Z-Channel: Scalable and Efficient Scheme in Zerocash

Yuncong Zhang[1,6], Yu Long[2,6], Zhen Liu[3,6], Zhiqiang Liu[4,6], and Dawu Gu[5,6]

[1] shjdzhangyuncong@sjtu.edu.cn
[2] longyu@sjtu.edu.cn
[3] liuzhen@sjtu.edu.cn
[4] liu-zq@cs.sjtu.edu.cn
[5] dwgu@sjtu.edu.cn
[6] Shanghai Jiao Tong University

Abstract. Decentralized ledger-based cryptocurrencies like Bitcoin present a way to construct payment systems without trusted banks. However, the anonymity of Bitcoin is fragile. Many altcoins and protocols are designed to improve Bitcoin on this issue, among which Zerocash is the first full-fledged anonymous ledger-based currency, using zero-knowledge proof, specifically zk-SNARK, to protect privacy. However, Zerocash suffers two problems: poor scalability and low efficiency. In this paper, we address the above issues by constructing a micropayment system in Zerocash called Z-Channel. First, we improve Zerocash to support multisignature and time lock functionalities, and prove that the reconstructed scheme is secure. Then we construct Z-Channel based on the improved Zerocash scheme. Our experiments demonstrate that Z-Channel significantly improves the scalability and reduces the confirmation time for Zerocash payments.

Keywords: Cryptocurrency, Blockchain, Zerocash, Privacy, Instant payment

1 Introduction

Decentralized ledger-based cryptocurrencies like Bitcoin [1] present a way to construct payment systems without trusted banks. After Bitcoin, many digital currencies try to improve it in different aspects, including functionality [2–5], consensus scheme [6, 3], scalability and efficiency [7, 2], and privacy [8, 9], etc.

Privacy protection in ledger-based digital currencies has attracted tremendous attention [10]. Bitcoin has been thoroughly analyzed and its privacy is deemed fragile [11]. Analyzing the transaction graph, values and dates in the ledger possibly link Bitcoin addresses with real world identities. *Mixes* are designed to break the linkability in Bitcoin system. A mix is a trusted party who mixes coins from many users and gives different coins back to them. However, coin mixing is time-consuming and centralized, so a mix is required to be trustworthy. Therefore, decentralized mixes are constructed like TumbleBit [12], CoinSwap [13], CoinParty [14], CoinShuffle[15] and CoinJoin [16], and altcoins such

as Zerocoin [17], BlindCoin [8], Mixcoin [18] and Pinocchio coin [19], etc. However, these solutions still suffer drawbacks: 1) Insufficient performance. Most of them require more than one round of interactions between many parties. 2) Lack of functionality. They allows "washing" coins from time to time, but fail to hide everyday transactions.

In comparison, Zerocash [20] completely conceals the user identity and amount of payment in each and every transaction. Zerocash uses zero-knowledge proof, specifically zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs) [21, 22], to protect privacy. However, zero-knowledge proof worsens the scalability and efficiency problems which are already serious in ledger-based currencies. In fact, Zerocash transactions are even larger than those of Bitcoin, and verifying zk-SNARK proof takes longer than verifying a Bitcoin transaction.

For other ledger-based digital currencies, works have been trying to solve the scalability and efficiency issues. Changing the blocksize [23] straightforwardly increases the scalability, while compromising efficiency by higher network latency and longer verification time. The block merging proposed in MimbleWimble [24] requires a special structure for the blocks and transactions, sacrificing a majority of the digital currency functionalities. Currently, micropayment channel [25] is the most promising solution to both scalability and efficiency problems. Micropayment channel enables Bitcoin users to conduct payments securely off-chain, promising to support billions of users. However, nobody has proposed to construct a micropayment system on Zerocash [7].

1.1 Our contribution

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

We develop a micropayment scheme over Zerocash, *Z-Channel*. Z-Channel allows numerous users to perform high-frequency transactions off-chain in day-to-day routine, conducting payments 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 of Zerocash and propose a new scheme called *DAP Plus* (DAP+ for short). DAP+ enriches DAP with multisignature and time lock features needed by Z-Channel. We give the formal definition of the security of DAP+ scheme based on the original DAP scheme. We prove that DAP+ scheme is secure under this definition.

Moreover, we implement the zk-SNARK for the new NP statement, based on the code of ZCash, and instantiate the Z-Channel protocol. We benchmark the zero-knowledge proofs and the procedures in Z-Channel protocol. In our experiment, a payment can be issued within 3 milliseconds, which is significantly

[7] The work of BOLT (Blind Off-chain Lightweight Transactions) [26] 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, and does not specify the concrete construction over Zerocash.

faster than the original Zerocash payment, which requires several minutes for generating zero-knowledge proof, and dozens of minutes for ledger confirmation.

1.2 Paper organization

The remainder of the paper is organized as follows. Section 2 introduces the preliminaries needed for understanding our work. Section 3 presents DAP+ scheme. In Section 4, we describe the construction of Z-Channel. Section 5 analyzes the performance of Z-Channel. Section 6 concludes this paper.

2 Preliminaries

2.1 Background on zk-SNARKs

The zero-knowledge proving scheme in Zerocash is zk-SNARK (Succinct Non-interactive ARguments of Knowledge) [22]. Suppose Alice has an NP problem instance x and its witness w. She is proving to Bob that x is a valid instance, without revealing w to Bob. She inputs x and w in zk-SNARK to generate a proof π, and sends π instead of w to Bob. Bob then inputs x and π in zk-SNARK and is told if π is a valid proof of x. Let C be a circuit verifying an NP language $\mathcal{L}_C$. C takes as input an instance x and witness w, and outputs b indicating if w is a valid witness for x.

A zk-SNARK is a triple of algorithms (KeyGen, Prove, Verify) fulfilling the above procedure. The algorithm $\mathsf{KeyGen}(C)$ outputs a *proving key* pk and a *verification key* vk. The algorithm Prove takes as input an instance x, a witness w, and 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, the proof π, and vk, and outputs b indicating if he is convinced that $x \in \mathcal{L}_C$.

A zk-SNARK has the property of

1. **Correctness.** If the honest prover can convince the verifier;
2. **Proof-of-knowledge.** If the verifier accepting a proof implies the prover knowing the witness;
3. **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 [27].

2.2 The Zerocash Scheme

Zerocash is constructed by overlaying a Decentralized Anonymous Payment (DAP) scheme over Bitcoin or any other ledger-based cryptocurrencies, which we call the basecoin.

DAP introduces a new kind of coin called *shielded coin* (by contrast, we call the unspent outputs in basecoin *transparent coins*), denoted by $\mathbf{c} = (\mathsf{cm}, v, \rho, a_{\mathsf{pk}}, r, s)$,

where cm is an information-hiding trapdoor commitment, ρ is a random string for generating the unique serial number sn for this coin. ρ together with the denomination v and *shielded address* a_{pk} of the owner are concealed in cm. r and s are the trapdoors used in commitment.

DAP introduces two types of transactions to handle shielded coins: a *mint* transaction $\mathsf{tx}_{\mathsf{Mint}}$ transforms transparent coins into a shielded coin, and a *pour* transaction $\mathsf{tx}_{\mathsf{Pour}}$ conducts payments between shielded coins. $\mathsf{tx}_{\mathsf{Pour}}$ could also transform part of the input shielded coins back to transparent coins.

A mint transaction $\mathsf{tx}_{\mathsf{Mint}} = (\mathsf{cm}, v, k, s)$ takes transparent coins as input, and produces one shielded coin $\mathbf{c} = (\mathsf{cm}, v, a_{\mathsf{pk}}, r, s)$ [8]. The commitment is conducted in two steps: all the data except v are committed into an intermediary commitment k (with trapdoor r), which is then committed together with v to obtain cm (with trapdoor s). The second commitment is opened, i.e. k, s and v are appended in $\mathsf{tx}_{\mathsf{Mint}}$ for others to verify v, while other information are concealed in k.

A pour transaction $\mathsf{tx}_{\mathsf{Pour}} = (\mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, v_{\mathsf{pub}}, \pi_{\mathsf{POUR}}, *)$ takes two shielded coins $\mathbf{c}_1^{\mathsf{old}}$ and $\mathbf{c}_2^{\mathsf{old}}$ as input, and produces two newly generated shielded coins $\mathbf{c}_1^{\mathsf{new}}$ and $\mathbf{c}_2^{\mathsf{new}}$, and a (possibly zero-value) transparent coin of value v_{pub}. $\mathsf{tx}_{\mathsf{Pour}}$ reveals the commitments to new shielded coins, i.e. $\mathsf{cm}_1^{\mathsf{new}}$ and $\mathsf{cm}_2^{\mathsf{new}}$, and the serial numbers of the old coins to prevent trying to spend them again. The validity of $\mathsf{tx}_{\mathsf{Pour}}$ is proved by zero-knowledge proof π_{POUR} for the following statement: $\mathsf{sn}_1^{\mathsf{old}}$ and $\mathsf{sn}_2^{\mathsf{old}}$ are valid serial numbers whose ρ_i^{old} are respectively committed in $\mathsf{cm}_1^{\mathsf{old}}$ and $\mathsf{cm}_2^{\mathsf{old}}$ that exist on the ledger, and I can open the commitments; I can open $\mathsf{cm}_1^{\mathsf{new}}$ and $\mathsf{cm}_2^{\mathsf{new}}$; the input and the output are balanced, i.e. $v_1^{\mathsf{old}} + v_2^{\mathsf{old}} = v_1^{\mathsf{new}} + v_2^{\mathsf{new}} + v_{\mathsf{pub}}$; I am owner of the input coins, i.e. for each $i \in \{1, 2\}$, I know secret key $a_{\mathsf{sk},i}^{\mathsf{old}}$ corresponding to the address $a_{\mathsf{pk},i}^{\mathsf{old}}$ committed in $\mathbf{c}_i^{\mathsf{old}}$.

Above are the main ideas of Zerocash. [20] mentions and solves many other issues in implementing Zerocash, we only provide a brief description due to space limitation.

1. To prove the existence of a coin commitment cm on the ledger, all commitments are maintained in a Merkle-tree with root rt.
2. To protect all the public information in $\mathsf{tx}_{\mathsf{Pour}}$ (for example, the address of v_{pub}) from forgery, $\mathsf{tx}_{\mathsf{Pour}}$ is protected by a signature σ, whose verification key $\mathsf{pk}_{\mathsf{sig}}$ is generated on the fly, and protected by zero-knowledge proof.

Finally, the formal definition of DAP scheme consists of algorithms (Setup, $\mathsf{CreateAddress}$, Mint, Pour, Verify, $\mathsf{Receive}$). The Setup algorithm initializes a DAP instance by invoking the initializers in all the cryptographic building blocks (for example, KeyGen in zk-SNARK); the $\mathsf{CreateAddress}$ algorithm is executed by each user to generate a shielded address and its key $(a_{\mathsf{pk}}, a_{\mathsf{sk}})$; the Mint algorithm outputs a mint transaction and the resulting shielded coin; the Pour algorithm outputs a pour transaction and the new shielded coins; the Verify algorithm

[8] We neglect the transaction fees.

checks the validity of a mint or pour transaction; finally, the Receive algorithm scans a ledger and outputs all the shielded coins belonging to a given shielded address.

2.3 Micropayment Channel

Micropayment channel [25] allows two parties to make payments to each other without publishing transactions on the ledger. A basic micropayment channel scheme consists of three protocols: establish channel, update channel, and close channel. For convenience, we use Alice and Bob in the following description of a complete execution of a micropayment channel. We use A and B in the subscript for a coin of address Alice or Bob (AB for a coin in shared address). We use α and β to differentiate different versions of the same transaction, i.e. symmetric up to Alice and Bob.

Next, we present the execution procedure of a micropayment channel.

1. **Establish channel.**
   (a) Alice and Bob agree on (v_A, v_B), the currency they are willing to devote into the channel, and a shared address $\mathsf{addr}^{\mathsf{shr}}$.
   (b) They agree on a *sharing transaction* $\mathsf{tx}^{\mathsf{shr}}$, which transforms values v_A and v_B from Alice and Bob, to a coin $\mathbf{c}^{\mathsf{shr}}_{AB}$ in address $\mathsf{addr}^{\mathsf{shr}}$ of value $v_A + v_B$.
   (c) Alice signs a *closing transaction* $\mathsf{tx}^{\mathsf{cls}}_{\beta}$ for Bob, and Bob signs $\mathsf{tx}^{\mathsf{cls}}_{\alpha}$ for Alice. $\mathsf{tx}^{\mathsf{cls}}_{\alpha}$ transforms $\mathbf{c}^{\mathsf{shr}}_{AB}$ to two coins $\mathbf{c}^{\mathsf{cls}}_{\alpha,A}$ and $\mathbf{c}^{\mathsf{cls}}_{\alpha,B}$ of value v_A and v_B to Alice and Bob respectively. $\mathsf{tx}^{\mathsf{cls}}_{\beta}$ transforms $\mathbf{c}^{\mathsf{shr}}_{AB}$ to two coins $\mathbf{c}^{\mathsf{cls}}_{\beta,A}$ and $\mathbf{c}^{\mathsf{cls}}_{\beta,B}$ in the same way.
   (d) Finally, they publish $\mathsf{tx}^{\mathsf{shr}}$, and the channel is established. The *balance* of a new channel is (v_A, v_B).

   Remarks:
   - In case they do not have coins of the exact value before creating $\mathsf{tx}^{\mathsf{shr}}$, they can optionally conduct a funding procedure to prepare the coins. In this case, the input coins to $\mathsf{tx}^{\mathsf{shr}}$ are called *funding coins*, denoted by $\mathbf{c}^{\mathsf{fund}}_{A}$ and $\mathbf{c}^{\mathsf{fund}}_{B}$ respectively.
   - They sign $\mathsf{tx}^{\mathsf{cls}}$ before $\mathsf{tx}^{\mathsf{shr}}$, so that neither of them can lock the other's currency in the shared address forever.
   - The implementation of shared address varies for different cryptocurrencies. For Bitcoin, this is implemented by paying to multiple addresses. For Zerocash, however, this functionality is not implemented, and is what our work aims to provide.
2. **Update channel.** If Alice pays Bob by Δ, the balance of the channel should be updated to $(v_A - \Delta, v_B + \Delta)$. This procedure is executed without interacting with the ledger.
   (a) Alice signs a new closing transaction $\mathsf{tx}^{\mathsf{cls}'}_{\beta}$ for Bob, and Bob signs $\mathsf{tx}^{\mathsf{cls}'}_{\alpha}$ for Alice. $\mathsf{tx}^{\mathsf{cls}'}_{\alpha}$ transforms $\mathbf{c}^{\mathsf{shr}}_{AB}$ to two coins $\mathbf{c}^{\mathsf{cls}'}_{\alpha,A}$ and $\mathbf{c}^{\mathsf{cls}'}_{\alpha,B}$ of value $v_A - \Delta$ and $v_B + \Delta$ to Alice and Bob respectively; similar for $\mathsf{tx}^{\mathsf{cls}'}_{\beta}$.

(b) Alice signs a *revoking transaction* $\mathsf{tx}_B^{\mathsf{rev}}$ for Bob, and Bob signs $\mathsf{tx}_A^{\mathsf{rev}}$ for Alice. $\mathsf{tx}_B^{\mathsf{rev}}$ transforms $\mathbf{c}_{\alpha,A}^{\mathsf{cls}}$ to a coin $\mathbf{c}_B^{\mathsf{rev}}$ for Bob; $\mathsf{tx}_A^{\mathsf{rev}}$ transforms $\mathbf{c}_{\beta,B}^{\mathsf{cls}}$ to a coin $\mathbf{c}_A^{\mathsf{rev}}$ for Alice.

Remarks:

- Each update is associated with a sequence number which increases by one with each update. And the sequence number of the transactions in each update are identical to that of the update.
- After an update, the previous closing transactions are rendered obsolete. The revoking transactions prevents any of the parties from publishing obsolete closing transaction, by giving all his/her coin in the channel to the other party.
- To prevent the revoking transaction from being surpassed by a transaction immediately following the obsolete closing transaction, the coin $\mathbf{c}_{\alpha,A}^{\mathsf{cls}}$ is locked by time T after $\mathsf{tx}_\alpha^{\mathsf{cls}}$ is published, while $\mathsf{tx}_B^{\mathsf{rev}}$ overrides the time lock. Implementation of such fine access control over a coin is left to the cryptocurrencies. For Bitcoin, the pay-to-script feature suffices to do the job. For Zerocash, the current scheme cannot accomplish this, which is another issue solved in our work.

3. **Close channel.** Either Alice or Bob can close the channel any time after the channel is established, without interacting with the other party. To close the channel, Alice or Bob publishes his/her own (alpha or beta) version of the most updated closing transaction, and waits for time T before redeeming his/her closing coin. The transactions taking the closing coin are called redeem transactions.

Fig.1 presents an example of execution of micropayment 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.

2.4 Distributed Signature Generation Scheme

The naive implementation of multisignature scheme in Bitcoin, i.e. counting the number of signatures, reveals some data which compromises the privacy if used in Zerocash. We implement the multisignature feature in an alternative way, namely the *distributed signature generation scheme* [28]. 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

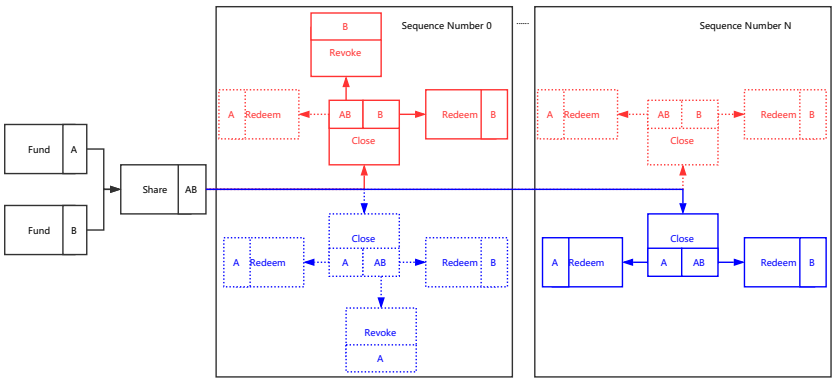

Fig. 1. Transactions and coins in a closed micropayment channel. The transactions that are finally confirmed on the ledger are represented in solid. This figure presents two examples: 1) (Blue) Bob publishes the latest beta version ending the channel in legal way or 2) (Red) Alice publishes an outdated alpha version, and Bob taking away all the coins for punishment.

broadcasts it to other parties. Anyone obtaining all the shares can recover the complete signature σ. This signature can be verified by pk and is indistinguishable from the signatures directly signed by sk.

3 DAP Plus: Improved Decentralized Anonymous Payment Scheme

Our construction of Z-Channel relies on two functionalities: multisignature and time lock. 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 time lock features.

3.1 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 to a public key lock in the coin. In Zerocash, a shielded coin $\mathbf{c}$ consists of a commitment cm and some secret data necessary for opening cm. The commitment involves the following data: the shielded address a_{pk} owned by Bob, the denomination v and a random string ρ (used for generating serial number sn). In DAP+, we require Alice to additionally commit a *public key lock* pklk into cm. pklk is a properly encoded public key of some public signature scheme.

For implementing multisignature functionality, we suggest that it is a *distributed signature generation scheme* described in last section, to enable multiple users to share a public key which is indistinguishable from a public key generated by a single user. For now we simply assume that Bob generates a pair of keys locally and sends the public key pklk to Alice for her to commit into cm. To fix the length of the committed data in cm, Alice commits the hash of pklk, denoted by $\mathsf{pkh} = \mathsf{Hash}(\mathsf{pklk})$ instead of pklk. When Bob tries to spend this coin, he has to append to the transaction a signature σ which is verified by pklk. We denote the data protected by this signature (for example, the entire transaction, or a short fixed string) by a function $\mathsf{ToBeLocked}()$, and leave it to be determined by the application that builds on top of DAP+ scheme.

To allow other parties to verify the signature, pklk should be disclosed as the coin is spent. The anonymity of Bob against Alice is thus compromised, since Alice would immediately perceive when Bob spends the coin, by identifying pklk published in the transaction. To solve this problem, we let Bob commit pkh into a commitment pkcm, with his secret key a_{sk} as trapdoor, and sends pkcm to Alice. Therefore, Alice does not know either pklk or its hash pkh, but she is still able to commit pkh into cm in an indirect way, i.e. committing pkcm into cm. We modify the zero-knowledge NP statement POUR in [20] for the pour transaction so that Alice only needs to prove that she knows pkcm for the new coins. When Bob spends his coin, however, he has to prove that the revealed pkh is correctly committed in the coins to spend, with his knowledge of a_{sk}.

Commit a time lock in coin. Next, we commit a time lock tlk 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 $\mathsf{BH}(\mathsf{cm}) + \mathsf{tlk} < \mathsf{MBH}$ 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 compromise the privacy of Alice.

We solve this issue by noting that Alice does not have to prove that $\mathsf{BH}(\mathsf{cm}) + \mathsf{tlk} < \mathsf{MBH}$, but $\mathsf{BH}(?) + \mathsf{tlk} < \mathsf{MBH}$ where $\mathsf{BH}(?)$ is the block height of something that is guaranteed to be later than cm on the ledger and 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 time lock. If a coin commits a public key lock pklk and time lock tlk, we say the coin is *locked* by pklk with tlk blocks. If tlk is set to the maximum time lock MTL, then we say the coin is locked by pklk forever. We denote a pair of public key commitment and time lock by $\mathsf{lock} = (\mathsf{pkcm}, \mathsf{tlk})$, and a pair of public key lock and signature by $\mathsf{unlock} = (\mathsf{pklk}, \sigma)$. We say unlock *unlocks* a lock if pklk 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 time lock. That is to say, the transaction is valid either when the time lock expires or a valid unlock is provided. To say it in another way, a coin is locked by tlk 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 $\mathsf{ovd}||(\mathsf{BH}(\mathsf{rt}) + \mathsf{tlk} < \mathsf{MBH})$ is true, where $||$ means logical OR.

Note that this logic can be easily modified, without modifying the NP statement POUR. 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 time lock. We will use a slightly modified version of logic in Z-Channel, but for simplicity, we only describe constructing with basic OR logic in this section.

3.2 Construction of DAP Plus Scheme

A *DAP Plus scheme* is a tuple of polynomial-time algorithms (Setup, $\mathsf{CreateAddress}$, $\mathsf{CreatePKCM}$, $\mathsf{MintPlus}$, $\mathsf{PourPlus}$, $\mathsf{VerifyPlus}$, $\mathsf{ReceivePlus}$). 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 [20]. To save space, we only present the differences in the construction of these algorithms compared to the corresponding ones in the original DAP scheme. For interested readers we refer the complete construction to Appendix A and [20].

We first present the cryptographic building blocks mentioned subsequently.

- Information hiding trapdoor commitment COMM.
- Collision resistance and flexible-input-length hash function Hash.
- Distributed public signature scheme $(\mathcal{G}_{\mathsf{dst}}, \mathcal{K}_{\mathsf{dst}}, \mathcal{S}_{\mathsf{dst}}, \mathcal{V}_{\mathsf{dst}})$, where $\mathcal{G}_{\mathsf{dst}}$ is for generating global public parameter $\mathsf{pp}_{\mathsf{dst}}$, $\mathcal{K}_{\mathsf{dst}}$ is the key generation algorithm, $\mathcal{S}_{\mathsf{dst}}$ is the signing algorithm and $\mathcal{V}_{\mathsf{dst}}$ is the verification algorithm.

Next, we present the detailed difference in the construction of the algorithms in DAP+ scheme compared to those in DAP scheme. For simplicity, we use subscript 1..2 to represent a pair each with subscript 1 and 2. For example, $\mathbf{c}^{\mathsf{old}}_{1..2}$ represents $\mathbf{c}^{\mathsf{old}}_1, \mathbf{c}^{\mathsf{old}}_2$.

System setup. Given security parameter λ, the algorithm Setup generates a set of public parameters pp. 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.

Apart from the executions mentioned in the original Setup algorithm in DAP scheme, in DAP+ this algorithm does the following:

1. Compute $\mathsf{pp}_{\mathsf{dst}} = \mathcal{G}_{\mathsf{dst}}()$.
2. Add $\mathsf{pp}_{\mathsf{dst}}$ to pp.

Create address. Given public parameter pp, the algorithm $\mathsf{CreateAddress}$ outputs a new shielded address and its secret key in a pair $(a_{\mathsf{pk}}, a_{\mathsf{sk}})$. The construction of $\mathsf{CreateAddress}$ in DAP+ is exactly the same to that in DAP.

Create public key commitment. Given public parameter pp and address secret key $\mathsf{addr}_{\mathsf{sk}}$, the algorithm $\mathsf{CreatePKCM}$ generates a key pair for the distributed signature scheme, and a commitment for the public key.

This algorithm is new in DAP+ scheme, so we present the complete construction as follows:

1. Compute $(\mathsf{pk}_{\mathsf{dst}}, \mathsf{sk}_{\mathsf{dst}}) = \mathcal{K}_{\mathsf{dst}}(\mathsf{pp}_{\mathsf{dst}})$.
2. Compute $\mathsf{pkh} := \mathsf{Hash}(\mathsf{pk}_{\mathsf{dst}})$.
3. Parse $\mathsf{addr}_{\mathsf{sk}}$ as $(a_{\mathsf{sk}}, \mathsf{sk}_{\mathsf{enc}})$, compute $\mathsf{pkcm} := \mathsf{COMM}_{a_{\mathsf{sk}}}(\mathsf{pkh})$.
4. Output $\mathsf{pk}_{\mathsf{dst}}, \mathsf{sk}_{\mathsf{dst}}, \mathsf{pkcm}$.

For complete anonymity, each time Alice tries to generate a coin (via $\mathsf{MintPlus}$ 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. For privacy, each generated pkcm must be used only once. It is recommended that a user stores the output tuples in the wallet, and whenever a new coin is received, mark the tuple containing the corresponding pkcm as already used. A coin that uses a pkcm already used should be considered invalid.

Mint coin. The $\mathsf{MintPlus}$ algorithm outputs a shielded coin and a mint transaction, which transforms some transparent coins into shielded coins with equal value.

Compared to the Mint algorithm in DAP scheme, the $\mathsf{MintPlus}$ algorithm behaves differently in the following respects.

1. Additionally take as input a lock lock.
2. Additionally commit lock into the intermediary coin commitment, i.e. compute
$$k := \mathsf{COMM}_r(a_{\mathsf{pk}}, \rho, \mathsf{lock}).$$
3. Add lock to the output coin $\mathbf{c}$.

Pour algorithm. The $\mathsf{PourPlus}$ algorithm outputs two shielded coins and a pour transaction, which transfers values from two input shielded coins into two new shielded coins, and optionally transfers part of the input value back to a transparent coin.

Compared to the Pour algorithm in DAP scheme, the $\mathsf{PourPlus}$ algorithm makes the following modifications.

- Input:
  1. Additionally take as input the minimal block height MBH.
  2. Input two Merkle-roots $\mathsf{rt}_{1..2}$ instead of one rt, i.e. use separate roots for two old coins.
  3. For each new coin $\mathbf{c}_i^{\mathsf{new}}$ additionally input a lock $\mathsf{lock}_i^{\mathsf{new}}$.
  4. Each old coin $\mathbf{c}_i^{\mathsf{old}}$ additionally contains a lock $\mathsf{lock}_i^{\mathsf{old}}$.
  5. For each old coin $\mathbf{c}_i^{\mathsf{old}}$ additionally input a (possibly empty) secret key $\mathsf{sk}_{\mathsf{dst},i}$.
- Procedure:
  1. Replace the part of generating new coin with the procedure of $\mathsf{MintPlus}$.
  2. Replace the zero-knowledge proof with one of the new statement (see paragraph "NP statement").
  3. In the part of preventing forgery, add the following to the message to be protected: $\mathsf{MBH}, \mathsf{pklk}_{1..2}^{\mathsf{old}}$.
  4. Add the unlock procedure:
     i. Compute $\mathsf{msg} := \mathsf{ToBeLocked}()$.
     ii. Let $\mathsf{ovd}_i := \mathsf{BH}(\mathsf{rt}_i) + \mathsf{tlk}_i^{\mathsf{old}} \geq \mathsf{MBH}$.
     iii. Compute[9] $\sigma_i := \mathcal{S}_{\mathsf{dst}}(\mathsf{sk}_{\mathsf{dst},i}, \mathsf{msg})$ if ovd_i, or let $\sigma_i := \perp$ if not ovd_i.
     iv. Let $\mathsf{unlock}_i := (\mathsf{pklk}_i^{\mathsf{old}}, \sigma_i)$.
- Output:
  1. In each output coin $\mathbf{c}_i^{\mathsf{new}}$, add the lock $\mathsf{lock}_i^{\mathsf{new}}$.
  2. Add to the pour transaction $\mathsf{MBH}, \mathsf{unlock}_{1..2}$.

Verify Transactions. Given public parameters pp, a transaction tx and a ledger L, the $\mathsf{VerifyPlus}$ algorithm outputs a bit b indicating if a given transaction is valid on a ledger.

If tx is a mint transaction, $\mathsf{VerifyPlus}$ behaves exactly as the Verify algorithm in DAP scheme.

If tx is a pour transaction, $\mathsf{VerifyPlus}$ behaves differently in the following respects.

1. Check the minimum block height MBH, if it is larger than the current block height, output $b := 0$ and exit.
2. In the part of preventing forgery, add the following to the message against which the signature is verified: MBH and $\mathsf{pklk}_{1..2}^{\mathsf{old}}$.
3. Check the validity of unlock:
   (a) If the signature σ_i in unlock_i is empty, set ovd_i to false, for $i = 1, 2$.
   (b) If the signature σ_i in unlock_i is not empty, compute $\mathsf{msg} = \mathsf{ToBeLocked}()$ and check $\mathcal{V}_{\mathsf{dst}}(\mathsf{pklk}_i, \mathsf{msg}, \sigma_i)$ for $i = 1, 2$. If any check fails, output $b := 0$ and exit.
4. Check the zero-knowledge proof according to the new NP statement.

Receive coins. Given public parameter pp, a shielded address and its key $(a_{\mathsf{pk}}, a_{\mathsf{sk}})$, and a ledger L, the $\mathsf{ReceivePlus}$ algorithm scans the ledger and outputs coins on the ledger belonging to a given shielded address.

[9] This procedure may be executed distributedly, where the input $\mathsf{sk}_{\mathsf{dst},i}$ is shared by more than one parties, and σ_i is synthesized from the shared signatures.

Compared to the Receive algorithm in DAP scheme, after finding out a coin belonging to the given address, the ReceivePlus algorithm additionally checks the pkcm in the coin to make sure that it is in the wallet and not marked as already used.

NP statement. We modify the NP statement POUR as follows:

- Public input:
  1. Use two Merkle-roots $\mathsf{rt}_{1..2}$ instead of one rt.
  2. Add minimum block height MBH.
  3. For each old coin, add $\mathsf{pkh}_i^{\mathsf{old}} = \mathsf{Hash}(\mathsf{pklk}_i^{\mathsf{old}})$ and ovd_i computed as in PourPlus and VerifyPlus algorithm.
- Private input: add the locks $\mathsf{lock}_i^{\mathsf{old}}$ and $\mathsf{lock}_i^{\mathsf{new}}$ in the corresponding coins.
- Statement:
  1. For each new coin, replace the commitment validity check with the following equation

$$\mathsf{cm}_i^{\mathsf{new}} = \mathsf{COMM}_{s_i^{\mathsf{new}}}(v_i^{\mathsf{new}}, \mathsf{COMM}_{r_i^{\mathsf{new}}}(a_{\mathsf{pk},i}^{\mathsf{new}}, \rho_i^{\mathsf{new}}, \mathsf{lock}_i^{\mathsf{new}})).$$

  2. For each old coin, replace the commitment validity check with the following equation

$$\mathsf{cm}_i^{\mathsf{old}} = \mathsf{COMM}_{s_i^{\mathsf{old}}}(v_i^{\mathsf{old}}, \mathsf{COMM}_{r_i^{\mathsf{old}}}(a_{\mathsf{pk},i}^{\mathsf{old}}, \rho_i^{\mathsf{old}}, \mathsf{COMM}_{a_{\mathsf{sk},i}^{\mathsf{old}}}(\mathsf{pkh}_i^{\mathsf{old}}), \mathsf{tlk}_i^{\mathsf{old}})).$$

  3. For each old coin, the time lock either expires or is overridden, i.e.

$$\mathsf{ovd}_i || (\mathsf{BH}(\mathsf{rt}_i) + \mathsf{tlk}_i^{\mathsf{old}} < \mathsf{MBH})$$

3.3 Security of DAP Plus Scheme

The security of DAP+ scheme is defined in a similar way as that of DAP scheme. We refer to Appendix C for the complete security definition and Appendix D for the security proof.

4 Z-Channel

We present the micropayment system over Zerocash, which we call *Z-Channel*. Z-Channel follows the structure of micropayment channel presented in Section 2.3. We first give the main idea of Z-Channel, then present the complete protocol.

4.1 Main Idea of Z-Channel

In the micropayment scheme, the parties generate many transactions during each update. In Zerocash, due to zero-knowledge proof, this will be slow. We consider letting the parties hold a summary of the transaction instead of a complete one. Define the *note* of a pour transaction to be the tuple $(\mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, *)$,

where $*$ is data of the public output[10]. The note specifies the behavior of the pour transaction. Recall that we left the $\mathsf{ToBeLocked}()$ function in DAP+ scheme to be defined by the application. We let this function to return the note of the pour transaction.

Context of a Z-Channel. If Alice and Bob negotiate the random data (namely $r, s, \rho, (a_{\mathsf{sk}}, a_{\mathsf{pk}})$) needed in every coin in the channel, the communication cost is tremendous. We consider letting them negotiate a random string seed, and generate all random data with a pseudorandom function. We assign a unique tag to each random string for distinction. We use the superscripts and subscripts of the coin to denote the tag, for example, $\mathsf{tag}^{\mathsf{cls}}_{\beta,A}$ denotes the tag of $\mathbf{c}^{\mathsf{cls}}_{\beta,A}$.

In the protocol, only a limited number of transactions (six, to be specific) will be published on the ledger, a limited number of public keys suffice to ensure the uniqueness of public key locks in each published transaction. They can be determined at the start of the protocol. We define the *context* of a Z-Channel ctx to be a tuple of seed and all the public key locks. Given the context and the denomination, each coin in the Z-Channel is completely determined, i.e. we can define the procedure $\mathbf{c} := \mathsf{GetCoin}(\mathsf{ctx}, v, \mathsf{tag})$ where tag specifies which coin to compute.

Relationship between time lock and public key lock. The closing coins $\mathbf{c}^{\mathsf{cls}}_{\alpha,A}$ and $\mathbf{c}^{\mathsf{cls}}_{\beta,B}$ are locked by T blocks, by the default specification of DAP+ scheme, the coin is spendable when either lock is resolved, so both Alice and Bob can spend the coins after T blocks. We have to modify the logic relationship between time lock and public key lock. We define two functions $\mathsf{ToBeLockedS}() := 0 \| \mathsf{ToBeLocked}()$ and $\mathsf{ToBeLockedW}() := 1 \| \mathsf{ToBeLocked}()$. We require that a valid pour transaction contains a signature verified by the public key lock on either $\mathsf{ToBeLockedS}()$ or $\mathsf{ToBeLockedW}()$. Furthermore, if the signature is verified on $\mathsf{ToBeLockedS}()$, we call it a *strong signature*, otherwise it is *weak*; we specify that only a strong signature can override the time lock.

When Alice signs $\mathsf{tx}^{\mathsf{cls}}_{\beta}$ for Bob, she simultaneously signs $\mathsf{tx}^{\mathsf{rdm}}_{B}$ which sends $\mathbf{c}^{\mathsf{cls}}_{\beta,B}$ to $\mathbf{c}^{\mathsf{rdm}}_{B}$ owned by Bob, with a weak signature. Denote the procedure of signing the notes for the other party in the update of sequence number seq with balance (v_A, v_B) by $(\sigma_1, \sigma_2) := \mathsf{SignNote}(v_A, v_B, \mathsf{seq})$. When Bob signs $\mathsf{tx}^{\mathsf{rev}}_{A}$ for Alice, which sends $\mathbf{c}^{\mathsf{cls}}_{\beta,B}$ to $\mathbf{c}^{\mathsf{rev}}_{A}$, he signs with strong signature. Therefore, if the closing coin is not revoked, after publishing $\mathsf{tx}^{\mathsf{cls}}_{\beta}$, Bob can wait T blocks before publishing $\mathsf{tx}^{\mathsf{rdm}}_{B}$ and get his coin back, while Alice can never get $\mathbf{c}^{\mathsf{cls}}_{\beta,B}$. If a revoked $\mathsf{tx}^{\mathsf{cls}}_{\beta}$ is published, Alice publishes $\mathsf{tx}^{\mathsf{rev}}_{A}$ which immediately takes $\mathbf{c}^{\mathsf{cls}}_{\beta,B}$ away. Table 1 summarizes all the public keys and time locks of each coin.

4.2 Construction of Z-Channel Protocol

A Z-Channel Protocol ZCP is a tuple of subprotocols ($\mathsf{Establish}$, Update, Close). We present the construction of the subprotocols in Algorithms 1,2 and 3. In Algorithm 1 and Algorithm 2 we divide (by horizontal rule) the procedures into

[10] In Z-Channel, the public output is always zero, so we neglect it in the sequel

$\mathbf{c}$	pklk	tlk	$\mathbf{c}$	pklk	tlk	$\mathbf{c}$	pklk	tlk	$\mathbf{c}$	pklk	tlk
$\mathbf{c}_A^{\mathsf{fund}}$	$\mathsf{pk}_A^{\mathsf{fund}}$	MTL	$\mathbf{c}_B^{\mathsf{fund}}$	$\mathsf{pk}_B^{\mathsf{fund}}$	MTL	$\mathbf{c}^{\mathsf{shr}}$	$\mathsf{pk}_{AB}^{\mathsf{shr}}$	MTL			
$\mathbf{c}_{\alpha,A,i}^{\mathsf{cls}}$	$\mathsf{pk}_{AB}^{\mathsf{cls}}$	T	$\mathbf{c}_{\beta,A,i}^{\mathsf{cls}}$	$\mathsf{pk}_A^{\mathsf{cls}}$	MTL	$\mathbf{c}_{\beta,B,i}^{\mathsf{cls}}$	$\mathsf{pk}_{AB}^{\mathsf{cls}}$	T	$\mathbf{c}_{\alpha,B,i}^{\mathsf{cls}}$	$\mathsf{pk}_B^{\mathsf{cls}}$	MTL
$\mathbf{c}_A^{\mathsf{rdm}}$	$\mathsf{pk}_A^{\mathsf{rdm}}$	MTL	$\mathbf{c}_B^{\mathsf{rdm}}$	$\mathsf{pk}_B^{\mathsf{rdm}}$	MTL	$\mathbf{c}_A^{\mathsf{rev}}$	$\mathsf{pk}_A^{\mathsf{rev}}$	MTL	$\mathbf{c}_B^{\mathsf{rev}}$	$\mathsf{pk}_B^{\mathsf{rev}}$	MTL

Table 1. Coin lock specifications in Z-Channel. The public keys with single subscript are generated by the corresponding parties locally and sent to the other. Those with double subscripts are generated in distributed way. MTL is the maximum time lock.

groups. In each group the procedures are executed regardless of the presented order, while different groups should 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, any of them can immediately abort the protocol[11].

Establish the channel. Alice and Bob agree on the context (seed and all public keys) of a Z-Channel. After that, they publish the funding coins and the share coin. This protocol is formalized in Algorithm 1.

Algorithm 1: Establish Protocol

Alice and Bob agree on seed, v_A and v_B;
Alice and Bob distributedly generate $\mathsf{pk}_{AB}^{\mathsf{shr}}$ and $\mathsf{pk}_{AB}^{\mathsf{cls}}$;

Alice generates $\mathsf{pk}_A^{\mathsf{fund}}, \mathsf{pk}_A^{\mathsf{cls}}, \mathsf{pk}_A^{\mathsf{rdm}}, \mathsf{pk}_A^{\mathsf{rev}}$;
Bob generates $\mathsf{pk}_B^{\mathsf{fund}}, \mathsf{pk}_B^{\mathsf{cls}}, \mathsf{pk}_B^{\mathsf{rdm}}, \mathsf{pk}_B^{\mathsf{rev}}$;

Let $\mathsf{ctx} := (\mathsf{seed}, \mathsf{pk}_A^{\mathsf{fund}}, \mathsf{pk}_A^{\mathsf{cls}}, \mathsf{pk}_A^{\mathsf{rdm}}, \mathsf{pk}_A^{\mathsf{rev}}, \mathsf{pk}_B^{\mathsf{fund}}, \mathsf{pk}_B^{\mathsf{cls}}, \mathsf{pk}_B^{\mathsf{rdm}}, \mathsf{pk}_B^{\mathsf{rev}}, \mathsf{pk}_{AB}^{\mathsf{shr}}, \mathsf{pk}_{AB}^{\mathsf{cls}})$;
Alice computes $\mathsf{SignNote}(v_B, v_A, 0)$;
Bob computes $\mathsf{SignNote}(v_A, v_B, 0)$;

Alice signs $(\mathsf{sn}_A^{\mathsf{fund}}, \mathsf{sn}_B^{\mathsf{fund}}, \mathsf{cm}^{\mathsf{shr}}, \mathsf{cm}^{\mathsf{dmy}})$;
Bob signs $(\mathsf{sn}_A^{\mathsf{fund}}, \mathsf{sn}_B^{\mathsf{fund}}, \mathsf{cm}^{\mathsf{shr}}, \mathsf{cm}^{\mathsf{dmy}})$;

Alice publishes $\mathbf{c}_A^{\mathsf{fund}} := \mathsf{GetCoin}(\mathsf{ctx}, v_A, \mathsf{tag}_A^{\mathsf{fund}})$
Bob publishes $\mathbf{c}_B^{\mathsf{fund}} := \mathsf{GetCoin}(\mathsf{ctx}, v_B, \mathsf{tag}_B^{\mathsf{fund}})$

Alice/Bob publishes $\mathbf{c}^{\mathsf{shr}}$;

[11] When the channel is already established, to abort means executing the Close protocol.

Update the state of channel. To update the channel, Alice and Bob sign notes for new closing transactions for each other. After that, they sign revocations for each other to revoke the old version of closing transactions. This protocol is formalized in Algorithm 2.

Algorithm 2: 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 $(\mathsf{sn}^{\mathsf{cls}}_{\alpha,A,i}, \mathsf{sn}^{\mathsf{dmy}}, \mathsf{cm}^{\mathsf{rev}}_{B}, \mathsf{cm}^{\mathsf{dmy}})$;
Bob signs $(\mathsf{sn}^{\mathsf{cls}}_{\beta,B,i}, \mathsf{sn}^{\mathsf{dmy}}, \mathsf{cm}^{\mathsf{rev}}_{A}, \mathsf{cm}^{\mathsf{dmy}})$;

Close the channel. Let Alice be the party that actively closes the channel. Alice publishes the most updated closing transaction. Then they publish redeeming transactions to take away their coins. Alice waits for T blocks before publishing the redeeming transaction. This protocol is formalized in Algorithm 3.

Algorithm 3: Close Protocol

Alice publishes $\mathbf{c}^{\mathsf{cls}}_{\alpha}$;
Bob publishes $\mathbf{c}^{\mathsf{rdm}}_{B}$;
Alice waits T blocks and publishes $\mathbf{c}^{\mathsf{rdm}}_{A}$;

4.3 Security of Z-Channel Protocol

Due to space limitation, we refer to Appendix B for the security definition and proof.

5 Performance Analysis

We measure the performance of DAP+ scheme and Z-Channel. For the DAP+ scheme, we construct new circuit based on that of ZCash and benchmark the performance of zk-SNARK on key generation, proving and verification. 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+ slightly increases the key sizes and running times.

For Z-Channel, we implement the protocol and benchmark the computation time. The result shows that Z-Channel increases the ledger scalability and reduces the payment confirmation time significantly.

5.1 Instantiation of DAP Plus and Z-Channel

Instantiation of DAP Plus. Our implementation of DAP+ is based on that of ZCash [29], which is the most popular implementation of DAP scheme. ZCash follows the idea of DAP scheme, but modifies the algorithms and data structures dramatically. Despite that, our improvements in DAP+ can be applied directly to ZCash. For details of ZCash we refer interested readers to [29].

We implement the distributed signature generation scheme with EC-Schnorr signature [28]. We take SHA256 as the public key hash function Hash. We compute pkcm with trapdoor a_{sk} (which is 252-bit string in ZCash), by taking the SHA256 compression of their concatenation $a_{\mathsf{sk}} \| \mathsf{pkh}$ prefixed by four zero-bits. The time lock is set 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, we take the following simple procedures:

1. For key generation, Alice generates random big integer a and computes $A = aG$ locally, where G is the generator of the elliptic curve group used in the EC-Schnorr signature scheme, 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; the complete signature is $\sigma = (e, s_1 + s_2)$.

For the consensus of secret seed, assume Alice and Bob have a secure communication channel. Alice and Bob generate random 256-bit strings a and b; Alice commits a to Bob, Bob sends b to Alice, and Alice sends a to Bob; the seed is $\mathsf{seed} = a \oplus b$.

5.2 Performance of Zero-Knowledge Proof in DAP Plus

We construct the circuit of the new NP statement for zk-SNARK based on the code of ZCash. Table 2 shows the performance of the zero-knowledge proof procedures, in comparison with that of the original DAP scheme. The modifications introduced in DAP+ scheme slightly (around 0.1% to 8%) increase the key sizes and the time consumption, as expected.

5.3 Performance of Z-Channel Protocol Between Single Pairs

In testing performance of a single Z-Channel, we run the Z-Channel clients on localhost to minimize the effect of real network latency, and simulate different network latencies. The time for updating the channel is the key in improving

	#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 PK size DAP+ PK size	465 MB 516 MB
DAP KeyGen Time	5	340.44s	6.2270s	348.15s	333.38s	DAP VK size	773 B
DAP+ KeyGen Time	5	367.48s	4.3756s	372.48s	362.76s	DAP+ VK size	932 B

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 2. Performance of Zero-Knowledge Proof

the payment efficiency of Zerocash. As for the establishment and closure procedures, the time is dominated by the ledger confirmation time, which on average is several minutes for Zerocash. Since we have already benchmarked the zero-knowledge procedures, here we only benchmark the computation needed in Z-Channel protocol, to make sure their time consumption is negligible compared to that of zero-knowledge procedure and ledger confirmation. Table 3 shows the result.

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

Table 3. Performance of Z-Channel

6 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 time lock 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. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. (2008)
2. Eyal, I., Gencer, A.E., Sirer, E.G., Van Renesse, R.: Bitcoin-ng: A scalable blockchain protocol. In: 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), USENIX Association (2016) 45–59
3. King, S., Nadal, S.: Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. self-published paper, August **19** (2012)
4. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer (2015) 281–310
5. Kroll, J.A., Davey, I.C., Felten, E.W.: The economics of bitcoin mining, or bitcoin in the presence of adversaries. In: Proceedings of WEIS. Volume 2013., Citeseer (2013)
6. Sompolinsky, Y., Zohar, A.: Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. IACR Cryptology ePrint Archive **2013**(881) (2013)
7. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper **151** (2014)
8. Valenta, L., Rowan, B.: Blindcoin: Blinded, accountable mixes for bitcoin. In: International Conference on Financial Cryptography and Data Security, Springer (2015) 112–126
9. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: Security and Privacy (SP), 2016 IEEE Symposium on, IEEE (2016) 839–858
10. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In: Security and Privacy (SP), 2015 IEEE Symposium on, IEEE (2015) 104–121
11. Reid, F., Harrigan, M.: An analysis of anonymity in the bitcoin system. In: Security and privacy in social networks. Springer (2013) 197–223
12. Heilman, E., Baldimtsi, F., Alshenibr, L., Scafuro, A., Goldberg, S.: Tumblebit: An untrusted tumbler for bitcoin-compatible anonymous payments. IACR Cryptology ePrint Archive **2016** (2016) 575
13. Maxwell, G.: Coinswap: Transaction graph disjoint trustless trading. CoinSwap: Transactiongraphdisjointtrustlesstrading (2013)
14. Ziegeldorf, J.H., Grossmann, F., Henze, M., Inden, N., Wehrle, K.: Coinparty: Secure multi-party mixing of bitcoins. In: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, ACM (2015) 75–86
15. Ruffing, T., Moreno-Sanchez, P., Kate, A.: Coinshuffle: Practical decentralized coin mixing for bitcoin. In: European Symposium on Research in Computer Security, Springer (2014) 345–364
16. Maxwell, G.: Coinjoin: Bitcoin privacy for the real world. In: Post on Bitcoin Forum. (2013)
17. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from bitcoin. In: Security and Privacy (SP), 2013 IEEE Symposium on, IEEE (2013) 397–411
18. Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J.A., Felten, E.W.: Mixcoin: Anonymity for bitcoin with accountable mixes. In: International Conference on Financial Cryptography and Data Security, Springer (2014) 486–504

19. Danezis, G., Fournet, C., Kohlweiss, M., Parno, B.: 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) 27–30
20. Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: IEEE Symposium on Security and Privacy. (2014) 459–474
21. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: Snarks for c: Verifying program executions succinctly and in zero knowledge. In: Advances in Cryptology–CRYPTO 2013. Springer (2013) 90–108
22. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer (2013) 626–645
23. Andresen, G.: Blocksize economics. bitcoinfoundation.org (2014)
24. Jedusor, T.: Mimblewimble, 2016, defunct hidden service
25. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments. (2016)
26. Green, M., Miers, I.: Bolt: Anonymous payment channels for decentralized currencies. Cryptology ePrint Archive, Report 2016/701 (2016) http://eprint.iacr.org/2016/701.
27. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive arguments for a von neumann architecture. IACR Cryptology ePrint Archive **2013** (2013) 879
28. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. In: International Conference on the Theory and Applications of Cryptographic Techniques, Springer (1999) 295–310
29. Hopwood, D., Bowe, S., Hornby, T., Wilcox, N.: Zcash protocol specification (2017)

A Construction of DAP Plus Scheme

We present the full construction of DAP+ here for completeness (the modification is shown in bold font).

A *DAP Plus 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, $\mathbf{c}^{\mathsf{old}}_{1..2}$ represents $\mathbf{c}^{\mathsf{old}}_1, \mathbf{c}^{\mathsf{old}}_2$.

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 $a_{\mathsf{pk}} = \mathsf{PRF}^{\mathsf{addr}}_{a_{\mathsf{sk}}}(0)$. 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 pklk. 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 $\mathsf{addr}_{\mathsf{sk}}$
- *Output:*
  - a pair of public/private keys
  - tuple $(\mathsf{pklk}, \mathsf{pkcm})$

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

For privacy, each generated pkcm must be used only once. It is recommended that a user stores the output tuples $(\mathsf{pklk}, \mathsf{pkcm})$ in a table PKCM. When receiving a coin from the ledger (as described in $\mathsf{ReceivePlus}$ 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 $\mathsf{MintPlus}$ algorithm generates a coin and a mint transaction.

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

The Mint algorithm in Zerocash is invoked to generate a Mint transaction which spends unspent output in basecoin and outputs a coin commitment. $\mathsf{MintPlus}$ modifies the original algorithm, by additionally committing a public key commitment pkcm and a time lock tlk. The other parts of the algorithm are left unmodified. The details of the $\mathsf{MintPlus}$ algorithm are presented in Alg.4.

Algorithm 4: $\mathsf{MintPlus}$ Algorithm

Parse $\mathsf{addr}_{\mathsf{pk}}$ as $(a_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}})$;
Randomly sample a $\mathsf{PRF}^{\mathsf{sn}}$ seed ρ;
Randomly sample the 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 $\mathsf{tx}_{\mathsf{Mint}} := (\mathsf{cm}, v, m, s)$;
Output $\mathbf{c}$ and $\mathsf{tx}_{\mathsf{Mint}}$.

Pour algorithm. The $\mathsf{PourPlus}$ 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. $\mathsf{PourPlus}$ generates two coins and a pour transaction.

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

- A coin $\mathbf{c}$ and the address key $\mathsf{addr}_{\mathsf{sk}}$ 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. $\mathsf{pklk}, \mathsf{sk}$ where sk is private key of pklk and $\mathsf{COMM}_{a_{\mathsf{sk}}}(\mathsf{pklk})$ 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 $(\mathsf{addr}_{\mathsf{pk}}, v, \mathsf{lock})$.

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** $\{\mathsf{CoinWitness}_i = (\mathbf{c}_i^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},i}^{\mathsf{old}}, \mathsf{rt}_i, \mathsf{path}_i, \mathsf{pklk}_i^{\mathsf{old}}, \mathsf{sk}_i)\}_{i=1}^2$
  - **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 $\mathsf{tx}_{\mathsf{Pour}}$

PourPlus algorithm modifies the original Pour algorithm, by publishing the public key lock $\mathsf{pklk}_i^{\mathsf{old}}$ previously committed in each input coin. For each $\mathsf{pklk}_i^{\mathsf{old}}$ append the corresponding signature if needed. The details of the PourPlus algorithm are presented in Alg.5.

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.6.

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 $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$
  - **public key commitment set** PKCM
  - ledger L

[12] 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.

Algorithm 5: 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 $h_i := \mathsf{PRF}^{\mathsf{pk}}_{a_{\mathsf{sk},i}^{\mathsf{old}}}((i-1)\|h_{\mathsf{sig}})$;
- **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 $\mathsf{sn}_i^{\mathsf{old}} := \mathsf{PRF}^{\mathsf{sn}}_{a_{\mathsf{sk},i}^{\mathsf{old}}}(\rho_i^{\mathsf{old}})$;
  - **Compute** $\mathsf{pkh}_i^{\mathsf{old}} := \mathsf{Hash}(\mathsf{pklk}_i^{\mathsf{old}})$;
  - **Compute** $\mathsf{ovd}_i := \mathsf{BH}(\mathsf{rt}_i) + \mathsf{tlk}_i^{\mathsf{old}} \geq \mathsf{MBH}$;
- **end**
- Set $\boldsymbol{x} := (\mathsf{rt}_{1..2}, \mathsf{sn}_{1..2}^{\mathsf{old}}, \mathsf{pkh}_{1..2}^{\mathsf{old}}, \mathsf{cm}_{1..2}^{\mathsf{new}}, v_{\mathsf{pub}}, h_{\mathsf{sig}}, h_{1..2}, \mathsf{MBH}, \mathsf{ovd}_{1..2})$;
- Set $\boldsymbol{a} := (\mathsf{path}_{1..2}, a_{\mathsf{sk},1..2}^{\mathsf{old}}, \mathbf{c}_{1..2}^{\mathsf{old}}, \mathbf{c}_{1..2}^{\mathsf{new}})$;
- Compute $\pi_{\mathsf{POUR}} := \mathsf{Prove}(\mathsf{pk}_{\mathsf{POUR}}, \boldsymbol{x}, \boldsymbol{a})$;

Procedure `PreventForgery()`
- Set $M := (\boldsymbol{x}, \pi_{\mathsf{POUR}}, \mathsf{MBH}, \mathbf{C}_{1..2}, \mathsf{pklk}_{1..2}^{\mathsf{old}})$;
- Compute $\sigma := \mathcal{S}_{\mathsf{sig}}(\mathsf{sk}_{\mathsf{sig}}, M)$;

Procedure `Unlock()`
- **Set** $\mathsf{msg} := \mathsf{ToBeLocked}()$;
- **for** $i \in \{1,2\}$ **do**
  - **if** ovd_i **then**
    - **Compute**[12] $\sigma_i = \mathcal{S}_{\mathsf{dst}}(\mathsf{sk}_i, \mathsf{msg})$;
  - **else**
    - **Set** $\sigma_i = \perp$;
  - **end**
  - **Set** $\mathsf{unlock}_i = (\mathsf{pklk}_i^{\mathsf{old}}, \sigma_i)$;
- **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 $\mathbf{c}_{1..2}^{\mathsf{new}}, \mathsf{tx}_{\mathsf{Pour}}$;

Algorithm 6: VerifyPlus Algorithm

if tx *is of type* $\mathsf{tx}_{\mathsf{Mint}}$ **then**
  Parse $\mathsf{tx}_{\mathsf{Mint}}$ as (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 $(\mathsf{rt}_{1..2}, \mathsf{sn}^{\mathsf{old}}_{1..2}, \mathsf{cm}^{\mathsf{new}}_{1..2}, v_{\mathsf{pub}}, \mathsf{MBH}, *)$ and $*$ as $(\mathsf{pk}_{\mathsf{sig}}, h_{1..2}, \pi_{\mathsf{POUR}}, \mathbf{C}_{1..2}, \sigma, \mathsf{unlock}_{1..2})$;
  If $\mathsf{sn}^{\mathsf{old}}_1$ or $\mathsf{sn}^{\mathsf{old}}_2$ appears on L or $\mathsf{sn}^{\mathsf{old}}_1 = \mathsf{sn}^{\mathsf{old}}_2$, output $b := 0$ and exit;
  If the Merkle tree root rt_1 or 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 unlock_i **as** $(\mathsf{pklk}_i, \sigma_i)$;
    Compute $\mathsf{pkh}^{\mathsf{old}}_i := \mathsf{Hash}(\mathsf{pklk}_i)$;
    Set $\mathsf{ovd}_i := (\sigma_i \neq \perp)$;
    if ovd_i ***and*** $\mathcal{V}_{\mathsf{dst}}(\mathsf{pklk}_i, \mathsf{msg}, \sigma_i) = 0$ **then**
      output $b := 0$ **and exit**
    end
  end
  Set $\boldsymbol{x} := (\mathsf{rt}_{1..2}, \mathsf{sn}^{\mathsf{old}}_{1..2}, \mathsf{pkh}^{\mathsf{old}}_{1..2}, \mathsf{cm}^{\mathsf{new}}_{1..2}, v_{\mathsf{pub}}, h_{\mathsf{sig}}, h_{1..2}, \mathsf{MBH}, \mathsf{ovd}_{1..2})$;
  Set $M := (\boldsymbol{x}, \pi_{\mathsf{POUR}}, \mathsf{MBH}, \mathbf{C}_{1..2}, \mathsf{pklk}^{\mathsf{old}}_1, \mathsf{pklk}^{\mathsf{old}}_2)$;
  If $\mathcal{V}_{\mathsf{sig}}(\mathsf{pk}_{\mathsf{sig}}, M, \sigma) = 0$ output $b := 0$ and exit;
  If $\mathsf{Verify}(\mathsf{vk}_{\mathsf{POUR}}, \boldsymbol{x}, \pi_{\mathsf{POUR}}) = 0$ output $b := 0$ and exit;
  Set $\mathsf{msg} := \mathsf{ToBeLocked}()$;
  Output $b := 1$;
end

- *Output:* set of received coins

ReceivePlus modifies the original Receive algorithm, by additionally checking that the public key commitment pkcm is previously generated by CreatePKCM and never used before. The details of the ReceivePlus algorithm are presented in Alg.7.

Algorithm 7: 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 lock **as** $(\mathsf{pkcm}, \mathsf{tlk})$;
      Check that 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

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

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

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

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

such that:

- For each $i \in \{1, 2\}$:
  - The path_i is a valid authentication path for leaf $\mathsf{cm}^{\mathsf{old}}_i$ with respect to root rt_i, in a CRH-based Merkle tree.
  - The private key $a^{\mathsf{old}}_{\mathsf{sk},i}$ matches the public address of $a^{\mathsf{old}}_{\mathsf{pk},i}$.
  - The serial number $\mathsf{sn}^{\mathsf{old}}_i$ is computed correctly, i.e. $\mathsf{sn}^{\mathsf{old}}_i = \mathsf{PRF}^{\mathsf{sn}}_{a^{\mathsf{old}}_{\mathsf{sk},i}}(\rho^{\mathsf{old}}_i)$.
  - **The coin $\mathbf{c}^{\mathsf{old}}_i$ is well formed, i.e.** $\mathsf{cm}^{\mathsf{old}}_i = \mathsf{COMM}_{s^{\mathsf{old}}_i}(v^{\mathsf{old}}_i, \mathsf{COMM}_{r^{\mathsf{old}}_i}(a^{\mathsf{old}}_{\mathsf{pk},i}, \rho^{\mathsf{old}}_i, \mathsf{COMM}_{a^{\mathsf{old}}_{\mathsf{sk},i}}(\mathsf{pkh}^{\mathsf{old}}_i), \mathsf{tlk}^{\mathsf{old}}_i))$.

- **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 time lock expires or is overridden, i.e.** $\mathsf{ovd}_i||(\mathsf{BH}(\mathsf{rt}_i) + \mathsf{tlk}_i^{\mathsf{old}}[k_i] < \mathsf{MBH})$.

- Balance is preserved: $v_1^{\mathsf{new}} + v_2^{\mathsf{new}} + v_{\mathsf{pub}} = v_1^{\mathsf{old}} + v_2^{\mathsf{old}}$

B 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 section C. The definitions of the two properties *currency security* and *channel privacy* are presented in subsection B.1 and B.2.

Our main theorem claims that the construction of Z-Channel Protocol in section 4 is secure.

Theorem 1. *The protocols presented in Alg.1, Alg.2 and Alg.3 form a secure Z-Channel Protocol.*

Proof: By Lemma 1, the protocols satisfy currency security. By Lemma 2, 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.

B.1 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 section 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 $\mathsf{CreateAddress}$ 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 $v_{A,i+1}$ [13]. 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.

Lemma 1. *The protocols presented in Alg.1, Alg.2 and Alg.3 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 4 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 section C), which is negligible by Theorem 2.

[13] 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.

B.2 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 2. *The protocols presented in Alg.1, Alg.2 and Alg.3 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 $\mathsf{L-IND}$ game (which is used to define the ledger-indistinguishability of DAP+, see section C). As a result, the probability that $\mathcal{A}$ wins this game is bound by the probability that $\mathcal{A}$ wins $\mathsf{L-IND}$ game which is negligible by Theorem 2.

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 section.

C Completeness and Security of DAP+ Scheme

In this section, 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 [20]. 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 $\mathsf{L-IND}$, $\mathsf{TR-NM}$ 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 $\mathsf{L}-\mathsf{IND}$ 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 $\mathsf{TR}-\mathsf{NM}$ 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.

Regarding the experiments $\mathsf{L}-\mathsf{IND}$, $\mathsf{TR}-\mathsf{NM}$ and BAL, we design them similarly to those in [20], and the major modifications are listed below.

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

- $Q = (\mathbf{CreatePKCM}, K)$
  (a) Invoke $\mathsf{CreatePKCM}(\mathsf{pp})$ to obtain the tuple $(\mathsf{sk}, \mathsf{pklk}, \mathsf{pkcm})$.
  (b) Store $(\mathsf{sk}, \mathsf{pklk}, \mathsf{pkcm})$ 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 time lock 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 time lock 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 time lock $\mathsf{tlk}_i^{\mathsf{new}}$.
- If the address is in ADDR, $\mathcal{O}^{DAP+}$ checks that $\mathsf{pkcm}_i^{\mathsf{new}}$ is in PKCM and not in $\mathsf{OLDPKCM}$, 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 $\mathsf{OLDPKCM}$, 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 $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{sk}, \mathsf{pklk}, \mathsf{pkcm})$ in OLDPKCM.
- For Pour query, $\mathcal{O}$ looks up the table OLDPKCM to find the tuple $(\mathsf{addr}^{\mathsf{old}}_{\mathsf{pk},i}, \mathsf{sk}^{\mathsf{old}}_i, \mathsf{pklk}^{\mathsf{old}}_i, \mathsf{pkcm}^{\mathsf{old}}_i)$ for each $\mathsf{addr}^{\mathsf{old}}_{\mathsf{pk},i}$, include $\mathsf{pklk}^{\mathsf{old}}_i$ in the pour transaction $\mathsf{tx}_{\mathsf{Pour}}$. If ovd_i is false, $\mathcal{O}$ checks that $\mathsf{tlk}^{\mathsf{old}}_i$ 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 $\mathsf{pklk}^{\mathsf{old}}_i$ 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 $\mathsf{L}-\mathsf{IND}$ 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 $\mathsf{L}-\mathsf{IND}$ 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 2. *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 [20]. Here we only present the modifications to the original one.

Modify the simulation experiment The simulated experiment $\Im_{\mathbf{sim}}$ proceed as in [20], except for the following modification:

1. **Answering** CreatePKCM **queries.** To answer Q, $\mathcal{C}$ behaves as in $\mathsf{L}-\mathsf{IND}$, except for the following modification: after obtaining $(\mathsf{sk}, \mathsf{pklk}, \mathsf{pkcm})$, $\mathcal{C}$ 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 $m = \mathsf{COMM}_r(a_{\mathsf{pk}}, \rho, \mathsf{pkcm}, \mathsf{tlk})$. Afterwards, $\mathcal{C}$ does the same for Q'.

   Remark 1. There is no need to modify the Pour queries except for the modifications mentioned in [20], which already discard the information of pkcm and tlk 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 pklk looked up from 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 $\mathsf{pkcm}_i^{\mathsf{old}}$.

Difference between $\Im_{\mathbf{sim}}$ and hybrid experiment $\Im_3$ Let $q_{\mathbf{CP}}$ be the total number of CreatePKCM queries issued by $\mathcal{A}$. In addition to those described in [20] appendix D.1, we additionally let the experiment $\Im_{\mathbf{sim}}$ modifies $\Im_3$ 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 [20] appendix D.1 as follows:

$$\left|\mathbf{Adv}^{\Im_{\mathbf{sim}}} - \mathbf{Adv}^{\Im_3}\right| \leq (q_{\mathbf{M}} + 4 \cdot q_{\mathbf{P}} + q_{\mathbf{CP}}) \cdot \mathbf{Adv}^{\mathsf{COMM}}$$

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 [20].

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, $\mathbf{Adv}_{\Pi,\mathcal{A}}^{\mathsf{INCOMP}}(\lambda) < negl(\lambda)$, where $\mathbf{Adv}_{\Pi,\mathcal{A}}^{\mathsf{INCOMP}}(\lambda)$ $:= 2 \cdot Pr[\mathsf{INCOMP}\ (\Pi, \mathcal{A}, \lambda) = 1] - 1$ 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* $\mathsf{L}-\mathsf{IND}$ *secure, if for every poly(λ)-size adversary $\mathcal{A}$ and sufficiently large λ, $\mathbf{Adv}_{\Pi,\mathcal{A}}^{\mathsf{L-IND}}(\lambda) < negl(\lambda)$, where $\mathbf{Adv}_{\Pi,\mathcal{A}}^{\mathsf{L-IND}}(\lambda) :=$ $2 \cdot Pr[\mathsf{L}-\mathsf{IND}(\ \Pi, \mathcal{A}, \lambda) = 1] - 1$ is $\mathcal{A}$'s advantage in the* $\mathsf{L}-\mathsf{IND}$ *experiment.*

Definition 9. *We say that a DAP+ scheme* $\Pi = ($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, $\mathbf{Adv}^{\mathsf{TR-NM}}_{\Pi,\mathcal{A}}(\lambda) < negl(\lambda)$, *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* $\Pi = ($Setup, CreatePKCM, CreateAddress, MintPlus, PourPlus, VerifyPlus, ReceivePlus*) is* BAL *secure, if for every poly(*λ*)-size adversary* $\mathcal{A}$ *and sufficiently large* λ, $\mathbf{Adv}^{\mathsf{BAL}}_{\Pi,\mathcal{A}}(\lambda) < negl(\lambda)$, *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 [20] 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 $(\mathsf{sk}, \mathsf{pklk}, \mathsf{pkcm})$;
(v) OLDPKCM, a set of tuples of $(\mathsf{addr_{pk}}, \mathsf{pkcm}, \mathsf{pklk})$.

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 pklk (with secret key list being *sklist*) of size K.
3. Compute $\mathsf{pkcm} = \mathsf{COMM}_u(\mathsf{Hash}(\mathsf{pklk}))$.
4. Store $(\mathsf{sk}, \mathsf{pklk}, \mathsf{pkcm})$ in table PKCM.
5. Output pkcm.

Other internal storages apart from PKCM stay unchanged.

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

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

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

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 $(\mathsf{addr}^{\mathsf{old}}_{\mathsf{pk},i}, \mathsf{sk}_i, \mathsf{pklk}^{\mathsf{old}}_i, \mathsf{pkcm}^{\mathsf{old}}_i)$ 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 tlk_i be the time lock 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 $\mathsf{BH}(rt_i) + \mathsf{tlk}_i < \mathsf{MBH}$.
   (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 $(\mathsf{pklk}^{\mathsf{new}}_i, u^{new}_i, \mathsf{pkcm}^{\mathsf{new}}_i)$ be the tuple found in PKCM. Remove $\mathsf{pkcm}^{\mathsf{new}}_i$ from PKCM and stores $(\mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},i}, \mathsf{sk}_i, \mathsf{pklk}^{\mathsf{new}}_i, \mathsf{pkcm}^{\mathsf{new}}_i)$ 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 $(\mathbf{c}^{\mathsf{new}}_1, \mathbf{c}^{\mathsf{new}}_2, \mathsf{tx}_{\mathsf{Pour}}) := \mathsf{Pour}(\mathsf{pp}, v_{\mathsf{pub}}, \mathbf{c}^{\mathsf{old}}_{1..2}, \mathsf{addr}^{\mathsf{old}}_{\mathsf{sk},1..2}, \mathsf{rt}_{1..2}, \mathsf{path}_{1..2}, \mathsf{pklk}^{\mathsf{old}}_{1..2}, \mathsf{sk}_{1..2}, \mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},1..2}, v^{\mathsf{new}}_{1..2}, \mathsf{lock}^{\mathsf{new}}_{1..2})$.
4. Verify that $\mathsf{Verify}(\mathsf{pp}, \mathsf{tx}_{\mathsf{Pour}}, L)$ outputs 1.
5. Add the coins $\mathbf{c}^{\mathsf{new}}_{1..2}$ to COIN.
6. Add the pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ to L.
7. Output $\perp$.

If any of the above operations fail, the output is $\perp$ (and L, ADDR, COIN, PKCM, OLDPKCM remain unchanged).
$Q = (\mathsf{Insert}, \mathsf{tx})$

1. Verify that $\mathsf{Verify}(\mathsf{pp}, \mathsf{tx}, L)$ outputs 1. (Else, abort.)

2. Add the mint/pour transaction tx to L.
3. Run $\mathsf{ReceivePlus}$ for all addresses $\mathsf{addr_{pk}}$ in ADDR;
4. For each output coin from $\mathsf{ReceivePlus}$
   (a) Let pkcm be the public key commitment stored in it.
   (b) Let $(\mathsf{sk}, \mathsf{pklk}, \mathsf{pkcm})$ be the first tuple in PKCM with the public key commitment pkcm (if not exists, aborts).
   (c) Remove this tuple from PKCM;
   (d) Add $(\mathsf{addr_{pk}}, \mathsf{sk}, \mathsf{pklk}, \mathsf{pkcm})$ to $\mathsf{OLDPKCM}$.
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 [20] 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 $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, 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 $(L_{\mathsf{left}}, L_{\mathsf{right}})$, 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 diffrent 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 $\mathsf{CreatePKCM}$ 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 $\mathsf{CreateAddress}$, 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 $\mathsf{CreatePKCM}$, 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 $\mathsf{addr}_{\mathsf{pk},1}$ and $\mathsf{addr}_{\mathsf{pk},2}$ 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 time locks 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').

D Proof of Security

Here we present the complete proof of Theorem 2. The proofs to transaction non-malleability and balance are trivially similar to the ones in [20], we omit them here. For proof of ledger indistingsuishability, we construct a simulation $\ni_{\mathbf{sim}}$ in which the adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$, as in the $\mathsf{L}-\mathsf{IND}$ experiment. However $\ni_{\mathbf{sim}}$ modifies the $\mathsf{L}-\mathsf{IND}$ 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 $\ni_{\mathbf{sim}}$ is 0. Then we show that $\mathbf{Adv}_{\Pi,\mathcal{A}}^{\mathsf{L}-\mathsf{IND}}(\lambda)$ is only negligibly larger than $\mathcal{A}$'s advantage in $\ni_{\mathbf{sim}}$.

The simulation experiment. The simulation $\ni_{\mathbf{sim}}$ works as follows. First, $\mathcal{C}$ samples $b \in \{0, 1\}$ and $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, with the following modifications: the zk-SNARK keys are generated by $(\mathsf{pk}_{\mathsf{POUR}}, \mathsf{vk}_{\mathsf{POUR}}, \mathsf{trap}) \leftarrow \mathsf{Sim}(1^\lambda, C_{\mathsf{POUR}})$, 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, $\ni_{\mathbf{sim}}$ proceeds in steps and at each step $\mathcal{C}$ present $\mathcal{A}$ two ledgers $(L_{\mathsf{left}}, L_{\mathsf{right}})$, 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 $\mathsf{L}-\mathsf{IND}$. The reaction of challenger $\mathcal{C}$ is different from that in $\mathsf{L}-\mathsf{IND}$, as described as follows:

1. **Answering** CreateAddress **queries**. In this case, $Q = Q' = \mathsf{CreateAddress}$. To answer Q, $\mathcal{C}$ behaves as in $\mathsf{L}-\mathsf{IND}$, except for the following modification: after obtaining $(\mathsf{addr_{pk}}, \mathsf{addr_{sk}}) \leftarrow \mathsf{CreateAddress}(\mathsf{pp})$, $\mathcal{C}$ 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, $Q = Q' = \mathsf{CreatePKCM}$. To answer Q, $\mathcal{C}$ behaves as in $\mathsf{L}-\mathsf{IND}$, except for the following modification: after obtaining $(\mathsf{sk}, \mathsf{pklk}, \mathsf{pkcm})$, $\mathcal{C}$ 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'.
3. **Answering** Mint **queries**. In this case, $Q = (\mathsf{Mint}, v, \mathsf{addr_{pk}})$ and $Q' = (\mathsf{Mint}, v, \mathsf{addr'_{pk}})$. To answer Q, $\mathcal{C}$ behaves as in $\mathsf{L}-\mathsf{IND}$, except for the following modification: Compute $m = \mathsf{COMM}_r(\tau)$ for a random string τ of the suitable length, instead of $m = \mathsf{COMM}_r(a_{\mathsf{pk}}, \rho, \mathsf{pklk}, \mathsf{tlk})$. Afterwards, $\mathcal{C}$ does the same for Q'.
4. **Answering** Pour **queries**. In this case, Q and Q' both have the form $(\mathsf{Pour}, \mathsf{idx}^{\mathsf{old}}_{1..2}, \mathsf{addr}^{\mathsf{old}}_{\mathsf{pk},1..2}, \mathsf{ovd}_{1..2}, v^{\mathsf{new}}_{1..2}, \mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},1..2}, \mathsf{lock}^{\mathsf{new}}_{1..2}, v_{\mathsf{pub}})$. To answer Q, $\mathcal{C}$ modifies in the following ways:
   (a) For each $j \in \{1, 2\}$:
      i. Uniformly sample random $\mathsf{sn}^{\mathsf{old}}_j$.
      ii. Randomly sample a list of pairs of public/private keys pklk_j, compute $\mathsf{pkh}^{\mathsf{old}}_j := \mathsf{Hash}(\mathsf{pklk}_j)$.
      iii. If $\mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},j}$ is in ADDR:
         A. sample a coin commitment $\mathsf{cm}^{\mathsf{new}}_j$ on a random input;
         B. run $\mathcal{K}_{\mathsf{enc}}(\mathsf{pp}_{\mathsf{enc}}) \rightarrow (\mathsf{pk}_{\mathsf{enc}}, \mathsf{sk}_{\mathsf{enc}})$ and compute $\mathbf{C}^{\mathsf{new}}_j := \mathcal{E}_{\mathsf{enc}}(\mathsf{pk}_{\mathsf{enc}}, r)$ for a random r of suitable length.
      iv. Otherwise, calculate $(\mathsf{cm}^{\mathsf{new}}_j, \mathbf{C}^{\mathsf{new}}_j)$ 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 $\pi_{\mathsf{POUR}} := \mathsf{Sim}(\mathsf{trap}, \boldsymbol{x})$, where $\boldsymbol{x} := (\mathsf{rt}_{1..2}, \mathsf{sn}^{\mathsf{old}}_{1..2}, \mathsf{pkh}^{\mathsf{old}}_{1..2}, \mathsf{cm}^{\mathsf{new}}_{1..2}, v_{\mathsf{pub}}, h_{\mathsf{sig}}, h_{1..2}, \mathsf{MBH}, \mathsf{ovd}_{1..2})$.
   Afterwards, $\mathcal{C}$ does the same for Q'.
5. Answering Insert queries. In this case, $Q = (\mathsf{Insert}, \mathsf{tx})$ and $Q = (\mathsf{Insert}, \mathsf{tx}')$. The answer to each query proceeds as in the $\mathsf{L}-\mathsf{IND}$ 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 $\mathfrak{I}_{\mathbf{sim}}$ is 0.

Indistinguishability from Real Experiment.

We construct a sequence of hybrid experiments $(\mathfrak{I}_{\mathbf{real}}, \mathfrak{I}_1, \mathfrak{I}_2, \mathfrak{I}_3, \mathfrak{I}_{\mathbf{sim}})$, in each of these experiments a challenger $\mathcal{C}$ conducts a different modification of the $\mathsf{L}-\mathsf{IND}$ experiment. We define $\mathfrak{I}_{\mathbf{real}}$ to be the original $\mathsf{L}-\mathsf{IND}$ experiment, and $\mathfrak{I}_{\mathbf{sim}}$ to be the simulation described above. Given experiment $\mathfrak{I}$, we define $\mathbf{Adv}^{\mathfrak{I}}$

to be the absolute value of the difference between the $\mathsf{L}-\mathsf{IND}$ advantage of $\mathcal{A}$ in $\ni$ and that in $\ni_{\mathbf{real}}$. 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 $\ni_1$. The experiment $\ni_1$ modifies $\ni_{\mathbf{real}}$ by simulating the zk-SNARKs. More precisely, we modify $\ni_{\mathbf{real}}$ so that $\mathcal{C}$ simulates each zk-SNARK proof, as follows. At the beginning of the experiment, instead of invoking $\mathsf{KeyGen}(1^\lambda, C_{\mathsf{POUR}})$, $\mathcal{C}$ invokes $\mathsf{Sim}(1^\lambda, C_{\mathsf{POUR}})$ and obtains ($\mathsf{pk}_{\mathsf{POUR}}$, $\mathsf{vk}_{\mathsf{POUR}}$, trap). At each subsequent invocation of the Pour algorithm, $\mathcal{C}$ computes $\pi_{\mathsf{POUR}} \leftarrow \mathsf{Sim}(\mathsf{trap}, x)$, 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 $\ni_{\mathbf{real}}$. Hence $\mathbf{Adv}^{\ni_1} = 0$.
2. Experiment $\ni_2$. The experiment $\ni_2$ modifies $\ni_1$ by replacing the ciphertexts in a pour transaction by encryptions of random strings. Each time $\mathcal{A}$ issues a Pour query where one of $(\mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},1}, \mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},2})$ is in ADDR, the ciphertexts $\mathbf{C}^{\mathsf{new}}_1$, $\mathbf{C}^{\mathsf{new}}_2$ are generated as follows:
   (a) $(\mathsf{pk}^{\mathsf{new}}_{\mathsf{enc}}, \mathsf{sk}^{\mathsf{new}}_{\mathsf{enc}}) \leftarrow \mathcal{K}_{\mathsf{enc}}(\mathsf{pp}_{\mathsf{enc}})$;
   (b) for each $j \in \{1,2\}$, $\mathbf{C}^{\mathsf{new}}_j := \mathcal{E}_{\mathsf{enc}}(\mathsf{pk}^{\mathsf{new}}_{\mathsf{enc}}, j, r)$ where r is a message randomly and uniformly sampled from plaintext space.
   By Lemma 3, $\left|\mathbf{Adv}^{\ni_2} - \mathbf{Adv}^{\ni_1}\right| \leq 4 \cdot q_{\mathbf{P}} \cdot \mathbf{Adv}^{\mathbf{Enc}}$.
3. Experiment $\ni_3$. The experiment $\ni_3$ modifies $\ni_2$ 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 $\mathsf{sn}^{\mathsf{old}}_1$, $\mathsf{sn}^{\mathsf{old}}_2$ 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 4, $\left|\mathbf{Adv}^{\ni_3} - \mathbf{Adv}^{\ni_2}\right| \leq q_{\mathbf{CA}} \cdot \mathbf{Adv}^{\mathsf{PRF}}$
4. Experiment $\ni_{\mathbf{sim}}$. The experiment $\ni_{\mathbf{sim}}$ is already described above. For comparison, we explain how it differs from $\ni_3$: 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 $\mathsf{addr}^{\mathsf{new}}_{\mathsf{pk},j}$ is in ADDR, $\mathsf{cm}^{\mathsf{new}}_j$ is substituted with a commitment to a random input.

By Lemma 5, $\left|\mathbf{Adv}^{\eth_{\mathbf{sim}}} - \mathbf{Adv}^{\eth_3}\right| \leq (q_{\mathbf{M}} + 4 \cdot q_{\mathbf{P}} + q_{\mathbf{CP}}) \cdot \mathbf{Adv}^{\mathsf{COMM}}$

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

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

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

Below, we sketch proofs for the lemmas used above.

Lemma 3. *Let AdvEnc be the maximum of: $\mathcal{A}$'s advantage in the IND-CCA experiment against the encryption scheme* $\mathbf{Enc}$*, and $\mathcal{A}$'s advantage in the IK-CCA experiment against the encryption scheme* $\mathbf{Enc}$*. 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 3 is exactly the same to the proof of Lemma D.1 in [20], so we omit it here.

Lemma 4. *Let* $\mathbf{Adv}^{\mathsf{PRF}}$ *be $\mathcal{A}$'s advantage in distinguishing the pseudorandom function* PRF *from a random function. Then, after $q_{\mathbf{CA}}$* $\mathsf{CreateAddress}$ *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 $\mathbf{H}$, 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 $\mathsf{addr}_{\mathsf{pk}} = (a_{\mathsf{pk}}, \mathsf{pk}_{\mathsf{enc}})$ with $\mathsf{addr}_{\mathsf{pk}} = (\tau, \mathsf{pk}_{\mathsf{enc}})$ where τ is a random string of the appropriate length. On each subsequent Pour query $\mathsf{tx}_{\mathsf{Pour}}$, for each $i \in 1, 2$, if $\mathsf{addr}^{\mathsf{old}}_{\mathsf{pk},i} = \mathsf{addr}_{\mathsf{pk}}$ 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 $\eth_2$. Let a_{sk} be the secret key generated by the oracle in the first $\mathsf{CreateAddress}$ query. In $\eth_2$ (as in $\eth_{\mathbf{real}}$):

1. $a_{\mathsf{pk}} := \mathsf{PRF}^{\mathsf{addr}}_{a_{\mathsf{sk}}}(0)$;
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 $\ni_2$ 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 $\ni_2$; 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 $\ni_2$ to $\ni_3$ is at most $q_{\mathbf{CA}} \cdot \mathbf{Adv}^{\mathsf{PRF}}$. The final hybrid is equal to $\ni_3$, we obtain that $\left|\mathbf{Adv}^{\ni_3} - \mathbf{Adv}^{\ni_2}\right| \leq q_{\mathbf{CA}} \cdot \mathbf{Adv}^{\mathsf{PRF}}$.

Lemma 5. *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,* $\left|\mathbf{Adv}^{\ni_{\mathbf{sim}}} - \mathbf{Adv}^{\ni_3}\right| \leq (q_{\mathbf{M}} + 4 \cdot q_{\mathbf{P}} + q_{\mathbf{CP}}) \cdot \mathbf{Adv}^{\mathsf{COMM}}$.

Proof sketch. We only provide a short sketch, because the structure of the argument is similar to the one used to prove Lemma 4 above.

For the first Mint or Pour query, replace the "internal" commitment $m := \mathsf{COMM}_r(a_{\mathsf{pk}}, \rho, \mathsf{pklk}, \mathsf{tlk})$ 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 $\ni_2$ 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 $\ni_{\mathbf{sim}}$ and get that $\left|\mathbf{Adv}^{\ni_{\mathbf{sim}}} - \mathbf{Adv}^{\ni_3}\right| \leq (q_{\mathbf{M}} + 4 \cdot q_{\mathbf{P}} + q_{\mathbf{CP}}) \cdot \mathbf{Adv}^{\mathsf{COMM}}$.