MANTA: a Plug and Play Private DeFi Stack

Shumo Chu[1,2], Yu Xia[3], and Zhenfei Zhang[2] *

[1] UC Santa Barbara
[2] Manta Network
{shumo,zhenfei}@manta.network
[3] MIT CSAIL
yuxia@mit.edu

June 3, 2021

Abstract. We propose MANTA, a plug and play private DeFi stack that consists of $\text{MANTA}_{\texttt{DAP}}$, a multi-asset decentralized anonymous payment scheme and $\text{MANTA}_{\texttt{DAX}}$, an automated market maker (AMM) based decentralized anonymous exchange scheme. Compared with existing privacy preserving cryptocurrencies such as Zcash and Monero, MANTA supports multiple base assets and allows the privatized assets to be exchanged anonymously via $\text{MANTA}_{\texttt{DAX}}$. We think this is a major step forward towards building a privacy preserving DeFi stack. Thanks to the efficiency of modern NIZKs (non-interactive zero-knowledge proof systems) and our carefully crafted design, MANTA is efficient: our benchmarks reports a 15 second, off-line zero-knowledge proof (ZKP) generation time, and a 6 millisecond, on-line proof verification time.

1 Introduction

In an ideal world, cryptocurrencies create a decentralized token economy that protects user privacy. The popular cryptocurrencies that use permissionless consensus protocols, such as Bitcoin [Nak], Ethereum [Woo], and Polkadot [dot], are pseudo-anonymous: although public keys are not explicitly attached to real-world identities, the transaction history is public, which means any leakage of the link between users' identities and public addresses are permanent and can be revealed by link analysis on transaction history. [KYMM18, YKM19, TBP20, GKRN18].

To solve this privacy issue, several privacy-preserving protocols have been propose to solve this privacy issue, most notably ring signature based Monero [vS13], and non-interactive zero-knowledge proof (NIZK) based Zcash [BCG+14]. While these cryptocurrencies provide various degrees of privacy to end users, there are two remaining issues of these protocols. First, they are *mono-asset* private cryptocurrencies: they have either a single kind of fungible tokens (Monero) or can only privatize a single kind of fungible tokens (Zcash). These protocols cannot provide privacy off-the-shelf for the wide range of emerging cryptocurrencies. Second, these protocols lack the extension to decentralized finance (DeFi) [def21], an ever popular application of cryptocurrencies. DeFi brings innovative permissionless financial tools to end users who don't have access to typical Wall Street tools and assets. Most significant DeFi applications include decentralized exchanges [AZR20], synthetic assets [BJSW18], and crypto loan [LH19].

In this paper, we propose MANTA, a protocol for private DeFi stack that solves both the issues of supporting arbitrary private assets and supporting DeFi infrastructure. At its core, MANTA contains two layers: $\text{MANTA}_{\texttt{DAP}}$, a *multi-asset* anonymous payment protocol, and $\text{MANTA}_{\texttt{DAX}}$, an automated market maker (AMM) styled anonymous exchange protocol built on top of $\text{MANTA}_{\texttt{DAP}}$. MANTA protocols can be implemented in any multi-asset blockchain systems, or implemented as a parachain in a Polkadot like relay-chain/parachain systems. Figure 1 shows the architecture of $\text{MANTA}_{\texttt{DAX}}$ when implemented as a parachain.

To achieve provable privacy for end-users, both $\text{MANTA}_{\texttt{DAP}}$ and $\text{MANTA}_{\texttt{DAX}}$ use various cryptographic primitives such as authenticated encryption, efficient commitment schemes for group elements, in conjunction with non-interactive zero-knowledge proof systems (NIZKs) [GGPR13, PHGR13, BCG+13, Gro16]. We formalize the privacy guarantee and prove both the soundness and privacy of MANTA.

[*] Authors are ordered alphabetically.

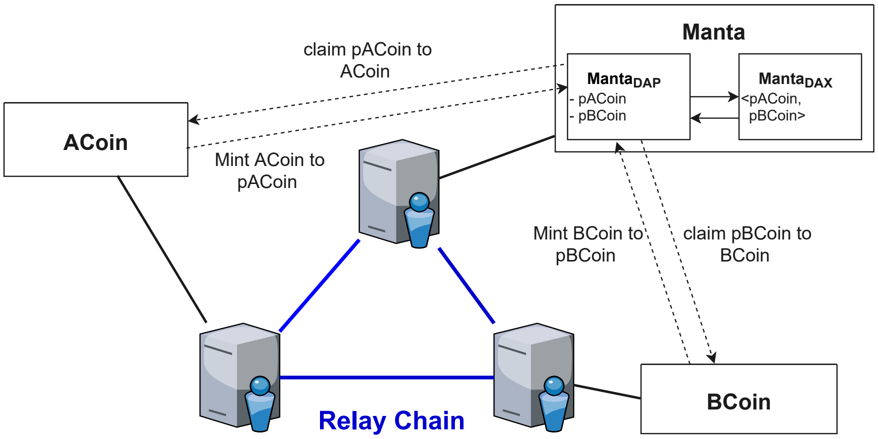

Fig. 1: Manta architecture (implemented as a parachain)

1.1 Our Contributions

- We propose Manta, the first privacy preserving DeFi stack based on NIZK. Manta consists of two layers: Manta$_{\texttt{DAP}}$, which allows users to mint and transfer assets of their choice to private tokens, and Manta$_{\texttt{DAX}}$, which allows users to exchange the private tokens anonymously.
- We propose Manta$_{\texttt{DAP}}$, a multi-asset decentralized anonymous payment protocol that hides addresses, transaction amounts, and asset types. Compared with existing solutions such as ZCash, Manta$_{\texttt{DAP}}$'s major innovation is the support of multiple, "bring-your-own" assets and the design of hidding asset IDs during the transactions. Since Manta$_{\texttt{DAP}}$ aggregates the transaction volumes of all assets, every asset transaction receives better privacy, especially the long tailed ones.
- We propose Manta$_{\texttt{DAX}}$, an AMM [But18] based decentralized anonymous exchange protocol. We choose AMM due to its simplicity: this simplicity leads to high gas efficiency and high capital efficiency both in theory and practice [AC20, AZR20, AEC20]. Manta$_{\texttt{DAX}}$ guarantees the privacy of both traders and liquidity providers. Additionally, the liquidity token that the liquidity provider gains is also private and tradable. This could further increase Manta$_{\texttt{DAX}}$'s capital efficiency [AEC21].
- We give a formal definition of the soundness, completeness, and privacy of both Manta$_{\texttt{DAP}}$ and Manta$_{\texttt{DAX}}$, and prove the security of our scheme.

1.2 How does Manta work?

We use a toy example to demonstrate a typical Manta workflow. Figure 2 shows the workflow of using Manta for private payment and exchange. In this example, Alice starts with a wallet, which holds 20 $CoinA$ under a public address $\texttt{PAddr}_1$, and a private coin $pCoin_0$, with a face value of 30 $CoinA$.

Mint. Now, Alice wants to mint her public coin $CoinA_1$, under a public address $\texttt{PAddr}_1$, to a private coin $pCoin_1$. Alice generates a private coin $pCoin_1$ in her wallet. $pCoin_1$ consists of the following parts:

- $\texttt{SAddr}_1$, a secretive address that is never revealed. This is similar to "Shielded Address" in ZCash.
- $\texttt{cm}_1$, a commitment to asset identifier ($CoinA$), amount (20), void number ($\texttt{vn}_1$, see the description next), and some auxiliary data.
- $\texttt{vn}_1$, a void number that is unique to $pCoin_1$ and is only revealed when the $pCoin_1$ is spent (either transferred, reclaimed to public coin, or exchanged).
- $CoinA$, the asset identifier of this private coin.
- 20, the nomination of this private coin.

Then, Alice sends a $\mathsf{tx}_{\mathsf{mint}}$ that contains the public address ($\texttt{PAddr}_{A_1}$), the commitment to the private coin ($\texttt{cm}_1$), and the mint amount (20). Upon receiving $\mathsf{tx}_{\mathsf{mint}}$, the ledger will update its state by including the commitment $\texttt{cm}_1$ to its $\texttt{CMList}$, and increase the pool size by 20.

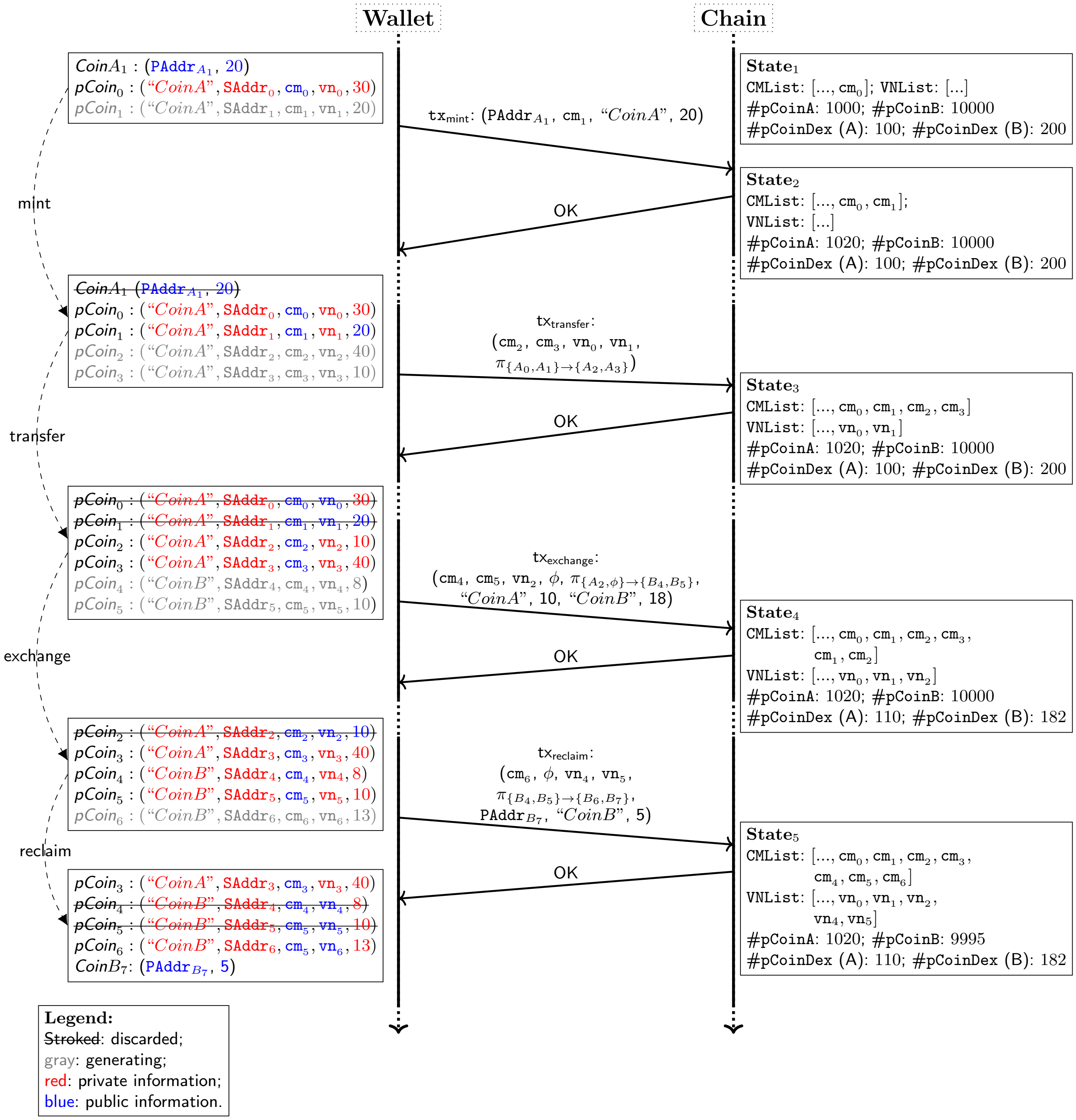
Wallet
Chain
$CoinA_1$: ($PAddr_{A_1}$, 20)
$pCoin_0$: ("CoinA", $SAddr_0$, cm_0, vn_0, 30)
$pCoin_1$: ("CoinA", $SAddr_1$, cm_1, vn_1, 20)
tx_{mint}: ($PAddr_{A_1}$, cm_1, "CoinA", 20)
$State_1$
CMList: [..., cm_0]; VNList: [...]
#pCoinA: 1000; #pCoinB: 10000
#pCoinDex (A): 100; #pCoinDex (B): 200
mint
OK
$State_2$
CMList: [..., cm_0, cm_1];
VNList: [...]
#pCoinA: 1020; #pCoinB: 10000
#pCoinDex (A): 100; #pCoinDex (B): 200
~~$CoinA_1$ ($PAddr_{A_1}$, 20)~~
$pCoin_0$: ("CoinA", $SAddr_0$, cm_0, vn_0, 30)
$pCoin_1$: ("CoinA", $SAddr_1$, cm_1, vn_1, 20)
$pCoin_2$: ("CoinA", $SAddr_2$, cm_2, vn_2, 40)
$pCoin_3$: ("CoinA", $SAddr_3$, cm_3, vn_3, 10)
$tx_{transfer}$:
(cm_2, cm_3, vn_0, vn_1,
$\pi_{\{A_0,A_1\}\to\{A_2,A_3\}}$)
transfer
OK
$State_3$
CMList: [..., cm_0, cm_1, cm_2, cm_3]
VNList: [..., vn_0, vn_1]
#pCoinA: 1020; #pCoinB: 10000
#pCoinDex (A): 100; #pCoinDex (B): 200
~~$pCoin_0$: ("CoinA", $SAddr_0$, cm_0, vn_0, 30)~~
~~$pCoin_1$: ("CoinA", $SAddr_1$, cm_1, vn_1, 20)~~
$pCoin_2$: ("CoinA", $SAddr_2$, cm_2, vn_2, 10)
$pCoin_3$: ("CoinA", $SAddr_3$, cm_3, vn_3, 40)
$pCoin_4$: ("CoinB", $SAddr_4$, cm_4, vn_4, 8)
$pCoin_5$: ("CoinB", $SAddr_5$, cm_5, vn_5, 10)
$tx_{exchange}$:
(cm_4, cm_5, vn_2, ϕ, $\pi_{\{A_2,\phi\}\to\{B_4,B_5\}}$,
"CoinA", 10, "CoinB", 18)
exchange
OK
$State_4$
CMList: [..., cm_0, cm_1, cm_2, cm_3,
cm_4, cm_5]
VNList: [..., vn_0, vn_1, vn_2]
#pCoinA: 1020; #pCoinB: 10000
#pCoinDex (A): 110; #pCoinDex (B): 182
~~$pCoin_2$: ("CoinA", $SAddr_2$, cm_2, vn_2, 10)~~
$pCoin_3$: ("CoinA", $SAddr_3$, cm_3, vn_3, 40)
$pCoin_4$: ("CoinB", $SAddr_4$, cm_4, vn_4, 8)
$pCoin_5$: ("CoinB", $SAddr_5$, cm_5, vn_5, 10)
$pCoin_6$: ("CoinB", $SAddr_6$, cm_6, vn_6, 13)
$tx_{reclaim}$:
(cm_6, ϕ, vn_4, vn_5,
$\pi_{\{B_4,B_5\}\to\{B_6,B_7\}}$,
$PAddr_{B_7}$, "CoinB", 5)
reclaim
OK
$State_5$
CMList: [..., cm_0, cm_1, cm_2, cm_3,
cm_4, cm_5, cm_6]
VNList: [..., vn_0, vn_1, vn_2,
vn_4, vn_5]
#pCoinA: 1020; #pCoinB: 9995
#pCoinDex (A): 110; #pCoinDex (B): 182
$pCoin_3$: ("CoinA", $SAddr_3$, cm_3, vn_3, 40)
~~$pCoin_4$: ("CoinB", $SAddr_4$, cm_4, vn_4, 8)~~
~~$pCoin_5$: ("CoinB", $SAddr_5$, cm_5, vn_5, 10)~~
$pCoin_6$: ("CoinB", $SAddr_6$, cm_5, vn_6, 13)
$CoinB_7$: ($PAddr_{B_7}$, 5)
Legend:
~~Stroked~~: discarded;
gray: generating;
red: private information;
blue: public information.

Fig. 2: Manta overview

We remark that it is computationally impossible to link $pCoin_0$ and $pCoin_1$ in the above example.

Transfer. Next, suppose Alice wishes to conduct a private transfer with her private coins, for example, $pCoin_0$ and $pCoin_1$. Alice *pours* these private coins to new private addresses. For simplicity, in this example, we assume that these new secretive addresses are till owned by herself. A transfer transaction takes two private coins and will produce two new private coins with new addresses. More specifically, she sends $\mathsf{tx}_{\mathsf{transfer}}$ to Manta ledger that contains:

- $\mathtt{cm}_2$ and $\mathtt{cm}_3$, the commitments to the new coins.
- $\mathtt{vn}_0$ and $\mathtt{vn}_1$, the void numbers of the old coins. Now, these void numbers are revealed and the tokens are voided.
- $\pi_{\{A_0,A_1\}\to\{A_2,A_3\}}$, a zero-knowledge proof that proves the transaction is valid and authenticated (more details will be given later).

Upon the recipient of the transaction, Manta ledger checks validity of the zero-knowledge proof ($\pi_{\{A_0,A_1\}\to\{A_2,A_3\}}$) and ensures that $\mathtt{vn}_0$ and $\mathtt{vn}_1$ have never been revealed before. Note, since we have revealed only $\mathtt{vn}_0$ and $\mathtt{vn}_1$ (instead of $\mathtt{cm}_0$ and $\mathtt{cm}_1$), and a ZKP ($\pi_{\{A_0,A_1\}\to\{A_2,A_3\}}$), the adversary cannot link the revealed void numbers ($\mathtt{vn}_0$ and $\mathtt{vn}_1$) with the commitments of the old coins ($\mathtt{cm}_0$ and $\mathtt{cm}_1$).

Exchange. Exchange works similarly as transfer. The main difference is that, in an exchange, the output $pCoin$ is for a different asset type, and the ZKP includes a statement of the total number of assets for both $CoinA$ and $CoinB$. This transaction is accepted if the ZKP verifies and the total amounts preserves the AMM ledger invariant, i.e., $\#\mathtt{pCoinDex}\ (\mathrm{A}) \times \#\mathtt{pCoinDex}\ (\mathrm{B}) = k$.

Concretely, in our case, $k = 2 \times 10^4$. Alice wants to exchange 10 $CoinA$ to $CoinB$. To maintain the constant invariant, Alice is expecting 18 $CoinB$s in return (rounding to integers for simplicity), for which she will split into two coins, denoted by $pCoinB_1$ and $pCoinB_2$.

Note that the result from [AEC21, Whi20] claimed that privacy preserving AMM is impossible. Therefore we assume that the attacker learns that the value of $pCoin_2$ is 10; that of $pCoin_4$ and $pCoin_5$ combined is 18. Nonetheless, $pCoin_2$ remains anonymous, since only $\mathtt{vn}_2$ is revealed. On the other hand, both face values and the addresses of $pCoin_4$ and $pCoin_5$ remain private.

Reclaim. Reclaim is a process through which the user claims back the base coin from the private coin. In our example, Alice wants to reclaim 5 $CoinB$s, and put the reminder of $pCoin_4$ and $pCoin_5$ to $pCoin_6$. The workflow is identical to a transfer workflow, with the only different that one of the output coins $CoinB_7$ is public.

Provide Liquidity. Manta allows for both public and private liquidity providing. Private liquidity providing will hide the identity of the liquidity provider; while leaking the amount of the asset that the provider injects to the pool. In particular, the liquid share are present by the so-called LP tokens, which itself is a fungible token, and can therefore be privatized via the Manta$_{\mathtt{DAP}}$ protocol. This additional step allows us to hide both the identity and the amount the provider provides to a liquid pool. This functionality is absent in Figure 2. We will give more details in section 5.

Auditability and Regulation Compliance. Auditablity and regulation compliance is a legit concern of any decentralized exchange, especially privacy preserving ones. Manta offers a self audit feature that allows an user to demonstrate the transaction records to IRS or other government agencies by providing the zero-knowledge proof. In the future, we plan to incorporate selective disclosure functionality to the auditing feature so that a finer granularity's of control can be achieved while keeping regulation compliance.

1.3 Related Works

Privacy Preserving Decentralized Payment Protocols. Two most notable privacy preserving crytpocurrency is Zcash [BCG+14] and Monero [vS13]. The security and privacy guarantee of them are based on different cryptographic primitives: Zcash is based on NIZK (or zkSNARK), and Monero is based on ring signature. Manta$_{\mathtt{DAP}}$'s protocol is similar to Zcash, but with two major differences: First, Manta$_{\mathtt{DAP}}$ supports multiple assets as base tokens and hides the identities of tokens as well. This gives strong privacy guarantee even for long tail assets. Second, Manta$_{\mathtt{DAP}}$'s implementation leverages sharded accumulator design that improves the protocol's performance under concurrent transactions. Besides clean sheet designs, Solidus [CZJ+17] and Zether [BAZB20] builds decentralized payment system leveraging existing permissionless ledgers such as ethereum. Due to the design constraints, account based decentralized anonymous payment protocol cannot provide the same level of privacy as UTXO based design like $\text{Manta}_{\texttt{DAP}}$ and Zcash. Besides permissionless, layer-1 solutions, Chiesa et al. [CGL+17] proposed a anonymous micro-payment (offline) protocol without cryptographic non-double-spending guarantee. Green and Miers proposed Bolt [GM17], a anonymous payment channel protocol for processing anonymous payment in Layer 2. zkLedger [NVV18] present a permissioned decentralized ledger design with auditability. These works are orthogonal to Manta, and explored different design spaces.

Other Privacy Preserving Decentralized Protocols. Beyond payment, many work explores privacy preserving applications on decentralized ledgers. For example, Ekiden [ZHC+20] proposed a privacy preserving smart contract framework that leverages trusted execution environment. Despite the limitation of relying on hardware security assumptions, it still can provide privacy guarantee to many meaningful applications. Ouroboros Crypsinous [KKKZ19] and Anonymous lottery [BMSZ20] proposed changes to Proof of Stake (PoS) consensus protocols that can make validators' stakes confidential. However, Kohlweiss et al. [KMNS21] demonstrates that the privacy guarantee will break if the network delays can be leveraged by the attackers and also proved the impossibility result. There are also domain specific protocols that build on top of NIZK and decentralized ledgers, such as ZebraLancer [LTW20], a privacy preserving decentralized crowd sourcing protocol.

Decentralized Exchanges and Automated Market Makers. Centralize exchanges bring back counter-party risk that cryptocurrencies try to eliminate and also naturally prone to front-running [Gra21] [4]. Due to the high on-chain computation cost, implementing a traditional order-book styled exchange in the decentralized setting could cause prohibitive gas cost. As a result, exchanges like IDEX [ide21] still keep the order-books and order-matching off-chain, and only do the settlement on chain. The aforehand mentioned issues of centralized exchanges mostly remains. An elegant solution to this problem is a scheme named automated market maker (AMM) [But18] from Vitalik Buterin. Instead of trading against counter-parties, a trader is trading against a convex curve of exchange ratio. For each trading pair, the exchange keeps the ratio of the two assets according to the curve. For superb gas efficiency and good capital efficiency, this design gains many attractions and results in billion dollar day trading volumes [AZR20, cur21]. To the authors best knowledge, $\text{Manta}_{\texttt{DAX}}$ is the first privacy preserving decentralized exchange protocol based on AMM. AMM's simplicity and gas efficiency could naturally result in cheap prover cost in NIZK. Compared with privacy preserving decentralized proposals like ZEXE [BCG+20], $\text{Manta}_{\texttt{DAX}}$ leaks slightly more information (the AMM position on the curve is leaked, but the trader and liqudity provider's identity is still confidential) and and trade for simplicity and constraints size of the circuits (hence the prover time).

2 Background

We briefly recall blockchain related notions in this section. We defer to section A for the introduction for zk-SNARKs and other related cryptographic primitives.

Blockchain and Decentralized Ledger. We model a blockchain as a byzantine fault tolerance replicated state machine with append only state, a.k.a ledger. When interacting with the blockchain, we call the entity who initiates the interaction (e.g., sending a transfer request) the user; the entity who verifies the interaction and logs it into the blockchain the validator (also known as miner in other contents). Users interact with the blockchain by sending amendment requests to the ledger. The amendment is appended to the database once validators approve the request. For simplicity, we assume 1) the ledger is synchronized, and the block finality is instant; 2) the validators are trusted for liveness and completeness. The underlying consensus protocol that validators employ is indeed orthogonal to this paper. What is also of out the scope of this paper is the governance token for the underlying blockchain. We nonetheless assume that the senders of our protocol holds enough governance tokens to send the transactions.

[4] Decentralized exchanges also suffer (a different set of) front-running issues [DGK+19].

Public/Private Assets and Asset Models. The focus of this paper are the customized assets issued over the blockchain. Throughout the paper, we will be dealing with two assets, with asset-ids $CoinA$ and $CoinB$, as an example. Extension to asset types beyond two is straightforward. We assume that those two assets are stored on a blockchain ledger in an account model. Denoted by $pCoin$-s the private coin of the assets. Those $pCoin$-s are stored on chain in an unspent transaction output (UTXO) model that is adopted by Bitcoin, and is essential to privacy-preserving ledgers. The ledger also maintains a public information on how many assets are converted into private, for example $\#\mathtt{pCoinA}$.

Automated Market Maker and Liquidity Pool Our AMM based exchange employs a liquidity pool for a trading pair, instantiated with two parameters: $\#\mathtt{pCoinDex}(A)$ and $\#\mathtt{pCoinDex}(B)$, which are the current number of assets that the pool holds. Any exchange against this pool needs to maintain the invariant, denoted by $T := \#\mathtt{pCoinDex}(A) \times \#\mathtt{pCoinDex}(B)$. On the other hand, when a liquidity provider injects or withdraws liquidity to/from the pool, the operation needs to preserve the ratio: $\#\mathtt{pCoinDex}(A) : \#\mathtt{pCoinDex}(B)$. Under this content, the entity who trades against the pool is a user; and that who provides liquidity to the pool is a liquidity provider.

When dealing with a two party interactive protocol, we refer to the user who sends/initiates the transfer as Alice, and counterpart as Bob.

3 Manta Security Model

3.1 Manta's security intuition

Now, we briefly explain why Manta's payment and exchange is private. We can make following observations in Figure 2:

- The secretive addresses (this is similar to shielded address in ZCash), e.g. $\mathtt{SAddr}_i$ are never revealed.
- The public information, e.g., cm-s and vn-s cannot be linked.
- All operations (except for mint) consume two old coins (UTXOs) and generate two new coins (UTXOs). Under the assumption that mint leaks the value of a single commitment, and exchange leaks the values of the summation of the two commitment (exchange have to leak the value of the trade otherwise the participant would be blind on the exchange ratio), the overall scheme remains private. Even more so, when conducting a transfer, exchange, or reclaim, user does not identify which commitment is been used; instead, she proves that " I own a certain commitment that was submitted to the ledger earlier, whose the void number is vn". Thus, knowing the face value of a commitment does not help the attacker de-anonymize the sender.
- For $\mathsf{tx}_{\mathsf{transfer}}$, an attacker cannot identify the asset identifier of the transfer. This allows us to mix transfers for all asset types, providing significantly better privacy for long-tail assets while improving the overall privacy for every asset.

This also explains why Manta's privacy guarantee is not contradict with the negative theoretical results of private AMM [AEC21]. In a nutshell, Manta guards the privacy despite the leak of trading price.

3.2 Formal Definition of Decentralized Anonymous Payment

A decentralized anonymous payment scheme is a tuple of algorithms (Setup, GenMint, Mint, GenTransfer, Transfer, GenReclaim, Reclaim).

(a) $\mathtt{Setup}(1^\lambda) \rightarrow (\sigma, \tau)$ takes place before the protocol starts. It sets up the parameters for the underlying NIZK system and grabs the common reference string and the simulator trapdoor τ.
(b) $\mathtt{GenMint}(\mathtt{LS}, CoinA, a_{pk}, a_{sk}, \delta, v, 1^\lambda) \rightarrow (\mathtt{tx}_{\mathsf{mint}}, \mathtt{aux})$ is called by the user to initiate a mint transaction. It takes in the current ledger state LS, a public coin $CoinA$, a public and private key pair a_{pk} and a_{sk}, the asset identifier δ, the proposed value of the private coin v, and randomness. It outputs a mint transaction $\mathtt{tx}_{\mathsf{mint}}$ and an auxiliary information aux that the user keeps private.
(c) $\mathtt{Mint}(\mathtt{LS}, \mathtt{tx}_{\mathsf{mint}}) \rightarrow (\mathtt{LS}', \{\top, \bot\})$ is called by the validator to verify and execute the mint transaction. It outputs a bit indicating whether the verification and execution succeed. Upon successful execution, the ledger state will be updated from LS to LS$'$.

(d) $\texttt{GenTransfer}(\texttt{LS}, c_1^{\texttt{old}}, c_2^{\texttt{old}}, a_{sk}, b_{sk}, d_{pk}, e_{pk}, \sigma, \texttt{aux}, 1^\lambda)$
$\rightarrow (\texttt{tx}_{\mathsf{transfer}}, \texttt{aux})$ is called by the user to create a transfer transaction. It takes in the current ledger state $\texttt{LS}$, two private coins $c_1^{\texttt{old}}, c_2^{\texttt{old}}$, two private keys a_{sk} and b_{sk}, the recipient public keys d_{pk} and e_{pk}, the common reference string σ, and the auxiliary information $\texttt{aux}$. It outputs a transfer transaction $\texttt{tx}_{\mathsf{transfer}}$.

(e) $\texttt{Transfer}(\texttt{LS}, \texttt{tx}_{\mathsf{transfer}}) \rightarrow (\texttt{LS}', \{\top, \bot\})$ is called by the validator to verify and execute the transfer transaction. It outputs a bit indicating whether the verification and execution succeed. Upon successful execution, the ledger state will be updated from $\texttt{LS}$ to $\texttt{LS}'$.

(f) $\texttt{GenReclaim}(\texttt{LS}, c_1^{\texttt{old}}, c_2^{\texttt{old}}, a_{sk}, b_{sk}, d_{pk}, \texttt{PAddr}, \delta, v, \sigma, \texttt{aux}, 1^\lambda)$
$\rightarrow (\texttt{tx}_{\mathsf{reclaim}}, \texttt{aux})$ is called by the user to create a reclaim transaction. It takes in the current ledger state $\texttt{LS}$, two private coin $c_1^{\texttt{old}}$ and $c_2^{\texttt{old}}$, the private keys a_{sk} and b_{sk}, the recipient public key d_{pk}, public key $\texttt{PAddr}$ of the public coin, the asset identifier δ, the value v of the public coin, the common reference string σ, and the auxiliary information $\texttt{aux}$. It outputs a transfer transaction $\texttt{tx}_{\mathsf{reclaim}}$.

(g) $\texttt{Reclaim}(\texttt{LS}, \texttt{tx}_{\mathsf{reclaim}}) \rightarrow (\texttt{LS}', \{\top, \bot\})$ is called by the validator to verify and execute the reclaim transaction. It outputs a bit indicating whether the verification and execution succeed. Upon successful execution, the ledger state will be updated from $\texttt{LS}$ to $\texttt{LS}'$.

Now we formally define the privacy property of Decentralized Anonymous Payment.

Definition 1 (Decentralized Anonymous Payment Privacy). *Definition deferred to Definition 16.*

We also require the completeness of the decentralized anonymous payment scheme. If a coin has not been spent before, it can be spent. Specifically, for any correctly minted coin $\texttt{pCoin}$, it is transferrable and reclaimable. The same holds for a correctly received coin.

Definition 2 (Decentralized Anonymous Payment Completeness). *Definition deferred to Definition 17.*

Then we define the security of the decentralized anonymous payment scheme. The guarantee is similar to that of zerocash [BCG+14]. Namely, assuming the underlying consensus protocol is ideal, a user is not able to double-spend a private coin, or double-reclaim a private coin. More formally, we have

Definition 3 (Decentralized Anonymous Payment Security). *Definition deferred to Definition 18.*

3.3 Formal Definition of DAX with Public liquidity provider

A decentralized anonymous exchange with public liquidity provider is a tuple of algorithms $\pi_{\texttt{DAX}} = (\texttt{Setup}, \texttt{GenExchange}, \texttt{Exchange}, \texttt{LP})$ with an oracle access to a decentralized anonymous payment scheme $\pi_{\texttt{DAP}}$.

(h) $\texttt{Setup}(1^\lambda) \rightarrow (\sigma, \tau)$ takes place before the protocol starts. It sets up the parameters for the underlying NIZK system and grabs the common reference string and the simulator trapdoor τ. Besides, it also calls $\pi_{\texttt{DAP}}.\texttt{Setup}$ to set up the DAP scheme.

(i) $\texttt{LP}$ is an interface for a public liquidity pool used for every pair of supported assets.

(j) $\texttt{GenExchange}(\texttt{LS}, c_1^{\texttt{old}}, c_2^{\texttt{old}}, \delta', v', a_{sk}, b_{sk}, d_{pk}, e_{pk}, \sigma, \texttt{aux}, 1^\lambda)$
$\rightarrow (\texttt{tx}_{\mathsf{exchange}}, \texttt{pCoinB})$ is called by the user to create an exchange transaction. It takes in two private coin $c_1^{\texttt{old}}, c_2^{\texttt{old}}$, a target coin type δ', the combined value of the new coin v', the related keys $a_{sk}, b_{sk}, d_{pk}, e_{pk}$, the common reference string σ, and the auxiliary information $\texttt{aux}$. It outputs an exchange transaction $\texttt{tx}_{\mathsf{exchange}}$ and a private coin of type δ' $pCoinB$.

(k) $\texttt{Exchange}(\texttt{LS}, \texttt{tx}_{\mathsf{exchange}}) \rightarrow (\texttt{LS}', \{\top, \bot\})$ is called by the validator to verify and execute the exchange transaction. It outputs a bit indicating whether the verification and execution succeed.

Now we formally define the privacy property of a Decentralized Anonymous Exchange (DAX).

Definition 4 (Decentralized Anonymous Exchange Privacy with Public Liquidity Provider). *Definition deferred to Definition 15.*

3.4 Formal Definition of DAX with Private liquidity provider

A decentralized anonymous exchange with private liquidity provider is a tuple of algorithms $\pi_{\texttt{DAX*}} = (\texttt{Setup}, \texttt{GenPrivInit}, \texttt{ApplyPrivInit}, \texttt{GenPrivProvide}, \texttt{ApplyPrivProvide}, \texttt{GenPrivWithdraw}, \texttt{ApplyPrivWithdraw})$ with an oracle access to a decentralized anonymous payment scheme $\pi_{\texttt{DAP}}$, and a DAX scheme with public liquidity providers.

(l) $\texttt{Setup}(1^\lambda) \to (\sigma, \tau)$ takes place before the protocol starts. It sets up the parameters for the underlying zkSNARK system and grabs the common reference string and the simulator trapdoor τ. Besides, it also calls $\pi_{\texttt{DAP}}.\texttt{Setup}$ to set up the DAP scheme; $\pi_{\texttt{DAX}}$ to set up the DAX scheme with public liquidity providers.

(m) $\texttt{GenPrivInit}(c_1^{\texttt{old}}, \delta_1, c_2^{\texttt{old}}, \delta_2, a_{sk}, b_{sk}, d_{pk}) \to (\texttt{tx}_{\textsf{PrivInit}}, \texttt{aux})$ is called by a private liquidity provider to privately initialize the pool.

(n) $\texttt{ApplyPrivInit}(\texttt{tx}_{\textsf{PrivInit}}, x, y) \to \mathcal{P}$ is called by the validator to verify and apply a private initialization transaction, and update the pool.

(o) $\texttt{GenPrivProvide}(c_1^{\texttt{old}}, \delta_1, c_2^{\texttt{old}}, \delta_2, a_{sk}, b_{sk}, d_{pk}) \to$ $(\texttt{tx}_{\textsf{PrivProvide}}, \texttt{aux})$ is called by a private liquidity provider to privately provide liquidity to the pool. It operates similarly to $\texttt{GenPrivInit}$

(p) $\texttt{ApplyPrivProvide}(\texttt{tx}_{\textsf{PrivProvide}}, x, y, \mathcal{P}) \to \mathcal{P}'$ is called by the validator to verify and apply a private provide transaction, and update the pool.

(q) $\texttt{GenPrivWithdraw}(c_1^{\texttt{old}}, v_1^{\texttt{old}}, a_{sk}, b_{pk}, d_{pk}, \mathcal{P}) \to$ $(\texttt{tx}_{\textsf{PrivWithdraw}}, \texttt{aux}_1, \texttt{aux}_2)$: is called by a liquidity provider to privately withdraw the liquidity from the pool.

(r) $\texttt{ApplyPrivWithdraw}(\texttt{tx}_{\textsf{PrivWithdraw}}, \mathcal{P}) \to \mathcal{P}'$: is called from the validator to verify and apply a private withdraw transaction, and update the pool.

Now we formally define the privacy property of Decentralized Anonymous Exchange.

Definition 5 (Decentralized Anonymous Exchange Privacy with Private Liquidity Provider). *Definition deferred to Definition 16.*

We also require the completeness of the decentralized anonymous exchange scheme. Besides the completeness of the underlying decentralized anonymous payment scheme, we further require that if a coin has not been exchanged before, it can be exchanged. Specifically, for any correctly minted coin `pCoinA`, it is exchangeable. For a private exchange scheme with private liquidity provider, we require that an honest private liquidity provider is able to withdraw the liquidity after successfully providing the liquidity.

Definition 6 (Decentralized Anonymous Exchange Completeness). *Definition deferred to Definition 17.*

Informally, the soundness properties of the exchange scheme covers all the soundness properties of decentralized anonymous payment scheme, and that any adversary is not able to double-exchange the same private coin. For the exchange scheme with private liquidity provider, we further require that any adversary is not able to act as the private liquidity provider and double-withdraw with a single liquidity coin `pCoinLP`.

Definition 7 (Decentralized Anonymous Exchange Security). *Definition deferred to Definition 18.*

4 Manta$_{\texttt{DAP}}$: Decentralized Anonymous Payment

In this section, we present Manta$_{\texttt{DAP}}$, a decentralized anonymous payment (DAP). As alluded earlier, our payment protocol hides the asset identifiers so that an attacker will not be able to link a base asset type with a private transfer. Mixing all private transactions into a single transaction pool gives better privacy guarantees to those long-tail assets whose trading volume may be low on its own. This DAP scheme supports both minting and reclaim, therefore, allows for bidirectional transfers between private coins and base coins.

Addresses A user u generates an address key pair $(a_{\mathsf{pk}}, a_{\mathsf{sk}})$. The coins of u can be only spent with the knowledge of a_{sk}. To generate a key pair, u randomly samples a secret from the domain $a_{\mathsf{sk}} \overset{\$}{\leftarrow} 1^{\lambda}$, and sets $a_{\mathsf{pk}} := \mathtt{PRF}^{addr}_{a_{\mathsf{sk}}}(0)$. A user could generate and use any number of key pairs; each key pair produces a UTXO and shall only be used once.

4.1 Mint private assets

(b) $\mathtt{GenMint}(\mathtt{LS}, \mathit{CoinA}, a_{pk}, a_{sk}, \delta, v, 1^{\lambda}) \rightarrow (\mathtt{tx}_{\mathsf{mint}}, \mathit{pCoin})$

To mint a private asset with a type δ and a face value v, a user u needs to initiate a coin minting transaction tx_{mint} with a deposit of the public asset of value v [5]. A user generates and submits tx_{mint} to the ledger as the following:

- u samples a random number $\rho \overset{\$}{\leftarrow} 1^{\lambda}$, which is a secret string that determines the coins void number [6] $\mathtt{vn} := \mathtt{PRF}^{vn}_{a_{\mathsf{sk}}}(\rho)$. Note that neither ρ nor $\mathtt{vn}$ is included in tx_{mint}.
- u commits to the triple $(a_{\mathsf{pk}}, v, \rho)$ in two phases:
  - sample a random r, and compute $k := \mathtt{COMM}_r(a_{\mathsf{pk}}||\rho)$; then,
  - sample a random s, and compute $\mathtt{cm} := \mathtt{COMM}_s(\delta||v||k)$.
- u thus mints a private coin $c := (a_{\mathsf{pk}}, v, \rho, r, s, \mathtt{cm})$ and a mint transaction $tx_{mint} := (\delta, v, k, s, \mathtt{cm})$.

This design allows validators to verify $\mathtt{cm}$ in tx_{mint} with asset id δ and value v but doesn't disclose the address of the owner (a_{pk}) or the void number.

A private coin consists of a tuple $\mathit{pCoin} = (a_{\mathsf{pk}}, \delta, v, \rho, r, s, \mathtt{cm})$, where a_{pk} is the public key of the owner, δ is the asset id of the coin, and v is the value of the private coin. They collectively form the $\mathtt{aux}$ that the user needs to spend this coin at a later stage.

(c) $\mathtt{Mint}(\mathtt{LS}, \mathtt{tx}_{\mathsf{mint}}) \rightarrow (\mathtt{LS}', \{\top, \bot\})$

Upon receiving tx_{mint}, the validator

- checks that the sender of tx_{mint} has v assets of δ;
- checks $\mathtt{cm} = \mathtt{COMM}_s(\delta||v||k)$;
- deducts v assets of δ from sender's account;
- adds $\mathtt{cm}$ to the accumulator that represents the ledger state acc.

The on-chain cost of the mint operation is dominated by the commitment scheme. It does not involve any zero-knowledge proof operations.

4.2 Transfer private coins

(d) $\mathtt{GenTransfer}(\mathtt{LS}, c_1^{\mathtt{old}}, c_2^{\mathtt{old}}, a_{sk}, b_{sk}, d_{pk}, e_{pk}, \sigma, \mathtt{aux}, 1^{\lambda})$
$\rightarrow (\mathtt{tx}_{\mathsf{transfer}}, c_3^{\mathtt{new}}, c_4^{\mathtt{new}})$

Private coins can be transferred and spent using the $\mathtt{transfer}$ operation, which takes a set of input private coins to be consumed, and transfers their total value into a set of new output coins: the total value of output coins equals the total value of the input coins. To be consistent with Figure 2, in the following, we will use two input coins and two output coins.

For example, suppose a user u, with address key pairs $(a^{\mathtt{old}}_{\mathsf{pk}}, a^{\mathtt{old}}_{\mathsf{sk}})$ and $(b^{\mathtt{old}}_{\mathsf{pk}}, b^{\mathtt{old}}_{\mathsf{sk}})$, tries to transfer his old coins

$$\begin{aligned} c_1^{\mathtt{old}} &= (a^{\mathtt{old}}_{\mathsf{pk}}, \delta_1^{\mathtt{old}}, v_1^{\mathtt{old}}, \rho_1^{\mathtt{old}}, r_1^{\mathtt{old}}, s_1^{\mathtt{old}}, \mathtt{cm}_1^{\mathtt{old}}) \\ c_2^{\mathtt{old}} &= (b^{\mathtt{old}}_{\mathsf{pk}}, \delta_2^{\mathtt{old}}, v_2^{\mathtt{old}}, \rho_2^{\mathtt{old}}, r_2^{\mathtt{old}}, s_2^{\mathtt{old}}, \mathtt{cm}_2^{\mathtt{old}}) \end{aligned}$$

[5] For simplicity, here we assume a 1 : 1 exchange ratio. A customized transaction fee may be charged during this process.

[6] Also known as the *serial number* in Zcash [BCG+14]; and *nullifier* in other contexts.

to two new coins $c_3^{\mathtt{new}}$ and $c_4^{\mathtt{new}}$ of type $\delta_3^{\mathtt{new}}$ and $\delta_4^{\mathtt{new}}$, under two new public keys $d_{\mathsf{pk}}^{\mathtt{new}}$ and $e_{\mathsf{pk}}^{\mathtt{new}}$. Note that those two keys are not present in `transfer`. This implies that the addresses could belong to u (i.e., a self-transaction), or someone else. For the sake of simplicity, we assume the receiver is also u.

To create such a `transfer` transaction, u samples trapdoors $\rho_3^{\mathtt{new}}$ and $\rho_4^{\mathtt{new}}$, and compute

$$\begin{aligned} k_3^{\mathtt{new}} &:= \mathtt{COMM}_{r_3^{\mathtt{new}}}(d_{\mathsf{pk}}^{\mathtt{new}}, ||\rho_3^{\mathtt{new}}), \\ k_4^{\mathtt{new}} &:= \mathtt{COMM}_{r_4^{\mathtt{new}}}(e_{\mathsf{pk}}^{\mathtt{new}}, ||\rho_4^{\mathtt{new}}), \\ \mathtt{cm}_3^{\mathtt{new}} &:= \mathtt{COMM}_{s_3^{\mathtt{new}}}(\delta_3^{\mathtt{new}}||v_3^{\mathtt{new}}||k_3^{\mathtt{new}}) \\ \mathtt{cm}_4^{\mathtt{new}} &:= \mathtt{COMM}_{s_4^{\mathtt{new}}}(\delta_4^{\mathtt{new}}||v_4^{\mathtt{new}}||k_4^{\mathtt{new}}) \end{aligned}$$

This creates new coins

$$\begin{aligned} c_3^{\mathtt{new}} &:= (d_{\mathsf{pk}}^{\mathtt{new}}, v_3^{\mathtt{new}}, \rho_3^{\mathtt{new}}, r_3^{\mathtt{new}}, s_3^{\mathtt{new}}, \mathtt{cm}_3^{\mathtt{new}}) \\ c_4^{\mathtt{new}} &:= (e_{\mathsf{pk}}^{\mathtt{new}}, v_4^{\mathtt{new}}, \rho_4^{\mathtt{new}}, r_4^{\mathtt{new}}, s_4^{\mathtt{new}}, \mathtt{cm}_4^{\mathtt{new}}) \end{aligned}$$

The user u also produces a NIZK proof $\pi_{\mathtt{transfer}}$ for the following `NP` statement, which will be called by `transfer`:

Definition 8 (`NP` Statement of `transfer`). *Given an accumulator acc that represents the ledger state, void numbers $\mathtt{vn}_1^{old}$ and $\mathtt{vn}_2^{old}$; and coin commitments $\mathtt{cm}_3^{new}$ and $\mathtt{cm}_4^{new}$, "I" know coins c_1^{old}, c_2^{old}, c_3^{new}, c_4^{new}, and secret keys a_{sk}^{old} and b_{sk}^{old}, such that:*

1. *The address and the secret key derive the public key: $a_{pk}^{old} = PRF_{a_{old}^{sk}}^{addr}(0)$, and $b_{pk}^{old} = PRF_{b_{old}^{sk}}^{addr}(0)$.*
2. *The old coins c_1^{old} and c_2^{old} are* well-formed*:*

$$\begin{aligned} k_1^{old} &= COMM_{r_1^{old}}(a_{pk}^{old}||\rho_1^{old}), \\ k_2^{old} &= COMM_{r_2^{old}}(b_{pk}^{old}||\rho_2^{old}), \\ cm_1^{old} &= COMM_{s_1^{old}}(\delta_1^{old}||v_1^{old}||k_1^{old}), \\ cm_2^{old} &= COMM_{s_2^{old}}(\delta_2^{old}||v_2^{old}||k_2^{old}). \end{aligned}$$

3. *The old coin's commitments, although* not *presented in `transfer`, are members of acc, i.e., $cm_1^{old}, cm_2^{old} \in acc$.*
4. *The new coins c_3^{new} and c_4^{new} are also* well-formed*:*

$$\begin{aligned} cm_3^{new} &= COMM_{s_3^{new}}(\delta_3^{new}||v_3^{new}||k_3^{new}), \\ cm_4^{new} &= COMM_{s_4^{new}}(\delta_4^{new}||v_4^{new}||k_4^{new}). \end{aligned}$$

5. *The old and new coins have a same asset identifier δ.*
6. *The old and new coins have a same combined value: $v_3^{new} + v_4^{new} = v_1^{old} + v_2^{old}$.*
7. *The new values are non-negative: $v_3^{new} \geq 0$ and $v_4^{new} \geq 0$.*

To complete the transaction, the user u sends a $tx_{\mathtt{transfer}} := (acc, \mathtt{vn}_1^{\mathtt{old}}, \mathtt{vn}_2^{\mathtt{old}}, \mathtt{cm}_3^{\mathtt{new}}, \mathtt{cm}_4^{\mathtt{new}}, \pi_{\mathtt{transfer}})$ to the ledger.

(e) $\mathtt{Transfer}(\mathtt{LS}, \mathtt{tx}_{\mathsf{transfer}}) \rightarrow (\mathtt{LS}', \{\top, \bot\})$

Upon receiving such a $tx_{\mathtt{transfer}}$, the validators check the validity of the transaction:

- $\mathtt{vn}_1^{\mathtt{old}}$ and $\mathtt{vn}_2^{\mathtt{old}}$ has never been used in a previous transaction in the ledger (otherwise it is a double spend)
- the NIZK verifier verifies the validity of $\pi_{\mathtt{transfer}}$.

Upon successful validation, the validators will append $\mathtt{vn}_1^{\mathtt{old}}, \mathtt{vn}_2^{\mathtt{old}}$, $\mathtt{cm}_3^{\mathtt{new}}$, and $\mathtt{cm}_4^{\mathtt{new}}$ to the ledger, and update the accumulator accordingly.

We make the follow remarks. From the sender's point of view, the proof does not specify which $\mathtt{cm}_1^{\mathtt{old}}$ $\mathtt{cm}_2^{\mathtt{old}}$ the coins are transferred from. Instead, it proves the existences of such coins. This breaks the link between old and new commitments, and is a key requirement for anonymity.

From the receiver's point of view, when the receiver is a different entity than u, the receiver needs to share with u, in an offline manner, the following quantities: $\{s_i^{\mathtt{new}}, k_i^{\mathtt{new}}\}_{i\in\{3,4\}}$, d_{pk}^{new} and e_{pk}^{new}. Those are one-time, none personal identifiable information (PII); and therefore do not leak the information of the receiver regardless if `transfer` goes through or not. On the other hand, the corresponding d_{sk}^{new} and e_{sk}^{new} are not shared with u. This guarantees that the receiver is the sole entity who is able to spend the coin later.

Stop double spending Manta prevents double spending by binding the void numbers with commitments and enforcing that `transfer` transaction reals the void numbers of the input coins. Concretely, the ledger maintains two lists, represented by two accumulators, namely, acc_{all} and acc_{spend}. acc_{all} contains all commitments that have ever appeared; while acc_{spend} contains `vn`s for all spend tokens. When generating a new `transfer`, the sender needs to prove that (the commitments of) the coins it is about to spend are in acc_{all}, in zero-knowledge; and (the `vn`s of) the coin is not in acc_{spend}. When the `transfer` is accepted, $\mathtt{vn}^{old}$ will be revealed and added to acc_{spend}.

Note that this creates a link between new commitment $\mathtt{cm}^{new}$s and $\mathtt{vn}^{old}$s since they all appear in a same `transfer`. This, however, does not break the anonymity, since `cm`s hide `vn`-s so one cannot link `cm`s with their corresponding `vn`s.

4.3 Reclaim public coins from private coins.

(f) $\mathtt{GenReclaim}(\mathtt{LS}, c_1^{\mathtt{old}}, c_2^{\mathtt{old}}, a_{sk}, b_{sk}, d_{pk}, \mathtt{PAddr}, \delta, v, \sigma, \mathtt{aux}, 1^\lambda)$
$\rightarrow (\mathtt{tx}_{\mathsf{reclaim}}, c_3^{\mathtt{new}})$

One can reuse the `GenTransfer` interface with a simple modification: making one of the output coins public. More concretely, the output coins include a private coin c_3^{new}, a public address $\mathtt{PAddr}_4$ the asset identifier δ, and a value v_4.

Definition 9 (NP Statement of `reclaim`). *Given an accumulator acc that represents the ledger state, void numbers* $\mathtt{vn}_1^{old}$ *and* $\mathtt{vn}_2^{old}$*; and coin commitment* $\mathtt{cm}_3^{new}$ *and a public coin* $(\mathtt{PAddr}_4, v_4)$ *for an asset identifier* δ*, "I" know coins* c_1^{old}*,* c_2^{old}*,* c_3^{new}*, and secret keys* $a_{\mathtt{sk}}^{old}$ *and* $b_{\mathtt{sk}}^{old}$*, such that:*

1. *The address and the secret key derive the public key:* $a_{\mathtt{pk}}^{old} = \mathtt{PRF}^{\mathtt{addr}}_{a^{\mathtt{sk}}_{old}}(0)$*, and* $b_{\mathtt{pk}}^{old} = \mathtt{PRF}^{\mathtt{addr}}_{b^{\mathtt{sk}}_{old}}(0)$*.*
2. *The old coins* c_1^{old} *and* c_2^{old} *are* well-formed*, identical to Definition 8.*
3. *The new coins* c_3^{new} *is also* well-formed*:*

$$\mathtt{cm}_3^{new} = \mathtt{COMM}_{s_3^{new}}(\delta_3^{new}||v_3^{new}||k_3^{new}).$$

4. *The old and new coins share a same asset identifier* δ*.*
5. *The old and new coins have a same combined value:* $v_3^{new} + v_4 = v_1^{old} + v_2^{old}$*.*
6. *The new value in* c_3^{new} *is non-negative:* $v_3^{new} \geq 0$*.*

Finally, the user sends $\mathtt{tx}_{\mathsf{reclaim}} = (\mathtt{LS}, \mathtt{vn}_1, \mathtt{vn}_2, \mathtt{cm}_3, \pi, \mathtt{PAddr}, v_4)$ to the ledger.

(g) $\mathtt{Reclaim}(\mathtt{LS}, \mathtt{tx}_{\mathsf{reclaim}}) \rightarrow (\mathtt{LS}', \{\top, \bot\})$

Upon receiving such a $tx_{\mathtt{reclaim}}$, the validators check the validity of the transaction:

- $\mathtt{vn}_1^{\mathtt{old}}$ and $\mathtt{vn}_2^{\mathtt{old}}$ has never been used in a previous transaction in the ledger (otherwise it is a double spend)
- the zkSNARK verifier verifies the validity of $\pi_{\mathtt{reclaim}}$.

Upon successful validation, the validators will append $\mathtt{vn}_1^{\mathtt{old}}, \mathtt{vn}_2^{\mathtt{old}}$ and $\mathtt{cm}_3^{\mathtt{new}}$ to the ledger, and update the accumulator accordingly. It will also credit the `PAddr` the reclaimed amount of Asset.

4.4 Security Proof

Theorem 1. *Assume $\texttt{COMM}$ is computationally hiding and binding, the zkSNARK scheme is zero-knowledge, and $\texttt{PRF}$ is pseudorandom, the proposed decentralized anonymous payment scheme is private.*

Proof. We prove by first constructing a simulator that satisfies definition 12. Then we prove that for any computationally efficient adversary Adv, it cannot distinguish between the view in the real world and the view from the ideal world.

The simulator is shown in Figure 3. Before proving the desired privacy argument, we first prove a lemma that specially looks at the games with the same initial configurations and transaction op-codes, namely, with equal $\sigma, \tau, N, M, \{sk_i\}, \{pk_i\}, \texttt{aux}_i$ and a sequence of transactions types $\langle \texttt{op}_1, \ldots, \texttt{op}_M \rangle$.

Lemma 1. *Assume $\texttt{COMM}$ is computationally hiding and binding, the zkSNARK scheme is zero-knowledge, and $\texttt{PRF}$ is pseudorandom, for any $\sigma, \tau \leftarrow \texttt{Setup}(1^\lambda)$, $N = poly(\lambda), M = poly(\lambda)$, any sampled set of N user identities $\{pk_i, sk_i\}_{i \in [N]}$, N private string $\{\texttt{aux}_i\}_{i \in [N]}$, and any sequence of transaction op-codes $\langle \texttt{op}_1, \ldots, \texttt{op}_M \rangle$, for an instance of* $\mathrm{Real}_{\mathrm{Adv}}(1^\lambda)$ *and* $\mathrm{Ideal}_{\mathrm{Adv}}(1^\lambda)$*, we have*

$$\begin{aligned}
&\big| \Pr[\mathrm{Real}_{\mathrm{Adv}}(1^\lambda) = 1 | \sigma, \tau, N, M, \{sk_i\}, \{pk_i\}, \texttt{aux}_i, \langle \texttt{op}_1, \ldots, \texttt{op}_M \rangle] \\
&- \Pr[\mathrm{Ideal}_{\mathrm{Adv,Sim}}(1^\lambda) = 1 | \sigma, \tau, N, M, \{sk_i\}, \{pk_i\}, \texttt{aux}_i, \langle \texttt{op}_1, \ldots, \texttt{op}_M \rangle] \big| \\
&\le \mathrm{negl}(\lambda)
\end{aligned}$$

Now we prove by a hybrid argument that, for any computationally adversary Adv, it cannot distinguish with a non-negligible probability an instance of $\mathrm{Real}_{\mathrm{Adv}}(1^\lambda)$ and an instance of $\mathrm{Ideal}_{\mathrm{Adv}}(1^\lambda)$.

Without loss of generality, denote the $\texttt{view}$ from $\mathrm{Real}_{\mathrm{Adv}}(1^\lambda)$ as $\texttt{view}^{(R)} := [\sigma, \tau, N, M, \{pk_i\}, \{(\texttt{tx}_{\mathsf{i}}^{(R)}, \texttt{LS}_i^{(R)})\}_{i=1}^M]$, and denote by $\texttt{view}^{(I)} := [\sigma, \tau, N, M, \{pk_i\}, \{(\texttt{tx}_{\mathsf{i}}^{(I)}, \texttt{LS}_i^{(I)})\}_{i=1}^M]$ the $\texttt{view}$ from $\mathrm{Ideal}_{\mathrm{Adv}}(1^\lambda)$.

Formally, we construct a series of intermediate states of the view $\texttt{view}_i, i \in 0, 1, \ldots, M$.

$\texttt{view}_i := [\sigma, \tau, N, M, \{pk_i\}, (\texttt{tx}_{\mathsf{1}}^{(R)}, \texttt{LS}_1^{(R)}), \ldots, (\texttt{tx}_{\mathsf{i}}^{(R)}, \texttt{LS}_i^{(R)}),\ (\texttt{tx}_{\mathsf{i+1}}^{(I)}, \texttt{LS}_{i+1}^{(I)}), \ldots, (\texttt{tx}_{\mathsf{M}}^{(I)}, \texttt{LS}_M^{(I)})]$. Note that $\texttt{view}^{(R)} = \texttt{view}_0$ and $\texttt{view}^{(I)} = \texttt{view}_M$.

Assume there is an adversary Adv that can distinguish $\texttt{view}^{(R)}$ and $\texttt{view}^{(I)}$ with advantage $\mathrm{poly}_0(\lambda)$ for some polynomial poly_0:

$$\big| \Pr[\mathrm{Adv}(\texttt{view}^{(R)}) = 1] - \Pr[\mathrm{Adv}(\texttt{view}^{(I)}) = 1] \big| = \mathrm{poly}_0(\lambda)$$

Then we have

$$\begin{aligned}
&\big| \Pr[\mathrm{Adv}(\texttt{view}_0) = 1] - \Pr[\mathrm{Adv}(\texttt{view}_M) = 1] \big| \\
&= \big| \sum_{i=1}^{M} (\Pr[\mathrm{Adv}(\texttt{view}_{i-1}) = 1] - \Pr[\mathrm{Adv}(\texttt{view}_i) = 1]) \big| \\
&= \mathrm{poly}_0(\lambda)
\end{aligned}$$

which is

$$\sum_{i=1}^{M} \big| \Pr[\mathrm{Adv}(\texttt{view}_{i-1}) = 1] - \Pr[\mathrm{Adv}(\texttt{view}_i) = 1] \big| \ge \mathrm{poly}_0(\lambda)$$

Therefore, there must be an $i_0 \in [M]$ s.t. $\big| \Pr[\mathrm{Adv}(\texttt{view}_{i_0 - 1}) = 1] - \Pr[\mathrm{Adv}(\texttt{view}_{i_0}) = 1] \big| \ge \mathrm{poly}_0(\lambda)/M$. Now we discuss the transaction type of $\texttt{tx}_{\mathsf{i}_0}$:

- If $\texttt{tx}_{\mathsf{i}_0}$ is a mint transaction, we can construct an adversary Adv' based on Adv to challenge the hiding property of $\texttt{COMM}$:
  Given a pair of commitments $\texttt{cm}_0 = \texttt{COMM}(m_0), \texttt{cm}_1 = \texttt{COMM}(m_1)$, $m_0 \neq m_1$. we replace $\texttt{tx}_{\mathsf{i}_0}^{(R)}$ with (δ, v, k, s, cm_0) and replace $\texttt{tx}_{\mathsf{i}_0}^{(I)}$ with (δ, v, k, s, cm_0), where δ, v, s are sampled accordingly, and k is sampled from the space of $\texttt{COMM}$. Calling Adv on $\texttt{view}_{i_0 - 1}$ and $\texttt{view}_{i_0}$ will yield a distinguishing advantage of at least $\mathrm{poly}_0(\lambda)/M$, which contradicts with the assumption of computational hiding property of $\texttt{COMM}$.

- If $\texttt{tx}_{\mathsf{i}_0}$ is a transfer transaction, we can construct an adversary Adv' based on Adv to challenge either the hiding properties of $\texttt{COMM}$, or the zero-knowledge property of the zkSNARK:
  Given two pair of commitments $\texttt{cm}_0 = \texttt{COMM}(m_0), \texttt{cm}_1 = \texttt{COMM}(m_1), \texttt{cm}_2 = \texttt{COMM}(m_2), \texttt{cm}_3 = \texttt{COMM}(m_3)$ and a pair of zkSNARK proofs $\pi_0 = \mathrm{Sim}_{zk}(R, \tau, \phi), \pi_1 = \mathrm{Prove}_{zk}(R, \tau, \phi, w)$, where R is an arbitrary relation, τ is the simulator backdoor, ϕ is the statement, and w is the hidden witness, we replace $\texttt{tx}_{\mathsf{i}_0}^{(R)}$ with $(\texttt{LS}, \texttt{vn}_1, \texttt{vn}_2, \texttt{cm}_0, \texttt{cm}_2, \pi_1)$ and replace $\texttt{tx}_{\mathsf{i}_0}^{(I)}$ with $(\texttt{LS}, \texttt{vn}_1, \texttt{vn}_2, \texttt{cm}_1, \texttt{cm}_3, \pi_0)$, where $\texttt{LS}$ is a random group element, $\texttt{vn}_1$ and $\texttt{vn}_2$ are sampled accordingly. Calling Adv on $\texttt{view}_{i_0-1}$ and $\texttt{view}_{i_0}$ will yield a distinguishing advantage of at least $\mathrm{poly}_0(\lambda)/M$, which indicates that either the computational hiding property of $\texttt{COMM}$ or the zero-knowledge property of zkSNARK is broken.
- If $\texttt{tx}_{\mathsf{i}_0}$ is a reclaim transaction, we break the property similarly as the transfer transaction case.

In any of these cases, we have a contradiction with the assumptions. Therefore, the lemma is proved.

Given Lemma 1, it is easy to see that, since the initial parameters are sampled in the same way in both $\mathrm{Real}_{\mathrm{Adv}}(1^\lambda)$ and $\mathrm{Ideal}_{\mathrm{Adv}}(1^\lambda)$, the theorem holds due to Equation 1.

$$
\begin{aligned}
&\big|\Pr[\mathrm{Real}_{\mathrm{Adv}}(1^\lambda) = 1] - \Pr[\mathrm{Ideal}_{\mathrm{Adv},\mathrm{Sim}}(1^\lambda) = 1]\big| \\
&\qquad = \Big| \sum_{\sigma,\tau,N,M,\{sk_i\},\{pk_i\},\texttt{aux}_i,\langle \texttt{op}_1,\ldots,\texttt{op}_M\rangle} \Pr[\mathrm{Real}_{\mathrm{Adv}}(1^\lambda) = 1 | \sigma, \tau, N, M, \{sk_i\}, \{pk_i\}, \texttt{aux}_i, \langle \texttt{op}_1, \ldots, \texttt{op}_M\rangle] \\
&\qquad - \sum_{\sigma,\tau,N,M,\{sk_i\},\{pk_i\},\texttt{aux}_i,\langle \texttt{op}_1,\ldots,\texttt{op}_M\rangle} \Pr[\mathrm{Ideal}_{\mathrm{Adv},\mathrm{Sim}}(1^\lambda) = 1 | \sigma, \tau, N, M, \{sk_i\}, \{pk_i\}, \texttt{aux}_i, \langle \texttt{op}_1, \ldots, \texttt{op}_M\rangle]\Big| \\
&\qquad \leq \|\{\sigma, \tau, N, M, \{sk_i\}, \{pk_i\}, \texttt{aux}_i, \langle \texttt{op}_1, \ldots, \texttt{op}_M\rangle\}\| \cdot \mathrm{negl}(\lambda) = \mathrm{negl}(\lambda).
\end{aligned} \tag{1}
$$

The proofs for the completeness and soundness of the decentralized anonymous payment scheme follow from [BCG+14].

5 Manta$_{\texttt{DAX}}$: Decentralized Anonymous Exchange

In this section, we describe Manta$_{\texttt{DAX}}$, a *Decentralized Anonymous eXchange* (DAX) scheme that extends the DAP scheme (section 4) to support AMM [7] style swap. In principle, our idea is applicable to other types of decentralized exchange. We use AMM in this paper for its elegant simplicity. In a decentralized exchange, there are three entities:

- The users who participate the exchange;
- The pool of trading pairs, which determines the ratio of the trading;
- liquidity provider who supply equal values to the pool.

Without loss of generality, we assume that there already exist two types of coins, denoted by $\texttt{CoinA}$ and $\texttt{CoinB}$, privatized through our Manta$_{\texttt{DAP}}$ protocol in section 4. We also have a pool, denoted by $\mathcal{P} := (\#\texttt{pCoinDex}(A), \#\texttt{pCoinDex}(B), \mathcal{D})$, consists of a pair of values $(\#\texttt{pCoinDex}(A), \#\texttt{pCoinDex}(B))$, and a key-value store $\mathcal{D}$. The key-value store $\mathcal{D}$ is yet another ledger that stores the address of the liquidity providers and their shares of the pool.

We start with a simple construction where the liquidity providers are public; and omit the method of liquidity providing. Then we will proceed to a more complex version that ensures liquidity providers' privacy. Those two methods are compatible with each other. In a real world deployment, one may provide the liquidity through either public or private channel.

In both cases, $\#\texttt{pCoinDex}(A)$ and $\#\texttt{pCoinDex}(B)$ log the current number of *CoinA* and *CoinB* in the pool. In the public setting, the key-value store records the identity of the liquidity provider, and its

[7] AMM can be viewed as a trading pair that always maintains an invariant on the balances of the assets. Please refer [But18] for a detailed explanation.

```
$\mathrm{Sim}(\texttt{op}, \tau, \sigma, \mathrm{arg}, 1^\lambda)$:

  – If op is opMint:
      Extract $< \texttt{LS}, CoinA, \delta, v > \leftarrow \mathrm{arg}$
      Sample $k$ uniformly from the domain of COMM.
      Sample $s$ randomly, and compute $\texttt{cm} \leftarrow \texttt{COMM}_s(\delta \| v \| k)$.
      Return $\texttt{tx}_{\mathsf{mint}} = (\delta, v, k, s, \texttt{cm}), \texttt{aux} = (\texttt{pCoin} := (a_{\mathsf{pk}}, \delta, v, \rho, r, s, \texttt{cm}), \texttt{vn})$.
  – If op is opTransfer:
      Extract $< \texttt{LS} > \leftarrow \mathrm{arg}$
      Sample $v_3, v_4,\ \delta_3,\ \delta_4, r_3, r_4, \rho_3, \rho_4, pk_3, pk_4$ randomly.
      Compute $k_3 = \texttt{COMM}_r(pk_3 \| \rho_3),\ k_4 = \texttt{COMM}_r(pk_4 \| \rho_4)$
      Sample $s_3, s_4$ randomly, and compute $\texttt{cm}_3 \leftarrow \texttt{COMM}_{s_3}(\delta_3 \| v_3 \| k_3), \texttt{cm}_4 \leftarrow \texttt{COMM}_{s_4}(\delta_4 \| v_4 \| k_4)$.
      Sample uniquely $\texttt{vn}_1, \texttt{vn}_2$.
      Compute $\pi \leftarrow \mathrm{Sim}_{zk}(\tau, \texttt{LS}, \texttt{vn}_1, \texttt{vn}_2, \texttt{cm}_3, \texttt{cm}_4)$.
      Return $\texttt{tx}_{\mathsf{transfer}} = (\texttt{LS}, \texttt{vn}_1, \texttt{vn}_2, \texttt{cm}_3, \texttt{cm}_4, \pi), \texttt{aux} = (c_3^{\texttt{new}} := (pk_3, \delta_3, v_3, \rho_3, r_3, s_3, \texttt{cm}_3), c_4^{\texttt{new}} := (pk_4, \delta_4, v_4, \rho_4, r_4, s_4, \texttt{cm}_4))$.
  – If op is opReclaim:
      Extract $< \texttt{LS}, \texttt{PAddr}, v > \leftarrow \mathrm{arg}$
      Sample $v_3, \delta_3, r_3, \rho_3, pk_3$ randomly.
      Compute $k_3 = \texttt{COMM}_r(pk_3 \| \rho_3)$.
      Sample $s_3$ randomly, and compute $\texttt{cm}_3 \leftarrow \texttt{COMM}_{s_3}(\delta_3 \| v_3 \| k_3)$.
      Sample uniquely $\texttt{vn}_1$ and $\texttt{vn}_2$.
      Compute $\pi \leftarrow \mathrm{Sim}_{zk}(\tau, \texttt{LS}, \texttt{vn}_1, \texttt{vn}_2, \texttt{cm}_3, \texttt{PAddr}, v)$.
      Return $\texttt{tx}_{\mathsf{reclaim}} = (\texttt{LS}, \texttt{vn}_1, \texttt{vn}_2, \texttt{cm}_3, \pi, \texttt{PAddr}, v), \texttt{aux} = (c_3^{\texttt{new}} := (pk_3, \delta_3, v_3, \rho_3, r_3, s_3, \texttt{cm}_3))$.
```

Fig. 3: Simulator Sim for the decentralized anonymous payment scheme.

contribution to the pool. In the private setting, the key-value store records a one-time address from the liquidity provider (the liquidity provider's ID remains hidden, nonetheless), and its contribution to the pool. This pool is operated by the validators. In other words, users and liquidity providers can only interact with the pool through blockchain transactions.

The ledger state of Manta$_{\texttt{DAX}}$ can therefore be defined as a tuple $\mathcal{S} = (\texttt{LS}, \mathcal{P})$:

- LS,: the ledger state of private coins, i.e., the accumulators that store all the cm-s and vn-s.
- $\mathcal{P}$: the ledger state of the exchange pair of *CoinA* and *CoinB*. Here, we adopted a simplified design, with "$\#\texttt{pCoinDex}(A) \times \#\texttt{pCoinDex}(B) = T$" market maker scheme [But18].

Similar to a normal automated market maker setting, $\mathcal{P}$ needs to maintain the invariant T a constant before and after an exchange. $\mathcal{P}$ also needs to allow for liquidity supplying and withdrawal, which change the invariant T while maintaining the ratio "$\#\texttt{pCoinDex}(A) : \#\texttt{pCoinDex}(B)$".

5.1 DAX with public liquidity providers

liquidity pool In a nutshell, the liquidity pool can be viewed as a ledger, with many account entries (liquidity providers), in the form of a key value store, where the key is the address of the liquidity provider, and the value is the share, for instance, in terms of percentage, of the pool. For our purpose, we may quantify the share percentage, and use yet another token, i.e., LPCoin to represent the share. More concretely, a liquidity provider, Luis, who holds ℓ number of LPCoin owns $\frac{\ell}{\texttt{LPCoin}_{total}}$ fraction of the pool. Looking ahead, now that we have tokenized the shares, we will be able to use the Manta$_{\texttt{DAP}}$ protocol to shield the liquidity providers' identities, as well as the number of LPCoins they hold.

Let us now formalize some notations.

- We stick with a trading pair of CoinA and CoinB, where the trading ratio is denoted by $\mathcal{R}$; the pool invariant is denote by T.

– An exchange will take in x number of `CoinA`, and outputs $y := \mathcal{R}x$ number of `CoinB` (or vise versa). This procedure will preserve the invariant T, while updating the trading ratio $\mathcal{R}$.
– A liquidity providing(withdrawn) will take input (or output, respectively) x number of `CoinA`, and $y := \mathcal{R}x$ number of `CoinB`, and add (subtract) them to the pool. This procedure will preserve the trading ratio $\mathcal{R}$, while updating the invariant T.

It is therefore sufficient to view the liquidity pool as a tuple $\mathcal{P} = (\mathcal{R}, T, \texttt{LP}_{total}, L_{\texttt{LP}})$.

(i) `LP` interfaces.

A public liquidity provider interacts with the pool through the following interfaces.

– $\texttt{LP.init}(id, x, y) \to \mathcal{P}$: this function initialize the pool with x number of `CoinA` and y number of `CoinB`. It then sets $\mathcal{R} = y/x$, $T = xy$, $amount = \sqrt{T}$; build a ledger $L_{\texttt{LP}}$ with an entry $(id : amount)$ and set $\texttt{LP}_{total} = amount$. It finally output $\mathcal{P} = (x, y, \mathcal{R}, T, \texttt{LP}_{total}, L_{\texttt{LP}})$.
– $\texttt{LP.provide}(id_0, x_0, y_0, \mathcal{P}) \to \mathcal{P}'$: this function firstly checks that $x_0/y_0 = \mathcal{P}.\mathcal{R}$. It then sets
  - $\mathcal{P}.x \mathrel{+}= x_0$;
  - $\mathcal{P}.y \mathrel{+}= y_0$;
  - $amount_0 = \sqrt{x_0 y_0}$;
  - $\mathcal{P}.\texttt{LP}_{total} \mathrel{+}= amount_0$;

  It also adds the entry $(id_0 : amount_0)$ to the ledger $L_{\texttt{LP}}$ if id_0 does not exist; or increase its balance by $amount_0$ if it exists. It finally outputs the updated $\mathcal{P}$ as $\mathcal{P}'$.
– $\texttt{LP.withdraw}(id_1, amount_1, \mathcal{P}) \to \mathcal{P}'$: this function firstly checks that id_1's balance is no less than $amount_1$. It then sets
  - $x_1 = \frac{amount_1}{\mathcal{P}.\texttt{LP}_{total}} \cdot \mathcal{P}.x$;
  - $y_1 = \frac{amount_1}{\mathcal{P}.\texttt{LP}_{total}} \cdot \mathcal{P}.y$;
  - $\mathcal{P}.x \mathrel{-}= x_1$;
  - $\mathcal{P}.y \mathrel{-}= y_1$;
  - $\mathcal{P}.\texttt{LP}_{total} \mathrel{-}= amount_1$;

  It reduces the balance of id_1 by $amount_0$ on the ledger $L_{\texttt{LP}}$. It finally outputs the updated $\mathcal{P}$ as $\mathcal{P}'$.
– $\texttt{LP.exchange}(c_1^{\texttt{old}}, c_2^{\texttt{old}}, \mathcal{P}_{\delta^{\text{old}},\delta}) \to (v^{\texttt{new}}, \mathcal{P}')$: this function updates the states of the liquidity pool by exchanging two coins $c_1^{\texttt{old}}, c_2^{\texttt{old}}$ with asset type δ^{old} to asset type δ. It returns the sum of the values of the new coins and the updated liquidity pool state.

5.2 Exchange

To deal with public liquidity providers, from the user's point of view, it is sufficient to assume that T is a constant, and ignore the liquidity supplying and withdraw for simplicity. Without losing of generality, we show how to privately exchange a single `CoinA` for a single `CoinB`.

(j) $\texttt{GenExchange}(\texttt{LS}, c_1^{\texttt{old}}, c_2^{\texttt{old}}, \delta', v', a_{sk}, b_{sk}, d_{pk}, e_{pk}, \sigma, \texttt{aux}, 1^\lambda)$
$\to (\texttt{tx}_{\textsf{exchange}}, \texttt{pCoinB})$

At a high level, suppose a user u, with address key pairs $(a_{\textsf{pk}}^{\texttt{old}}, a_{\textsf{sk}}^{\texttt{old}})$, $(b_{\textsf{pk}}^{\texttt{old}}, b_{\textsf{sk}}^{\texttt{old}})$, wants to exchange its old `pCoins`

$$c_1^{\texttt{old}} := (a_{\textsf{pk}}^{\texttt{old}}, \delta^{\texttt{old}}, v_1^{\texttt{old}}, \rho_1^{\texttt{old}}, r_1^{\texttt{old}}, s_1^{\texttt{old}}, \texttt{cm}_1^{\texttt{old}})$$
$$c_2^{\texttt{old}} := (b_{\textsf{pk}}^{\texttt{old}}, \delta^{\texttt{old}}, v_2^{\texttt{old}}, \rho_2^{\texttt{old}}, r_2^{\texttt{old}}, s_2^{\texttt{old}}, \texttt{cm}_2^{\texttt{old}})$$

with $\delta^{\texttt{old}}$ = "CoinA", for new `pCoins` of type $\delta^{\texttt{new}}$, denoted by $c_3^{\texttt{new}}$ and $c_4^{\texttt{new}}$, at address $d_{\textsf{pk}}^{\texttt{new}}$ and $e_{\textsf{pk}}^{\texttt{new}}$ [8]. The user then creates the new coins

$$c_3^{\texttt{new}} := (d_{\textsf{pk}}^{\texttt{new}}, \delta^{\texttt{new}}, v_3^{\texttt{new}}, \rho_3^{\texttt{new}}, r_3^{\texttt{new}}, s_3^{\texttt{new}}, \texttt{cm}_3^{\texttt{new}})$$
$$c_4^{\texttt{new}} := (e_{\textsf{pk}}^{\texttt{new}}, \delta^{\texttt{new}}, v_4^{\texttt{new}}, \rho_4^{\texttt{new}}, r_4^{\texttt{new}}, s_4^{\texttt{new}}, \texttt{cm}_4^{\texttt{new}})$$

with $\delta^{\texttt{new}}$ = "CoinB". The rest input field follows those of a `transfer` protocol. The user u also produces a zero-knowledge proof $\pi_{\textsf{exchange}}$ for the following `NP` statement:

[8] u does not need the corresponding secret key to complete the exchange. In fact, the address may be owned by another entity, which makes the `exchange` protocol simultaneously a `transfer` protocol.

Definition 10 (NP Statement of π_{exchange}). *Given an accumulator of ledger acc, serial numbers vn_1^{old} and vn_2^{old}, two coin types δ^{old} and δ^{new}, two values v^{old} and v^{new}, and coin commitments cm_3^{new} and cm_4^{new}, "I" know coins c_1^{old}, c_2^{old} , c_3^{new} c_4^{new}, and secret keys a_{sk}^{old} and b_{sk}^{old} such that:*

1. *Knowledge of the secret key, identical to Definition 8.*
2. *c_1^{old} and c_2^{old} are* well formed*, also identical to Definition 8.*
3. *$cm_1^{old} \in acc$ and $cm_2^{old} \in acc$.*
4. *c_3^{new} and c_4^{new} are* well formed*, identical to Definition 8.*
5. *$v^{old} = v_1^{old} + v_2^{old}$ and $v^{new} = v_3^{new} + v_4^{new}$.*

The user u then sends an exchange transaction $tx_{\text{exchange}} := (acc, \texttt{vn}_1^{\text{old}}, \texttt{vn}_2^{\text{old}}, \texttt{cm}_3^{\text{new}}, \texttt{cm}_4^{\text{new}}, \pi_{\text{exchange}}, \delta^{\text{old}}, v^{\text{old}}, \delta^{\text{new}}, v^{\text{new}})$ to the ledger.

(k) $\texttt{Exchange}(\texttt{LS}, \texttt{tx}_{\text{exchange}}) \rightarrow (\texttt{LS}', \{\top, \bot\})$

Upon receiving a tx_{exchange}, the validator checks the validity of the transaction: the transaction is valid only if $\texttt{vn}_1^{\text{old}}$ and $\texttt{vn}_2^{\text{old}}$ have never been used in a previous transaction (otherwise it is a double spend); the zkSNARK verifier verifies the validity of π_{exchange}; and $v^{new} : v^{old} = \mathcal{R}$. In addition to adding vn-s and cm-s to the accumulators, the validator also updates $\mathcal{P}$'s state with new #pCoinDex(A) and #pCoinDex(B) as follows:

- $\mathcal{P}.x \mathrel{+}= v^{\text{old}}$;
- $\mathcal{P}.y \mathrel{-}= v^{\text{new}}$;
- $\mathcal{P}.\mathcal{R} = \mathcal{P}.x/\mathcal{P}.y$;

As stated in subsection 3.1, in our DAX the values of the exchange is public. This is the requirement of AMM, or else, $\mathcal{P}$ is not able to update its state and maintains functional. Our solution to bypass this issue is similar to our DAP protocol. In analogue, one may relate to the mint function, which also publicly reveals the face values for commitments. By itself we already get good anonymity, since even the face value is leaked, the identity of the owner of the coin remains hidden. In addition, as shown in Figure 2, our `exchange` protocol, similar to `transfer` protocol, takes a set of input coins and output a set of output coins. Only the combined value for input coins and that of output coins is leaked, while the face value for individual coin remains hidden.

5.3 Private Liquidity Provider

Now we are ready to present the private liquidity providing protocol.

(m) $\texttt{GenPrivInit}(c_1^{\text{old}}, \delta_1, c_2^{\text{old}}, \delta_2, a_{sk}, b_{sk}, d_{pk}) \rightarrow (\texttt{tx}_{\text{PrivInit}}, \texttt{aux})$

A liquid provider, with c_1^{old} of δ_1 and c_2^{old} of δ_2 contributes her tokens to the liquidity pool for a trading pair δ_1 and δ_2. In return, she is expecting a new liquidity token c_3^{new} with $\delta^{\text{new}} = \texttt{pCoinLP}$, and some `aux` information that allows her to spend this coin once it is finalized on chain.

The liquid provider first generate the new token

$$c_3^{\text{new}} := (d_{\text{pk}}^{\text{new}}, \delta^{\text{new}}, v_3^{\text{new}}, \rho_3^{\text{new}}, r_3^{\text{new}}, s_3^{\text{new}}, \texttt{cm}_3^{\text{new}}),$$

where $v_3^{\text{new}} = \sqrt{v_1^{\text{old}} v_2^{\text{old}}}$; v_1 and v_2 are the amount of assets in c_1^{old} and c_2^{old}, respectively. She then generate a zero-knowledge proof $\pi_{\texttt{PrivInit}}$ for the following statements:

1. Knowledge of the secret key, identical to Definition 8.
2. c_1^{old} and c_2^{old} are *well formed*, also identical to Definition 8.
3. c_3^{new} is *well formed*, identical to Definition 8.
4. $v_3^{\text{new}} = \sqrt{v_1^{\text{old}} v_2^{\text{old}}}$.

The liquidity provider then sends an transaction $tx_{PrivInit} :=$
$(acc, \texttt{vn}_1^{\text{old}}, \texttt{vn}_2^{\text{old}}, \texttt{cm}_3^{\text{new}}, \pi_{\texttt{PrivInit}}, \delta_1^{\text{old}}, \delta_2^{\text{old}}, v_1^{\text{old}}, v_2^{\text{old}}, v_3^{\text{new}})$ to the validator.

(n) $\texttt{ApplyPrivInit}(\texttt{tx}_{\textsf{PrivInit}}, x, y) \rightarrow \mathcal{P}$

This function firstly checks that $c_1^{\texttt{old}}$ and $c_2^{\texttt{old}}$ contains $v_1^{\texttt{old}}$ and $v_2^{\texttt{old}}$ number of `CoinA` and `CoinB`, respectively. Then it initializes the pool with $v_1^{\texttt{old}}$ number of `CoinA` and $v_2^{\texttt{old}}$ number of `CoinB` and sets $\mathcal{R} = v_2^{\texttt{old}}/v_1^{\texttt{old}}$, $T = v_1^{\texttt{old}}v_2^{\texttt{old}}$, $amount = \sqrt{T}$. It also checks that $c_3^{\texttt{old}}$ contains exactly $v_3^{\texttt{new}}$ number of `CoinLP`. It finally build a private liquidity pool via outputing $\mathcal{P} = (v_1^{\texttt{old}}, v_2^{\texttt{old}}, \mathcal{R}, T, \texttt{LP*}_{total}, L_{\texttt{LP*}})$.

(o) $\texttt{GenPrivProvide}(c_1^{\texttt{old}}, \delta_1, c_2^{\texttt{old}}, \delta_2, a_{sk}, b_{sk}, d_{pk})$
$\rightarrow (\texttt{tx}_{\textsf{PrivProvide}}, \texttt{aux})$

This function is identical to the `GenPrivInit` function.

(p) $\texttt{ApplyPrivProvide}(\texttt{tx}_{\textsf{PrivProvide}}, x, y, \mathcal{P}) \rightarrow \mathcal{P}'$

Upon receiving a $tx_{PrivProvide}$ transaction, this function firstly checks that $c_1^{\texttt{old}}$ and $c_2^{\texttt{old}}$ contains $v_1^{\texttt{old}}$ and $v_2^{\texttt{old}}$ number of `CoinA` and `CoinB`, respectively. Then it checks that $v_1^{\texttt{old}}/v_2^{\texttt{old}} = \mathcal{P}.\mathcal{R}$, and injects the liquidity to the pool with $v_1^{\texttt{old}}$ number of `CoinA` and $v_2^{\texttt{old}}$ number of `CoinB` as follows:

- $\mathcal{P}.x \mathrel{+}= v_1^{\texttt{old}}$;
- $\mathcal{P}.y \mathrel{+}= v_2^{\texttt{old}}$;
- $amount_0 = \sqrt{v_1^{\texttt{old}}v_2^{\texttt{old}}}$;
- $\mathcal{P}.\texttt{LP*}_{total} \mathrel{+}= amount_0$;

and add $\texttt{pCoinLP}_0$ to $L_{\texttt{LP*}}$. It finally outputs the updated $\mathcal{P}$ as $\mathcal{P}'$.

(q) $\texttt{GenPrivWithdraw}(c_1^{\texttt{old}}, v_1^{\texttt{old}}, a_{sk}, \mathcal{P}) \rightarrow (\texttt{tx}_{\textsf{PrivWithdraw}}, \texttt{aux}_1, \texttt{aux}_2)$

A liquid provider, with $c_1^{\texttt{old}}$ of `pCoinLP`, withdraws her tokens from the liquidity pool for a trading pair δ_1 and δ_2. In return, she is expecting two new token $c_2^{\texttt{new}}$ and $c_3^{\texttt{new}}$ with $\delta_2^{\texttt{new}} = \texttt{pCoinA}$ and $\delta_3^{\texttt{new}} = \texttt{pCoinB}$, respectively; and some `aux` information that allows her to spend these coins once it is finalized on chain.

The liquid provider first generate the new token

$$c_2^{\texttt{new}} := (b_{\texttt{pk}}^{\texttt{new}}, \delta_2^{\texttt{new}}, v_2^{\texttt{new}}, \rho_2^{\texttt{new}}, r_2^{\texttt{new}}, s_2^{\texttt{new}}, \texttt{cm}_2^{\texttt{new}}),$$

$$c_3^{\texttt{new}} := (d_{\texttt{pk}}^{\texttt{new}}, \delta_3^{\texttt{new}}, v_3^{\texttt{new}}, \rho_3^{\texttt{new}}, r_3^{\texttt{new}}, s_3^{\texttt{new}}, \texttt{cm}_3^{\texttt{new}}).$$

She then generates a zero-knowledge proof $\pi_{\texttt{PrivWithdraw}}$ for the following statements

1. Knowledge of the secret key, identical to Definition 8.
2. $c_1^{\texttt{old}}$ is *well formed*, also identical to Definition 8.
3. $c_2^{\texttt{new}}$ and $c_3^{\texttt{new}}$ are *well formed*, identical to Definition 8.
4. $v_1^{\texttt{old}} = \sqrt{v_2^{\texttt{mew}}v_3^{\texttt{mew}}}$.

and send a transaction $tx_{PrivWithdraw} :=$
$(acc, \texttt{vn}_1^{\texttt{old}}, \texttt{cm}_2^{\texttt{new}}, \texttt{cm}_3^{\texttt{new}}, \pi_{\texttt{PrivWithdraw}}, \delta_2^{\texttt{new}}, \delta_3^{\texttt{new}}, v_1^{\texttt{old}}, v_2^{\texttt{new}}, v_3^{\texttt{new}})$ to the validators.

(r) $\texttt{ApplyPrivWithdraw}(\texttt{tx}_{\textsf{PrivWithdraw}}, \mathcal{P}) \rightarrow \mathcal{P}'$

Upon receiving a $\texttt{tx}_{\textsf{PrivWithdraw}}$ transaction, this function firstly checks that $c_1^{\texttt{old}}$ contains $v_1^{\texttt{old}}$ number of `pCoinLP`. Then it checks that $v_2^{\texttt{new}}/v_3^{\texttt{new}} = \mathcal{P}.\mathcal{R}$, and removes the liquidity to the pool with $v_2^{\texttt{new}}$ number of `CoinA` and $v_3^{\texttt{new}}$ number of `CoinB` as follows:

- $x_1 = \frac{v_1^{\texttt{old}}}{\mathcal{P}.\texttt{LP*}_{total}} \cdot \mathcal{P}.x$;
- $y_1 = \frac{v_1^{\texttt{old}}}{\mathcal{P}.\texttt{LP*}_{total}} \cdot \mathcal{P}.y$;
- $\mathcal{P}.x \mathrel{-}= x_1$;
- $\mathcal{P}.y \mathrel{-}= y_1$;
- $\mathcal{P}.\texttt{LP*}_{total} \mathrel{-}= v_1^{\texttt{old}}$;

and remove $\texttt{pCoinLP}_0$ to $L_{\texttt{LP*}}$. It finally outputs the updated $\mathcal{P}$ as $\mathcal{P}'$.

(s) $\texttt{Exchange}(c_1^{\texttt{old}}, c_2^{\texttt{old}}, \mathcal{P}_{\delta^{\texttt{old}},\delta}) \rightarrow (v^{\texttt{new}}, \mathcal{P}')$: this function is the same as the `exchange` function of the public liquidity pool.

	Component	Payload breakup	Bytes
Mint	$\delta, v, \texttt{cm}, k, s$	$2 \times \texttt{u64} + 3 \times \texttt{u256}$	112
Transfer	$k_1, \texttt{vn}_1, \texttt{acc}_1$ $k_2, \texttt{vn}_2, \texttt{acc}_2$	$3 \times \texttt{u256}$	608
	$k_3, \texttt{cm}_3, \texttt{cipher}_3$ $k_4, \texttt{cm}_4, \texttt{cipher}_4$	$2 \times \texttt{u256} + 1 \times \texttt{u384}$	
	$\pi_{\texttt{transfer}}$	$2 \times G_1 + 1 \times G_2$	
Exchange	$k_1, \texttt{vn}_1, \texttt{acc}_1$ $k_2, \texttt{vn}_2, \texttt{acc}_2$	$3 \times \texttt{u256}$	640
	$k_3, \texttt{cm}_3, \texttt{cipher}_3$ $k_4, \texttt{cm}_4, \texttt{cipher}_4$	$2 \times \texttt{u256} + 1 \times \texttt{u384}$	
	$\pi_{\texttt{transfer}}$	$2 \times G_1 + 1 \times G_2$	
	$\delta^{old}, v^{\texttt{old}}, \delta^{new}, v^{\texttt{new}}$	$4 \times \texttt{u64}$	
Reclaim	δ, v	$2 \times \texttt{u64}$	512
	$k_1, \texttt{vn}_1, \texttt{acc}_1$ $k_2, \texttt{vn}_2, \texttt{acc}_2$	$3 \times \texttt{u256}$	
	$k_3, \texttt{cm}_3, \texttt{cipher}_3$	$2 \times \texttt{u256} + 1 \times \texttt{u384}$	
	$\pi_{\texttt{reclaim}}$	$2 \times G_1 + 1 \times G_2$	

Table 1: Communication cost

5.4 Security Proof

Proof deferred to section C.

6 Empirical Evaluation

	CRS		Cost	
	PK size	VK size	Proving	Verifying
transfer	226 MB	2312 bytes	14.97 s	5.86 ms
exchange	226 MB	2504 bytes	14.95 s	6.06 ms
reclaim	224 MB	2312 bytes	14.88 s	5.44 ms

Table 2: Unit benchmark

	x86 Native	WASM	e. TPS (N)	e. TPS (W)
Null Trans.	12 μs	28 μs		
Init.	0.7 ms	2.9 ms		
Mint	3.1 ms	20 ms	107.5	16.7
Transfer	10.9 ms	109 ms	30.6	3
Exchange	11 ms	110 ms	30.3	3
Reclaim	8.3 ms	89 ms	40.2	3.75

Table 3: Integrated benchmark

System Implementation. We implement our scheme in Rust, using the Arkworks library [Tea21] and the Substrate framework [Tec21]. We deploy our code to a publicly accessible testnet [9]. Our code is release to the public domain under the GPL license. The underlying cryptographic library allows users to develop their own Substrate based blockchain applications with zero-knowledge proofs, and therefore may be of independent interest. The data reported in this section use BLS12-381 [Bow17] and Jubjub curves [HBHW21, 5.4.8.3], Pedersen commitment [HBHW21, 5.4.1.7] and Groth16 proof systems [Gro16]. However, thanks to Arkworks, our code is configurable with other curves, commitments and provers.

[9] `https://github.com/manta-network`

Concrete Performance. We first report the payload one needs to send for each type of transactions. Mint always build one tokens, at a cost of 112 bytes of payload. Transfer and exchange each consumes two old tokens and produces two new tokens, at a cost of over 600 bytes of payload. Reclaim consumes two old tokens, and produces one new token, at a cost of 512 bytes of payload. Table 1 gives the break down cost of each payload.

Then we report our benchmark results. We benchmark MANTA over an AMD 5900x CPU, 32 GBytes of memory and an Ubuntu 20.04 system. We disabled Rust's standard library, as well as instructions such as `AVX-2`s, in order to be compatible with the WASM runtimes.

There are two sets of benchmarks. The first one is unit benchmarks, in which we measure the cost of generating and verifying zero-knowledge proofs for each set of statements. We uses Rust's `Criterion` crate to conduct this benchmark, and the result are presented in Table 2. Overall, the proving time and verification time are around 15 seconds and 6 milliseconds, respectively.

The second benchmark is to integrate our protocol to the blockchain, deploy the test network, and benchmark the end-to-end latency. For this benchmark, we use `frame-benchmark` toolchain from Substrate. The data are collected from a local node. This helps us to remove the network latency and have a better view of the zk-SNARK related latency.

As shown in Table 3, to set up the base line, a null transfer on our network is almost instant: it takes 12 μs and 28 μs when compiled to the x86-64 native executable and webassembly (WASM) [Ros21] executable, respectively. In general, ZKP related transactions, such as Transfer and Exchange, takes around 10 ms to complete with Rust compiler. This result agrees with the unit benchmark, in that the integrated benchmark indeed takes additional time for various operations such as I/O of the ledger state and verification key deserialization, etc. For WASM compiler, our benchmark shows a 10x slowdown. We leave WASM optimization to future work.

With those data, we estimate the transaction per second (TPS) on both native runtime and WASM runtime. We have configured our block finality time to 6 seconds, within which, 2 seconds is dedicated to validation operations. We follow this configuration from Polkadot. Nonetheless, customized setting will give different performance. Under this setting, within a block, we can stretch 2 $second/3.1\ ms = 645.2$ Mint transactions. In other words, $645.2/6\ seconds = 107.5$ TPS. The TPS for other operations are estimated in a similar fashion.

7 Acknowledgement

We would like to thank Qiudong Xia for his help on the performance benchmark and Gabriela Brown for her suggestions on improving the paper.

References

AC20. Guillermo Angeris and Tarun Chitra. Improved price oracles: Constant function market makers. In *AFT*, pages 80–91. ACM, 2020.

AEC20. Guillermo Angeris, Alex Evans, and Tarun Chitra. When does the tail wag the dog? curvature and market making, 2020.

AEC21. Guillermo Angeris, Alex Evans, and Tarun Chitra. A note on privacy in constant function market makers, 2021.

AZR20. Hayden Adams, Noah Zinsmeister, and Dan Robinson. Uniswap v2 core. https://uniswap.org/whitepaper.pdf, 2020.

BAZB20. Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *Financial Cryptography*, volume 12059 of *Lecture Notes in Computer Science*, pages 423–443. Springer, 2020.

BCG$^+$13. Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for C: verifying program executions succinctly and in zero knowledge. In *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2013.

BCG$^+$14. Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society, 2014.

BCG^{+}20. Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: enabling decentralized private computation. In *IEEE Symposium on Security and Privacy*, pages 947–964. IEEE, 2020.

BCI^{+}13. Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, volume 7785 of *Lecture Notes in Computer Science*, pages 315–333. Springer, 2013.

BJSW18. Samuel Brooks, Anton Jurisevic, Michael Spain, and Kain Warwick. A decentralised payment network and stablecoin. https://synthetix.io/uploads/synthetix_whitepaper.pdf, 2018.

BMSZ20. Foteini Baldimtsi, Varun Madathil, Alessandra Scafuro, and Linfeng Zhou. Anonymous lottery in the proof-of-stake setting. In *CSF*, pages 318–333. IEEE, 2020.

Bow17. Sean Bowe. Bls12-381: New zk-snark elliptic curve construction. https://electriccoin.co/blog/new-snark-curve/, 2017.

But18. Vitalik Buterin. Improving front running resistance of x*y=k market makers. https://ethresear.ch/t/improving-front-running-resistance-of-x-y-k-market-makers/1281, 2018.

CGL^{+}17. Alessandro Chiesa, Matthew Green, Jingcheng Liu, Peihan Miao, Ian Miers, and Pratyush Mishra. Decentralized anonymous micropayments. In *EUROCRYPT (2)*, volume 10211 of *Lecture Notes in Computer Science*, pages 609–642, 2017.

cur21. curve. Curve.fi. https://curve.fi/, 2021.

CZJ^{+}17. Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed E. Kosba, Ari Juels, and Elaine Shi. Solidus: Confidential distributed ledger transactions via PVORM. In *CCS*, pages 701–717. ACM, 2017.

def21. Decentralized finance. https://en.wikipedia.org/wiki/Decentralized_finance, 2021.

DGK^{+}19. Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges, 2019.

dot. Polkadot: Decentralized web 3.0 blockchain interoperability platform. https://polkadot.network/.

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645, 2013.

GKRN18. Steven Goldfeder, Harry A. Kalodner, Dillon Reisman, and Arvind Narayanan. When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies. *Proc. Priv. Enhancing Technol.*, 2018(4):179–199, 2018.

GM17. Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *CCS*, pages 473–489. ACM, 2017.

GMR85. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, pages 291–304. ACM, 1985.

Gra21. Steve Graves. Sec chair gensler: Crypto exchanges need 'protections' from front-running. https://decrypt.co/72088/sec-chair-gensler-crypto-exchanges-need-protections-front-running, 2021.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT (2)*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

HBHW21. Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification version 2021.2.2 [overwinter+sapling+blossom+heartwood+canopy]. https://github.com/zcash/zips/blob/master/protocol/protocol.pdf, 2021.

ide21. idex. Idex high performance decentralized exchange. https://idex.io/, 2021.

KKKZ19. Thomas Kerber, Aggelos Kiayias, Markulf Kohlweiss, and Vassilis Zikas. Ouroboros crypsinous: Privacy-preserving proof-of-stake. In *IEEE Symposium on Security and Privacy*, pages 157–174. IEEE, 2019.

KMNS21. Markulf Kohlweiss, Varun Madathil, Kartik Nayak, and Alessandra Scafuro. On the anonymity guarantees of anonymous proof-of-stake protocols. *IACR Cryptol. ePrint Arch.*, 2021:409, 2021.

KYMM18. George Kappos, Haaroon Yousaf, Mary Maller, and Sarah Meiklejohn. An empirical analysis of anonymity in zcash. In *USENIX Security Symposium*, pages 463–477. USENIX Association, 2018.

LH19. Robert Leshner and Geoffrey Hayes. Compound: The money market protocol. https://compound.finance/documents/Compound.Whitepaper.pdf, 2019.

LTW20. Yuan Lu, Qiang Tang, and Guiling Wang. Zebralancer: Decentralized crowdsourcing of human knowledge atop open blockchain, 2020.

Nak. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf.

NVV18. Neha Narula, Willy Vasquez, and Madars Virza. zkledger: Privacy-preserving auditing for distributed ledgers. In *NSDI*, pages 65–80. USENIX Association, 2018.

PHGR13. Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society, 2013.

Ros21. Andreas Rossberg. Webassembly specification release 1.1 (draft 2021-05-17). https://webassembly.github.io/spec/core/, 2021.

TBP20. Florian Tramèr, Dan Boneh, and Kenny Paterson. Remote side-channel attacks on anonymous transactions. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2739–2756. USENIX Association, August 2020.

Tea21. Arkworks Team. arkworks:an ecosystem for developing and programming with zksnarks. https://github.com/arkworks-rs, 2021.

Tec21. Parity Technologies. Substrate developer hub: Blockchain development for innovators. https://substrate.dev/, 2021.

vS13. Nicolas van Saberhagen. Cryptonote v 2.0. https://github.com/monero-project/research-lab/blob/master/whitepaper/whitepaper.pdf, 2013.

Whi20. Barry WhiteHat. Why you can't build a private uniswap with zkps., 2020.

Woo. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger.

YKM19. Haaroon Yousaf, George Kappos, and Sarah Meiklejohn. Tracing transactions across cryptocurrency ledgers. In *USENIX Security Symposium*, pages 837–850. USENIX Association, 2019.

ZHC$^+$20. Fan Zhang, Warren He, Raymond Cheng, Jernej Kos, Nicholas Hynes, Noah M. Johnson, Ari Juels, Andrew Miller, and Dawn Song. The ekiden platform for confidentiality-preserving, trustworthy, and performant smart contracts. *IEEE Secur. Priv.*, 18(3):17–27, 2020.

A Background: cryptography

A.1 Zero knowledge proofs

In cryptography, a zero-knowledge proof protocol is a proof system, in which one party, a.k.a Alice (the prover), proves to another party, a.k.a Bob (the verifier), that she knows a value x, without conveying any information apart from the fact she knows x. By the seminal work from Goldwasser, Micali, and Rackoff [GMR85], we know that this notion of knowledge can be generalized to any NP statement.

In recent years, emerging from a pure theoretical concept, zero-knowledge proof systems have become practical, thanks to many great work in this space [Gro16, GGPR13, BCG$^+$13, PHGR13]. The major cryptographic primitive used in this paper is a special kind of NIZK: publicly-verifiable preprocessing zero-knowledge Succinct Non-interactive ARgument of Knowledge or zkSNARK for short. We informally define zkSNARK as follows.

For a finite field $\mathbb{F}$, an $\mathbb{F}$-arithmetic circuit, where the inputs, outputs and intermediate values are all in $\mathbb{F}$. We consider circuits that have an input $x \in \mathcal{F}^n$ and an auxiliary input $a \in \mathcal{F}^h$, namely a *witness*. We define arithmetic circuit satisfiability as follows:

Definition 11. *The arithmetic circuit satisfiability problem of an $\mathcal{F}$-arithmetic circuit C: $\mathcal{F}^n \times \mathcal{F}^h \rightarrow \mathcal{F}^l$ is captured by the relation $\mathcal{R}_C = \{(x, a) \in \mathcal{F}^n \times \mathcal{F}^h : C(x, a) = 0^l\}$, where the language $\mathcal{L}_C = \{x \in \mathcal{F}^n : \exists a \in \mathcal{F}^h$ s.t. $C(x, a) = 0^l\}$.*

Given a field $\mathcal{F}$, a zkSNARK for $\mathcal{F}-$arithmetic circuit satisfiability is defined by a triple of a polynomial-time algorithms ($\texttt{KeyGen}$, $\texttt{Prove}$, $\texttt{Verify}$):

1. $\texttt{KeyGen}(1^\lambda, C) \rightarrow (\texttt{pk}, \texttt{vk})$. Taken a security parameter λ (e.g. 128 bits) and an $\mathcal{F}$-arithmetic circuit C, $\texttt{KeyGen}$ probabilistically samples a *proving key* $\texttt{pk}$ and a *verification key* $\texttt{vk}$. Both keys are published as public parameters and can be used for any number of times, to prove/verify the memberships in $\mathcal{L}_C$.
2. $\texttt{Prove}(\texttt{pk}, x, a) \rightarrow \pi$. Taken a proving key $\texttt{pk}$ and any $(x, a) \in \mathcal{R}_C$ as input, the *prover* $\texttt{Prove}$ outputs a non-interactive proof π for the statement $x \in \mathcal{L}_C$.
3. $\texttt{Verify}(\texttt{vk}, x, \pi) \rightarrow b$. Taken a verification key $\texttt{vk}$, x , and a proof π as input, the *verifier* $\texttt{Verify}$ outputs 1 if it is convinced that $x \in \mathcal{L}_C$, and outputs 0 otherwise.

A zkSNARK satisfies the following properties:

Completeness. For every security parameter λ, any $\mathcal{F}-$arithmetic circuit C, and any $(x, a) \in \mathcal{R}_C$, an honest prover must convince the verifier. Namely, output 1 with probability $1 - \text{negl}(\lambda)$ with the following: $(\texttt{pk}, \texttt{vk}) \leftarrow \texttt{KeyGen}(1^\lambda, C)$, $\pi \leftarrow \texttt{Prove}(\texttt{pk}, x, a)$, $b \leftarrow \texttt{Verify}(\texttt{vk}, x, \pi)$.

Soundness. If the verifier accepts a proof from a bounded prover, then the prover must know the secret input corresponding to the witness of the given instance [10].

Succinctness. An honestly-generated proof π has $O_\lambda(1)$ bits and $\texttt{Verify}(\texttt{vk}, x, \pi)$ runs in time $O_\lambda(|x|)$ (O_λ hides a fixed polynomial factor in λ).

Zero knowledge. An honestly-generated proof is perfect zero knowledge: there exists a poly(λ)$-$size simulator $\texttt{Sim}$, who has no access to the secret inputs, can generate a simulated proof, such that all stateful poly(λ)$-$bounded distinguishers $\mathcal{D}$ cannot distinguish this proof from an honest proof.

A.2 Other cryptographic primitives

In addition to zkSNARK, we use the following cryptographic primitives:

- $\texttt{COMM}$, a non-interactive commitment scheme that is both hiding and binding. For example, given a random seed r and a message m, the commitment is $c := \texttt{COMM}_r(m)$. c can be *opened* by revealing r and m, which verifies the commitment.
- pseudorandom functions. More specifically, we use three labeled pseudorandom functions that may be instantiated from a same core function. For a seed x, we derive $\texttt{PRF}_x^{addr}(\cdot)$, $\texttt{PRF}_x^{vn}(\cdot)$, and $\texttt{PRF}_x^{pk}(\cdot)$, which will be used to generate addresses, void numbers and public keys, accordingly.

[10] The formal definition of soundness is based on the concept of extractor, which can be found in [BCI$^+$13].

- Cryptographic accumulators, which support efficient, zero-knowledge, membership proofs. Looking ahead, we will be using Merkle tree based constructions.
- an (elliptic curve) integrated encryption scheme ($\texttt{ECIES}$), which is build on top of (elliptic curve) Diffie-Hellman key exchange, a key derivation function and a blockcipher such as AES. This primitive is useful when the senders encrypt the face values of the tokens under receivers public keys.

B Additional Definitions

B.1 DAP

We first describe two experiments, involving a challenger, an adversary Adv, and a simulator Sim.

Definition 12 (Decentralized Anonymous Payment Privacy). *A decentralized anonymous payment scheme is private if and only if for any p.p.t. adversary* Adv*, we have*

$$\left|\Pr[\mathrm{Real}^{\pi_{DAP}}_{\mathrm{Adv}}(1^{\lambda}) = 1] - \Pr[\mathrm{Ideal}^{\pi_{DAP}}_{\mathrm{Adv},\mathrm{Sim}}(1^{\lambda}) = 1]\right| \leq \mathrm{negl}(\lambda)$$

$\mathrm{Real}^{\pi_{\mathsf{DAP}}}_{\mathrm{Adv}}(1^{\lambda})$:

- The challenger runs the setup: $\sigma, \tau \leftarrow \texttt{Setup}(1^{\lambda})$.
- The challenger samples $N, M, A \leftarrow poly(\lambda)$, and a set of N user identities $\{pk_i, sk_i\}_{i\in[N]}$, a set of asset IDs $\{\delta_i\}$, and N private string $\{\texttt{aux}_i\}_{i\in[N]}$.
- Initialize $\texttt{view} \leftarrow [\sigma, \tau, N, M, \{pk_i\}]$, and initial ledger state $\texttt{LS}$.
- Repeat the following for M times:
  - Sample user index $u \leftarrow [N]$.
  - The challenger samples an op code $\texttt{op}$ from $\{\texttt{opMint}, \texttt{opTransfer}, \texttt{opReclaim}\}$.
  - If $\texttt{op}$ is $\texttt{opMint}$:
    Sample a coin value v, an asset ID δ, and a public coin *Coin*.
    Let $(\texttt{tx}_{\mathsf{mint}}, \texttt{aux}) \leftarrow \texttt{GenMint}(\texttt{LS}, CoinA, pk_u, sk_u, \delta, v, 1^{\lambda})$.
    Let $(\texttt{LS}', b) \leftarrow \texttt{Mint}(\texttt{LS}, \texttt{tx}_{\mathsf{mint}})$
    If $b = \bot$, stop the game and return $\mathrm{Adv}(\texttt{view}, 1^{\lambda})$.
    Update the ledger state $\texttt{LS} \leftarrow \texttt{LS}'$
    Append $(\texttt{tx}_{\mathsf{mint}}, \texttt{LS}')$ to $\texttt{view}$, and append $\texttt{aux}$ to $\texttt{aux}_u$.
  - If $\texttt{op}$ is $\texttt{opTransfer}$:
    Sample private coins c_1^{old} and c_2^{old} with private key sk_a, sk_b from $\texttt{aux}_u$, and target addresses pk_d, pk_e.
    Let $(\texttt{tx}_{\mathsf{transfer}}, \texttt{aux}) \leftarrow \texttt{GenTransfer}(\texttt{LS}, c_1^{\mathsf{old}}, c_2^{\mathsf{old}}, sk_a, sk_b, pk_d, pk_e, \sigma, \texttt{aux}, 1^{\lambda})$.
    Let $(\texttt{LS}', b) \leftarrow \texttt{Transfer}(\texttt{LS}, \texttt{tx}_{\mathsf{transfer}})$
    If $b = \bot$, stop the game and return $\mathrm{Adv}(\texttt{view}, 1^{\lambda})$.
    Update the ledger state $\texttt{LS} \leftarrow \texttt{LS}'$
    Append $(\texttt{tx}_{\mathsf{transfer}}, \texttt{LS}')$ to $\texttt{view}$, and append $\texttt{aux}$ to $\texttt{aux}_u$.
  - If $\texttt{op}$ is $\texttt{opReclaim}$:
    Sample private coins c_1^{old} and c_2^{old} with private key sk_a, sk_b from $\texttt{aux}_u$, a target public key pk_d, a public address $\texttt{PAddr}$, and a public coin value v
    Let $(\texttt{tx}_{\mathsf{reclaim}}, \texttt{aux}) \leftarrow \texttt{GenReclaim}(\texttt{LS}, c_1^{\mathsf{old}}, c_2^{\mathsf{old}}, sk_a, sk_b, pk_d, \texttt{PAddr}, \delta, v, \sigma, \texttt{aux}, 1^{\lambda})$.
    Let $(\texttt{LS}', b) \leftarrow \texttt{Reclaim}(\texttt{LS}, \texttt{tx}_{\mathsf{reclaim}})$
    If $b = \bot$, stop the game and return $\mathrm{Adv}(\texttt{view}, 1^{\lambda})$.
    Update the ledger state $\texttt{LS} \leftarrow \texttt{LS}'$
    Append $(\texttt{tx}_{\mathsf{transfer}}, \texttt{LS}')$ to $\texttt{view}$, and append $\texttt{aux}$ to $\texttt{aux}_u$.
- The challenger returns $\mathrm{Adv}(\texttt{view})$.

Fig. 4: Experiments $\mathrm{Real}^{\pi_{\mathsf{DAP}}}_{A}(1^{\lambda})$

$\text{Ideal}^{\pi_{\mathsf{DAP}}}_{\text{Adv},\text{Sim}}(1^{\lambda})$:

- The challenger runs the setup: $\sigma, \tau \leftarrow \mathtt{Setup}(1^{\lambda})$.
- The challenger samples $N, M, A \leftarrow \mathit{poly}(\lambda)$, and a set of N user identities $\{pk_i, sk_i\}_{i \in [N]}$, a set of asset IDs $\{\delta_i\}$, and N private string $\{\mathtt{aux}_i\}_{i \in [N]}$.
- Initialize $\mathtt{view} \leftarrow [\sigma, \tau, N, M, \{pk_i\}]$, and initial ledger state $\mathtt{LS}$.
- Repeat the following for M times:
  - Sample user index $u \leftarrow [N]$.
  - The challenger samples an op code $\mathtt{op}$ from $\{\mathtt{opMint}, \mathtt{opTransfer}, \mathtt{opReclaim}\}$.
  - If $\mathtt{op}$ is $\mathtt{opMint}$:
Sample a coin value v, an asset ID δ, and a public coin Coin.
Let $(\mathtt{tx}_{\mathsf{mint}}, \mathtt{aux}) \leftarrow \text{Sim}(\mathtt{op}, \tau, \sigma, < \mathtt{LS}, \mathit{CoinA}, \delta, v >, 1^{\lambda}))$.
Let $(\mathtt{LS}', b) \leftarrow \mathtt{Mint}(\mathtt{LS}, \mathtt{tx}_{\mathsf{mint}})$
If $b = \perp$, stop the game and return $\text{Adv}(\mathtt{view}, 1^{\lambda})$.
Update the ledger state $\mathtt{LS} \leftarrow \mathtt{LS}'$
Append $(\mathtt{tx}_{\mathsf{mint}}, \mathtt{LS}')$ to $\mathtt{view}$, and append $\mathtt{aux}$ to $\mathtt{aux}_u$.
  - If $\mathtt{op}$ is $\mathtt{opTransfer}$:
Sample private coins c_1^{old} and c_2^{old} with private key sk_a, sk_b from $\mathtt{aux}_u$, and target addresses pk_d, pk_e.
Let $(\mathtt{tx}_{\mathsf{transfer}}, \mathtt{aux}) \leftarrow \text{Sim}(\mathtt{op}, \tau, \sigma, < \mathtt{LS} >, \sigma, 1^{\lambda})$.
Let $(\mathtt{LS}', b) \leftarrow \mathtt{Transfer}(\mathtt{LS}, \mathtt{tx}_{\mathsf{transfer}})$
If $b = \perp$, stop the game and return $\text{Adv}(\mathtt{view}, 1^{\lambda})$.
Update the ledger state $\mathtt{LS} \leftarrow \mathtt{LS}'$
Append $(\mathtt{tx}_{\mathsf{transfer}}, \mathtt{LS}')$ to $\mathtt{view}$, and append $\mathtt{aux}$ to $\mathtt{aux}_u$.
  - If $\mathtt{op}$ is $\mathtt{opReclaim}$:
Sample private coins c_1^{old} and c_2^{old} with private key sk_a, sk_b from $\mathtt{aux}_u$, a target public key pk_d, a public address $\mathtt{PAddr}$, and a public coin value v
Let $(\mathtt{tx}_{\mathsf{reclaim}}, \mathtt{aux}) \leftarrow \text{Sim}(\mathtt{op}, \tau, \sigma, < \mathtt{LS}, \mathtt{PAddr}, \delta, v >, 1^{\lambda})$.
Let $(\mathtt{LS}', b) \leftarrow \mathtt{Reclaim}(\mathtt{LS}, \mathtt{tx}_{\mathsf{reclaim}})$
If $b = \perp$, stop the game and return $\text{Adv}(\mathtt{view}, 1^{\lambda})$.
Update the ledger state $\mathtt{LS} \leftarrow \mathtt{LS}'$
Append $(\mathtt{tx}_{\mathsf{transfer}}, \mathtt{LS}')$ to $\mathtt{view}$, and append $\mathtt{aux}$ to $\mathtt{aux}_u$.
- The challenger returns $\text{Adv}(\mathtt{view})$.

Fig. 5: Experiments $\text{Ideal}^{\pi_{\mathsf{DAP}}}_{\text{Adv},\text{Sim}}(1^{\lambda})$

$$\Pr\left[\mathtt{Transfer}(\mathtt{LS}, \mathtt{tx}_{\mathsf{transfer}}) \rightarrow (\mathtt{LS}', \top) : \begin{array}{c} \mathtt{Setup}(1^{\lambda}) \rightarrow (\sigma, \tau), \\ \mathtt{GenMint}(\mathtt{LS}, \mathit{CoinA}, a_{pk}, a_{sk}, \delta, v, 1^{\lambda}) \rightarrow (\mathtt{tx}_{\mathsf{mint}}, \mathtt{aux}), \\ \mathtt{Mint}(\mathtt{LS}, \mathtt{tx}_{\mathsf{mint}}) \rightarrow (\mathtt{LS}', \top), \\ \mathtt{GenTransfer}(\mathtt{LS}, \mathit{pCoinA}, a_{pk}, a_{sk}, b_{pk}, \sigma, \mathtt{aux}, 1^{\lambda}) \rightarrow (\mathtt{tx}_{\mathsf{transfer}}) \end{array}\right] = 1 \tag{2}$$

$$\Pr\left[\mathtt{Reclaim}(\mathtt{LS}, \mathtt{tx}_{\mathsf{reclaim}}) \rightarrow (\mathtt{LS}', \top) : \begin{array}{c} \mathtt{Setup}(1^{\lambda}) \rightarrow (\sigma, \tau), \\ \mathtt{GenMint}(\mathtt{LS}, \mathit{CoinA}, a_{pk}, a_{sk}, \delta, v, 1^{\lambda}) \rightarrow (\mathtt{tx}_{\mathsf{mint}}, \mathtt{aux}), \\ \mathtt{Mint}(\mathtt{LS}, \mathtt{tx}_{\mathsf{mint}}) \rightarrow (\mathtt{LS}', \top), \\ \mathtt{GenReclaim}(\mathtt{LS}, \mathit{pCoinA}, a_{pk}, a_{sk}, \mathtt{PAddr}, v, \sigma, \mathtt{aux}, 1^{\lambda}) \rightarrow (\mathtt{tx}_{\mathsf{reclaim}}) \end{array}\right] = 1 \tag{3}$$

$$\Pr\left[\mathtt{Transfer}(\mathtt{LS}, \mathtt{tx}_{\mathsf{transfer}}) \rightarrow (\mathtt{LS}', \top) : \begin{array}{c} \mathtt{Setup}(1^{\lambda}) \rightarrow (\sigma, \tau), \\ \mathtt{GenMint}(\mathtt{LS}, \mathit{CoinA}, a_{pk}, a_{sk}, \delta, v, 1^{\lambda}) \rightarrow (\mathtt{tx}_{\mathsf{mint}}, \mathtt{aux}), \\ \mathtt{Mint}(\mathtt{LS}, \mathtt{tx}_{\mathsf{mint}}) \rightarrow (\mathtt{LS}', \top), \\ \mathtt{GenTransfer}(\mathtt{LS}, \mathit{pCoinA}, a_{pk}, a_{sk}, b_{pk}, \sigma, \mathtt{aux}, 1^{\lambda}) \rightarrow (\mathtt{tx}_{\mathsf{transfer}}) \\ \mathtt{Transfer}(\mathtt{LS}, \mathtt{tx}_{\mathsf{transfer}}) \rightarrow (\mathtt{LS}', \top) \\ \mathtt{GenTransfer}(\mathtt{LS}, \mathit{pCoinA}, b_{pk}, b_{sk}, c_{pk}, \sigma, \mathtt{aux}, 1^{\lambda}) \rightarrow (\mathtt{tx}_{\mathsf{transfer}}) \end{array}\right] = 1 \tag{4}$$

$$\Pr\left[\texttt{Reclaim}(\texttt{LS}, \texttt{tx}_{\textsf{reclaim}}) \to (\texttt{LS}', \top) : \begin{array}{c} \texttt{Setup}(1^\lambda) \to (\sigma, \tau), \\ \texttt{GenMint}(\texttt{LS}, \mathit{CoinA}, a_{pk}, a_{sk}, \delta, v, 1^\lambda) \to (\texttt{tx}_{\textsf{mint}}, \texttt{aux}), \\ \texttt{Mint}(\texttt{LS}, \texttt{tx}_{\textsf{mint}}) \to (\texttt{LS}', \top), \\ \texttt{GenTransfer}(\texttt{LS}, \mathit{pCoinA}, a_{pk}, a_{sk}, b_{pk}, \sigma, \texttt{aux}, 1^\lambda) \to (\texttt{tx}_{\textsf{transfer}}) \\ \texttt{Transfer}(\texttt{LS}, \texttt{tx}_{\textsf{transfer}}) \to (\texttt{LS}', \top) \\ \texttt{GenReclaim}(\texttt{LS}, \mathit{pCoinA}, b_{pk}, b_{sk}, \texttt{PAddr}, v, \sigma, \texttt{aux}, 1^\lambda) \to (\texttt{tx}_{\textsf{reclaim}}) \end{array}\right] = 1 \qquad (5)$$

Definition 13 (Decentralized Anonymous Payment Completeness). *A decentralized anonymous payment scheme is complete if and only if for any keypairs $a_{pk}, a_{sk}, b_{pk}, b_{sk}, c_{pk}$, any valid asset id δ and value v, the following probabilities are 1.*

- *Correctly minted coin is transferrable, i.e. Equation 2.*
- *Correctly minted coin is reclaimable, i.e., Equation 3.*
- *Correctly received coin is transferrable, i.e., Equation 4.*
- *Correctly received coin is reclaimable, i.e., Equation 5.*

$$\Pr\left[\begin{array}{c} \texttt{tx}_{\textsf{transfer}} \text{ transfers } \mathit{pCoinA} \text{ to } b_{pk} \\ \texttt{tx}'_{\textsf{transfer}} \text{ transfers } \mathit{pCoinA} \text{ to } b'_{pk} \\ b_{pk} \neq b'_{pk} \\ \texttt{Transfer}(\texttt{LS}, \texttt{tx}_{\textsf{transfer}}) \to (\texttt{LS}', \top) \\ \texttt{Transfer}(\texttt{LS}', \texttt{tx}'_{\textsf{transfer}}) \to (\texttt{LS}'', \top) \end{array} : \begin{array}{c} \texttt{Setup}(1^\lambda) \to (\sigma, \tau), \\ \texttt{GenMint}(\texttt{LS}, \mathit{CoinA}, a_{pk}, a_{sk}, \delta, v, 1^\lambda) \to (\texttt{tx}_{\textsf{mint}}, \texttt{aux}), \\ \texttt{Mint}(\texttt{LS}, \texttt{tx}_{\textsf{mint}}) \to (\texttt{LS}', \top), \\ \mathrm{Adv}(\mathit{pCoinA}, a_{pk}, a_{sk}, 1^\lambda) \to b_{pk}, b'_{pk}, \texttt{tx}_{\textsf{transfer}}, \texttt{tx}'_{\textsf{transfer}} \end{array}\right] \leq \mathrm{negl}(\lambda) \qquad (6)$$

$$\Pr\left[\begin{array}{c} \texttt{tx}_{\textsf{reclaim}} \text{ Reclaims } \mathit{pCoinA} \text{ to } b_{pk} \\ \texttt{tx}'_{\textsf{reclaim}} \text{ Reclaims } \mathit{pCoinA} \text{ to } b'_{pk} \\ b_{pk} \neq b'_{pk} \\ \texttt{Reclaim}(\texttt{LS}, \texttt{tx}_{\textsf{reclaim}}) \to (\texttt{LS}', \top) \\ \texttt{Reclaim}(\texttt{LS}', \texttt{tx}'_{\textsf{reclaim}}) \to (\texttt{LS}'', \top) \end{array} : \begin{array}{c} \texttt{Setup}(1^\lambda) \to (\sigma, \tau), \\ \texttt{GenMint}(\texttt{LS}, \mathit{CoinA}, a_{pk}, a_{sk}, \delta, v, 1^\lambda) \to (\texttt{tx}_{\textsf{mint}}, \texttt{aux}), \\ \texttt{Mint}(\texttt{LS}, \texttt{tx}_{\textsf{mint}}) \to (\texttt{LS}', \top), \\ \mathrm{Adv}(\mathit{pCoinA}, a_{pk}, a_{sk}, 1^\lambda) \to b_{pk}, b'_{pk}, \texttt{tx}_{\textsf{reclaim}}, \texttt{tx}'_{\textsf{reclaim}} \end{array}\right] \leq \mathrm{negl}(\lambda) \qquad (7)$$

Definition 14 (Decentralized Anonymous Payment Security). *A decentralized anonymous payment scheme is secure if and only if for any computationally bounded adversary* Adv*, given oracle access to the scheme, the probabilities in Equation 6 and Equation 7 are negligible.*

B.2 DAX with public Liquidity provider

Similar to DAP privacy, we construct two experiments, involving a challenger, an adversary Adv, and a simulator Sim. For simplicity, our definition only captures a single exchange pair `pCoinA` and `pCoinB`.

Definition 15 (Decentralized Anonymous Exchange Privacy with Public Liquidity Provider). *A decentralized anonymous exchange scheme with public liquidity provider is private if and only if for any p.p.t. adversary* Adv*, we have*

$$\left|\Pr[\mathrm{Real}^{\pi_{DAX}}_{\mathrm{Adv}}(1^\lambda) = 1] - \Pr[\mathrm{Ideal}^{\pi_{DAX}}_{\mathrm{Adv},\mathrm{Sim}}(1^\lambda) = 1]\right| \leq \mathrm{negl}(\lambda)$$

B.3 DAX with private Liquidity provider

Similar to DAP privacy, we construct two experiments, involving a challenger, an adversary Adv, and a simulator Sim. For simplicity, our definition only captures a single exchange pair *pCoinA* and *pCoinB*.

Definition 16 (Decentralized Anonymous Exchange Privacy with Private Liquidity Provider).

A decentralized anonymous exchange scheme is private if and only if for any p.p.t. adversary Adv*, we have*

$$\left| \Pr[\mathrm{Real}^{\pi_{\mathit{DAX*}}}_{\mathrm{Adv}}(1^\lambda) = 1] - \Pr[\mathrm{Ideal}^{\pi_{\mathit{DAX*}}}_{\mathrm{Adv},\mathrm{Sim}}(1^\lambda) = 1] \right| \leq \mathrm{negl}(\lambda)$$

Definition 17 (Decentralized Anonymous Exchange Completeness). *A decentralized anonymous payment scheme (with public/private liquidity provider) is complete if and only if for any user with keypair a_{pk}, a_{sk} and any target a'_{pk} with valid asset ids $\delta$and δ', and value v, Equation 8 holds; for a private exchange scheme with private liquidity provider, Equation 9 also holds;*

Definition 18 (Decentralized Anonymous Exchange Security). *A decentralized exchange payment scheme is secure if and only if for any computationally bounded adversary* Adv*, any asset id δ, δ', any value x, y, given oracle access to the scheme, the probabilities in Equation 10 and Equation 11 are negligible.*

C Security Proof for Private Liquidity Provider

Figure 10 shows a feasible simulator for the private decentralized exchange protocol with private liquidity provider. The simulator for the private decentralized exchange protocol with public liquidity provider is exactly the same. We can show that our private decentralized exchange protocol with private liquidity provider and that with public liquidity provider are private. Both of the proofs are similar to theorem 1.

The proofs for the completeness and soundness of the decentralized anonymous payment scheme are similar to [BCG+14].

$\mathrm{Real}^{\pi_{\mathtt{DAX}}}_{\mathrm{Adv}}(1^{\lambda})$:

- The challenger runs the setup: $\sigma, \tau \leftarrow \mathtt{Setup}(1^{\lambda})$.
- The challenger samples $N, M, x, y \leftarrow poly(\lambda)$, a set of N user identities $\{pk_i, sk_i\}_{i\in[N]}$, a set of asset IDs $\{\delta_i\}$, and N private string $\{\mathtt{aux}_i\}_{i\in[N]}$.
- Initialize the liquidity pool $P_{\delta_i,\delta_j} \leftarrow \mathtt{LP}.init(0, x, y)$ for every asset pair (δ_i, δ_j), adversary view $\mathtt{view} \leftarrow [\sigma, \tau, N, M, \{pk_i\}, (\mathtt{LP}_{\delta_i,\delta_j})_{i,j}]$, and initial ledger state $\mathtt{LS}$.
- Repeat the following for M times:
  - Sample user index $u \leftarrow [N]$.
  - Sample an asset id $\delta \in \{\delta_i\}$.
  - The challenger samples an op code $\mathtt{op}$ from $\{\mathtt{opMint}, \mathtt{opTransfer}, \mathtt{opReclaim}, \mathtt{opExchange}, \mathtt{opProvide}, \mathtt{opWithdraw}\}$.
  - If $\mathtt{op}$ is $\mathtt{opMint}$:
    Sample a public coin $Coin$ and a coin value v. Let $(\mathtt{tx}_{\mathsf{mint}}, \mathtt{aux}) \leftarrow \mathtt{GenMint}(\mathtt{LS}, CoinA, pk_u, sk_u, \delta, v, 1^{\lambda})$.
    Let $(\mathtt{LS}', b) \leftarrow \mathtt{Mint}(\mathtt{LS}, \mathtt{tx}_{\mathsf{mint}})$
    If $b = \perp$, stop the game and return $\mathrm{Adv}(\mathtt{view}, 1^{\lambda})$.
    Update the ledger state $\mathtt{LS} \leftarrow \mathtt{LS}'$
    Append $(\mathtt{tx}_{\mathsf{mint}}, \mathtt{LS}')$ to $\mathtt{view}$, and append $\mathtt{aux}$ to $\mathtt{aux}_u$.
  - If $\mathtt{op}$ is $\mathtt{opTransfer}$:
    Sample private coins $c_1^{\mathtt{old}}$ and $c_2^{\mathtt{old}}$ with private key sk_a, sk_b from $\mathtt{aux}_u$, and target addresses pk_d, pk_e.
    Let $(\mathtt{tx}_{\mathsf{transfer}}, \mathtt{aux}) \leftarrow \mathtt{GenTransfer}(\mathtt{LS}, c_1^{\mathtt{old}}, c_2^{\mathtt{old}}, sk_a, sk_b, pk_d, pk_e, \sigma, \mathtt{aux}, 1^{\lambda})$.
    Let $(\mathtt{LS}', b) \leftarrow \mathtt{Transfer}(\mathtt{LS}, \mathtt{tx}_{\mathsf{transfer}})$
    If $b = \perp$, stop the game and return $\mathrm{Adv}(\mathtt{view}, 1^{\lambda})$.
    Update the ledger state $\mathtt{LS} \leftarrow \mathtt{LS}'$
    Append $(\mathtt{tx}_{\mathsf{transfer}}, \mathtt{LS}')$ to $\mathtt{view}$, and append $\mathtt{aux}$ to $\mathtt{aux}_u$.
  - If $\mathtt{op}$ is $\mathtt{opReclaim}$:
    Sample private coins $c_1^{\mathtt{old}}$ and $c_2^{\mathtt{old}}$ with private key sk_a, sk_b from $\mathtt{aux}_u$, a target public key pk_d, a public address $\mathtt{PAddr}$, and a public coin value v
    Let $(\mathtt{tx}_{\mathsf{reclaim}}, \mathtt{aux}) \leftarrow \mathtt{GenReclaim}(\mathtt{LS}, c_1^{\mathtt{old}}, c_2^{\mathtt{old}}, sk_a, sk_b, pk_d, \mathtt{PAddr}, \delta, v, \sigma, \mathtt{aux}, 1^{\lambda})$.
    Let $(\mathtt{LS}', b) \leftarrow \mathtt{Reclaim}(\mathtt{LS}, \mathtt{tx}_{\mathsf{reclaim}})$
    If $b = \perp$, stop the game and return $\mathrm{Adv}(\mathtt{view}, 1^{\lambda})$.
    Update the ledger state $\mathtt{LS} \leftarrow \mathtt{LS}'$
    Append $(\mathtt{tx}_{\mathsf{transfer}}, \mathtt{LS}')$ to $\mathtt{view}$, and append $\mathtt{aux}$ to $\mathtt{aux}_u$.
  - If $\mathtt{op}$ is $\mathtt{opExchange}$:
    Sample a private coin $c_1^{\mathtt{old}}, c_2^{\mathtt{old}}$ with private key sk_a, sk_b from $\mathtt{aux}_u$. Determine the coin type $\delta^{\mathtt{old}}$. Sample the target public key pk_d, pk_e.
    Let $(v^{\mathtt{new}}, \mathcal{P}_{\delta^{\mathtt{old}},\delta}) \leftarrow \mathtt{LP.exchange}(c_1^{\mathtt{old}}, c_2^{\mathtt{old}}, \mathcal{P}_{\delta^{\mathtt{old}},\delta})$.
    Let $(\mathtt{tx}_{\mathsf{exchange}}, \mathtt{aux}) \leftarrow \mathtt{GenExchange}(\mathtt{LS}, c_1^{\mathtt{old}}, c_2^{\mathtt{old}}, \delta, v^{\mathtt{new}}, sk_a, sk_b, pk_d, pk_e, \sigma, \mathtt{aux}, 1^{\lambda})$.
    Let $(\mathtt{LS}', b) \leftarrow \mathtt{Exchange}(\mathtt{LS}, \mathtt{tx}_{\mathsf{exchange}})$
    If $b = \perp$, stop the game and return $\mathrm{Adv}(\mathtt{view}, 1^{\lambda})$.
    Update the ledger state $\mathtt{LS} \leftarrow \mathtt{LS}'$ and the coin counters $\mathcal{P}_{\delta^{\mathtt{old}},\delta}.x$ and $\mathcal{P}_{\delta^{\mathtt{old}},\delta}.y$
    Append $(\mathtt{tx}_{\mathsf{exchange}}, \mathtt{LS}', \mathcal{P}_{\delta^{\mathtt{old}},\delta})$ to $\mathtt{view}$, and append $\mathtt{aux}$ to $\mathtt{aux}_u$.
  - If $\mathtt{op}$ is $\mathtt{opProvide}$:
    Sample $id \leftarrow poly(\lambda), x, y \leftarrow poly(\lambda)$.
    Let $\mathcal{P} \leftarrow \mathtt{LP.provide}(id, x, y, \mathcal{P})$. Append $\mathcal{P}_{\delta^{\mathtt{old}},\delta}$ to $\mathtt{view}$.
  - If $\mathtt{op}$ is $\mathtt{opWithdraw}$:
    Sample $id \leftarrow poly(\lambda), a \leftarrow poly(\lambda)$.
    Let $\mathcal{P} \leftarrow \mathtt{LP.withdraw}(id, a, \mathcal{P})$. Append $\mathcal{P}_{\delta^{\mathtt{old}},\delta}$ to $\mathtt{view}$.
- The challenger returns $\mathrm{Adv}(\mathtt{view})$.

Fig. 6: Experiments $\mathrm{Real}^{\pi_{\mathtt{DAX}}}_{A}(1^{\lambda})$

Ideal$^{\pi_{\mathtt{DAX}}}_{\mathrm{Adv},\mathrm{Sim}}(1^\lambda)$:

- The challenger runs the setup: $\sigma, \tau \leftarrow \mathtt{Setup}(1^\lambda)$.
- The challenger samples $N, M, x, y \leftarrow poly(\lambda)$, a set of N user identities $\{pk_i, sk_i\}_{i \in [N]}$, a set of asset IDs $\{\delta_i\}$, and N private string $\{\mathtt{aux}_i\}_{i \in [N]}$.
- Initialize the liquidity pool $P_{\delta_i,\delta_j} \leftarrow \mathtt{LP}.init(0, x, y)$ for every asset pair (δ_i, δ_j), adversary view $\mathtt{view} \leftarrow [\sigma, \tau, N, M, \{pk_i\}, (\mathtt{LP}_{\delta_i,\delta_j})_{i,j}]$, and initial ledger state $\mathtt{LS}$.
- Repeat the following for M times:
    - Sample user index $u \leftarrow [N]$.
    - Sample an asset id $\delta \in \{\delta_i\}$.
    - The challenger samples an op code $\mathtt{op}$ from $\{\mathtt{opMint}, \mathtt{opTransfer}, \mathtt{opReclaim}, \mathtt{opExchange}, \mathtt{opProvide}, \mathtt{opWithdraw}\}$.
    - If $\mathtt{op}$ is $\mathtt{opMint}$:
      Sample a public coin $Coin$ and a coin value v.
      Let $(\mathtt{tx}_{\mathsf{mint}}, \mathtt{aux}) \leftarrow \mathrm{Sim}(\mathtt{op}, \tau, \sigma, < \mathtt{LS}, CoinA, \delta, v >, 1^\lambda))$.
      Let $(\mathtt{LS}', b) \leftarrow \mathtt{Mint}(\mathtt{LS}, \mathtt{tx}_{\mathsf{mint}})$
      If $b = \perp$, stop the game and return $\mathrm{Adv}(\mathtt{view}, 1^\lambda)$.
      Update the ledger state $\mathtt{LS} \leftarrow \mathtt{LS}'$
      Append $(\mathtt{tx}_{\mathsf{mint}}, \mathtt{LS}')$ to $\mathtt{view}$, and append $\mathtt{aux}$ to $\mathtt{aux}_u$.
    - If $\mathtt{op}$ is $\mathtt{opTransfer}$:
      Sample private coins c_1^{old} and c_2^{old} with private key sk_a, sk_b from $\mathtt{aux}_u$, and target addresses pk_d, pk_e.
      Let $(\mathtt{tx}_{\mathsf{transfer}}, \mathtt{aux}) \leftarrow \mathrm{Sim}(\mathtt{op}, \tau, \sigma, < \mathtt{LS} >, \sigma, 1^\lambda)$.
      Let $(\mathtt{LS}', b) \leftarrow \mathtt{Transfer}(\mathtt{LS}, \mathtt{tx}_{\mathsf{transfer}})$
      If $b = \perp$, stop the game and return $\mathrm{Adv}(\mathtt{view}, 1^\lambda)$.
      Update the ledger state $\mathtt{LS} \leftarrow \mathtt{LS}'$
      Append $(\mathtt{tx}_{\mathsf{transfer}}, \mathtt{LS}')$ to $\mathtt{view}$, and append $\mathtt{aux}$ to $\mathtt{aux}_u$.
    - If $\mathtt{op}$ is $\mathtt{opReclaim}$:
      Sample private coins c_1^{old} and c_2^{old} with private key sk_a, sk_b from $\mathtt{aux}_u$, a target public key pk_d, a public address $\mathtt{PAddr}$, and a public coin value v
      Let $(\mathtt{tx}_{\mathsf{reclaim}}, \mathtt{aux}) \leftarrow \mathrm{Sim}(\mathtt{op}, \tau, \sigma, < \mathtt{LS}, \mathtt{PAddr}, \delta, v >, 1^\lambda)$.
      Let $(\mathtt{LS}', b) \leftarrow \mathtt{Reclaim}(\mathtt{LS}, \mathtt{tx}_{\mathsf{reclaim}})$
      If $b = \perp$, stop the game and return $\mathrm{Adv}(\mathtt{view}, 1^\lambda)$.
      Update the ledger state $\mathtt{LS} \leftarrow \mathtt{LS}'$
      Append $(\mathtt{tx}_{\mathsf{transfer}}, \mathtt{LS}')$ to $\mathtt{view}$, and append $\mathtt{aux}$ to $\mathtt{aux}_u$.
    - If $\mathtt{op}$ is $\mathtt{opExchange}$:
      Sample a private coin $c_1^{\mathrm{old}}, c_2^{\mathrm{old}}$ with private key sk_a, sk_b from $\mathtt{aux}_u$. Determine the coin type δ^{old}. Sample the target public key pk_d, pk_e.
      Let $(v^{\mathtt{new}}, \mathcal{P}_{\delta^{\mathtt{old}},\delta}) \leftarrow \mathtt{LP.exchange}(c_1^{\mathtt{old}}, c_2^{\mathtt{old}}, \mathcal{P}_{\delta^{\mathtt{old}},\delta})$.
      Let $(\mathtt{tx}_{\mathsf{exchange}}, \mathtt{aux}) \leftarrow \mathrm{Sim}(\mathtt{op}, \tau, \sigma, < \mathtt{LS}, (\mathcal{P}_{\delta^{\mathtt{old}},\delta}) >, \sigma, 1^\lambda)$.
      Let $(\mathtt{LS}', b) \leftarrow \mathtt{Transfer}(\mathtt{LS}, \mathtt{tx}_{\mathsf{exchange}})$
      If $b = \perp$, stop the game and return $\mathrm{Adv}(\mathtt{view}, 1^\lambda)$.
      Update the ledger state $\mathtt{LS} \leftarrow \mathtt{LS}'$ and the coin counters $\#\mathtt{pCoinA}$ and $\$\mathtt{pCoinB}$
      Append $(\mathtt{tx}_{\mathsf{exchange}}, \mathtt{LS}', \mathcal{P}_{\delta^{\mathtt{old}},\delta})$ to $\mathtt{view}$, and append $\mathtt{aux}$ to $\mathtt{aux}_u$.
    - If $\mathtt{op}$ is $\mathtt{opProvide}$:
      Sample $id \leftarrow poly(\lambda), x, y \leftarrow poly(\lambda)$.
      Let $\mathcal{P} \leftarrow \mathtt{LP.provide}(id, x, y, \mathcal{P})$.
      Append $\mathcal{P}_{\delta^{\mathtt{old}},\delta}$ to $\mathtt{view}$.
    - If $\mathtt{op}$ is $\mathtt{opWithdraw}$:
      Sample $id \leftarrow poly(\lambda), a \leftarrow poly(\lambda)$.
      Let $\mathcal{P} \leftarrow \mathtt{LP.withdraw}(id, a, \mathcal{P})$.
      Append $\mathcal{P}_{\delta^{\mathtt{old}},\delta}$ to $\mathtt{view}$.
- The challenger returns $\mathrm{Adv}(\mathtt{view})$.

Fig. 7: Experiments Ideal$^{\pi_{\mathtt{DAX}}}_{\mathrm{Adv},\mathrm{Sim}}(1^\lambda)$

$\text{Real}^{\pi_{\texttt{DAX*}}}_{\text{Adv}}(1^\lambda)$:

- The challenger runs the setup: $\sigma, \tau \leftarrow \texttt{Setup}(1^\lambda)$.
- The challenger samples $N, N', D, M, x, y \leftarrow poly(\lambda)$, a set of N user identities $\{pk_i, sk_i\}_{i\in[N]}$, a set of asset IDs $\{\delta_i\}_{i\in[D]}$, and N private string $\{\texttt{aux}_i\}_{i\in[N]}$.
- Run the DAP scheme for N' steps.
- For every asset pair (δ_i, δ_j):
  - Sample a user id $id \in [N]$ for the liquidity pool initialization.
  - If $\texttt{aux}_{id}$ does not contain asset type δ_i and δ_j at the same time, stop the game.
  - Sample $\texttt{pCoinA}$, $\texttt{pCoinB}$ of type δ_i and δ_j, with value x, y from $\texttt{aux}_{id}$.
  - $(\texttt{tx}_{\textsf{PrivInit}}, \texttt{pCoinLP}) \leftarrow \texttt{LP*}.GenPrivInit(pCoinA, pCoinB), x, y, pCoinLP, \pi_{init})$.
  - Initialize the liquidity pool $\mathcal{P}_{\delta_i,\delta_j} \leftarrow \texttt{LP*}.ApplyPrivInit(\texttt{tx}_{\textsf{PrivInit}}, x, y)$.
- Initialize the adversary view $\texttt{view} \leftarrow [\sigma, \tau, N, M, \{pk_i\}, (\mathcal{P}_{\delta_i,\delta_j})_{i,j}]$, and the initial ledger state $\texttt{LS}$.
- Repeat the following for M times:
  - Sample user index $u \leftarrow [N]$.
  - Sample private liquidity provider $id \leftarrow [N]$.
  - Sample an asset id $\delta \in \{\delta_i\}_{i\in[D]}$.
  - The challenger samples an op code $\texttt{op}$ from $\{\texttt{opMint}, \texttt{opTransfer}, \texttt{opReclaim}, \texttt{opExchange}, \texttt{opProvide}, \texttt{opWithdraw}\}$.
  - If $\texttt{op}$ is $\texttt{opMint}$:
    Sample a public coin $Coin$ and a coin value v.
    Let $(\texttt{tx}_{\textsf{mint}}, \texttt{aux}) \leftarrow \texttt{GenMint}(\texttt{LS}, CoinA, pk_u, sk_u, \delta, v, 1^\lambda)$.
    Let $(\texttt{LS}', b) \leftarrow \texttt{Mint}(\texttt{LS}, \texttt{tx}_{\textsf{mint}})$
    If $b = \perp$, stop the game and return $\text{Adv}(\texttt{view}, 1^\lambda)$.
    Update the ledger state $\texttt{LS} \leftarrow \texttt{LS}'$
    Append $(\texttt{tx}_{\textsf{mint}}, \texttt{LS}')$ to $\texttt{view}$, and append $\texttt{aux}$ to $\texttt{aux}_u$.
  - If $\texttt{op}$ is $\texttt{opTransfer}$:
    Sample private coins c_1^{old} and c_2^{old} with private key sk_a, sk_b from $\texttt{aux}_u$, and target addresses pk_d, pk_e.
    Let $(\texttt{tx}_{\textsf{transfer}}, \texttt{aux}) \leftarrow \texttt{GenTransfer}(\texttt{LS}, c_1^{\text{old}}, c_2^{\text{old}}, sk_a, sk_b, pk_d, pk_e, \sigma, \texttt{aux}, 1^\lambda)$.
    Let $(\texttt{LS}', b) \leftarrow \texttt{Transfer}(\texttt{LS}, \texttt{tx}_{\textsf{transfer}})$
    If $b = \perp$, stop the game and return $\text{Adv}(\texttt{view}, 1^\lambda)$.
    Update the ledger state $\texttt{LS} \leftarrow \texttt{LS}'$
    Append $(\texttt{tx}_{\textsf{transfer}}, \texttt{LS}')$ to $\texttt{view}$, and append $\texttt{aux}$ to $\texttt{aux}_u$.
  - If $\texttt{op}$ is $\texttt{opReclaim}$:
    Sample private coins c_1^{old} and c_2^{old} with private key sk_a, sk_b from $\texttt{aux}_u$, a target public key pk_d, a public address $\texttt{PAddr}$, and a public coin value v
    Let $(\texttt{tx}_{\textsf{reclaim}}, \texttt{aux}) \leftarrow \texttt{GenReclaim}(\texttt{LS}, c_1^{\text{old}}, c_2^{\text{old}}, sk_a, sk_b, pk_d, \texttt{PAddr}, \delta, v, \sigma, \texttt{aux}, 1^\lambda)$.
    Let $(\texttt{LS}', b) \leftarrow \texttt{Reclaim}(\texttt{LS}, \texttt{tx}_{\textsf{reclaim}})$
    If $b = \perp$, stop the game and return $\text{Adv}(\texttt{view}, 1^\lambda)$.
    Update the ledger state $\texttt{LS} \leftarrow \texttt{LS}'$
    Append $(\texttt{tx}_{\textsf{transfer}}, \texttt{LS}')$ to $\texttt{view}$, and append $\texttt{aux}$ to $\texttt{aux}_u$.
  - If $\texttt{op}$ is $\texttt{opExchange}$:
    Sample a private coin $c_1^{\text{old}}, c_2^{\text{old}}$ with private key sk_a, sk_b from $\texttt{aux}_u$. Determine the coin type δ^{old}. Sample the target public key pk_d, pk_e.
    Let $(v^{\texttt{new}}, \mathcal{P}_{\delta^{\text{old}},\delta}) \leftarrow \texttt{LP*.Exchange}(c_1^{\text{old}}, c_2^{\text{old}}, \mathcal{P}_{\delta^{\text{old}},\delta})$.
    Let $(\texttt{tx}_{\textsf{exchange}}, \texttt{aux}) \leftarrow \texttt{GenExchange}(\texttt{LS}, c_1^{\text{old}}, c_2^{\text{old}}, \delta, v^{\texttt{new}}, sk_a, sk_b, pk_d, pk_e, \sigma, \texttt{aux}, 1^\lambda)$.
    Let $(\texttt{LS}', b) \leftarrow \texttt{Exchange}(\texttt{LS}, \texttt{tx}_{\textsf{exchange}})$
    If $b = \perp$, stop the game and return $\text{Adv}(\texttt{view}, 1^\lambda)$.
    Update the ledger state $\texttt{LS} \leftarrow \texttt{LS}'$ and the coin counters $\mathcal{P}_{\delta^{\text{old}},\delta}.x$ and $\mathcal{P}_{\delta^{\text{old}},\delta}.y$
    Append $(\texttt{tx}_{\textsf{exchange}}, \texttt{LS}', \mathcal{P}_{\delta^{\text{old}},\delta})$ to $\texttt{view}$, and append $\texttt{aux}$ to $\texttt{aux}_u$.
  - If $\texttt{op}$ is $\texttt{opProvide}$:
    If $\texttt{aux}_{id}$ has smaller than 2 types of coin, stop the game.
    Sample $\texttt{pCoinA}, \texttt{pCoinB}$ with value x and y, and type δ and δ' from $\texttt{aux}_{id}$.
    $\texttt{tx}_{\textsf{PrivProvide}}, \texttt{pCoinLP} \leftarrow \texttt{LP*.GenPrivProvide}(\texttt{pCoinA}, \texttt{pCoinB}, \mathcal{P})$.
    Let $\mathcal{P}_{\delta,\delta'} \leftarrow \texttt{LP.ApplyPrivProvide}(\texttt{tx}_{\textsf{PrivProvide}}, x, y, \mathcal{P})$. Update the ledger state $\texttt{LS} \leftarrow \texttt{LS}'$
    Append $(\texttt{tx}_{\textsf{exchange}}, \texttt{LS}', (\mathcal{P}_{\delta,\delta'}))$ to $\texttt{view}$, and append $\texttt{pCoinLP}$ to $\texttt{aux}_{id}$.
  - If $\texttt{op}$ is $\texttt{opWithdraw}$:
    Sample a private coin $\texttt{pCoinLP}$ with value t from $\texttt{aux}_{id}$, and determine the corresponding asset pair δ and δ'.
    $(\texttt{tx}_{\textsf{PrivWithdraw}}, \texttt{pCoinA}_1, \texttt{pCoinB}_1) \leftarrow \texttt{LP*.GenPrivWithdraw}(\texttt{pCoinLP}_1, t, \mathcal{P})$.
    Let $\mathcal{P}_{\delta,\delta'} \leftarrow \texttt{LP.ApplyPrivWithdraw}(\texttt{tx}_{\textsf{PrivWithdraw}}, t, \mathcal{P})$. Append $(\texttt{tx}_{\textsf{exchange}}, \texttt{LS}', \mathcal{P}_{\delta,\delta'})$ to $\texttt{view}$, and append $\texttt{pCoinA}, \texttt{pCoinB}$ to $\texttt{aux}_{id}$.
- The challenger returns $\text{Adv}(\texttt{view})$.

Fig. 8: Experiments $\text{Real}^{\pi_{\texttt{DAX*}}}(1^\lambda)$

$\text{Ideal}^{\pi_{\texttt{DAX*}}}_{\text{Adv},\text{Sim}}(1^{\lambda})$:

- The challenger runs the setup: $\sigma, \tau \leftarrow \texttt{Setup}(1^{\lambda})$.
- The challenger samples $N, N', D, M, x, y \leftarrow poly(\lambda)$, a set of N user identities $\{pk_i, sk_i\}_{i\in[N]}$, a set of asset IDs $\{\delta_i\}_{i\in[D]}$, and N private string $\{\texttt{aux}_i\}_{i\in[N]}$.
- Run the DAP scheme for N' steps.
- For every asset pair (δ_i, δ_j):
  - Sample a user id $id \in [N]$ for the liquidity pool initialization.
  - If $\texttt{aux}_{id}$ does not contain asset type δ_i and δ_j at the same time, stop the game.
  - Sample $\texttt{pCoinA}$, $\texttt{pCoinB}$ of type δ_i and δ_j, with value x, y from $\texttt{aux}_{id}$.
  - $(\texttt{tx}_{\mathsf{PrivInit}}, \texttt{pCoinLP}) \leftarrow \text{Sim}(\texttt{opPrivInit}, \tau, \sigma, < x, y, \delta_i, \delta_j >), 1^{\lambda})$.
  - Initialize the liquidity pool $\mathcal{P}_{\delta_i,\delta_j} \leftarrow \texttt{LP*}.ApplyPrivInit(\texttt{tx}_{\mathsf{PrivInit}}, x, y)$.
- Initialize the adversary view $\texttt{view} \leftarrow [\sigma, \tau, N, M, \{pk_i\}, (\mathcal{P}_{\delta_i,\delta_j})_{i,j}]$, and the initial ledger state $\texttt{LS}$.
- Repeat the following for M times:
  - Sample user index $u \leftarrow [N]$.
  - Sample private liquidity provider $id \leftarrow [N]$.
  - Sample an asset id $\delta \in \{\delta_i\}_{i\in[D]}$.
  - The challenger samples an op code $\texttt{op}$ from $\{\texttt{opMint}, \texttt{opTransfer}, \texttt{opReclaim}, \texttt{opExchange}, \texttt{opProvide}, \texttt{opWithdraw}\}$.
  - If $\texttt{op}$ is $\texttt{opMint}$:
    Sample a public coin $Coin$, a coin value v, and a coin type $\texttt{CoinType} \in \{A, B\}$.
    Let $(\texttt{tx}_{\mathsf{mint}}, \texttt{aux}) \leftarrow \text{Sim}(\texttt{op}, \tau, \sigma, < \texttt{LS}, CoinA, \delta, v >, 1^{\lambda}))$.
    Let $(\texttt{LS}', b) \leftarrow \texttt{Mint}(\texttt{LS}, \texttt{tx}_{\mathsf{mint}})$
    If $b = \perp$, stop the game and return $\text{Adv}(\texttt{view}, 1^{\lambda})$.
    Update the ledger state $\texttt{LS} \leftarrow \texttt{LS}'$
    Append $(\texttt{tx}_{\mathsf{mint}}, \texttt{LS}')$ to $\texttt{view}$, and append $\texttt{aux}$ to $\texttt{aux}_u$.
  - If $\texttt{op}$ is $\texttt{opTransfer}$:
    Sample private coins $c_1^{\texttt{old}}$ and $c_2^{\texttt{old}}$ with private key sk_a, sk_b from $\texttt{aux}_u$, and target addresses pk_d, pk_e.
    Let $(\texttt{tx}_{\mathsf{transfer}}, \texttt{aux}) \leftarrow \text{Sim}(\texttt{op}, \tau, \sigma, < \texttt{LS} >, \sigma, 1^{\lambda})$.
    Let $(\texttt{LS}', b) \leftarrow \texttt{Transfer}(\texttt{LS}, \texttt{tx}_{\mathsf{transfer}})$
    If $b = \perp$, stop the game and return $\text{Adv}(\texttt{view}, 1^{\lambda})$.
    Update the ledger state $\texttt{LS} \leftarrow \texttt{LS}'$
    Append $(\texttt{tx}_{\mathsf{transfer}}, \texttt{LS}')$ to $\texttt{view}$, and append $\texttt{aux}$ to $\texttt{aux}_u$.
  - If $\texttt{op}$ is $\texttt{opReclaim}$:
    Sample private coins $c_1^{\texttt{old}}$ and $c_2^{\texttt{old}}$ with private key sk_a, sk_b from $\texttt{aux}_u$, a target public key pk_d, a public address $\texttt{PAddr}$, and a public coin value v
    Let $(\texttt{tx}_{\mathsf{reclaim}}, \texttt{aux}) \leftarrow \text{Sim}(\texttt{op}, \tau, \sigma, < \texttt{LS}, \texttt{PAddr}, \delta, v >, 1^{\lambda})$.
    Let $(\texttt{LS}', b) \leftarrow \texttt{Reclaim}(\texttt{LS}, \texttt{tx}_{\mathsf{reclaim}})$
    If $b = \perp$, stop the game and return $\text{Adv}(\texttt{view}, 1^{\lambda})$.
    Update the ledger state $\texttt{LS} \leftarrow \texttt{LS}'$
    Append $(\texttt{tx}_{\mathsf{transfer}}, \texttt{LS}')$ to $\texttt{view}$, and append $\texttt{aux}$ to $\texttt{aux}_u$.
  - If $\texttt{op}$ is $\texttt{opExchange}$:
    Sample a private coin $c_1^{\texttt{old}}, c_2^{\texttt{old}}$ with private key sk_a, sk_b from $\texttt{aux}_u$. Determine the coin type $\delta^{\texttt{old}}$. Sample the target public key pk_d, pk_e.
    Let $(v^{\texttt{new}}, \mathcal{P}_{\delta^{\texttt{old}},\delta}) \leftarrow \texttt{LP*.Exchange}(c_1^{\texttt{old}}, c_2^{\texttt{old}}, \mathcal{P}_{\delta^{\texttt{old}},\delta})$.
    Let $(\texttt{tx}_{\mathsf{exchange}}, \texttt{aux}) \leftarrow \text{Sim}(\texttt{op}, \tau, \sigma, < \texttt{LS}, (\mathcal{P}_{\delta^{\texttt{old}},\delta}) >, \sigma, 1^{\lambda})$.
    Let $(\texttt{LS}', b) \leftarrow \texttt{Transfer}(\texttt{LS}, \texttt{tx}_{\mathsf{exchange}})$
    If $b = \perp$, stop the game and return $\text{Adv}(\texttt{view}, 1^{\lambda})$.
    Update the ledger state $\texttt{LS} \leftarrow \texttt{LS}'$ and the coin counters $\#\texttt{pCoinA}$ and $\$\texttt{pCoinB}$
    Append $(\texttt{tx}_{\mathsf{exchange}}, \texttt{LS}', \mathcal{P}_{\delta^{\texttt{old}},\delta})$ to $\texttt{view}$, and append $\texttt{aux}$ to $\texttt{aux}_u$.
  - If $\texttt{op}$ is $\texttt{opProvide}$:
    If $\texttt{aux}_{id}$ has smaller than 2 types of coin, stop the game. Sample $\texttt{pCoinA}, \texttt{pCoinB}$ with value x and y, and type δ and δ' from $\texttt{aux}_{id}$.
    $\texttt{tx}_{\mathsf{PrivProvide}}, \texttt{pCoinLP} \leftarrow \text{Sim}(\texttt{op}, \tau, \sigma, < x, y, \mathcal{P}_{\delta,\delta'} >, 1^{\lambda}))$.
    Let $\mathcal{P}_{\delta,\delta'} \leftarrow \texttt{LP.ApplyPrivProvide}(\texttt{tx}_{\mathsf{PrivProvide}}, x, y, \mathcal{P}_{\delta,\delta'})$. Update the ledger state $\texttt{LS} \leftarrow \texttt{LS}'$
    Append $(\texttt{tx}_{\mathsf{exchange}}, \texttt{LS}', \mathcal{P}_{\delta,\delta'})$ to $\texttt{view}$, and append $\texttt{pCoinLP}$ to $\texttt{aux}_{id}$.
  - If $\texttt{op}$ is $\texttt{opWithdraw}$:
    Sample a private coin $\texttt{pCoinLP}$ with value t from $\texttt{aux}_{id}$, and determine the corresponding asset pair δ and δ'
    $(\texttt{tx}_{\mathsf{PrivWithdraw}}, \texttt{pCoinA}_1, \texttt{pCoinB}_1) \leftarrow \text{Sim}(\texttt{op}, \tau, \sigma, < t, \mathcal{P}_{\delta,\delta'} >, 1^{\lambda})$.
    Let $\mathcal{P} \leftarrow \texttt{LP.ApplyPrivWithdraw}(\texttt{tx}_{\mathsf{PrivWithdraw}}, t, \mathcal{P}_{\delta,\delta'})$. Append $(\texttt{tx}_{\mathsf{exchange}}, \texttt{LS}', \mathcal{P}_{\delta,\delta'})$ to $\texttt{view}$, and append $\texttt{pCoinA}, \texttt{pCoinB}$ to $\texttt{aux}_{id}$.
- The challenger returns $\text{Adv}(\texttt{view})$.

Fig. 9: Experiments $\text{Ideal}^{\pi_{\texttt{DAX*}}}_{\text{Adv},\text{Sim}}(1^{\lambda})$

$$\Pr\left[\begin{array}{c} \mathtt{Exchange}(\mathtt{LS},\mathtt{tx}_{\mathsf{exchange}}) \\ \rightarrow (\mathtt{LS}',\top) \end{array}\middle| \begin{array}{c} \mathtt{Setup}(1^{\lambda})\rightarrow(\sigma,\tau),\\ \mathtt{GenMint}(\mathtt{LS}, CoinA, a_{pk}, a_{sk}, \delta, v, 1^{\lambda}) \rightarrow (\mathtt{tx}_{\mathsf{mint}}, \mathtt{aux}),\\ \mathtt{Mint}(\mathtt{LS},\mathtt{tx}_{\mathsf{mint}})\rightarrow(\mathtt{LS}',\top),\\ \mathtt{GenExchange}(\mathtt{LS}, pCoinA, \delta', v', a_{pk}, a_{sk}, a'_{pk}, \sigma, \mathtt{aux}, 1^{\lambda}) \rightarrow (\mathtt{tx}_{\mathsf{exchange}}, pCoinB)\end{array}\right] = 1 \tag{8}$$

$$\Pr\left[\begin{array}{c} \mathtt{LP*.ApplyPrivWithdraw}(\mathtt{LS},\mathtt{tx}_{\mathsf{PrivWithdraw}}) \\ \rightarrow (\mathtt{LS}',\top) \end{array}\middle| \begin{array}{c} \mathtt{Setup}(1^{\lambda})\rightarrow(\sigma,\tau),\\ \mathtt{GenMint}(\mathtt{LS}, CoinA, a_{pk}, a_{sk}, \delta, x, 1^{\lambda}) \rightarrow (\mathtt{tx}_{\mathsf{mint}}, \mathsf{pCoinA}),\\ \mathtt{Mint}(\mathtt{LS},\mathtt{tx}_{\mathsf{mint}})\rightarrow(\mathtt{LS}',\top),\\ \mathtt{GenMint}(\mathtt{LS}, CoinA, a_{pk}, a_{sk}, \delta, y, 1^{\lambda}) \rightarrow (\mathtt{tx}_{\mathsf{mint}}, \mathsf{pCoinB}),\\ \mathtt{Mint}(\mathtt{LS},\mathtt{tx}_{\mathsf{mint}})\rightarrow(\mathtt{LS}',\top),\\ \mathtt{LP*.GenPrivProvide}(\mathsf{pCoinA},\mathsf{pCoinB},\mathcal{P}) \rightarrow (\mathtt{tx}_{\mathsf{PrivProvide}}, \mathsf{pCoinLP})\\ \mathtt{LP*.ApplyPrivProvide}(\mathtt{tx}_{\mathsf{PrivProvide}}, x, y, \mathcal{P}) \rightarrow \mathcal{P}'\\ \mathtt{LP*.GenPrivWithdraw}(\mathsf{pCoinLP}, \sqrt{xy}, \mathcal{P}') \rightarrow (\mathtt{tx}_{\mathsf{PrivWithdraw}}, \mathsf{pCoinA}_1, \mathsf{pCoinB}_1)\end{array}\right] = 1 \tag{9}$$

$$\Pr\left[\begin{array}{c} \mathtt{tx}_{\mathsf{exchange}} \text{ exchanges } pCoinA \text{ to } pCoinB \\ \mathtt{tx}'_{\mathsf{exchange}} \text{ exchanges } pCoinA \text{ to } pCoinB\text{'} \\ pCoinB \neq pCoinB' \\ \mathtt{Transfer}(\mathtt{LS},\mathtt{tx}_{\mathsf{exchange}}) \rightarrow (\mathtt{LS}',\top) \\ \mathtt{Transfer}(\mathtt{LS}',\mathtt{tx}'_{\mathsf{exchange}}) \rightarrow (\mathtt{LS}'',\top) \end{array}\middle| \begin{array}{c} \mathtt{Setup}(1^{\lambda})\rightarrow(\sigma,\tau),\\ \mathtt{GenMint}(\mathtt{LS}, CoinA, a_{pk}, a_{sk}, \delta, v, 1^{\lambda}) \rightarrow (\mathtt{tx}_{\mathsf{mint}}, \mathtt{aux}),\\ \mathtt{Mint}(\mathtt{LS},\mathtt{tx}_{\mathsf{mint}})\rightarrow(\mathtt{LS}',\top),\\ \mathrm{Adv}(pCoinA, a_{pk}, a_{sk}, 1^{\lambda}) \rightarrow pCoinB, pCoinB', \mathtt{tx}_{\mathsf{exchange}}, \mathtt{tx}'_{\mathsf{exchange}}\end{array}\right] \leq \mathrm{negl}(\lambda) \tag{10}$$

$$\Pr\left[\begin{array}{c} \mathtt{tx}_{\mathsf{PrivWithdraw}} \text{ Withdraws } pCoinLP \text{ to } pCoinA \text{ and } pCoinB \\ \mathtt{tx}'_{\mathsf{PrivWithdraw}} \text{ Withdraws } pCoinLP \text{ to } pCoinA\text{' to } pCoinB\text{'} \\ b_{pk} \neq b'_{pk} \\ \mathtt{LP*.ApplyPrivWithdraw}(\mathtt{tx}_{\mathsf{PrivWithdraw}}, t, \mathcal{P}) \rightarrow (\mathcal{P}',\top) \\ \mathtt{LP*.ApplyPrivWithdraw}(\mathtt{tx}'_{\mathsf{PrivWithdraw}}, t, \mathcal{P}) \rightarrow (\mathcal{P}'',\top) \end{array}\right. \tag{11}$$

$$\left| \begin{array}{c} \mathtt{Setup}(1^{\lambda})\rightarrow(\sigma,\tau),\\ \mathtt{GenMint}(\mathtt{LS}, CoinA, a_{pk}, a_{sk}, \delta, x, 1^{\lambda}) \rightarrow (\mathtt{tx}_{\mathsf{mint}}, \mathtt{aux}),\\ \mathtt{Mint}(\mathtt{LS},\mathtt{tx}_{\mathsf{mint}})\rightarrow(\mathtt{LS}',\top),\\ \mathtt{GenMint}(\mathtt{LS}, CoinB, a_{pk}, a_{sk}, \delta', y, 1^{\lambda}) \rightarrow (\mathtt{tx}_{\mathsf{mint}}, \mathtt{aux}),\\ \mathtt{Mint}(\mathtt{LS}',\mathtt{tx}_{\mathsf{mint}})\rightarrow(\mathtt{LS}'',\top),\\ \mathtt{LP*.GenPrivProvide}(\mathsf{pCoinA},\mathsf{pCoinB},\mathcal{P}) \rightarrow (\mathtt{tx}_{\mathsf{PrivProvide}}, \mathsf{pCoinLP})\\ \mathtt{LP*.ApplyPrivProvide}(\mathtt{tx}_{\mathsf{PrivProvide}}, x, y, \mathcal{P})\\ \mathrm{Adv}(pCoinLP, a_{pk}, a_{sk}, 1^{\lambda}) \rightarrow pCoinA, pCoinB, pCoinA', pCoinB', \mathtt{tx}_{\mathsf{PrivWithdraw}}, \mathtt{tx}'_{\mathsf{PrivWithdraw}}\end{array}\right] \leq \mathrm{negl}(\lambda) \tag{12}$$

$\mathrm{Sim}(\mathtt{op}, \tau, \sigma, \mathrm{arg}, 1^\lambda)$:

- If $\mathtt{op}$ is $\mathtt{opMint}$:
  Extract $< \mathtt{LS}, CoinA, \delta, v > \leftarrow \mathrm{arg}$
  Sample k uniformly from the domain of $\mathtt{COMM}$.
  Sample s randomly, and compute $\mathtt{cm} \leftarrow \mathtt{COMM}_s(\delta \| v \| k)$.
  Return $\mathtt{tx}_{\mathsf{mint}} = (\delta, v, k, s, \mathtt{cm}), \mathtt{aux} = (\mathtt{pCoin} := (a_{\mathsf{pk}}, \delta, v, \rho, r, s, \mathtt{cm}), \mathtt{vn})$.
- If $\mathtt{op}$ is $\mathtt{opTransfer}$:
  Extract $< \mathtt{LS} > \leftarrow \mathrm{arg}$
  Sample $v_3, v_4,\ \delta_3,\ \delta_4, r_3, r_4, \rho_3, \rho_4, pk_3, pk_4$ randomly.
  Compute $k_3 = \mathtt{COMM}_r(pk_3 \| \rho_3),\ k_4 = \mathtt{COMM}_r(pk_4 \| \rho_4)$
  Sample s_3, s_4 randomly, and compute $\mathtt{cm}_3 \leftarrow \mathtt{COMM}_{s_3}(\delta_3 \| v_3 \| k_3), \mathtt{cm}_4 \leftarrow \mathtt{COMM}_{s_4}(\delta_4 \| v_4 \| k_4)$.
  Sample uniquely $\mathtt{vn}_1, \mathtt{vn}_2$.
  Compute $\pi \leftarrow \mathrm{Sim}_{zk}(\tau, \mathtt{LS}, \mathtt{vn}_1, \mathtt{vn}_2, \mathtt{cm}_3, \mathtt{cm}_4)$.
  Return $\mathtt{tx}_{\mathsf{transfer}} = (\mathtt{LS}, \mathtt{vn}_1, \mathtt{vn}_2, \mathtt{cm}_3, \mathtt{cm}_4, \pi), \mathtt{aux} = (c_3^{\mathtt{new}} := (pk_3, \delta_3, v_3, \rho_3, r_3, s_3, \mathtt{cm}_3), c_4^{\mathtt{new}} := (pk_4, \delta_4, v_4, \rho_4, r_4, s_4, \mathtt{cm}_4))$.
- If $\mathtt{op}$ is $\mathtt{opReclaim}$:
  Extract $< \mathtt{LS}, \mathtt{PAddr}, v > \leftarrow \mathrm{arg}$
  Sample $v_3, \delta_3, r_3, \rho_3, pk_3$ randomly.
  Compute $k_3 = \mathtt{COMM}_r(pk_3 \| \rho_3)$.
  Sample s_3 randomly, and compute $\mathtt{cm}_3 \leftarrow \mathtt{COMM}_{s_3}(\delta_3 \| v_3 \| k_3)$.
  Sample uniquely $\mathtt{vn}_1$ and $\mathtt{vn}_2$.
  Compute $\pi \leftarrow \mathrm{Sim}_{zk}(\tau, \mathtt{LS}, \mathtt{vn}_1, \mathtt{vn}_2, \mathtt{cm}_3, \mathtt{PAddr}, v)$.
  Return $\mathtt{tx}_{\mathsf{reclaim}} = (\mathtt{LS}, \mathtt{vn}_1, \mathtt{vn}_2, \mathtt{cm}_3, \pi, \mathtt{PAddr}, v), \mathtt{aux} = (c_3^{\mathtt{new}} := (pk_3, \delta_3, v_3, \rho_3, r_3, s_3, \mathtt{cm}_3))$.
- If $\mathtt{op}$ is $\mathtt{opExchange}$:
  Extract $< \mathtt{LS}, \mathcal{P}_{\delta^{\mathsf{old}}, \delta} > \leftarrow \mathrm{arg}$
  Get the coin numbers $\#\mathtt{pCoinDex}(\delta^{\mathrm{old}})$ and $\#\mathtt{pCoinDex}(\delta)$ from $\mathcal{P}_{\delta^{\mathsf{old}}, \delta}$.
  Sample v_1, v_2 randomly such that $\#\mathtt{pCoinDex}(\delta^{\mathrm{old}}) > v_1 + v_2$, and solve $v_3 + v_4$ from $(\#\mathtt{pCoinDex}(\delta^{\mathrm{old}}) - v_1 - v_2) \times (\mathtt{pCoinDex}(\delta) + v_3 + v_4) = T$.
  Sample $r_3, \rho_3, pk_3, r_4, \rho_4, pk_4$ randomly. Compute $k_3 = \mathtt{COMM}_r(pk_3 \| \rho_3),\ k_4 = \mathtt{COMM}_r(pk_4 \| \rho_4)$.
  Sample s_3, s_4 randomly, and compute $\mathtt{cm}_3 \leftarrow \mathtt{COMM}_{s_3}(\delta \| v_3 \| k_3), \mathtt{cm}_4 \leftarrow \mathtt{COMM}_{s_4}(\delta \| v_4 \| k_4)$.
  Sample uniquely $\mathtt{vn}_1, \mathtt{vn}_2$.
  Compute $\pi \leftarrow \mathrm{Sim}_{zk}(\tau, \mathtt{LS}, \mathtt{vn}_1, \mathtt{vn}_2, \mathtt{cm}_3, \mathtt{cm}_4, \delta^{\mathrm{old}}, \delta)$.
  Return $\mathtt{tx}_{\mathsf{exchange}} = (\mathtt{LS}, \mathtt{vn}_1, \mathtt{vn}_2, \mathtt{cm}_3, \mathtt{cm}_4, \pi, \delta^{\mathrm{old}}, v_1 + v_2, \delta, v_3 + v_4)$.
- If $\mathtt{op}$ is $\mathtt{opPrivInit}$:
  Extract $< x, y, \delta_i, \delta_j > \leftarrow \mathrm{arg}$. Sample private coins $\mathtt{pCoinA}$ and $\mathtt{pCoinB}$ with values x and y, and type δ_i and δ_j.
  Sample a private $\mathtt{pCoinLP}$ with value $\sqrt{xy}$. Compute $\pi \leftarrow \mathrm{Sim}_{zk}(\tau, \mathtt{LS}, \mathtt{vn}_A, \mathtt{vn}_B, x, y, \delta_i, \delta_j)$. Return $\mathtt{tx}_{\mathsf{PrivInit}} = (\mathtt{LS}, \mathtt{vn}_A, \mathtt{vn}_B, x, y, \pi)$.
- If $\mathtt{op}$ is $\mathtt{opProvide}$:
  Extract $< x, y, \mathcal{P}_{\delta, \delta'} > \leftarrow \mathrm{arg}$. Extract δ, δ' from $\mathcal{P}_{\delta, \delta'}$. Sample private coins $\mathtt{pCoinA}$ and $\mathtt{pCoinB}$ with values x and y, and type δ_i and δ_j.
  Sample a private $\mathtt{pCoinLP}$ with value $\sqrt{xy}$. Compute $\pi \leftarrow \mathrm{Sim}_{zk}(\tau, \mathtt{LS}, \mathtt{vn}_A, \mathtt{vn}_B, x, y, \mathtt{cm}_{LP}, \delta_i, \delta_j, \mathcal{P}_{\delta, \delta'})$.
  Return $\mathtt{tx}_{\mathsf{PrivProvide}} = (\mathtt{LS}, \mathtt{vn}_A, \mathtt{vn}_B, x, y, \mathcal{P}_{\delta, \delta'}, \pi)$.
- If $\mathtt{op}$ is $\mathtt{opWithdraw}$:
  Extract $< t, \mathcal{P}_{\delta, \delta'} > \leftarrow arg$. Extract δ, δ' from $\mathcal{P}_{\delta, \delta'}$. Sample a private $\mathtt{pCoinLP}$ with value t. Sample private coins $\mathtt{pCoinA}$ and $\mathtt{pCoinB}$ with values x and y s.t. $xy = t^2$. Compute $\pi \leftarrow \mathrm{Sim}_{zk}(\tau, \mathtt{LS}, \mathtt{vn}_{LP}, \mathtt{cm}_A, \mathtt{cm}_B, t, x, y, \delta_i, \delta_j, \mathcal{P}_{\delta, \delta'})$. Return $\mathtt{tx}_{\mathsf{PrivWithdraw}} = (\mathtt{LS}, \mathtt{vn}_{LP}, \mathtt{cm}_A, \mathtt{cm}_B, t, x, y, \mathcal{P}_{\delta, \delta'}, \pi)$.

Fig. 10: Simulator Sim for the decentralized anonymous exchange scheme.