protocols satisfy currency security. By Lemma 5.3, the protocols satisfy channel privacy. Thus concludes the proof.

In the next subsections, we will discuss the two properties of our construction of Z-Channel based on the following assumption: Alice and Bob always have a secure communication channel established between them whenever needed. Specifically, we require that the channel is secure against eavesdropping and man-in-the-middle attacks.

A. Currency Security

Definition 3: A Z-Channel Protocol ZCP is said to satisfy *currency security* if for any adversary $\mathcal{A}$, the probability for him to win the *Z-Channel Currency* game ZCC is negligible.

The game ZCC is conducted as follows: a challenger $\mathcal{C}$ maintains a DAP+ oracle $\mathcal{O}^{DAP+}$, which maintains a DAP+ scheme on a ledger L (the detailed description of $\mathcal{O}^{DAP+}$ is presented in subsection V-C). $\mathcal{C}$ also maintains a sequence number i which is initially set to 0. The game takes as parameter a list of balances $\mathsf{vs} = \{(v_{A,i}, v_{B,i})\}_{i=0}^{n}$ specifying the number of updates and the balances of each update.

At the beginning, $\mathcal{C}$ executes Setup algorithm to initialize the ledger and sends the resulting public parameter pp to $\mathcal{A}$. Then $\mathcal{C}$ and $\mathcal{A}$ each executes CreateAddress algorithm to generate a shielded address, and sends the address to each other. Denote the address for $\mathcal{A}$ by $\mathsf{addr}_{\mathsf{pk},A}$ and that of $\mathcal{C}$ by $\mathsf{addr}_{\mathsf{pk},B}$. $\mathcal{C}$ queries $\mathcal{O}^{DAP+}$ to mint two coins for each address with value $v_{A,0}$ and $v_{B,0}$ respectively.

Then they conduct the ZCP protocol by querying $\mathcal{O}^{DAP+}$ to insert the transactions and updating the balances as specified by vs. $\mathcal{A}$ can send $\mathcal{C}$ any data at any time, and if it is a pour transaction, $\mathcal{C}$ directly passes it to $\mathcal{O}^{DAP+}$. After each insertion, $\mathcal{C}$ presents $\mathcal{A}$ the resulting ledger. $\mathcal{C}$ aborts and outputs 0 whenever $\mathcal{O}^{DAP+}$ aborts due to an invalid transaction inserted to ledger. After each update, $\mathcal{C}$ updates sequence number $i := i + 1$.

$\mathcal{C}$ starts the closing subprotocol when $i = n$ or anytime when $\mathcal{A}$ sends a closing transaction or any unexpected data. After the closing subprotocol is started, $\mathcal{C}$ stops receiving data from $\mathcal{A}$, except the transactions, which he still has to pass to $\mathcal{O}^{DAP+}$. $\mathcal{C}$ outputs 1 if $\mathcal{A}$ successfully inserts a transaction to the ledger which transfers value v to $\mathsf{addr}_{\mathsf{pk},A}$, which is larger than both $v_{A,i}$ and $v_{A,i+1}$ [3]. Else, if $\mathcal{C}$ successfully closes the channel in expected manner, he outputs 0. $\mathcal{A}$ wins ZCC if $\mathcal{C}$ outputs 1.

The following lemma claims that our construction of ZCP satisfies currency security.

[3]We do not consider the loss of the denomination of a single payment a serious issue. Due to the fact that the actions of paying and receiving service is not atomic, the problem of fair exchange exists ubiquitously, and not just in ledger-based digital currencies. The common solutions to this problem such as trusted third party or smart contract are beyond the discussion of this paper. Therefore, we simply assume that loss of the amount of a single payment is tolerable.

Lemma 5.2: The protocols presented in Alg.7, Alg.8 and Alg.9 form a Z-Channel Protocol that satisfies currency security.

Sketch of Proof: If $\mathcal{A}$ only issues transactions permitted by the protocol, the analysis in Section IV for the design of the subprotocols already covers all the cases where $\mathcal{A}$ may cheat. Therefore, the probability that $\mathcal{A}$ wins the game is bound by the probability that $\mathcal{A}$ breaks the non-malleability and balance property of DAP Plus (see subsection V-C), which is negligible by Theorem 5.4.

B. Channel Privacy

Definition 4: A Z-Channel Protocol ZCP is said to satisfy *channel privacy* property if for any adversary $\mathcal{A}$, the probability for him to win the *Channel Privacy* game CP is negligible.

The game CP is conducted as follows: a challenger $\mathcal{C}$ maintains two DAP+ oracles $\mathcal{O}_0^{DAP+}$ and $\mathcal{O}_1^{DAP+}$, each of which maintains a ledger L_1 and L_2 respectively. At the beginning of this game, $\mathcal{C}$ randomly samples a bit b. Then $\mathcal{A}$ sends $\mathcal{C}$ a list of balances $\mathsf{vs} = \{(v_{A,i}, v_{B,i})\}_{i=0}^{n}$ and all the details to specify an execution of a ZCP, i.e. the version of the closing transaction to publish and the times for the executions of all the subprotocols, etc. $\mathcal{C}$ executes ZCP locally and inserts pour transactions to $\mathcal{O}_0^{DAP+}$. Each time $\mathcal{C}$ inserts a transaction to $\mathcal{O}_0^{DAP+}$, he simultaneously inserts a randomly generated pour transaction to $\mathcal{O}_1^{DAP+}$, which is publicly consistent to the one inserted to $\mathcal{O}_1^{DAP+}$. Finally, $\mathcal{C}$ presents $L_{\mathsf{left}} := L_b$ and $L_{\mathsf{right}} := L_{1-b}$ to $\mathcal{A}$, and $\mathcal{A}$ outputs a bit b'. $\mathcal{A}$ wins CP if $b' = b$.

The following lemma claims that our construction of ZCP satisfies channel privacy property.

Lemma 5.3: The protocols presented in Alg.7, Alg.8 and Alg.9 form a Z-Channel Protocol that satisfies channel privacy property.

Sketch of Proof: We claim that $\mathcal{A}$ perceives less information in CP game than in $\mathsf{L}-\mathsf{IND}$ game (which is used to define the ledger-indistinguishability of DAP+, see subsection V-C). As a result, the probability that $\mathcal{A}$ wins this game is bound by the probability that $\mathcal{A}$ wins $\mathsf{L}-\mathsf{IND}$ game which is negligible by Theorem 5.4.

Now we can safely conclude that the completeness and security of Z-Channel are based on those of DAP+ scheme, which we discuss in the next subsection.

C. Completeness and Security of DAP+ Scheme

In this subsection, we present the formal definition of the completeness and security of DAP+ scheme.

The completeness and security of DAP+ are defined similar to those of DAP scheme in [25]. The completeness of DAP is defined by INCOMP experiment. The security of DAP+ consists of the properties of *ledger indistinguishability*, *transaction non-malleability* and *balance*, which are defined by experiments $\mathsf{L}-\mathsf{IND}$, $\mathsf{TR}-\mathsf{NM}$ and BAL respectively. We use a modified version of the above mentioned experiments to define the completeness and security for DAP+ scheme.

Definition 5: We say that a DAP Plus scheme Π = (Setup, CreatePKCM, CreateAddress, MintPlus, PourPlus, VerifyPlus, ReceivePlus) is complete, if no polynomial-size adversary $\mathcal{A}$ wins INCOMP with more than negligible probability.

Definition 6: We say that a DAP Plus scheme Π = (Setup, CreatePKCM, CreateAddress, MintPlus, PourPlus, VerifyPlus, ReceivePlus) is secure, if it is secure under experiment $\mathsf{L-IND}$, $\mathsf{TR-NM}$ and BAL.

In the INCOMP experiment, an adversary $\mathcal{A}$ sends $\mathcal{C}$ a ledger L and two coins $\mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}$, and parameters needed to spend the coins. $\mathcal{C}$ tries to spend the two coins and gets a pour transaction $\mathsf{tx}_{\mathsf{Pour}}$. $\mathcal{A}$ wins if the L is a valid ledger, the parameters are valid with respect to L, the transaction $\mathsf{tx}_{\mathsf{Pour}}$ is consistent to the parameters, but $\mathsf{tx}_{\mathsf{Pour}}$ cannot be verified on the ledger. The completeness requires that $\mathcal{A}$ wins with negligible probability.

In the $\mathsf{L-IND}$ experiment, $\mathcal{C}$ samples a random bit b establishes two oracles $\mathcal{O}_0^{DAP+}$ and $\mathcal{O}_1^{DAP+}$, each of which maintains a DAP Plus scheme on a ledger L_0 and L_1 respectively. In each step $\mathcal{A}$ is presented with the two ledgers L_b and L_{b-1} and issues a pair of queries (Q, Q') to $\mathcal{C}$, which will be forwarded to the oracles $\mathcal{O}_0^{DAP+}$ and $\mathcal{O}_1^{DAP+}$ respectively. The queries Q and Q' satisfy *public consistency* that they matches in type and reveals the same information to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a guess b' and wins when $b' = b$. The ledger indistinguishability requires that the advantage of $\mathcal{A}$ is negligible.

In the $\mathsf{TR-NM}$ experiment, $\mathcal{A}$ interacts with one DAP Plus scheme oracle and then outputs a pour transaction $\mathsf{tx}'_{\mathsf{Pour}}$, and wins if there is a pour transaction $\mathsf{tx}_{\mathsf{Pour}} \neq \mathsf{tx}'_{\mathsf{Pour}}$ on the ledger such that $\mathsf{tx}_{\mathsf{Pour}}$ reveals the same serial number of $\mathsf{tx}'_{\mathsf{Pour}}$ and that if $\mathsf{tx}'_{\mathsf{Pour}}$ takes the place of $\mathsf{tx}_{\mathsf{Pour}}$ the ledger is still valid. The transaction non-malleability requires that $\mathcal{A}$ wins with negligible probability.

In the BAL experiment, $\mathcal{A}$ interacts with one DAP Plus scheme oracle and wins the game if the total value he can spend or has spent is greater than the value he has minted or received. The balance requires that $\mathcal{A}$ wins with negligible probability.

For the definitions of security under the above experiments, refer to the full version of this paper.Regarding the experiments $\mathsf{L-IND}$, $\mathsf{TR-NM}$ and BAL, we design them similarly to those in [25], and the major modifications are listed below.

Assume that $\mathcal{O}^{DAP+}$ maintains two tables PKCM, OLDPKCM (in addition to the tables mentioned in the original version). We add a new kind of query CreatePKCM as follows:

- $Q = (\mathbf{CreatePKCM}, K)$
  a) Invoke $\mathsf{CreatePKCM}(\mathsf{pp})$ to obtain the tuple $(\mathsf{sk}, \mathsf{pkL}, \mathsf{pkcm})$.
  b) Store $(\mathsf{sk}, \mathsf{pkL}, \mathsf{pkcm})$ in table PKCM.
  c) Output pkcm.

We modify the queries Mint, Pour as follows:

- For each $\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{old}}$, $\mathcal{A}$ provides boolean flag ovd_i to indicate whether to override the lock time by unlocking public key lock.
- The flag ovd_i in Q and Q' must be the same for each input coin, and if ovd_i is false, the selected lock time must be expired.
- For $\mathsf{addr}_{\mathsf{pk}}$ in Mint query or each $\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{new}}$ in Pour query, $\mathcal{A}$ provides a public key commitment $\mathsf{pkcm}_i^{\mathsf{new}}$ and a lock time $\mathsf{tL}_i^{\mathsf{new}}$.
- If the address is in ADDR, $\mathcal{O}^{DAP+}$ checks that $\mathsf{pkcm}_i^{\mathsf{new}}$ is in PKCM and not in OLDPKCM, and aborts if the check fails.
- If the address is not in ADDR, $\mathcal{O}^{DAP+}$ checks that $\mathsf{pkcm}_i^{\mathsf{new}}$ is not in either PKCM or OLDPKCM, and aborts if the check fails.
- If the Mint or Pour query is successful, $\mathcal{O}^{DAP+}$ removes all $\mathsf{pkcm}^{\mathsf{new}}$ mentioned from PKCM and stores the tuple $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{sk}, \mathsf{pkL}, \mathsf{pkcm})$ in OLDPKCM.
- For Pour query, $\mathcal{O}$ looks up the table OLDPKCM to find the tuple ($\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{old}}$, $\mathsf{sk}_i^{\mathsf{old}}$, $\mathsf{pkL}_i^{\mathsf{old}}$, $\mathsf{pkcm}_i^{\mathsf{old}}$) for each $\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{old}}$, include $\mathsf{pkL}_i^{\mathsf{old}}$ in the pour transaction $\mathsf{tx}_{\mathsf{Pour}}$. If ovd_i is false, $\mathcal{O}$ checks that $\mathsf{tL}_i^{\mathsf{old}}$ is less than current time, aborts if check fails. If ovd_i is true, $\mathcal{O}$ signs the transaction with the corresponding secret key of $\mathsf{pkL}_i^{\mathsf{old}}$ and include the signature in $\mathsf{tx}_{\mathsf{Pour}}$.

We remove the Receive query in the original definition of $\mathcal{O}^{DAP+}$ for the following reasons:

- The Receive query does not model a proper attacking scenario in real life. In fact, this query allows $\mathcal{A}$ to identify the coins belonging to an address for which $\mathcal{A}$ does not hold the secret key, which is *unreasonable* in real life.
- The Receive query compromises the ledger indistinguishability. We devise the following attack to the $\mathsf{L-IND}$ game making use of the information provided by Receive query. First, $\mathcal{A}$ issues two pairs of CreateAddress queries to receive two address public keys, for simplicity we denote the two addresses by Alice and Bob respectively. Then, $\mathcal{A}$ issues a pair of Mint queries to generate a coin for Alice in both ledgers. Next, $\mathcal{A}$ issues a pair of Pour queries (Q, Q') to $\mathcal{C}$. In Q $\mathcal{A}$ specifies that Alice pays her coin to Bob, while in Q' Alice pays the coin to herself. Finally, $\mathcal{A}$ issues a pair of Receive queries on Alice, and obtains the lists of coin commitments for the ledgers respectively. The oracle that returns an empty commitment list is the one maintaining ledger L_0. Thus $\mathcal{A}$ wins $\mathsf{L-IND}$ game with 100 percent probability.
- We considered keeping this query to keep the consistency between the queries and the algorithms. However, if we modify the receiving query to output the coins belonging to an address of $\mathcal{A}$, it would be redundant since $\mathcal{A}$ can simply execute the ReceivePlus algorithm on the ledgers locally. If we modify the query to simply tell $\mathcal{O}^{DAP+}$ to execute ReceivePlus algorithm on an address in ADDR but do not output the result, this query is also redundant since $\mathcal{O}^{DAP+}$ is already specified to execute ReceivePlus after each Mint, Pour and Insert query.

We modify the Insert query as follows: for each output coin, check that the pkcm in the coin is stored in PKCM, abort if not so; remove the corresponding tuple from PKCM and add to OLDPKCM.

The following theorem claims that our construction of DAP Plus scheme is complete and secure under the above definitions.

Theorem 5.4: The tuple (Setup, CreatePKCM, CreateAddress, MintPlus, PourPlus, VerifyPlus, ReceivePlus) is a complete and secure DAP Plus scheme.

The proof is similar to that of Theorem 4.1 in [25]. Here we only present the modifications to the original one. For the complete proof refer to the full version of this paper.

a) Modify the simulation experiment: The simulated experiment $\partial_{\mathbf{sim}}$ proceed as in [25], except for the following modification:

1) **Answering** CreatePKCM **queries.** To answer Q, $\mathcal{C}$ behaves as in $\mathsf{L-IND}$, except for the following modification: after obtaining $(\mathsf{sk}, \mathsf{pkL}, \mathsf{pkcm})$, $\mathcal{C}$ replaces pkcm with a random string of the appropriate length; then, $\mathcal{C}$ stores the tuple in PKCM and returns pkcm to $\mathcal{A}$. Afterwards, $\mathcal{C}$ does the same for Q'.
2) **Answering** Mint **queries.** Compute $m = \mathsf{COMM}_r(\tau)$ for a random string τ of the suitable length, instead of $m = \mathsf{COMM}_r(a_{\mathsf{pk}}, \rho, \mathsf{pkcm}, \mathsf{tL})$. Afterwards, $\mathcal{C}$ does the same for Q'.
   Remark *There is no need to modify the* Pour *queries except for the modifications mentioned in [25], which already discard the information of* pkcm *and* tL *in the commitment* $\mathsf{cm}_i^{\mathsf{new}}$ *and ciphertext* $\mathbf{C}_i^{\mathsf{new}}$. *For each* $\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{old}}$ *the simulated oracle puts the original* pkL *looked up from* $\mathsf{OLDPKCM}$ *in* $\mathsf{tx}_{\mathsf{Pour}}$. *It makes no difference to replace it by a newly generated one, since the one stored in the table is independent from the random string replacing* $\mathsf{pkcm}_i^{\mathsf{old}}$.

b) Difference between $\partial_{\mathbf{sim}}$ and hybrid experiment ∂_3: Let $q_{\mathbf{CP}}$ be the total number of CreatePKCM queries issued by $\mathcal{A}$. In addition to those described in [25] appendix D.1, we additionally let the experiment $\partial_{\mathbf{sim}}$ modifies ∂_3 in the following ways:

1) Each time $\mathcal{A}$ issues a CreatePKCM query, the commitment pkcm is substituted with a random string of suitable length.
2) Each time $\mathcal{A}$ issues a Mint query, the commitment k in $\mathsf{tx}_{\mathsf{Mint}}$ is substituted with a commitment to a random input.

Then we modify the Lemma D.3 in [25] appendix D.1 as follows:

$$\left|\mathbf{Adv}^{\partial_{\mathbf{sim}}} - \mathbf{Adv}^{\partial_3}\right| \leq (q_{\mathbf{M}} + 4 \cdot q_{\mathbf{P}} + q_{\mathbf{CP}}) \cdot \mathbf{Adv}^{\mathsf{COMM}}$$

VI. Performance Analysis

In this section, we run several experiments to measure the performance of DAP Plus scheme and Z-Channel. For the DAP Plus scheme, we construct new circuit based on that of ZCash and benchmark the performance of zk-SNARK on generating the keys, proving and verification with this circuit. For comparison, we also benchmark the performance of the original DAP scheme with the same environment. The result shows that the modification introduced in DAP Plus increases the key sizes and algorithm running times in a limited manner.

For Z-Channel, we implement the protocol and benchmark the computation introduced in the protocol. Then, we run a simulation of blockchain-based ledger on a large network of users, where a portion of the payments are conducted via Z-Channel. We measure the scalability and payment confirmation time of the ledger under different ratio of the Z-Channel payments. The result shows that Z-Channel increases the ledger scalability and reduces the payment confirmation time significantly.

A. Instantiation of DAP Plus and Z-Channel

Instantiation of DAP Plus. Instead of the instantiation proposed in the [25], we base our implementation of DAP Plus on that of ZCash [13], which is the most popular implementation and improvement of DAP scheme. ZCash basically follows the idea of DAP scheme, but modifies the algorithm details and the data structures dramatically. For example, ZCash deserts the Mint and Pour transactions, and instead appends on the basecoin transaction a variable-size list of *JoinSplit*. A JoinSplit plays the role of either a Mint or Pour transaction in the Zerocash paper. For details of ZCash we refer interested readers to [13]. Here we remark that our improvements made in DAP Plus can be applied directly to the implementation of ZCash, and brief the implementation of our modifications.

We implement the distributed signature generation scheme with EC-Schnorr Signature [10]. For computation of pkH, we take SHA256 as the public key hash function Hash. For computation of pkcm with trapdoor a_{sk} (which is 252-bit string in ZCash), we take the SHA256 compression of their concatenation $a_{\mathsf{sk}} \| \mathsf{pkH}$ prefixed by four zero-bits. The lock time is taken as a 64-bit integer. As in ZCash, we abandon the trapdoor s and compute the coin commitment as the SHA256 of the concatenation of all the coin data.

Instantiation of Z-Channel. For the distributed generation of Schnorr keys and signature, since the number of parties is limited to two, we take the following simple procedures: 1) For key generation, Alice generates random big integer a and computes $A = aG$ locally, and Bob generates b and $B = bG$; Alice commits A to Bob, Bob sends B to Alice, and Alice sends A to Bob; finally, the shared public key is $A+B$, and the shared secret key is $a+b$. 2) For signature generation, they first run a key generation procedure to agree on $K = k_1G + k_2G$, and Alice computes signature share by $e = H(x_K \| M)$, $s_1 = k_1 - ae$, $\sigma_1 = (e, s_1)$, where H is hash function and M is the message to sign; Bob computes σ_2 similarly; finally, one of them sends his/her share to the other as appropriate, and the complete signature is $\sigma = (e, s_1 + s_2)$.

For the consensus of secret seed, we assume that Alice and Bob have a secure communication channel, and take the following procedure: Alice generates random 256-bit string a and Bob generates b; Alice commits a to Bob, Bob sends b to Alice, and Alice sends a to Bob; finally they compute $\mathsf{seed} = a \oplus b$.

For the generation of random strings from seed, we assign each value with a unique tag, and compute the SHA256 of the tag prefixed with seed.

B. Performance of Zero-Knowledge Proof in DAP Plus

Our construction of the circuit for the zk-SNARK for the modified NP statement is based on the code of ZCash. Table II

	#Repeat	Mean	Std	Max	Min
Platform	Ubuntu 16.04 LTS 64 bit on Intel Core i7-5500U @ 2.40 GHz 7.7 GB Memory				
DAP KeyGen Time	5	340.44s	6.2270s	348.15s	333.38s
DAP+ KeyGen Time	5	367.48s	4.3756s	372.48s	362.76s
DAP PK size	465 MB				
DAP+ PK size	516 MB				
DAP VK size	773 B				
DAP+ VK size	932 B				

TABLE II: Performance of Zero-Knowledge Proof: System Setup

	#Repeat	Mean	Std	Max	Min
Platform	Ubuntu 17.04 64 bit on Intel Core i5-4590 @ 3.30 GHz 3.6 GB Memory				
DAP Prove Time	15	98.06s	0.4914s	99.490s	97.558s
DAP+ Prove Time	15	101.22s	2.5206s	107.52s	98.089s
DAP Verify Time	1500	23.43ms	0.509ms	25.4ms	23.3ms
DAP+ Verify Time	1500	23.46ms	0.128ms	26.3ms	23.4ms

TABLE III: Performance of Zero-Knowledge Proof: Proof Generation and Verification

and Table III shows the performance of the zero-knowledge proof procedures, in comparison with that of the original DAP scheme. For accuracy, we benchmark each procedure repeatedly and list the statistics.

The modifications introduced in DAP+ scheme slightly (around 0.1% to 8%) increase the key sizes as well as the time for key generation, proving and verification, which is as expected.

C. Performance of Z-Channel Protocol Between Single Pairs

We test the Z-Channel protocol by running two instances of Z-Channel client on localhost, to minimize the effect of real network latency, and measure the performance of Z-Channel in different network latencies. Our focus is on the time cost in updating the channel, since this is the key in improving the payment efficiency of Zerocash. As for the establishment and closure procedure, the time consumption is dominated by the time for the ledger to confirm the transactions (which on average is several minutes for Zerocash). In the experiment, we only test the computation time, to make sure that the cost brought by the key generation, seed negotiation, etc. is negligible in comparison to the transaction confirmation time.

Table IV shows the time consumption of the subprotocols neglecting network latency and ledger confirmation latency.

Fig.2 shows how network latency affects update time. After the Z-Channel is established, the update can be executed within milliseconds, which is nearly negligible in comparison to the confirmation time of original Zerocash, which is approximately a dozen minutes (in presence of six-block confirmation).

	#Repeat	Mean	Std	Max	Min
Platform	Ubuntu 17.04 64 bit on Intel Core i5-4590 @ 3.30 GHz 3.6 GB Memory				
Update Time	1000	3.778ms	1.238ms	22.5ms	3.467ms
Establish Time	1	26.59ms			
Close Time	1	0.3749ms			

TABLE IV: Performance of Z-Channel

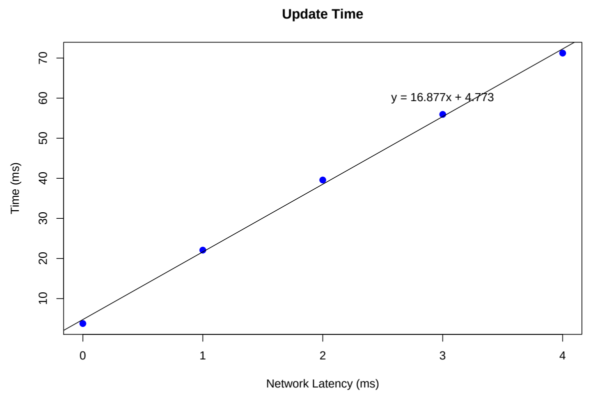


Fig. 2: Relationship between update time of Z-Channel and network latency. Note that the ratio of update time and network latency is close to 16, which is explained by the fact that each distributed signature generation requires four interactions between the parties, and they execute this procedure four times during each update.

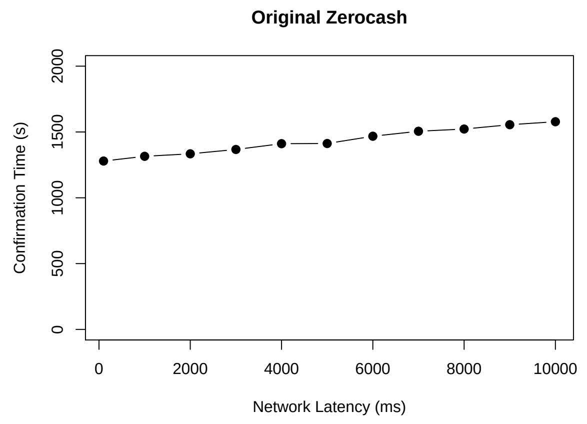
D. Simulation of Z-Channel in Large Network

We simulate a blockchain-based ledger maintained by n users. The users form a social network which is randomly generated such that on average each user is connected with D users. We call a pair of connected users "friends" and a pair of unconnected users "strangers". We assume that payments happen more often between friends than between strangers.

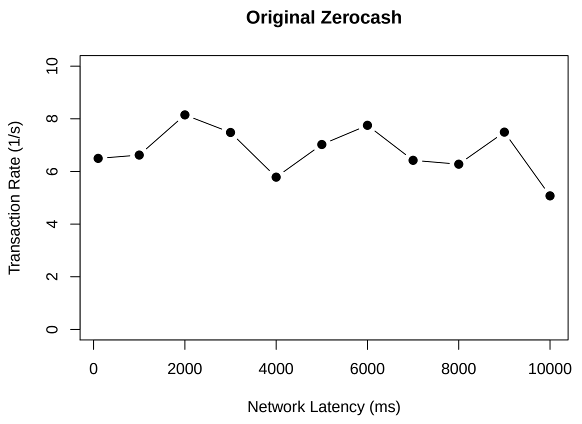
We simulate the payments by a Poisson procedure with parameter λ. For each payment event, the probability that it happens between friends is set to α. If it is between friends, randomly select a pair of connected users, else randomly select a pair of unconnected users. If a payment event happens between friends, and they have not established the Z-Channel, they establish it by issuing three transactions i.e. the two funding transactions and a share transaction. Assume the time for generating a transaction is p, and the time for a transaction to reach the ledger is r. To establish a channel, they insert two transactions in the transaction pool after time $p + r$, and after time $2p + r$ they insert another transaction. If they already have Z-Channel, the payment is immediately confirmed and the confirmation time is set to $16d$, where d is the network latency. If the payment event happens between strangers, they simply issue a transaction which is inserted to the transaction pool after time $p + r$.

We simulate the blockchain by another Poisson procedure with parameter μ. When each block is generated, it selects m transactions in the transaction pool in time order and packs them into a block, or all of them if there are less than m left. If the number of blocks is larger than the confirmation block number k, then each transaction k blocks ago is confirmed along with their corresponding payments. The confirmation time of each payment is set to current time minus the time the payment happens, and plus s which is the average time for a new block to reach a user.

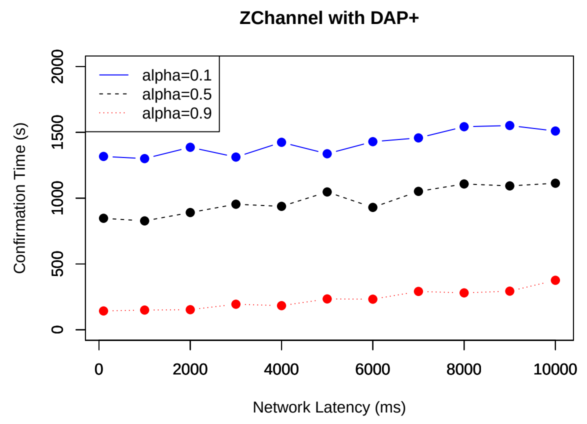
We assume that the payment rate λ is self-adaptive, according to the fact that when the ledger is congested, users will tend to other alternative payment methods, and when the ledger is not fully used, the fast confirmation will attract users back.



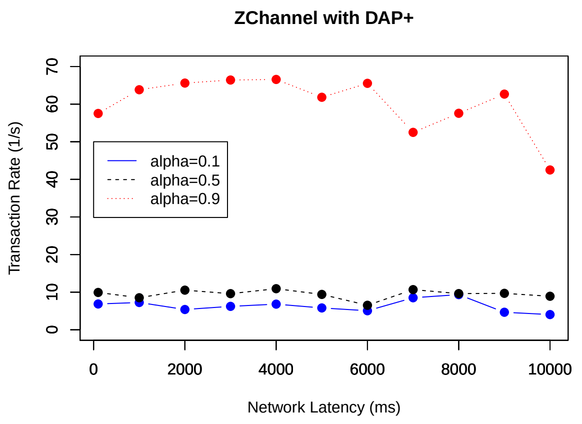
(a) Original Zerocash Confirm Time



(b) Original Zerocash Payment Rate



(c) DAP+ with Z-Channel Confirm Time



(d) DAP+ with Z-Channel Payment Rate

Fig. 3: Simulation result for original Zerocash and the DAP+ scheme with Z-Channel

Parameter	Value	Parameters	Value	Parameters	Value
n	100000	D	10	k	6
p (for DAP+)	101.22s	r	$5 \times d$	s	$27 \times d$
p (for DAP)	98.06s	μ	1/150s	N	900000
m	1500				

[1] The proving time p is set according to benchmark result of DAP and DAP+;
[2] the block volumne m is estimated according to online explorer of ZCash;
[3] the latency r and s is estimated according to the efficiency analysis of [25].

TABLE V: Simulation Parameters

Finally, λ will converge to a stable point, which is the amount of payments the ledger could provide, and is used to measure the scalability of the ledger. We measure the confirmation time of the ledger by taking the average confirmation time of the last N confirmed payments after λ converges.

In the simulation, we measure the payment rates and confirmation time under different network latencies, and different values of α measuring the tendency of conducting payments with friends. Other parameters are listed in Table V.

For comparison, we run the simulation for the original Zerocash, where no Z-Channel is established, and the social network is neglected. For Z-Channel, we run the simulation with different α, representing different social networks where users have different tendency to financially interact with friends or strangers. The simulation results are summarized in Fig.3.

We see that when α is as low as 0.1, the payment rates and confirmation time are close to those of original Zerocash. With higher α, the confirmation time drops significantly. When α is set to 0.9, i.e. only 10% of the payments are conducted outside Z-Channel, the average confirmation time also drops to 10% of that of original Zerocash, while the payment rate the ledger is able to handle is 10 times higher.

VII. Conclusion

We develop Z-Channel, a micropayment channel scheme over Zerocash. In particular, we improve the original DAP scheme of Zerocash and propose DAP Plus, which supports multisignature and lock time functionalities that are essential in implementing micropayment channels. We then construct the Z-Channel protocol, which allows numerous payments conducted and confirmed off-chain in short periods of time. The privacy protection provided by Z-Channel ensures that the identities of the parties and the balances of the channels and even the existence of the channel are kept secret. Finally, we implement Z-Channel protocol, and our experiments demonstrate that Z-Channel significantly improves the scalability and reduces the average payment time of Zerocash.

References

[1] G. Andresen, "Blocksize economics. bitcoinfoundation.org," 2014.

[2] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for c: Verifying program executions succinctly and in zero knowledge," in *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 90–108.

[3] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive arguments for a von neumann architecture," *IACR Cryptology ePrint Archive*, vol. 2013, p. 879, 2013.

[4] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 104–121.

[5] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten, "Mixcoin: Anonymity for bitcoin with accountable mixes," in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 486–504.

[6] G. Danezis, C. Fournet, M. Kohlweiss, and B. Parno, "Pinocchio coin: building zerocoin from a succinct pairing-based proof system," in *Proceedings of the First ACM workshop on Language support for privacy-enhancing technologies*. ACM, 2013, pp. 27–30.

[7] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, 2016, pp. 45–59.

[8] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 281–310.

[9] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 626–645.

[10] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 295–310.

[11] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," Cryptology ePrint Archive, Report 2016/701, 2016, http://eprint.iacr.org/2016/701.

[12] E. Heilman, F. Baldimtsi, L. Alshenibr, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted tumbler for bitcoin-compatible anonymous payments." *IACR Cryptology ePrint Archive*, vol. 2016, p. 575, 2016.

[13] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash protocol specification," 2017. [Online]. Available: https://github.com/zcash/zips/raw/master/protocol/protocol.pdf

[14] T. Jedusor, "Mimblewimble, 2016, defunct hidden service." [Online]. Available: https://scalingbitcoin.org/papers/mimblewimble.txt

[15] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," *self-published paper, August*, vol. 19, 2012.

[16] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 839–858.

[17] J. A. Kroll, I. C. Davey, and E. W. Felten, "The economics of bitcoin mining, or bitcoin in the presence of adversaries," in *Proceedings of WEIS*, vol. 2013. Citeseer, 2013.

[18] G. Maxwell, "Coinjoin: Bitcoin privacy for the real world," in *Post on Bitcoin Forum*, 2013.

[19] ——, "Coinswap: Transaction graph disjoint trustless trading," *CoinSwap: Transactiongraphdisjointtrustlesstrading*, 2013.

[20] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 2013, pp. 397–411.

[21] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[22] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[23] F. Reid and M. Harrigan, "An analysis of anonymity in the bitcoin system," in *Security and privacy in social networks*. Springer, 2013, pp. 197–223.

[24] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *European Symposium on Research in Computer Security*. Springer, 2014, pp. 345–364.

[25] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *IEEE Symposium on Security and Privacy*, 2014, pp. 459–474.

[26] Y. Sompolinsky and A. Zohar, "Accelerating bitcoin's transaction processing. fast money grows on trees, not chains." *IACR Cryptology ePrint Archive*, vol. 2013, no. 881, 2013.

[27] L. Valenta and B. Rowan, "Blindcoin: Blinded, accountable mixes for bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 112–126.

[28] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, 2014.

[29] J. H. Ziegeldorf, F. Grossmann, M. Henze, N. Inden, and K. Wehrle, "Coinparty: Secure multi-party mixing of bitcoins," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*. ACM, 2015, pp. 75–86.

Appendix

A. Definition of Security

We define the completeness, ledger indistinguishability, transaction non-malleability and balance in a way similar to definitions B.1, C.1 C.2 and C.3 in [25].

Definition 7: We say that a DAP+ scheme Π = (Setup, CreatePKCM, CreateAddress, MintPlus, PourPlus, VerifyPlus, ReceivePlus) is complete, if for every poly(λ)-size adversary $\mathcal{A}$ and sufficiently large λ, $\mathbf{Adv}^{\mathsf{INCOMP}}_{\Pi,\mathcal{A}}(\lambda) < \text{negl}(\lambda)$, where $\mathbf{Adv}^{\mathsf{INCOMP}}_{\Pi,\mathcal{A}}(\lambda) := 2\cdot Pr[\mathsf{INCOMP}\ (\Pi,\mathcal{A},\lambda) = 1] - 1$ is $\mathcal{A}$'s advantage in the INCOMP experiment.

Definition 8: We say that a DAP+ scheme Π = (Setup, CreatePKCM, CreateAddress, MintPlus, PourPlus, VerifyPlus, ReceivePlus) is $\mathsf{L-IND}$ secure, if for every poly(λ)-size adversary $\mathcal{A}$ and sufficiently large λ, $\mathbf{Adv}^{\mathsf{L-IND}}_{\Pi,\mathcal{A}}(\lambda) < \text{negl}(\lambda)$, where $\mathbf{Adv}^{\mathsf{L-IND}}_{\Pi,\mathcal{A}}(\lambda) := 2 \cdot Pr[\mathsf{L-IND}(\ \Pi,\mathcal{A},\lambda) = 1] - 1$ is $\mathcal{A}$'s advantage in the $\mathsf{L-IND}$ experiment.

Definition 9: We say that a DAP+ scheme Π = (Setup, CreatePKCM, CreateAddress, MintPlus, PourPlus, VerifyPlus, ReceivePlus) is $\mathsf{TR-NM}$ secure, if for every poly(λ)-size adversary $\mathcal{A}$ and sufficiently large λ, $\mathbf{Adv}^{\mathsf{TR-NM}}_{\Pi,\mathcal{A}}(\lambda) < \text{negl}(\lambda)$, where $\mathbf{Adv}^{\mathsf{TR-NM}}_{\Pi,\mathcal{A}}(\lambda) := 2 \cdot Pr[\mathsf{TR-NM}\ (\Pi,\mathcal{A},\lambda) = 1] - 1$ is $\mathcal{A}$'s advantage in the $\mathsf{TR-NM}$ experiment.

Definition 10: We say that a DAP+ scheme Π = (Setup, CreatePKCM, CreateAddress, MintPlus, PourPlus, VerifyPlus, ReceivePlus) is BAL secure, if for every poly(λ)-size adversary $\mathcal{A}$ and sufficiently large λ, $\mathbf{Adv}^{\mathsf{BAL}}_{\Pi,\mathcal{A}}(\lambda) <$

negl(λ), where $\mathbf{Adv}^{\mathsf{BAL}}_{\Pi,\mathcal{A}}(\lambda) := 2 \cdot Pr[\mathsf{BAL}\ (\Pi, \mathcal{A}, \lambda) = 1] - 1$ is $\mathcal{A}$'s advantage in the BAL experiment.

In each of the experiments, one or more oracles of the DAP scheme $\mathcal{O}^{DAP+}$ receives queries and output answers. A challenger $\mathcal{C}$ interacts with an adversary $\mathcal{A}$, forwards the queries from $\mathcal{A}$ to $\mathcal{O}^{DAP+}$ and the answers back to $\mathcal{A}$, and performs sanity checks. We modify the mechanism of the original $\mathcal{O}^{DAP+}$ in [25] to suit our new DAP+ scheme. Below, we first describe how this new oracle $\mathcal{O}^{DAP+}$ works.

The oracle $\mathcal{O}^{DAP+}$ is initialized by a list of public parameters pp and maintains state. Internally, $\mathcal{O}^{DAP+}$ stores the following:

(i) L, a ledger;
(ii) ADDR, a set of address key pairs;
(iii) COIN, a set of coins;
(iv) PKCM, a set of tuples of $(\mathsf{sk}, \mathsf{pkL}, \mathsf{pkcm})$;
(v) OLDPKCM, a set of tuples of $(\mathsf{addr_{pk}}, \mathsf{pkcm}, \mathsf{pkL})$.

Initially, L, ADDR, COIN, PKCM, OLDPKCM start out empty. The oracle $\mathcal{O}^{DAP+}$ accepts various types of queries, and each type of query modifes L, ADDR, COIN, PKCM, OLDPKCM in different ways and outputs differently. We now describe each type of query Q.

Q = (CreateAddress)

1) Compute ($\mathsf{addr_{pk}}$, $\mathsf{addr_{sk}}$) := CreateAddress(pp).
2) Add the address key pair ($\mathsf{addr_{pk}}$, $\mathsf{addr_{sk}}$) to ADDR.
3) Output the address public key $\mathsf{addr_{pk}}$.

Other internal storages apart from ADDR stay unchanged.

Q = (CreatePKCM,$\mathsf{addr_{pk}}$,K)

1) Randomly sample u.
2) Randomly sample a public key list pkL (with secret key list being $sklist$) of size K.
3) Compute $\mathsf{pkcm} = \mathsf{COMM}_u(\mathsf{Hash}(\mathsf{pkL}))$.
4) Store $(\mathsf{sk}, \mathsf{pkL}, \mathsf{pkcm})$ in table PKCM.
5) Output pkcm.

Other internal storages apart from PKCM stay unchanged.

Q = (Mint, v, $\mathsf{addr_{pk}}$, pkcm, tL)

1) Compute ($\mathbf{c}$, $\mathsf{tx_{Mint}}$) := Mint(pp, v, $\mathsf{addr_{pk}}$, tL).
2) Add the coin $\mathbf{c}$ to COIN.
3) If $\mathsf{addr_{pk}}$ is in ADDR, find tuple $(\mathsf{sk}, \mathsf{pkL}, \mathsf{pkcm})$ in table PKCM, aborts if cannot find, then removes the tuple from PKCM and stores $(\mathsf{addr_{pk}}, \mathsf{sk}, \mathsf{pkL}, \mathsf{pkcm})$ in OLDPKCM;
4) If $\mathsf{addr_{pk}}$ is not in ADDR, but pkcm can be found in PKCM or OLDPKCM, aborts;
5) Add the mint transaction $\mathsf{tx_{Mint}}$ to L.
6) Output $\perp$.

The internal storage ADDR stay unchanged.

Q = (Pour, $\mathsf{idx}^{\mathsf{old}}_{1..2}$, $\mathsf{addr}^{\mathsf{old}}_{\mathsf{pk},1..2}$, $\mathsf{ovd}_{1..2}$, $v^{\mathsf{new}}_{1..2}$, $\mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},1..2}$, $\mathsf{lock}^{\mathsf{new}}_{1..2}$, v_{pub})

1) Let MBH be the current block height.
2) For each $i \in \{1, 2\}$:
   a) Let $\mathsf{cm}^{\mathsf{old}}_i$ be the $\mathsf{idx}^{\mathsf{old}}_i$-th coin commitment in L.
   b) Let tx_i be the mint/pour transaction in L that contains $\mathsf{cm}^{\mathsf{old}}_i$.
   c) Let $\mathbf{c}^{\mathsf{old}}_i$ be the first coin in COIN with coin commitment $\mathsf{cm}^{\mathsf{old}}_i$.
   d) Let $\mathsf{pkcm}^{\mathsf{old}}_i$ be the public key commitment stored in $\mathbf{c}^{\mathsf{old}}_i$.
   e) Let $(\mathsf{addr}^{\mathsf{old}}_{\mathsf{pk},i}, \mathsf{sk}_i, \mathsf{pkL}^{\mathsf{old}}_i, \mathsf{pkcm}^{\mathsf{old}}_i)$ be the first tuple in OLDPKCM with public key commitment $\mathsf{pkcm}^{\mathsf{old}}_i$.
   f) Let ($\mathsf{addr}^{\mathsf{old}}_{\mathsf{pk},i}$,$\mathsf{addr}^{\mathsf{old}}_{\mathsf{sk},i}$) be the first key pair in ADDR with $\mathsf{addr}^{\mathsf{old}}_{\mathsf{pk},i}$ being $\mathbf{c}^{\mathsf{old}}_i$'s address.
   g) Let tL_i be the lock time stored in $\mathbf{c}^{\mathsf{old}}_i$.
   h) If ovd_i is false, let rt_i be the a randomly selected root in the Merkle tree root history later than $\mathsf{cm}^{\mathsf{old}}_i$ in L such that $\mathsf{BH}(rt_i) + \mathsf{tL}_i < \mathsf{MBH}$.
   i) If ovd_i is true, let rt_i be the a randomly selected root in the Merkle tree root history.
   j) Compute path_i, the authentication path from $\mathsf{cm}^{\mathsf{old}}_i$ to rt_i.
   k) If $\mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},i}$ is in ADDR, checks that $\mathsf{pkcm}^{\mathsf{new}}_i$ is in PKCM and not in OLDPKCM, and aborts if the check fails. Let $(\mathsf{pkL}^{\mathsf{new}}_i, u^{new}_i, \mathsf{pkcm}^{\mathsf{new}}_i)$ be the tuple found in PKCM. Remove $\mathsf{pkcm}^{\mathsf{new}}_i$ from PKCM and stores $(\mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},i}$, sk_i, $\mathsf{pkL}^{\mathsf{new}}_i$, $\mathsf{pkcm}^{\mathsf{new}}_i)$ in OLDPKCM.
   l) If $\mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},i}$ is not in ADDR, checks that $\mathsf{pkcm}^{\mathsf{new}}_i$ is not in either PKCM or OLDPKCM, and aborts if the check fails.
3) Compute ($\mathbf{c}^{\mathsf{new}}_1$, $\mathbf{c}^{\mathsf{new}}_2$, $\mathsf{tx_{Pour}}$) := Pour(pp, v_{pub}, $\mathbf{c}^{\mathsf{old}}_{1..2}$, $\mathsf{addr}^{\mathsf{old}}_{\mathsf{sk},1..2}$, $\mathsf{rt}_{1..2}$, $\mathsf{path}_{1..2}$, $\mathsf{pkL}^{\mathsf{old}}_{1..2}$, $\mathsf{sk}_{1..2}$, $\mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},1..2}$, $v^{\mathsf{new}}_{1..2}$, $\mathsf{lock}^{\mathsf{new}}_{1..2}$).
4) Verify that Verify(pp, $\mathsf{tx_{Pour}}$, L) outputs 1.
5) Add the coins $\mathbf{c}^{\mathsf{new}}_{1..2}$ to COIN.
6) Add the pour transaction $\mathsf{tx_{Pour}}$ to L.
7) Output $\perp$.

If any of the above operations fail, the output is $\perp$ (and L, ADDR, COIN, PKCM, OLDPKCM remain unchanged).

Q = (Insert, tx)

1) Verify that Verify(pp, tx, L) outputs 1. (Else, abort.)
2) Add the mint/pour transaction tx to L.
3) Run ReceivePlus for all addresses $\mathsf{addr_{pk}}$ in ADDR;
4) For each output coin from ReceivePlus
   a) Let pkcm be the public key commitment stored in it.
   b) Let $(\mathsf{sk}, \mathsf{pkL}, \mathsf{pkcm})$ be the first tuple in PKCM with the public key commitment pkcm (if not exists, aborts).
   c) Remove this tuple from PKCM;
   d) Add $(\mathsf{addr_{pk}}, \mathsf{sk}, \mathsf{pkL}, \mathsf{pkcm})$ to OLDPKCM.
5) Output $\perp$.

The address set ADDR stays unchanged.

With the above described oracle $\mathcal{O}^{DAP+}$, the definitions of ledger indistinguishability, transaction non-malleability and balance are defined by three games respectively: $\mathsf{L-IND}$, $\mathsf{TR-NM}$ and BAL. We now describe the above mentioned $\mathsf{L-IND}$ experiment. The other experiments $\mathsf{TR-NM}$ and BAL are similar to the original ones, refer to [25] for the details.

Given a DAP+ scheme Π, adversary $\mathcal{A}$, and security parameter λ, the (probabilistic) experiment $\mathsf{L-IND}(\Pi, \mathcal{A}, \lambda)$ consists of a series of interactions between $\mathcal{A}$ and a challenger

$\mathcal{C}$. At the end of this experiment, $\mathcal{C}$ outputs a bit in $\{0,1\}$ indicating whether $\mathcal{A}$ succeeds.

At the start of the experiment, $\mathcal{C}$ samples $b \in \{0,1\}$ at random, samples pp $\leftarrow$Setup(1^λ), and sends pp to $\mathcal{A}$; using pp, $\mathcal{C}$ initializes two DAP+ oracles $\mathcal{O}_0^{DAP+}$ and $\mathcal{O}_1^{DAP+}$.

Now $\mathcal{A}$ and $\mathcal{C}$ start interaction in steps. In each step, $\mathcal{C}$ provides to $\mathcal{A}$ two ledgers (L_{left}, L_{right}), where $L_{\mathsf{left}} := L_b$ is the current ledger in $\mathcal{O}_b^{DAP+}$ and $L_{\mathsf{right}} := L_{1-b}$ the ledger in $\mathcal{O}_{1-b}^{DAP+}$; then $\mathcal{A}$ sends to $\mathcal{C}$ a pair of queries (Q, Q'), which must be of the same type of query. $\mathcal{C}$ acts differently on differnt types of queries, as follows:

1) If the query is of type Insert, $\mathcal{C}$ forwards Q to $\mathcal{O}_b^{DAP+}$, and Q' to $\mathcal{O}_{1-b}^{DAP+}$. If the inserted query is a Pour query with one of the target address $\mathsf{addr_{pk}}$ in ADDR, the public key commitment pkcm committed in the coin must not be one generated by CreatePKCM previously.
2) For the other query types, $\mathcal{C}$ ensures that Q, Q' are *publicly consistent*, and then forwards Q to $\mathcal{O}_0^{DAP+}$, and Q' to $\mathcal{O}_1^{DAP+}$; assume the two oracle answer (a_0, a_1), $\mathcal{C}$ forwards to $\mathcal{A}$ (a_b, a_{1-b}).

At the end, $\mathcal{A}$ sends $\mathcal{C}$ a guess $b' \in \{0,1\}$. If $b = b'$, $\mathcal{C}$ outputs 1; else, $\mathcal{C}$ outputs 0.

Public consistency. As mentioned above, the pairs of queries $\mathcal{A}$ sends $\mathcal{C}$ must be of the same type and publicly consistent. We now define the public consistency. If Q, Q' are of type CreateAddress, the queries are automatically public consistent; further more, we require that in this case the address generated in both oracles are identity. If they are of type CreatePKCM, the queries are automatically public consistent. If they are of type Mint, then the minted value v in Q must equal the value in Q'. Finally, if they are Pour query, we require the following restrictions.

First, each of Q, Q' must be well-formed:

(i) the coins $\mathbf{c}_1^{\mathsf{old}}$, $\mathbf{c}_2^{\mathsf{old}}$ corresponding to the coin commitments (reference by the two indices $\mathsf{idx}_1^{\mathsf{old}}$, $\mathsf{idx}_2^{\mathsf{old}}$) in Q must appear in the coin table COIN, similar requirement for Q';
(ii) the coins $\mathbf{c}_1^{\mathsf{old}}$, $\mathbf{c}_2^{\mathsf{old}}$ referenced in Q must be unspent, similar requirement for Q';
(iii) the address public keys $\mathsf{addr_{pk,1}}$ and $\mathsf{addr_{pk,2}}$ in Q must match those in $\mathbf{c}_1^{\mathsf{old}}$, $\mathbf{c}_2^{\mathsf{old}}$, similar requirement for Q';
(iv) the balance equations must hold;
(v) the lock times of the old coins must be up, if not overriden;
(vi) the public key commitments pkcm_i must be one generated by PKCM previously and never used in previous queries and each must be unique in these queries Q and Q'.

Furthermore, Q, Q' must be consistent with respect to public information and $\mathcal{A}$'s view:

(i) the public values in Q and Q' must equal;
(ii) for each $i \in \{1,2\}$, if the i-th recipient addresses in Q is not in ADDR, then v_i^{new} in Q and Q' must equal (vice versa for Q');
(iii) for each $i \in \{1,2\}$, the i-th overriding flag ovd_i in Q must equal the corresponding flag in Q';
(iv) for each $i \in \{1,2\}$, if the i-th index in Q references a coin commitment in a transaction from a previously posted Insert query, then the corresponding index in Q' must also reference a coin commitment in a transaction posted in Insert query; additionally, v_i^{old} in Q and Q' must equal (vice versa for Q').

B. Proof of Security

Here we present the complete proof of Theorem 5.4. The proofs to transaction non-malleability and balance are trivially similar to the ones in [25], we omit them here. For proof of ledger indistingsuishability, we construct a simulation $\eth_{\mathbf{sim}}$ in which the adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$, as in the $\mathsf{L-IND}$ experiment. However $\eth_{\mathbf{sim}}$ modifies the $\mathsf{L-IND}$ experiment in a critical way: all answers sent by $\mathcal{C}$ to $\mathcal{A}$ are independent from the bit b, so the advantage of $\mathcal{A}$'s in $\eth_{\mathbf{sim}}$ is 0. Then we show that $\mathbf{Adv}_{\Pi,\mathcal{A}}^{\mathsf{L-IND}}(\lambda)$ is only negligibly larger than $\mathcal{A}$'s advantage in $\eth_{\mathbf{sim}}$.

The simulation experiment. The simulation $\eth_{\mathbf{sim}}$ works as follows. First, $\mathcal{C}$ samples $b \in \{0,1\}$ and pp $\leftarrow$ Setup(1^λ), with the following modifications: the zk-SNARK keys are generated by ($\mathsf{pk_{POUR}}$, $\mathsf{vk_{POUR}}$, trap)$\leftarrow$ Sim$(1^\lambda$, $C_{\mathsf{POUR}})$, instead of the usual way. Then, $\mathcal{C}$ sends pp to $\mathcal{A}$, and initializes two DAP+ oracles $\mathcal{O}_0^{DAP+}$ and $\mathcal{O}_1^{DAP+}$.

Afterwards, $\eth_{\mathbf{sim}}$ proceeds in steps and at each step $\mathcal{C}$ present $\mathcal{A}$ two ledgers (L_{left}, L_{right}), where $L_{\mathsf{left}} := L_b$ is the current ledger in $\mathcal{O}_b^{DAP+}$ and $L_{\mathsf{right}} := L_{1-b}$ the ledger in $\mathcal{O}_{1-b}^{DAP+}$; then $\mathcal{A}$ sends to $\mathcal{C}$ a message (Q, Q'), which consist of two queries of the same type. The requirement to these two queries is the same to that in $\mathsf{L-IND}$. The reaction of challenger $\mathcal{C}$ is different from that in $\mathsf{L-IND}$, as described as follows:

1) **Answering** CreateAddress **queries**. In this case, $Q = Q'$ = CreateAddress. To answer Q, $\mathcal{C}$ behaves as in $\mathsf{L-IND}$, except for the following modification: after obtaining ($\mathsf{addr_{pk}}$, $\mathsf{addr_{sk}}$)$\leftarrow$CreateAddress(pp), $\mathcal{C}$ replaces a_{pk} in $\mathsf{addr_{pk}}$ with a random string of the appropriate length; then, $\mathcal{C}$ stores ($\mathsf{addr_{pk}}$,$\mathsf{addr_{sk}}$) in ADDR and returns $\mathsf{addr_{pk}}$ to $\mathcal{A}$. Afterwards, $\mathcal{C}$ does the same for Q'.
2) **Answering** CreatePKCM **queries.** In this case, $Q = Q'$ = CreatePKCM. To answer Q, $\mathcal{C}$ behaves as in $\mathsf{L-IND}$, except for the following modification: after obtaining (sk, pkL, pkcm), $\mathcal{C}$ replaces pkcm with a random string of the appropriate length; then, $\mathcal{C}$ stores the tuple in PKCM and returns pkcm to $\mathcal{A}$. Afterwards, $\mathcal{C}$ does the same for Q'.
3) **Answering** Mint **queries**. In this case, Q = (Mint, v, $\mathsf{addr_{pk}}$) and Q' = (Mint, v, $\mathsf{addr'_{pk}}$). To answer Q, $\mathcal{C}$ behaves as in $\mathsf{L-IND}$, except for the following modification: Compute $m = \mathsf{COMM}_r(\tau)$ for a random string τ of the suitable length, instead of $m = \mathsf{COMM}_r(a_{\mathsf{pk}}, \rho, \mathsf{pkL}, \mathsf{tL})$. Afterwards, $\mathcal{C}$ does the same for Q'.
4) **Answering** Pour **queries**. In this case, Q and Q' both have the form (Pour, $\mathsf{idx}_{1..2}^{\mathsf{old}}$, $\mathsf{addr}_{\mathsf{pk},1..2}^{\mathsf{old}}$, $\mathsf{ovd}_{1..2}$, $v_{1..2}^{\mathsf{new}}$, $\mathsf{addr}_{\mathsf{pk},1..2}^{\mathsf{new}}$, $\mathsf{lock}_{1..2}^{\mathsf{new}}$, v_{pub}). To answer Q, $\mathcal{C}$ modifies in the following ways:
   a) For each $j \in \{1,2\}$:
      i) Uniformly sample random $\mathsf{sn}_j^{\mathsf{old}}$.

ii) Randomly sample a list of pairs of public/private keys pkL_j, compute $\mathsf{pkH}_j^{\mathsf{old}} := \mathsf{Hash}(\mathsf{pkL}_j)$.

iii) If $\mathsf{addr}_{\mathsf{pk},j}^{\mathsf{new}}$ is in ADDR:

A) sample a coin commitment $\mathsf{cm}_j^{\mathsf{new}}$ on a random input;

B) run $\mathcal{K}_{\mathsf{enc}}(\mathsf{pp}_{\mathsf{enc}}) \to (\mathsf{pk}_{\mathsf{enc}}, \mathsf{sk}_{\mathsf{enc}})$ and compute $\mathbf{C}_j^{\mathsf{new}} := \mathcal{E}_{\mathsf{enc}}(\mathsf{pk}_{\mathsf{enc}}, r)$ for a random r of suitable length.

iv) Otherwise, calculate ($\mathsf{cm}_j^{\mathsf{new}}$, $\mathbf{C}_j^{\mathsf{new}}$) as in the Pour algorithm.

b) Set h_1 and h_2 to be random strings of suitable length.

c) Compute all other values as in the Pour algorithm.

d) The pour proof is computed as π_{POUR}:=Sim(trap, $\vec{x}$), where $\vec{x}$:= ($\mathsf{rt}_{1..2}$, $\mathsf{sn}_{1..2}^{\mathsf{old}}$, $\mathsf{pkH}_{1..2}^{\mathsf{old}}$, $\mathsf{cm}_{1..2}^{\mathsf{new}}$, v_{pub}, h_{sig}, $h_{1..2}$, MBH, $\mathsf{ovd}_{1..2}$).

Afterwards, $\mathcal{C}$ does the same for Q'.

5) Answering Insert queries. In this case, Q = (Insert, tx) and Q = (Insert, tx$'$). The answer to each query proceeds as in the L − IND experiment.

In each of the above cases, the response to $\mathcal{A}$ is computed independently of the bit b. Thus, when $\mathcal{A}$ outputs a guess b', it must be the case that $Pr[b = b'] = 1/2$, i.e., $\mathcal{A}$'s advantage in $\eth_{\mathbf{sim}}$ is 0.

Indistinguishability from Real Experiment.

We construct a sequence of hybrid experiments ($\eth_{\mathbf{real}}$, $\eth_1$, $\eth_2$, $\eth_3$, $\eth_{\mathbf{sim}}$), in each of these experiments a challenger $\mathcal{C}$ conducts a different modification of the L − IND experiment. We define $\eth_{\mathbf{real}}$ to be the original L − IND experiment, and $\eth_{\mathbf{sim}}$ to be the simulation described above. Given experiment $\eth$, we define $\mathbf{Adv}^{\eth}$ to be the absolute value of the difference between the L − IND advantage of $\mathcal{A}$ in $\eth$ and that in $\eth_{\mathbf{real}}$. Also, let

1) $q_{\mathbf{CA}}$ be the number of CreateAddress queries issued by $\mathcal{A}$,
2) $q_{\mathbf{CP}}$ be the number of CreatePKCM queries issued by $\mathcal{A}$.
3) $q_{\mathbf{P}}$ be the number of Pour queries issued by $\mathcal{A}$,
4) $q_{\mathbf{M}}$ be the number of Mint queries issued by $\mathcal{A}$,

Finally, define $\mathbf{Adv}^{\mathbf{Enc}}$ to be $\mathcal{A}$'s advantage in **Enc**'s IND-CCA and IK-CCA experiments, $\mathbf{Adv}^{\mathsf{PRF}}$ to be $\mathcal{A}$'s advantage in distinguishing the pseudorandom function PRF from a random one, and $\mathbf{Adv}^{\mathsf{COMM}}$ to be $\mathcal{A}$'s advantage against the hiding property of COMM.

We now describe each of the hybrid experiments.

1) Experiment $\eth_1$. The experiment $\eth_1$ modifies $\eth_{\mathbf{real}}$ by simulating the zk-SNARKs. More precisely, we modify $\eth_{\mathbf{real}}$ so that $\mathcal{C}$ simulates each zk-SNARK proof, as follows. At the beginning of the experiment, instead of invoking KeyGen(1^λ, C_{POUR}), $\mathcal{C}$ invokes Sim(1^λ, C_{POUR}) and obtains ($\mathsf{pk}_{\mathsf{POUR}}$, $\mathsf{vk}_{\mathsf{POUR}}$, trap). At each subsequent invocation of the Pour algorithm, $\mathcal{C}$ computes π_{POUR} ←Sim(trap, x), without using any witnesses, instead of using Prove. Since the zk-SNARK system is perfect zero knowledge, the distribution of the simulated π_{POUR} is identical to that of the proofs computed in $\eth_{\mathbf{real}}$. Hence $\mathbf{Adv}^{\eth_1} = 0$.

2) Experiment $\eth_2$. The experiment $\eth_2$ modifies $\eth_1$ by replacing the ciphertexts in a pour transaction by encryptions of random strings. Each time $\mathcal{A}$ issues a Pour query where one of ($\mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}$, $\mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}$) is in ADDR, the ciphertexts $\mathbf{C}_1^{\mathsf{new}}$, $\mathbf{C}_2^{\mathsf{new}}$ are generated as follows:

a) $(\mathsf{pk}_{\mathsf{enc}}^{\mathsf{new}}, \mathsf{sk}_{\mathsf{enc}}^{\mathsf{new}}) \leftarrow \mathcal{K}_{\mathsf{enc}}(\mathsf{pp}_{\mathsf{enc}})$;

b) for each $j \in \{1, 2\}$, $\mathbf{C}_j^{\mathsf{new}} := \mathcal{E}_{\mathsf{enc}}(\mathsf{pk}_{\mathsf{enc}}^{\mathsf{new}}, j, r)$ where r is a message randomly and uniformly sampled from plaintext space.

By Lemma A.1, $\left|\mathbf{Adv}^{\eth_2} - \mathbf{Adv}^{\eth_1}\right| \leq 4 \cdot q_{\mathbf{P}} \cdot \mathbf{Adv}^{\mathbf{Enc}}$.

3) Experiment $\eth_3$. The experiment $\eth_3$ modifies $\eth_2$ by replacing all PRF-generated values with random strings:

a) each time $\mathcal{A}$ issues a CreateAddress query, the value a_{pk} within the returned $\mathsf{addr}_{\mathsf{pk}}$ is substituted with a random string of the same length;

b) each time $\mathcal{A}$ issues a Pour query, each of the serial numbers $\mathsf{sn}_1^{\mathsf{old}}$, $\mathsf{sn}_2^{\mathsf{old}}$ in $\mathsf{tx}_{\mathsf{Pour}}$ is substituted with a random string of the same length, and h_1 and h_2 with random strings of the same length.

By Lemma A.2, $\left|\mathbf{Adv}^{\eth_3} - \mathbf{Adv}^{\eth_2}\right| \leq q_{\mathbf{CA}} \cdot \mathbf{Adv}^{\mathsf{PRF}}$

4) Experiment $\eth_{\mathbf{sim}}$. The experiment $\eth_{\mathbf{sim}}$ is already described above. For comparison, we explain how it differs from $\eth_3$: all the commitments are replaced with commitments to random inputs:

a) each time $\mathcal{A}$ issues a CreatePKCM query, the commitment pkcm is substituted with a random string of suitable length; and

b) each time $\mathcal{A}$ issues a Mint query, the coin commitment cm in $\mathsf{tx}_{\mathsf{Mint}}$ is substituted with a commitment to a random input; and

c) each time $\mathcal{A}$ issues a Pour query, for each $j \in \{1, 2\}$, if the output address $\mathsf{addr}_{\mathsf{pk},j}^{\mathsf{new}}$ is in ADDR, $\mathsf{cm}_j^{\mathsf{new}}$ is substituted with a commitment to a random input.

By Lemma A.3, $\left|\mathbf{Adv}^{\eth_{\mathbf{sim}}} - \mathbf{Adv}^{\eth_3}\right| \leq (q_{\mathbf{M}} + 4 \cdot q_{\mathbf{P}} + q_{\mathbf{CP}}) \cdot \mathbf{Adv}^{\mathsf{COMM}}$

By summing over $\mathcal{A}$'s advantages in the hybrid experiments, we can bound $\mathcal{A}$'s advantage in $\eth_{\mathbf{real}}$ by

$$\begin{aligned}\mathbf{Adv}_{\Pi,\mathcal{A}}^{\mathsf{L-IND}}(\lambda) \quad \leq \quad & 4 \cdot q_{\mathbf{P}} \cdot \mathbf{Adv}^{\mathbf{Enc}} + q_{\mathbf{CA}} \cdot \mathbf{Adv}^{\mathsf{PRF}} + \\ & (q_{\mathbf{M}} + 4 \cdot q_{\mathbf{P}} + q_{\mathbf{CP}}) \cdot \mathbf{Adv}^{\mathsf{COMM}}\end{aligned}$$

which is negligible in λ. This concludes the proof of ledger indistinguishability.

Below, we sketch proofs for the lemmas used above.

Lemma A.1: Let AdvEnc be the maximum of: $\mathcal{A}$'s advantage in the IND-CCA experiment against the encryption scheme **Enc**, and $\mathcal{A}$'s advantage in the IK-CCA experiment against the encryption scheme **Enc**. Then after $q_{\mathbf{P}}$ Pour queries, $\left|\mathbf{Adv}^{\eth_2} - \mathbf{Adv}^{\eth_1}\right| \leq 4 \cdot q_{\mathbf{P}} \cdot \mathbf{Adv}^{\mathbf{Enc}}$.

The proof of Lemma A.1 is exactly the same to the proof of Lemma D.1 in [25], so we omit it here.

Lemma A.2: Let $\mathbf{Adv}^{\mathsf{PRF}}$ be $\mathcal{A}$'s advantage in distinguishing the pseudorandom function PRF from a random function. Then, after $q_{\mathbf{CA}}$ CreateAddress queries,
$\left|\mathbf{Adv}^{\eth_3} - \mathbf{Adv}^{\eth_2}\right| \leq q_{\mathbf{CA}} \cdot \mathbf{Adv}^{\mathsf{PRF}}$.

Proof sketch. We first constuct a hybrid **H**, intermediate between $\eth_2$ and $\eth_3$, in which we replace all values computed by the first oracle-generated key a_{sk} with random strings. On receiving $\mathcal{A}$'s first $\mathsf{CreateAddress}$ query, replace the public address $\mathsf{addr_{pk}} = (a_{\mathsf{pk}}, \mathsf{pk_{enc}})$ with $\mathsf{addr_{pk}}$ = $(\tau, \mathsf{pk_{enc}})$ where τ is a random string of the appropriate length. On each subsequent Pour query $\mathsf{tx_{Pour}}$, for each $i \in 1, 2$, if $\mathsf{addr}^{\mathsf{old}}_{\mathsf{pk},i} = \mathsf{addr_{pk}}$ then:

1) replace $\mathsf{sn}^{\mathsf{old}}_i$ with a random string of appropriate length;
2) replace each of h_1, h_2 with a random string of appropriate length;
3) simulate the zk-SNARK proof π_{POUR}.

We now argue that $\mathcal{A}$'s advantage in $\mathbf{H}$ is at most $\mathbf{Adv}^{\mathsf{PRF}}$ more than in ∂_2. Let a_{sk} be the secret key generated by the oracle in the first $\mathsf{CreateAddress}$ query. In ∂_2 (as in $\partial_{\mathbf{real}}$):

1) a_{pk} := $\mathsf{PRF}^{\mathsf{addr}}_{a_{\mathsf{sk}}}$ (0);
2) for each $i \in \{1, 2\}$, $\mathsf{sn}_i := \mathsf{PRF}^{\mathsf{sn}}_{a_{\mathsf{sk}}}\rho)$ for a random ρ;
3) for each $i \in \{1, 2\}$, $h_i := \mathsf{PRF}^{\mathsf{pk}}_{a_{\mathsf{sk}}}(i \| h_{\mathsf{sig}})$ and h_{sig} is unique.

Now let $\mathcal{O}$ be an oracle that implements either $\mathsf{PRF}_{a_{\mathsf{sk}}}$ or a random function. We show that if $\mathcal{A}$ distinguishes $\mathbf{H}$ from ∂_2 with probability ϵ, we can construct a distinguisher for the two implementations of $\mathcal{O}$. In fact, when $\mathcal{O}$ implements $\mathsf{PRF}_{a_{\mathsf{sk}}}$, the distribution of the experiment is identical to that of ∂_2; when $\mathcal{O}$ is a random function, the distribution is identical to $\mathbf{H}$. Therefore, $\mathcal{A}$'s advantage is at most $\mathbf{Adv}^{\mathsf{PRF}}$.

Finally, by the hybrid argument, we extend to all $q_{\mathbf{CA}}$ oracle-generated addresses; then, $\mathcal{A}$'s advantage gain from ∂_2 to ∂_3 is at most $q_{\mathbf{CA}} \cdot \mathbf{Adv}^{\mathsf{PRF}}$. The final hybrid is equal to ∂_3, we obtain that $\left|\mathbf{Adv}^{\partial_3} - \mathbf{Adv}^{\partial_2}\right| \leq q_{\mathbf{CA}} \cdot \mathbf{Adv}^{\mathsf{PRF}}$.

Lemma A.3: Let $\mathbf{Adv}^{\mathsf{COMM}}$ be $\mathcal{A}$'s advantage against the hiding property of COMM. After $q_{\mathbf{M}}$ Mint queries, $q_{\mathbf{P}}$ Pour queries and $q_{\mathbf{CP}}$ $\mathsf{CreatePKCM}$ queries, $\left|\mathbf{Adv}^{\partial_{\mathbf{sim}}} - \mathbf{Adv}^{\partial_3}\right| \leq (q_{\mathbf{M}} + 4 \cdot q_{\mathbf{P}} + q_{\mathbf{CP}}) \cdot \mathbf{Adv}^{\mathsf{COMM}}$.

Proof sketch. We only provide a short sketch, because the structure of the argument is similar to the one used to prove Lemma A.2 above.

For the first Mint or Pour query, replace the "internal" commitment $m := \mathsf{COMM}_r(a_{\mathsf{pk}}, \rho, \mathsf{pkL}, \mathsf{tL})$ with a $\mathsf{COMM}_r(\tau)$ where τ is a random string of appropriate length. Since ρ is random, $\mathcal{A}$'s advantage in distinguishing this modified experiment from ∂_2 is at most $\mathbf{Adv}^{\mathsf{COMM}}$. Then, if we modify all $q_{\mathbf{M}}$ Mint queries and all $q_{\mathbf{P}}$ Pour queries, by replacing the $q_{\mathbf{M}} + 2 \cdot q_{\mathbf{P}}$ internal commitments with random strings, we can bound $\mathcal{A}$'s advantage by $(q_{\mathbf{M}} + 2 \cdot q_{\mathbf{P}}) \cdot \mathbf{Adv}^{\mathsf{COMM}}$.

Next, similarly, replace the coin commitment in the first Pour with a commitment to a random value, then $\mathcal{A}$'s advantage in distinguishing this modified experiment from the above one is at most $\mathbf{Adv}^{\mathsf{COMM}}$. Then, we modify all $q_{\mathbf{P}}$ Pour queries, by replacing the $2 \cdot q_{\mathbf{P}}$ output coin commitments with random strings, we can update the bound to $\mathcal{A}$'s advantage to $(q_{\mathbf{M}} + 2 \cdot q_{\mathbf{P}}) \cdot \mathbf{Adv}^{\mathsf{COMM}}$.

Finally, we modify the $q_{\mathbf{CP}}$ $\mathsf{CreatePKCM}$ commitments to replace the resulting $q_{\mathbf{CP}}$ public key commitments by a random string of appropriate length, we obtain the experiment $\partial_{\mathbf{sim}}$ and get that $\left|\mathbf{Adv}^{\partial_{\mathbf{sim}}} - \mathbf{Adv}^{\partial_3}\right| \leq (q_{\mathbf{M}} + 4 \cdot q_{\mathbf{P}} + q_{\mathbf{CP}}) \cdot \mathbf{Adv}^{\mathsf{COMM}}$.