FPSwap: Fair and Privacy Friendly Swap of Digital Assets on Blockchain

Abstract

Bitcoin introduces a new type of digital currency that does not rely on a central system to process and maintain transactions. Benefiting from the concept of a decentralized ledger, users who do not know or trust each other can still conduct transactions in a peer-to-peer manner. Recently, cryptocurrencies have been introduced to enhance the privacy provided by Bitcoin. For example, CryptoNote, which implements the "Traceable Ring Signature" approach to achieve Untraceability, and Zcash seek to hide the content of a crypto note to enhance privacy while allowing the public to verify the validity of a payment transaction. However, all these schemes consider one-way transaction, and there are few mechanisms to support the two-way exchange of privacy-preserving notes or digital assets on the chain and simultaneously protect the privacy of the exchange operation. Existing approaches for fair exchange of assets with privacy mostly rely on off-chain or sidechain exchanges, escrow, or centralized services. Instead, in this paper, we propose a solution that supports the oblivious and private, fully on-chain fair exchange of crypto notes or confidential digital assets. We demonstrate this technology by extending zero-knowledge-based crypto notes. To address privacy and multi-asset support, we built a new zero-knowledge proof system and extended the crypto note format with a new property to represent various types of digital or "tokenized" assets. We extend the payment protocol to implement fair exchange operations through the orchestration of privacy-enhanced asset transfer and payment transactions. Based on possible scenarios during the exchange operation, we add new constraints and conditions to the zero-knowledge proof system used for public verification of transaction validity. We present experimental results from an implementation of our solution using the xJsnark tool.

Keywords: Blockchain, Cryptocurrencies, Private Exchange, Digital Asset, CryptoNote, Zero Knowledge

1. Introduction

Cryptocurrency systems based on decentralized ledgers require each participant to keep a local copy of the transaction history to prevent double-spending without relying on a centralized party. This design however exposes users to privacy concerns, as all transaction information becomes publicly available [1, 2].

To address this concern, several schemes (e.g., CryptoNote [3], Zcash [4], Monero [5], Zether [6]) proposed to enhance the privacy at the individual payment transaction level by avoiding the disclosure of payment-related information, e.g., the recipient wallet address and the amount of cryptocurrency transferred. However, privacy-preserving or oblivious exchange of digital assets, which is essential for a thriving ecosystem, have been so far largely ignored. By exchange, we mean giving a certain digital asset and receiving another asset in return. The exchange can be of identical or different asset types, e.g., Alice exchanges a note of X number of A coins for a note of Y number of B coins with Bob.

In this paper, we develop a unified framework to support both privacy-enhanced payment transactions and fair exchange of digital assets without using centralized mixing services, escrow-based or off-chain/side-chain approaches [7]. More specifically, we propose a ledger-based multi-asset system designed to support both the private exchange (swap) of assets and regular confidential asset transfer/spending operations.

Our solution aims to satisfy several requirements. First, the exchange/swap operation should be fair, i.e., at the end of the protocol, either both or none of the transfers are executed. Second, the system should support an exchange between different types of assets, which we call *private notes*. Third, the exchange needs to be oblivious and private, i.e., information posted on the blockchain should not leak the identities of the parties involved in an exchange, or the nature of the exchange (e.g., types or amount of assets, the status of the exchange).

To achieve this we introduce the concepts of *sibling note* and *negative value* to allow the linkage of discrete steps in exchange to guarantee fairness. We further leverage zero-knowledge proofs to hide the information of exchange transactions while allowing the public to verify the validity. We develop a single unified format for both asset transfer/payment transactions and asset exchange. As a result, an adversary cannot learn any information of an ongoing or completed asset exchange operation based on the data recorded on the ledger. The adversary cannot even tell if there is an exchange that has occurred

In summary, we provide the following contributions:

- We develop a unified and formal framework of privacy-preserving exchanges and payment transactions of confidential assets on a decentralized ledger that satisfies security and privacy requirements. The framework supports both one-way payment transactions and bi-directional exchanges under a single confidential asset transaction protocol, where the two operations are indistinguishable.

- We propose a concrete construction of privacy-preserving confidential asset exchange that supports multiple types

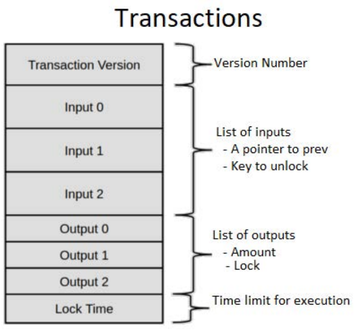

Figure 1: UTXO transactions.

of digital assets and meets the formal definitions by extending the Zcash design, protocol, and data models.

- We implement our solution using xJsnark's "program to circuits" tool and conduct experiments to show performance and complexity.

2. Background

In this section, we provide the necessary background for the rest of the paper.

Transactions are the fundamental building blocks in distributed ledgers or blockchains designed to support cryptocurrencies. Transactions represent state changes in a distributed ledger.

UTXO model: The UTXO model is a common design adopted by many cryptocurrencies, for example Bitcoin. In the UTXO model, cryptocurrencies do not use accounts or balances. Instead, individual coins (UTXOs also called UTXO notes) are transferred between users in a way similar to physical coins or cash. Each node in the system keeps track of the status of UTXOs, which includes detection of double spending.

When a transaction happens, it comprises of transaction version, which tells a node the set of rules to be used to verify it, followed by lock-time which specifies whether a transaction can be included in the blockchain right away or after some delay. Before a lock-time takes place, transaction output and inputs happen. A transaction output is locked, and input provides a key to unlock them. They are data records that contain an amount indicating the quantity of value to be transferred, a locking script, and other metadata. Locking scripts within a transaction output encode the conditions that must be satisfied in order for the amount to be spent. The transaction inputs contain a transaction hash, output index, unlock script, and other metadata. The transaction hash is a hash of the transaction containing the transaction output from where the value is to be drawn. Every transaction contains a pointer and an unlocking key. The pointer points to a previous transaction output which the key is used to unlock. The unlocking script contains the necessary information to the locking script of the referenced UTXO, which can be validated by other nodes in the blockchain network. Every time an unlock occurs, it is marked on the blockchain as "spent".

Every node in a blockchain network keeps track of the available UTXOs in a data structure called the UTXO pool. When a UTXO is matched with a transaction input, it is considered as spent and, therefore, is removed from the UTXO pool. This means that a UTXO can be only spent once. An example of the transaction can be seen in Fig 1

Oracle: Blockchains and smart contracts cannot access off-chain data. However, for many contractual agreements, it is vital to have relevant information outside of the chain to execute the agreement. This creates the need for data feeds designed to bring external data into a distributed ledger system. These data feeds are known as oracles. Oracles are vital within the blockchain ecosystem because they expand the scope in which a transaction system or smart contracts can operate. Without blockchain oracles, the systems would have very limited use as they would only have access to data from within their networks.

Oracles can act as trusted data sources for on-chain transactions. They can be applied to both smart contract and UTXO based systems. There are diverse design patterns for oracles that differ in terms of trust model, decentralization, interaction process, efficiency, cost, and etc (e.g., [8]). The centralized oracle model relies on a single trusted intermediary as a bridge for an external data source. This approach is very efficient and cost-effective. However, centralized oracles bring the concern about the trustworthiness and validity of the data that depend on a single trusted source. To solve such concerns, a decentralized oracle is developed. A decentralized oracle network is a group of independent blockchain oracles that provide data to a blockchain. Every independent node or oracle in the decentralized oracle network independently retrieves data from an off-chain source and brings it on-chain. These are then aggregated so that the system can arrive at a deterministic value of truth for that data point based on consensus [9]. The decentralized oracle model addresses the trust issue of centralized oracles at the cost of efficiency (e.g., higher latency and cost). While decentralized oracles aim to achieve trustlessness, it is important to note that just like trustless blockchain networks, decentralized oracles do not eliminate trust, but rather distribute it between many participants.

In terms of the interaction process, oracle services can apply different design patterns such as the request-response model (more suited for smart contracts), publish-subscribe model, or simple bulletin board model. In the request-response model, a smart contract with external data needs can send a request to the data provider's data feed smart contract. The request will be routed to the off-chain infrastructure that monitors data requests, retrieves the requested external data, and returns them to the on-chain smart contract. The publish-subscribe model relies on data broadcast service, which is ideally suited for real-time data feeds or data with periodical updates. Subscribers can enroll in the oracle smart contracts and listen for updates. In the bulletin board model, oracle providers can simply upload external event data or hashes of event data to the distributed ledger, for instance using transactions. The data can be queried or checked by transactions or smart contracts after it is recorded on-chain.

This paper does not try to fix any design limitation of an oracle but rather assumes the existence of one and aims to work with any oracle.

Zero-knowledge proof based UTXO model: In the default UTXO model, all the transactions are in public, which brings privacy concerns. To resolve such concern, zero-knowledge based transaction protocols such as Zcash/Zcoin are developed. A lot of the work that went to implement a private blockchain relies on zero-knowledge proofs, specially zk-SNARK [10] , which enables zero-knowledge proof in zcash. A zero-knowledge proof of knowledge is a protocol in which a prover can convince a verifier that some statement holds without revealing any information. Generally speaking, a zero-knowledge proof involves two parties, the *prover* and the *verifier*. For a statement, the *prover* can generate a proof to convince the *verifier* of the correctness of the statement. In this process, the *verifier* cannot learn anything except the fact that the statement is true (zero-knowledge feature). Zero-knowledge proofs can be either interactive or non-interactive. Interactive zero-knowledge proofs [11, 12] require the *prover* to communicate with the *verifier* over multiple rounds to finish the proof. Non-interactive zero-knowledge proofs (NIZK) [13, 14, 15] do not require multiple rounds of interactions between *prover* and *verifier* and are more suitable for scenarios where it is difficult for these parties to be online at the same time.

Leveraging practical non-interactive zero-knowledge-based tools, Zcash implements a decentralized UTXO based payment system with strong protection of transaction integrity and anonymity using collision-resistant hash (CRH) functions, commitment schemes, and secure signature schemes as building blocks.

When a UTXO note is spent, the spender (or payer) needs to create a zero-knowledge proof that there is a corresponding note commitment (commit) in the pool of commitments without disclosing which one it is. The proof also shows that the transaction is consistent and that the user knows all the secret keys, without revealing any additional information. In Zcash, such protected transactions are called shielded transactions.

3. Overview

In this section, we provide an overview of the proposed private exchange mechanism on a decentralized ledger.

3.1. Problem Statement

At a high level, the system is aimed at meeting the following requirements.

- **Supporting multiple digital assets:** The system should support multiple types of digital assets on the same decentralized ledger.
- **Privacy protected payment/asset transfer:** Similar to shielded payment transactions in Zcash/Zerocoin, the system should prevent unrelated parties to learn the content of transactions.
- **Publicly verifiable:** The public should be able to verify the validity of all transactions including both transfer and exchange of confidential assets supported by the system.
- **Atomic asset exchange:** An asset exchange operation between two users should be atomic. Although the exchange operation may involve multiple payment or asset transfer transactions. In the end, either the exchange succeeds with an ownership swap of each participant′s digital asset, or the operation terminates so that each participant still keeps his/her own asset before the exchange.
- **Private exchange or swap of confidential assets:** Private exchange means that asset swap transactions recorded on the ledger should appear to be indistinguishable from one another and reveal no information or knowledge about the exchange operation, including the presence of an asset exchange operation, users/accounts involved in the exchange, types of the assets and the corresponding amounts in the exchange. This means that exchange transactions should be hidden among the shielded payment transactions. Based on the history of transactions, an adversary should not be able to separate transactions involved in asset exchanges from other shielded asset transfer/payment transactions.
- **Unified transaction system for both payment/asset transfer and exchange operation:** The system should not use separate protocols or algorithms that are unique for payment transactions or exchange/swap operations. All transactions should be verified using the same mechanism and an identical procedure.

We assume that there are more than one type of confidential digital assets in the system; and owners of two different types of assets, Alice and Bob, who do not fully trust each other, decide to exchange the ownership of digital assets (e.g., privacy coins, privacy tokens) on a decentralized ledger.

If privacy is not a concern, we can easily implement a fair exchange operation using a smart contract, and the underlying distributed ledger can guarantee the atomic property of the operation, i.e., either the operation succeeds that Alice and Bob get the other's digital asset, or the operation fails that each one still keeps his/her own assets. However, such an approach does not provide any privacy protection regarding traceability and prevents transactions from being linked. Every node in the system can observe the exchange of information. Some key research challenges to be solved are: how to ensure both atomicity and full privacy for multi-step exchange operations that are composed of one-way asset transfer transactions (e.g., shielded transaction from payer to payee).

It is worth mentioning that all transactions involved in an exchange operation should be publicly verifiable (verified on the chain by nodes of the ledger network using consensus) and all the transactions should be oblivious. In other words, all transactions, regardless of their nature (e.g., payment transactions, different steps of exchange operations) should be indistinguishable to the public who verify them. These requirements exclude alternatives such as setting an off-chain payment channel for exchange, which includes techniques that rely on off-chain or side-chain payments for improving privacy. The work here aims at supporting the on-chain private exchange of digital assets where all the transactions, regardless of the nature as

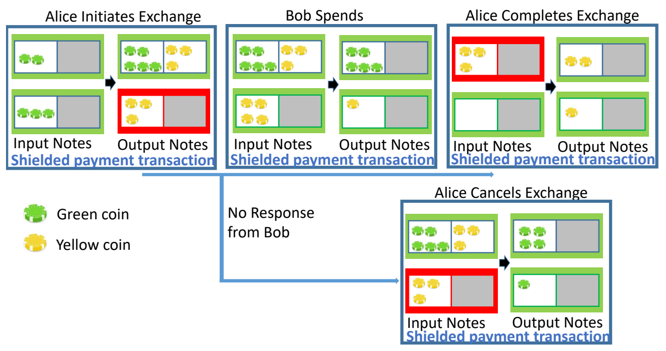

Figure 2: Example transaction sequence between Alice and Bob to complete exchange of assets.

simple asset transfer/payment or multi-step exchange, are indistinguishable.

To simplify discussions and experiments, we use zero knowledge proof-based privacy coin (e.g., Zerocash a.k.a Zcash, Zerocoin) as the baseline design. We extend the privacy coin data models and proving systems to support multiple assets, and oblivious fair exchange. Exchange is realized through a predefined sequence of privacy-protected (shielded) asset transfer-/payment transactions.

3.2. *Zero Knowledge Proof based Digital Assets*

A zero-knowledge proof of knowledge is a protocol in which a prover can convince a verifier that some statement holds without revealing any information. Roughly speaking, a zero- knowledge proof involves two parties, the *prover* and the *verifier*. For a statement, the *prover* can generate a proof to convince the *verifier* of the correctness of the statement. In this process, the *verifier* cannot learn anything except the fact that the statement is true (zero-knowledge feature). Zero-knowledge proofs can be either interactive or non-interactive. Interactive zero-knowledge proofs [11, 12] require the *prover* to communicate with the *verifier* over multiple rounds to finish the proof. Non-interactive zero-knowledge proofs (NIZK) [13, 14, 15] do not require multiple rounds of interactions between *prover* and *verifier* and are more suitable for scenarios where it is difficult for these parties to be online at the same time.

An important tool related to NIZK is zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK) [16, 17, 18]. We refer the readers to [17] for a formal definition of zk-SNARK. Leveraging practical ZK-SNARK, Zcash implements a decentralized payment system with strong protection of transaction integrity and anonymity built from collision-resistant hash (CRH) functions, commitment schemes, and pseudo-random functions. When a crypto-note (UTXO) is spent, the spender (or payer) needs to create a zero-knowledge proof that there is a corresponding note commitment in the pool of commitments without disclosing which one it is. The proof also shows that the transaction is consistent and that the user knows all the secret keys, without revealing any additional information.

3.3. *An Example of Exchange Operation*

The proposed oblivious and fair exchange scheme extends the Zcash data models, for instance, crypto-note definition, referred as a note for simplification, and the payment protocol. This subsection describes the high-level design concept with a concrete exchange example.

To support multiple confidential assets, the note format is expanded to include a new integer attribute that specifies the types of digital assets (e.g., digital currencies, tokens). Here we use colors to represent different types of assets (e.g., red coin, green coin, yellow coin). In addition, there are two types of notes, called primary note and sibling note. In Figure 2, note with a green surrounding box is a primary note; and note with a red surrounding box is a sibling note. A sibling note is uniquely associated with a primary note. Each payment transaction includes input notes and output notes where input notes are consumed and output notes are created. A primary note can be consumed by itself in a payment transaction while the associated sibling note can only be consumed either after the primary note has been consumed already in the past or together with the primary note in a transaction.

Furthermore, each note contains a debt part (the type of asset and amount) which requires that when the note is consumed in a transaction, the debt part has to be canceled out by an additional note with sufficient value that matches with the debt (both asset type and amount). For instance, if a note contains a debt of five red coins, when it is consumed (spent), the user has to mix it with a note containing at least five red coins as the inputs.

Assume that Alice wants to exchange five green coins with Bob's three yellow coins. The sequence of actions may be the following:

- Alice initiates an exchange operation by converting her assets into a pair of notes. In the first transaction, she provides two input notes (one with three green coins and one with two green coins). The transaction outputs two notes, one primary note with five green coins and three yellow coins as debt; and a sibling note with three yellow coins. Nodes in the ledger network verify the transactions and store commitments of the asset notes on the ledger.
- The primary note is encrypted and shared between Alice and Bob. Either Alice or Bob can spend the primary note.
- If Bob wants to complete the exchange, he issues a new transaction using the received primary note from Alice and one of his notes as the inputs. In this case, his input note contains four yellow coins. The transaction produces two output notes, one with five green coins and the second one with one yellow coin (after canceling the three yellow coin debt embedded in the note received from Alice).
- After Bob spends the note sent by Alice, Alice can spend the associated sibling note that contains three yellow coins, which completes the exchange.
- In case that Bob does not spend the note received from Alice within a bounded time limit, Alice can cancel the exchange by creating a new transaction that spends both the primary note and the associated sibling note, which

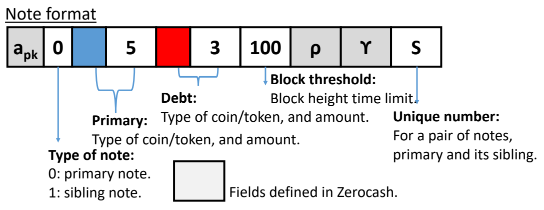

Figure 3: Private note structure.

cancels out the three yellow coin debt. The output notes will contain the five green coins that she puts in the primary note shared with Bob. As a result, the exchange is terminated. Bob can no longer spend the primary note received from Alice.

Details of the protocol design and verification algorithms, as well as security analyses, are provided in the following sections of the paper.

4. Definition of Oblivious Multi-Asset Protocol (OMAP)

The notations of OMAP are based on extending the notations of Zcash [19, 20]. In this paper, for simplification, we apply zk-SNARK and original Zcash design [19, 20] to achieve exemplary implementation of OMAP. However, it is worthwhile pointing out that it is possible to realize OMAP using alternative back-end zero-knowledge proving systems, for instance, zk-STARK [21], bulletproof [22], or approaches such as [23]. Discussion applying these different back-end zero-knowledge proving systems is orthogonal to the scope of this work, which focuses primarily on the protocol design of privacy-preserving multi-asset transaction and exchange systems.

Multi confidential asset ledger: There is a multi-asset ledger, L_{OMAP}, which records a sequence of transactions in append-only mode. The ledger supports multiple types of tokens or assets using extended note format, described later in this section. The ledger comprises ordered blocks created from a genesis block under a consensus mechanism. Each block has a block height. Further, we assume that blocks are generated with a relatively constant speed. Details of consensus mechanism and block generation can be either based on or follow Zcash design. Although our experiments are based on Zcash implementation, OMAP and L_{OMAP} can be adapted to any distributed ledger-based systems such as one based on Proof-of-Stake (PoS) or different flavors of BFTs.

Public parameters: In the case of the experiments in this paper, the system uses the same set of public parameters pp in Zcash design. They are generated either by a trusted party at the beginning or through a Multi-Party Computation ceremony [24, 25]. If OMAP is implemented over a proving system that doesn't require a trusted setup, the public parameters can be based on common public strings appropriate for the proving system.

Payment address and note format: In this work, we use the same design of payment address pair (a_{sk}, a_{pk}) where a_{sk} is used as spending key. For receiving shielded payment, a user needs to scan ledger L_{OMAP} using a_{sk}. The algorithm is similar to the one described in Zcash blockchain scanning. For each type of asset or token, there is a v_{pub}. In this experimental implementation, the value is stored in levelDB.

Table 1: Note attributes.

$pos_{1..N^{old}}$	Merkle tree positions of old notes, N=1 or 2
$path_{1..N^{old}}$	Merkle tree paths of old notes, N=1 or 2
a_{pk}	spending key
s	type of note (primary or sibling note)
$color_1$	color of primary coin (type of asset)
v_1	value of primary coin
$color_2$	color of debt coin (type of asset)
v_2	debt coin amount
bt	block height threshold
ρ	used to compute nullifier (disclosed to the public after spending)
γ	trapdoor
S	unique value for a pair of matching notes
cm	note commitment (comm)

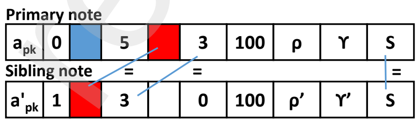

Figure 4: A pair of primary and sibling notes.

An OMAP note, n, is similar to Zcash note with extensions to support multiple confidential assets and oblivious exchange.

Figure 3 shows an example note. Attributes including a_{pk}, ρ, γ, and cm are the same as defined by the Zcash design. For each note, attribute s specifies the type of note (primary or sibling). For simplifying the discussion, we refer to the type of asset as color, for instance, black coin, red coin, which means that black coin and red coin represent different types of assets. The type and amount of primary asset are represented as: $color_1$ and v_1. The size of $color$ field determines the maximum number of assets supported by OMAP (asset type zero represents a null asset, which is used by dummy notes). For each note, there is a second pair of asset type and the amount that represent debt embedded in a note, $(color_2, v_2)$. When both $color_2$ and v_2 are zero, it means that the note is a regular note. When both $color_2$ and v_2 are not zero, it means that when the note is spent, it has to be paired with a note with its primary asset type matching with $color_2$ and the amount greater than or equal with v_2. The second pair $(color_2, v_2)$ is introduced for the purpose to support the atomic exchange of different types of assets between two OMAP users.

Attribute bt is a block height threshold, which is used for setting a time limit of asset exchange or swap. It is a threshold value that can be configured to decide when the counterparty in exchange has to spend a received note in order to move forward to the next step. When a user initiates an exchange, the user will create a note pair, see example in Figure 4. One note has s set as zero (primary note) and the second one with s set to 1 (sibling note). Attribute S is a unique value for each pair of matching notes, which means that a primary note and its sibling note share the same value S. Calculation of S is based on a CRH

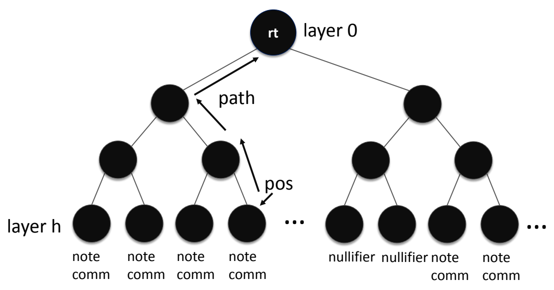

Figure 5: Combined Merkle tree for storing note commitments and nullifiers. Zero-knowledge proof is used to show that for any leaf node, there is a path to the root.

with a unique random input computed from the input notes (specific to a transaction). CRH stands for collision-resistant hash algorithm. In addition, for each pair of notes (primary and its sibling), the value of bt needs to be the same; $color_{1,2} = color_{2,1} \wedge v_{1,2} = v_{2,1}$; and $v_{2,2} = 0$.

When a note is spent, a nullifier value, nf, will be created using attribute ρ as input where nf is determined by $\mathtt{PRF}^{nf}_{a_{sk}}(\rho)$ where PRF stands for pseudo-random function. OMAP enforces that nullifiers must be unique in order to prevent double-spending. In addition, the uniqueness of nullifiers is used to support the atomic exchange of assets (see Section 5).

Note commitment and nullifier Tree: OMAP uses an incremental Merkle tree of fixed depth for note commitments and nullifiers. Different from Zcash where nullifiers are kept only for preventing double-spending, OMAP maintains a joint Merkle tree, M, that includes both note commitments and nullifiers. Alternatively, there could be a separate note commitment tree and nullifier tree so that there will be two Merkle tree roots, one for the note commitment tree and the second one for the nullifier tree. In this work, we assume that the two trees are combined. Each commit or nullifier has a (path, pos) where pos is the position in the tree. Leaf node $M_i^{MerkleDepth}$ is in the tree with given root rt = M_0^0, where M $\in (cm, nf)$.

Note that in this work, we restrict to cases of two input notes and two output notes. Spending cases of more than two input notes or two output notes can be reduced to transactions with two notes as input and output, subject of future extension. Similar to the Zcash design, input note or output note can be dummy note (without associated note commitment). In the case of dummy note, the asset type is zero.

Remark: Refer to [20] for a complete description of notations in the zero-knowledge proving system used by Zcash. The description is based on extending the Zcash note format and operations. Here we focus on algorithms and definitions that are extended or modified to support oblivious and fair exchange through $tr_{JoinSplit}$ transactions. Definitions that are not changed can be found in the original Zcash specification.

5. OMAP Protocol

5.1. Preliminary

The protocol is built by extending the design of zero-knowledge based privacy payment system such as Zcash. These privacy oriented blockchain designs apply zero-knowledge proving system as the underlying building block to protect privacy. A zero-knowledge proving system is a cryptography protocol that allows proving a particular claim/statement, dependent on two input datasets, public and witness, without disclosing information about the witness input other than that included in the claim/statement.

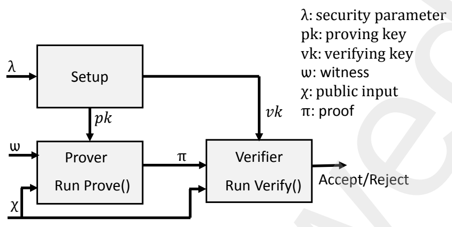

Figure 6: Diagram of zero-knowledge proving system.

A zero-knowledge proving system needs to satisfy the following security requirements: completeness, succinctness, and proof-of-knowledge.

Completeness: for security parameter λ, a $\mathbb{F}$, arithmetic circuit C, and any $(\chi,\ \omega) \in \mathcal{R}$ (satisfying inputs), honest prover can convince the verifier with probability $1 - negl(\lambda)$.

Succinctness: honestly generated proof π has $O_n(1)$ bits and Verify(vk, χ, π) runs in the $O_\lambda(|\chi|)$.

Proof-of-knowledge: If the verifier accepts a proof output by a bounded prover, then the prover knows witness for the given statement.

5.2. Transaction Algorithms

For completeness, this subsection lists the main algorithms.

In a traditional Zcash implementation transactions can contain transparent input, output, and scripts. They also contain a sequence of zero or more JoinSplits. Each of which takes in a transparent value and up to two input notes, and produces a transparent value and up to two output notes. The nullifiers of the input notes and the commitments of the output notes are revealed thus allowing them to be spent in future. Each JoinSplit also includes a computationally sound zero-knowledge proof-of-knowledge (SNARK) which proves:

1. The inputs and outputs balance (individually for each JoinSplit).
2. For each input note of non-zero value, some revealed commitment exists for that note.
3. The prover knew the private keys of the input notes.
4. The nullifiers and commitments are computed correctly.
5. The private keys of the input notes are cryptographically linked to a signature over the whole transaction, in such a way that the transaction cannot be modified by a party who did not know these private keys.
6. Each output note is generated in such a way that its nullifier will not collide with the nullifier of any other note.

Outside the SNARK, it is also checked that the nullifiers for the input notes had not already been revealed (i.e. they had not already been spent).

The main extensions here are related to the JoinSplit transaction. In addition to one-way payment (shielded to the transaction of Zcash UTXO note), JoinSplit is expanded to support additional spending/transaction cases for achieving atomic and oblivious exchange of different types of assets (notes). As described earlier, the design is extended to support multiple types of assets such as a variety of digital coins or tokens.

Setup$_{OMAP}$: Follows the same Zcash algorithm for constructing public parameters. $Setup_{OMAP}$ takes 1^λ as security parameter and outputs public parameters pp_{OMAP} that includes: $pk_{JoinSplit}$, $vk_{JoinSplit}$, pp_{enc}, pp_{sig}. $pk_{JoinSplit}$ and $vk_{JoinSplit}$ are a pair of proving and verifying keys for JoinSplit transactions. pp_{enc} is encryption key and pp_{sig} is signature key.

Algorithm 1: Public parameter generation algorithm.

```
Input  : Security parameter $\lambda$
Output: Public parameters $pp_{OMAP}$
1 Compute $C_{JoinSplit}$ at security parameter $\lambda$
2 Compute $(pk_{JoinSplit}, vk_{JoinSplit}) = KeyGen(1^\lambda, C_{JoinSplit})$
3 Create $pp_{enc} = \mathcal{G}_{enc}(1^\lambda)$
4 Create $pp_{sig} = \mathcal{G}_{sig}(1^\lambda)$
5 Output $pp_{OMAP} = (pk_{JoinSplit}, vk_{JoinSplit}, pp_{enc}, pp_{sig})$
```

CreateAddr$_{OMAP}$: It takes public parameters pp_{OMAP} as input and creates a pair of transmission key (a_{pk}, pk_{enc}) and receiving key (a_{pk}, sk_{enc}) where a note sent to a recipient is encrypted using pk_{enc} and it is retrieved by the recipient from the ledger using sk_{enc}.

In case of initializing the first JoinSplit transaction for an exchange operation, sender and recipient share the same spending key a_{sk} for spending the primary note. In this case, a_{sk} can be created from a shared secret between the sender who initiates the exchange and recipient, for instance
$a_{sk} = \mathtt{PRF}^{shared}_{a_{sk}}(sharedsecret \| h_{sig})$ where
$h_{sig} = \mathtt{CRH}(nf^{old}_1, nf^{old}_2, pk_{sig})$ and PRF can be SHA256.

Mint$_{OMAP}$: It takes public parameters pp_{OMAP}, public address pair (a_{pk}, pk_{enc}), π, asset $color$, value v where $color \in (1, ..., color_{max})$ and $v \in (0, ..., v_{max})$. It appends note n to the ledger L_{OMAP}, or output $\perp$ (reject). This algorithm is used to mint notes of different asset types based on the public values (each asset type and amount) and put the notes on L_{OMAP}.

JoinSplit$_{OMAP}$: JoinSplit$_{OMAP}$ takes a set of input values and creates two output notes, n^{new}_1 and n^{new}_2, and a payment/asset transfer transaction $tr_{JoinSplit}$. Algorithm 2 lists pseudo-code adapted from Zcash specifications. To support asset exchange, there are different JoinSplit transaction cases in contract with Zcash where there is only one type of JoonSplit transaction. Orchestration of different JoinSplit transactions can achieve atomic exchange of confidential assets between users. Details of the proving circuit unique to each JoinSplit transaction case are described in Section **??**. Here we focus on operations and procedures common to all the transactions. Note that we use asset

Algorithm 2: JoinSplit algorithm.

```
Input  : Public parameters $pp_{OMAP}$, Merkle tree root
         $rt$, input note $n^{old}_1$ and $n^{old}_2$, input note
         spending keys $a^{old}_{sk,1}$ and $a^{old}_{sk,2}$, output addresses
         $a^{new}_{pk,1}$ and $a^{new}_{pk,2}$, public value and asset type
         (color) - $color^{old}_{pub}$ and $v^{old}_{pub}$, ledger $L_{OMAP}$
Output: Two output notes $n^{new}_1$ and $n^{new}_2$; a transaction
         $tr_{JoinSplit}$
1  For each input note, compute nullifier $nf^{old}_i =$
   $PRF^{nf_{old}}_{a^{old}_{sk,i}}(\rho^{old}_i) = SHA256(a^{old}_{sk,i} \| \rho^{old}_i)$
2  Create signature key pair $(pk_{sig}, sk_{sig}) = \kappa_{sig}(pp_{sig})$
   where $\kappa_{sig}$ generates a key pair
3  Compute $h_{sig} = CRH(nf^{old}_1, nf^{old}_2, pk_{sig})$
4  Set $\rho^{new}_i = PRF^{\rho}_{\varphi}(i, h_{sig}) = SHA256(i \| \varphi \| h_{sig})$
5  For each new note, sample a random $\gamma^{new}_i$
6  Set new note as $n^{new}_i = (a^{new}_{pk,i}, s^{new}_i, color^{new}_{i,1}, v^{new}_{i,1},$
   $color^{new}_{i,2}, v^{new}_{i,2}, bt^{new}_i, \rho^{new}_i, \gamma^{new}_i, h_{sig})$
7  Compute note commitment $cm^{new}_i = \mathtt{NoteCommit}(n^{new}_i)$
8  Compute old note spending signature $h_i = PRF^{pk}_{a^{old}_{sk,i}}(i \| h_{sig})$
9  Encrypt new note as $N^{enc,new}_i = \mathcal{E}_{enc}(pk^{new}_{enc,i}, n^{new}_i)$
10 Set public input $\chi = (rt, nf^{old}_1, nf^{old}_2, cm^{new}_1, cm^{new}_2,$
   $v^{old}_{pub,color}, v^{new}_{pub,color}, block_n, h_{sig}, h_1, h_2)$
11 Set witness $\omega = (path_1, pos_1, path_2, pos_2, n^{old}_1, n^{old}_2,$
   $a^{old}_{sk,1}, a^{old}_{sk,2}, \varphi, dummy^{old}_1, dummy^{old}_2, n^{new}_1, n^{new}_2,$
   $n^{old}_{pair\_of\_1..N}, path_{pair\_of\_1..N}, pos_{pair\_of\_1..N}, a^{old}_{sk,pair\_of\_1..N},$
   $nf^{old}_{pair\_of\_1..N}, path_{nf}, pos_{nf})$
12 Compute proof $\pi_{JoinSplit} = Prove(pk_{JoinSplit}, \chi, \omega)$
13 Set transaction message $m = (\chi, \pi_{JoinSplit}, N^{enc,new}_1,$
   $N^{enc,new}_2)$
14 Set message signature $\delta = S_{sig}(sk_{sig}, m)$
15 Set $tx_{JoinSplit} = (rt, nf^{old}_1, nf^{old}_2, cm^{new}_1, cm^{new}_2, color^{new}_{pub},$
   $v^{new}_{pub}, *)$ where
16 $* = (pk_{sig}, h_1, h_2, \pi_{JoinSplit}, N^{enc,new}_1, N^{enc,new}_2, \delta)$
```

Table 2: Data fields defined in ω.

$pos_{1..N^{old}}$	Merkle tree positions of old notes, N=1 or 2
$path_{1..N^{old}}$	Merkle tree paths of old notes, N=1 or 2
$n^{old}_{1..N^{old}}$	old notes, N=1 or 2
$a^{old}_{sk,1..N^{old}}$	old note spending key, N=1 or 2
φ	random seed
$dummy^{old}_{1..N^{old}}$	old note is dummy note or not
$n^{new}_{1..N^{new}}$	new notes, N=1 or 2
$n^{old}_{pair\_of\_1..N}$	associated note with one of the old notes (primary and sibling pair)
$pos_{pair\_of\_1..N}$	Merkle tree position of associated note with one of the old notes (primary and sibling pair)
$path_{pair\_of\_1..N}$	Merkle tree path of the associated note above
$a^{old}_{sk,pair\_of\_1..N}$	spending key of the associated note (primary and sibling pair)
$nf^{old}_{pair\_of\_1..N}$	associated note nullifier (primary and sibling pair)
pos_{nf}	Merkle tree position of associated note nullifier (primary and sibling pair)
$path_{nf}$	Merkle tree path of the associated note nullifier

transfer and asset spending interchangeably in this paper.

When generating $\pi_{JoinSplit}$, the circuit verifies a set of constraints for the generated transaction, which is applied to generate a proof to be checked by a verifier.

Algorithm 3: Prove algorithm.

```
Input   : $pk_{JoinSplit}$, $\chi$, $\omega$
Output: $\pi_{JoinSplit}$
1 Input & output note type validity: verify that it is a
  valid transaction based on the input types and the
  output types - see Table 3 of Section ??
2 Verify common constraints below:
3 Merkle path validity: $(path_i, pos_i)$ valid tree path from
  NoteCommit $(n_i^{old})$ to Merkle tree root rt
4 Nullifier integrity: $nf_i^{old} = PRF^{nf_{old}}_{a^{old}_{sk,i}}(\rho_i^{old})$
5 Spending key validity: $a^{old}_{pk_i} = PRF^{addr}_{a^{old}_{sk,i}}(0)$
6 Uniqueness of $\rho_i^{new}$: $\rho_i^{new} = PRF^{\rho}_{\varphi}(i, h_{sig})$
7 Note commit validity: $cm_i^{new} = \mathtt{NoteCommit}(n_i^{new})$
8 Transaction case specific validity: verify unique
  constraints for each transaction/spending case as
  sub-circuit - details in Section 5.3
```

Verify$_{OMAP}$: It takes a set of inputs as described in Algorithm 4. It appends $tr_{JoinSplit}$, n_1^{new} and n_2^{new} to the ledger, or output $\perp$.

Receive$_{OMAP}$: Given public parameters pp_{OMAP}, Merkle tree root rt, recipient key pair (a_{sk}, a_{pk}), and the ledger L_{OMAP}, it outputs received note n^{new}, or output $\perp$. The algorithm follows the Zcash design. It scans the ledger and outputs received note for each JoinSplit transaction.

Algorithm 4: Verify algorithm.

```
Input   : Public parameters $pp_{OMAP}$, Merkle tree root
          $rt$, a JoinSplit transaction $tr_{JoinSplit}$, two notes
          $n_1^{new}$ and $n_2^{new}$, and ledger $L_{OMAP}$
Output: accept or reject
1 Parse $tr_{JoinSplit}$
2 Set $b1 \leftarrow Verify(vk_{JoinSplit}, \chi, \pi_{JoinSplit})$
3 Set $b2 \leftarrow v_{sig}(pk_{sig}, m, \delta)$ where m is transaction
  message and $\delta$ is message signature defined in
  Algorithm 2
4 Output $\perp$ if any of the following is true:
      • $b1 \wedge b2$ is false
      • $nf_1^{old}$ or $nf_2^{old}$ appears on $L_{OMAP}$
      • $nf_1^{old} = nf_2^{old}$
      • Merkle root rt not on $L_{OMAP}$
      • $h_{sig}$ does not match with $\mathtt{CRH}(nf_1^{old}, nf_2^{old}, pk_{sig})$
```

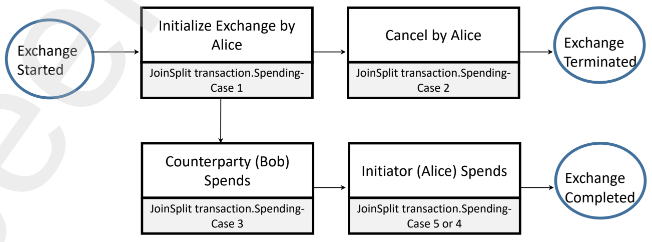
Exchange Started
Initialize Exchange by Alice
JoinSplit transaction.Spending-Case 1
Cancel by Alice
JoinSplit transaction.Spending-Case 2
Exchange Terminated
Counterparty (Bob) Spends
JoinSplit transaction.Spending-Case 3
Initiator (Alice) Spends
JoinSplit transaction.Spending-Case 5 or 4
Exchange Completed

Figure 7: Flow of transaction cases during asset exchange.

5.3. Transactions for Supporting Exchange

OMAP achieves exchange of assets with privacy between two parties using JoinSplit transactions. In order to support fair exchange, the JoinSplit transaction protocol described in Zcash design is extended with new spending scenarios. Accordingly, the zero-knowledge proving system for JoinSplit instance is also extended to include new constraints for each new type of JoinSplit spending scenario.

OMAP extends asset note format with new attributes. As a result, it creates multiple new JoinSplit spending/transaction types. The number of JoinSplit scenarios depends on, type of each input note (primary or sibling), type of each output note (primary or sibling), the value of the debt part in input notes (no debt or with debt). With the restriction of two input notes and two output notes, there are eighteen asset spending scenarios when ignoring the order of note types. This means that changing the position of a note within inputs or outputs will not affect the spending scenario (otherwise there will be total of thirty-two spending scenarios). Of the eighteen spending scenarios, there are eight allowed cases. They are listed in Table 3. The subsequent discussion provides details of each new spending case, in particular, constraints that must be satisfied by the zero-knowledge proof.

Default shielded payment transaction: This is the case of the

Table 3: Spending cases based on input and output note types (assume two input notes and two output notes). $\times$ indicates disallowed scenarios.

Input Types	Debt?	Output Types	Cases
$<0,0>$	No	$<0,0>$	Default shielded payment transaction
		$<0,1>$	Transaction case 1
		$<1,1>$	$\times$
	YES	$<0,0>$	Transaction case 3
		$<0,1>$	Transaction case 1
		$<1,1>$	$\times$
$<0,1>$ *or* $<1,0>$	NO	$<0,0>$	Transaction case 4
		$<0,1>$	Transaction case 4
		$<1,1>$	$\times$
	YES	$<0,0>$	Transaction case 5 or case 2
		$<0,1>$	Transaction case 2
		$<1,1>$	$\times$
$<1,1>$	NO		$\times$
	YES		$\times$

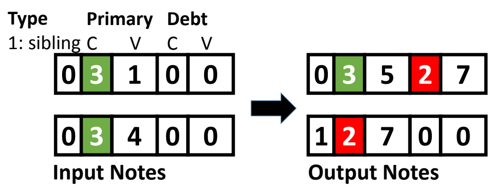

Figure 8: Transaction case 1 example (start exchange).

original JoinSplit transaction where one user sends notes to another user. Its definition is identical to JoinSplit transaction described in the Zcash design.

Exchange initialization: Without loss of generality, we assume that Alice initializes an exchange process. In this case (case 1 in Table 3), the ZK proving algorithm needs to show that the satisfying instance can meet the following constraints in addition to the requirements common to all JoinSplit transactions. Satisfying conditions for the transaction instance include:

$$\begin{cases} s_1^{old} = s_2^{old} = 0 \\ v_{1,2}^{old} = v_{2,2}^{old} = 0 \vee (v_{1,2}^{old} > 0 \wedge v_{2,2}^{old} = 0) \vee (v_{1,2}^{old} = 0 \wedge v_{2,2}^{old} > 0) \\ s_1^{new} = 0 \wedge s_2^{new} = 1 \wedge v_{1,2}^{new} > 0 \wedge v_{2,2}^{new} = 0 \end{cases}$$

Satisfying conditions determine specific requirements for the input and output notes during a transaction. During proof generation, it also helps the system to select an appropriate routine (sub arithmetic circuit) to execute. Suppose a note can be denoted simply as $[s, color_1, v_1, color_2, v_2]$; and Alice creates 5 units of green note (color code: 3) to exchange 7 units of red note (color code: 2) with Bob, we have what is shown in Fig 8.

Note that output note (sibling note) $[1, 2, 7, 0, 0]$ cannot be spent unless output note $[0, 3, 5, 2, 7]$ is spent already or spent together. $[0, 3, 5, 2, 7]$ has a negative part where $color_2 = 2$ and $v_2 = 7$. Additional satisfying conditions for this case, besides

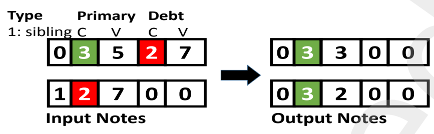

Figure 9: Transaction case 2 example (cancel exchange).

the original verification of privacy JoinSplit [20], include the following: when $v_{1,2}^{old} = v_{2,2}^{old} = 0$, and the second note is not a dummy note, the proving algorithm verifies:

$$\begin{cases} (color_{1,1}^{old} = color_{2,1}^{old} = color_{1,1}^{new}) \wedge color_{1,2}^{new} = color_{2,1}^{new} \\ v_{1,2}^{new} = v_{2,1}^{new} \wedge (v_{1,1}^{old} + v_{2,1}^{old} = v_{1,1}^{new}) \end{cases}$$

When $v_{1,2}^{old} > 0$ or $v_{2,2}^{old} > 0$, the proving algorithm verifies similar sets of satisfying conditions as additional sub-cases under this transaction scenario.

Either Alice or Bob can spend the created primary note. This means that both of them have access to the same note spending key. This can be achieved using several approaches. Both can either run a secret sharing protocol offline or encrypt exchanged messages using each other's address public key and store ciphertexts on the ledger or as part of the encrypted memo associated with a note.

Exchange cancellation by the initiator: The initiator, Alice, is free to cancel the swap before the other party takes actions (the system can enforce that Alice can only cancel after a predetermined time limit specified as block height threshold). This is equivalent to the case that Alice starts a second JoinSplit transaction to get her notes back before Bob does anything. In case of a cancellation transaction (case 2 in Table 3), the proving algorithm verifies the following satisfying conditions for the transaction instance:

$$\begin{cases} s_1^{old} = 0 \wedge s_2^{old} = 1 \wedge v_{1,2}^{old} > 0 \wedge v_{2,2}^{old} = 0 \\ (s_1^{new} = 0 \wedge s_2^{new} = 0 \wedge v_{1,2}^{new} = 0 \wedge v_{2,2}^{new} = 0) \vee \\ (s_1^{new} = 0 \wedge s_2^{new} = 1 \wedge v_{1,2}^{new} > 0 \wedge v_{2,2}^{new} = 0) \end{cases}$$

If Alice issues a new transaction to recover 5 units of green note and cancel the exchange, we have different sub cases.

When $v_{1,2}^{new} = v_{2,2}^{new} = 0$, the proving algorithm verifies the sets of satisfying conditions.

$$\begin{cases} color_{1,2}^{old} = color_{2,1}^{old} \wedge color_{1,1}^{new} = color_{2,1}^{new} = color_{1,1}^{old} \\ v_{1,2}^{old} = v_{2,1}^{old} \wedge v_{1,1}^{new} + v_{2,1}^{new} = v_{1,1}^{old} \end{cases}$$

Similarly, additional transaction sub-cases cover when $s_2^{new} = 1$ and $v_{1,2}^{new} > 0$, or situations that notes may change order of positions (which one is the first note).

Counterparty response with transaction: In order to spend the note that he receives, and complete the exchange, Bob needs to handle the note from Alice with debt. He must spend one of his notes that can cancel out the debt encoded in the note (both type of asset and amount) - case 3 in Table 3. The requirement

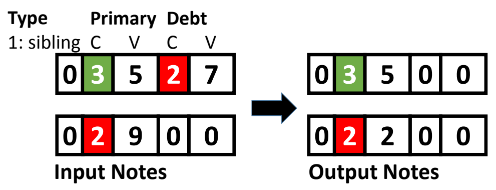

Figure 10: Transaction case 3 example (respond to exchange request).

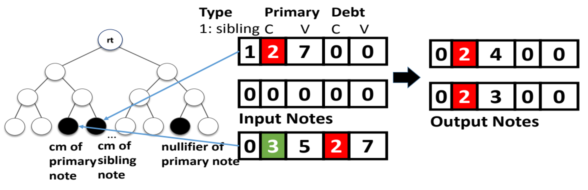

Figure 11: Transaction case 4 example (spend after exchange).

is reflected in the following satisfying conditions by the proving algorithm for the transaction instance that Bob creates:

$$\begin{cases} s_1^{old} = s_2^{old} = s_1^{new} = s_2^{new} = 0 \\ (v_{1,2}^{old} > 0 \wedge v_{2,2}^{old} = 0) \vee (v_{1,2}^{old} = 0 \wedge v_{2,2}^{old} > 0) \\ v_{1,2}^{new} = 0 \wedge v_{2,2}^{new} = 0 \end{cases}$$

In this transaction, Bob receives 5 units of green coins by sending 9 units of red coins in order to finish the exchange operation.

When $v_{1,2}^{old} > 0$ and $color_{1,1}^{old} = color_{1,1}^{new}$, the proving algorithm verifies:

$$\begin{cases} v_{1,1}^{old} = v_{1,1}^{new} \wedge v_{1,2}^{old} + v_{2,1}^{new} = v_{2,1}^{old} \\ color_{2,1}^{old} = color_{2,1}^{new} \wedge color_{1,2}^{old} = color_{2,1}^{old} \\ block_n \leq bt_1^{old} \end{cases}$$

The last condition means that Bob can only spend the note before the block height of bt_1^{old}. If not, Alice can cancel the exchange. This prevents the scenario that exchange neither completes nor terminates by a cancellation. The proving algorithm also needs to verify other sub-cases under this spending scenario when the order of notes changes, e.g., when $v_{2,2}^{old} > 0$ and $color_{1,1}^{old} = color_{2,1}^{new}$, and etc. After Bob spends, a nullifier will be created and inserted to the pool of nullifiers and the combined Merkle tree, which as a result allows Alice to continue and complete the exchange.

Exchange completion by the initiator: In this scenario, Alice spends after Bob. Alice knows when she can spend by scanning the pool of nullifiers. When she detects the expected nullifier, she can complete the exchange by creating a new transaction to spend the matching sibling note (case 4 in Table 3). The proving algorithm verifies the following satisfying constraints by the transaction instance:

$$\begin{cases} s_1^{old} = 1 \wedge s_2^{old} = 0 \wedge v_{1,2}^{old} = v_{2,2}^{old} = 0 \\ (s_1^{new} = 0 \wedge s_2^{new} = 0 \wedge v_{1,2}^{new} = 0 \wedge v_{2,2}^{new} = 0) \vee \\ (s_1^{new} = 0 \wedge s_2^{new} = 1 \wedge v_{1,2}^{new} > 0 \wedge v_{2,2}^{new} = 0) \end{cases}$$

Note that the first input note can only be spent if there is a matching nullifier for a note already spent by Bob that is the primary note associated with the first input note (See Case 3 transaction example). Additional satisfying requirements for this transaction scenario include (when the second note is not a dummy note):

$$\begin{cases} color_{1,1}^{old} = color_{2,1}^{old} = color_{1,1}^{new} = color_{2,1}^{new} \\ v_{1,1}^{old} + v_{2,1}^{old} = v_{1,1}^{new} + v_{2,1}^{new} \\ block_n > bt_1^{old} \wedge s_1^{old} = 1 \\ \exists\, note\ n_{pair\_of\_1}^{old} \wedge (h_{pair\_of\_1,sig}^{old} = h_{1,sig}^{old}) \\ (\exists\, cm_{pair\_of\_1}^{old}\ with\ Merkle\ path_{pair\_of\_1}, pos_{pair\_of\_1}\ w.r.t\ rt \\ \wedge\ cm_{pair\_of\_1}^{old} = NoteCommit(n_{pair\_of\_1}^{old})) \\ (\exists\, nf_{pair\_of\_1}^{old}\ with\ Merkle\ path_{nf}, pos_{nf}\ w.r.t\ rt \\ \wedge\, nf_{pair\_of\_1}^{old} = \mathtt{PRF}^{nf}_{a^{old}_{sk,pair\_of\_1}}(\rho_{pair\_of\_1}^{old})) \\ |pos_{pair\_of\_1} - pos_1| = 1\text{[1]} \end{cases}$$

Similar to Case 3 transaction, one can set a threshold using block height bt_1^{old} to restrict when the note can be spent. In addition, the proving algorithm needs to verify additional sub-cases of transaction instances such as $s_2^{new} = 1$, and etc.

Exchange completion by the initiator (a second scenario): In this scenario, Alice spends after Bob to complete the exchange (case 5 in Table 3). The proving algorithm needs to verify the following constraints on the input notes and output notes:

$$\begin{cases} color_{1,1}^{old} = color_{2,2}^{old} = color_{1,1}^{new} \wedge color_{2,1}^{old} = color_{2,1}^{new} \\ v_{1,1}^{old} = v_{2,2}^{old} + v_{1,1}^{new} \wedge v_{2,1}^{old} = v_{2,1}^{new} \\ block_n > bt_1^{old} \wedge block_n \leq bt_2^{old} \\ \exists\, note\ n_{pair\_of\_1}^{old} \wedge (h_{pair\_of\_1,sig}^{old} = h_{1,sig}^{old}) \\ (\exists\, cm_{pair\_of\_1}^{old}\ with\ Merkle\ path_{pair\_of\_1}, pos_{pair\_of\_1}\ w.r.t\ rt \\ \wedge\, cm_{pair\_of\_1}^{old} = NoteCommit(n_{pair\_of\_1}^{old})) \\ (\exists\, nf_{pair\_of\_1}^{old}\ with\ Merkle\ path_{nf}, pos_{nf}\ w.r.t\ rt \\ \wedge\, nf_{pair\_of\_1}^{old} = PRF^{nf}_{a^{old}_{sk,pair\_of\_1}}(\rho_{pair\_of\_1}^{old})) \\ |pos_{pair\_of\_1} - pos_1| = 1 \end{cases}$$

Additional conditions that need to be satisfied by this scenario: When $s_1^{new} = s_2^{new} = v_{1,2}^{new} = 0$, the proving algorithm needs to verify:

$$\begin{cases} s_1^{old} = 1 \wedge s_2^{old} = 0 \wedge v_{1,2}^{old} = 0 \wedge v_{2,2}^{old} > 0 \\ s_1^{new} = 0 \wedge s_2^{new} = 0 \wedge v_{1,2}^{new} = 0 \wedge v_{2,2}^{new} = 0 \end{cases}$$

6. Security Model and Analysis

In this paper, the described OMAP protocol for supporting the oblivious and fair exchange of confidential digital assets is

[1]The conditions guarantee that note commitment $cm_{pair\_of\_1}^{old}$ is the neighbor of its sibling/pairing note′ commitment cm_1^{old}.

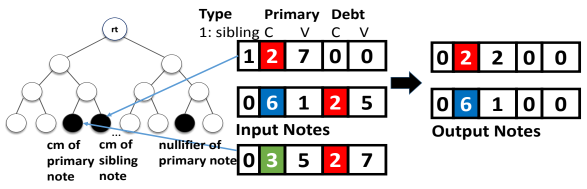

Figure 12: Transaction case 5 example (spend after exchange - another scenario).

based on extending the data models and transaction definitions of the original Zcash protocol. It achieves fair and privacy-preserving exchange through orchestrated payment transactions. Since the OMAP algorithms are within the Zcash framework, most of the properties shown in Zcash and related publications still hold for OMAP [19, 26, 27, 23], for instance, completeness. Because exchange transactions are realized on top of multiple shielded payment transactions, the security proofs for Zcash, shown to be secure for shielded transactions, are mostly applicable to OMAP with straightforward extensions.

Here the discussions focus on properties or requirements unique to the proposed exchange protocol. Note that it is assumed that the underlying distributed ledger system to support consensus and network communications is assumed to be secure and reliable, which is outside the scope of OMAP. We further assume that measures are taken to prevent attacks that may happen at different layers. Such attacks include but are not limited to, for instance, 51% attack, side-channel attacks, privacy compromise through analysis of network traffic patterns, attack to DNS, hard forks, quantum attack, and etc. In addition, we assume that the underlying zero-knowledge proving system is secure.

6.1. Requirements

The original Zcash privacy coin defines security as: ledger indistinguishability, transaction non-malleability, and balance.

Ledger indistinguishability: Ledger L reveals no information to the adversary $\mathcal{A}$ beyond publicly disclosed information.

Transaction non-malleability: No bounded adversary $\mathcal{A}$ can alter any of the data stored within a valid transaction. It prevents the adversary from modifying others' transactions before they are added to the ledger.

Balance: No bounded adversary $\mathcal{A}$ can own more notes than what he minted or received via transactions from others.

Specific to OMAP, we define the following security requirements:

Fairness: When users agree to conduct an exchange of assets, represented as confidential asset notes. The operation leads to two possibilities: it fails that each one still keeps his/her note, or it succeeds that each one possesses the other's note (asset type and amount). In other words, the exchange either completes or aborts.

Privacy: Though all transaction information is stored on the decentralized ledger, other users should not be able to learn any useful information about the exchange transaction. For payment, Zcash is already shown to meet the requirement of transaction indistinguishability. OMAP extends JoinSplit transactions with multiple spending cases. Therefore, transaction indistinguishability is extended to cover the requirement that, no bounded adversary $\mathcal{A}$ can distinguish different scenarios of JoinSplit transactions. This means that OMAP ledger reveals no information to the adversary $\mathcal{A}$ of transactions so that $\mathcal{A}$ can tell if a JoinSplit transaction is a regular payment, exchange initialization, exchange cancellation, and etc.

Balance: No bounded adversary $\mathcal{A}$ can own more notes than what he minted, or received via direct payment from others, or via exchange.

6.2. Analysis

We show that the constructed scheme satisfies all the features defined above, including, *fairness*, *indistinguishability*, and *balance*.

Fairness: This property requires that after an exchange is initiated, it either completes or aborts. At a high level, OMAP achieves fairness by reducing this requirement to assurance that double spending can be detected and prevented by the underlying ledger. According to OMAP, to start an exchange, Alice creates a pair of notes (one primary note and one sibling note). The primary note is shared between Alice and Bob, so either one of them can be spent (but not both). If fairness is not satisfied, this means that the system reaches a state that either both Alice and Bob succeeded in spending the primary note; or none of them can spend the primary note. For the first case, it is clear that it is equivalent to double spending and is prohibited by the consensus mechanism of the underlying ledger. For the second case, if neither Alice nor Bob has spent the primary note yet, then one of them can spend in the future. In case, there is a time limit, Alice can eventually spend the primary note, which terminates the exchange operation. The likelihood that someone else besides Alice and Bob spends the primary note is negligible and ensured by the security of the base protocol. Therefore, one can conclude, that with probability 1-negl, the exchange will either succeed or terminate.

Privacy: In OMAP, the exchange is achieved through multiple JoinSplit transactions (shielded to shielded payment transactions). Although OMAP defines multiple JoinSplit transaction scenarios, there is only one combined circuit for proving ZK satisfiability of a JoinSplit transaction instance. This guarantees that different JoinSplit spending cases within an exchange operation or between exchange operations or between an exchange operation and regular payment should be indistinguishable. They are different ZK satisfying instances of the same proving circuit. If an adversary $\mathcal{A}$ can break the indistinguishability property, it means that $\mathcal{A}$ can distinguish different proofs, which contradicts to the assumption that the underlying zero-knowledge proving system is secure.

Balance: This requires that no bounded adversary $\mathcal{A}$ can own more notes than what he receives via direct payment or an exchange. Note that OMAP implements fair exchange using JoinSplit payment transactions. One has to spend valid notes in order to initialize an exchange operation. The original Zcash

Table 4: Privacy coin setting

Field	Size	Field	Size
Merkle tree height	64 bits	Coin type	1 bit
Primary coin value	64 bits	Primary coin color	32 bits
Debt coin value	64 bits	Debt coin color	32 bits
Time threshold	64 bits		

protocol already shows that the balance requirement is satisfied by JoinSplit transactions. OMAP introduces sibling note. Somehow, the asset in a sibling note can be viewed as borrowed from the system. A sibling note cannot be spent by itself. According to JoinSplit algorithm, the condition to spend a sibling note is that either the associated primary note is spent already or it is spent together with the paired primary note in a JoinSplit transaction. This guarantees balance during the exchange operation. Assume that adversary $\mathcal{A}$ can spend a sibling note itself independent of the associated primary note. According to the OMAP JoinSplit ZK proof algorithm, this means that the adversary $\mathcal{A}$ must find another primary note with matching sequence number S. For a pair of primary note and sibling note, S is created by a CRH using input values unique to the transaction. If adversary $\mathcal{A}$ can spend a sibling note by pairing it with another note, it means that adversary $\mathcal{A}$ can break the CRH by finding a collision, which contradicts the assumption that CRH is collision-resistant.

7. Implementation and Experiments

This section presents the implementation of the proposed exchange scheme and evaluates its performance.

7.1. Arithmetic Circuit Design for OMAP

As discussed in the previous sections, zero-knowledge proving system allows anyone to verify a transaction without complete information from a prover. It means that the system does not need to provide private transaction information (e.g., amounts, asset types, and etc.). The verifier can still validate if the transaction instance satisfies the preset conditions. There are several tools and libraries available to achieve this goal. For example, Pinocchio [28], zk-SNARK [4], and Geppetto [29]. In common, zero-knowledge proving system requires a complex circuit to generate proofs and verifying keys for the verifiers.

In the experiment implantation, we used xJsnark [30]. xJsnark is a tool that achieves "program-to circuit" conversion, i.e., to compile a user-supplied program described in a Java-like source language into a compact circuit representation that is recognized by the existing SNARK libraries. It creates circuits in a libsnark compatible format so that the resulting SNARK can be executed using the libsnark back end. xJsnark implements several optimizations to improve the efficiency of frequent operations and reduce circuit size. It supports efficient short and long integer arithmetic and implements global optimizations for integer arithmetic. The compiler has a built-in optimizer of circuit minimization.

The existing code developed by xJnark that implements Zcash transactions is used as a baseline. The Zcash circuit implemented in xJsnark is very efficient. In fact, the compiled circuit under xJsnark is slightly better than the manually optimized implementation. The reason is due to the low-level arithmetic optimizations and the circuit minimizer. Necessary changes are made to implement the described OMAP system. A proving system is used to generate proof for each JoinSplit transaction based on the extended data models and algorithms. Note that we need to generate a key pair of <*pk*,*vk*> in a zero-knowledge proving system. Eventually, a verifier just needs the *vk*, public inputs χ, and the proof π to verify a transaction.

In the beginning, the circuit verifies the validity of the spending case by checking asset/coin types of both the inputs and outputs. Then, it verifies the conditions that are common for all the transaction/spending scenarios. Next, it splits the verification to individual transaction/spending case, as sub-circuit. Each sub-circuit handles unique constraints of different transaction cases and checks the satisfiability of the transaction information. The results are combined together using logical operation.

Table 5: Implementation results.

Constraints	9547660
QAP degree	8388608
PK size in bits	13906386458
VK size in bits	29787
Prove time	344s
Verification time	8ms

Table 6: Circuit size under different Merkle tree heights

	Height=16	Height=32	Height=64
# of constraints	2958988	5155212	9547660
QAP degree	2359296	4194304	8388608

7.2. Circuit Evaluation

We measure the size of the produced circuit, as well as proving the key and verifying the key. The results are listed in Table 5. We also evaluate the complexity of the circuit by analyzing the degree of Quadratic Arithmetic Programs (QAP). The degree of QAP is a metric to measure the complexity of a circuit (Refer to [31] for more details). We conducted experiments using an Intel Xeon computer that has 8 Xeon cores and 64GB ram. The installed system is Ubuntu 18.04. Comparing with the baseline implementation in xJsnark, the circuit size increases. This is primarily due to the verification of additional constraints in the transaction/spending scenarios, which requires extra validation of Merkle tree hash paths (both coin commit and nullifier). However, verification time does not increase significantly. It is almost identical (about 8ms vs. 7.5ms). Additional studies indicate that the circuit size increase is largely dominated by the multiple verifications of Merkle tree paths. This can be clearly observed when comparing circuit size under different Merkle tree heights. The circuit size grows almost linearly with the height of Merkle hash tree. Comparing with the cost of hash tree verification, the cost of other verification operations of constraints is almost negligible. The complexity of the proving circuit can be significantly reduced if the system applies k-anonymity instead of the default Merkle hash tree verification. k-anonymity has been implemented in privacy coin schemes such as Monero [32] and Zether [6]. It is also possible to use a different hash algorithm from $SHA256$. These circuit optimizations are complementary to the current design.

8. Related Works and Discussion

Zero-knowledge based privacy coins. Zcash [4] and Zerocoin [19] implement privacy-preserving transaction systems by applying zero-knowledge proof where the connection between the payer and payee in a transaction is cut off. It leaves no trail of both the entities involved in a payment transaction [4]. In recent times, we have seen several exchange systems built over the blockchain that somewhere in their systems adopt zero-knowledge proof to enhance privacy. Zero-knowledge-based digital asset and crypto-currency systems can be extended to support multiple confidential asset types (e.g., [33]). A scheme has been proposed where a single node can handle multiple asset types [34]. Quasi-homomorphic symmetric encryption was used to hide transaction information in a blockchain environment [35]. Techniques were also developed to shift the computation burden involved in privacy enhancement, especially when the blockchain participants are IoT devices with limited resources [36]. To our knowledge, none of them support the private and atomic exchange of confidential assets based on a unified transaction layer.

Sean Bowe *et al.* propose a decentralized-ledger-based system that allows transactions to hide all information about the offline computations but any node can still validate the computations even without whole details [37]. A use case of cryptocurrency exchange is mentioned in their research. It leverages Z_{EXE} and supports exchange. However, the system needs to execute a script to verify a preset list of conditions before sending the notes. In another word, it has to wait until it gets expected results from the commitment pool. In contrast, in our approach, notes are sent right away. It reduces the delay of transactions. In the literature, one can find applications of the zero-knowledge proof in other areas like privacy-preserving auditing [38].

There are efforts to port Zerocoin protocol to other recent zero-knowledge proof systems such as bulletproof [22]. Such efforts are orthogonal to the endeavor in this work as exchange protocol could be adapted to other zero-knowledge circuit proving systems.

Decentralized exchange. Note that our approach focuses on the exchange, or swap, of confidential assets supported by the same distributed ledger. There are plenty of projects that implement exchanges of assets from different blockchains. In the case of privacy coins, typically, such exchange can be completed using a centralized intermediary or escrow service provider. The escrow services are basically third-party implementations that allow for fast off-chain completion of the transaction and it has been implemented, as an example, in TumbleBit [39]. Our system does not require nor assume the existence of intermediaries or escrow services for atomic exchanges. Our approach also differs from asset transfer designs that rely on off-chain payment channels or payment networks (e.g., [40]. All the transactions during an exchange are verifiable by the participating nodes of a blockchain system that implements the design and are recorded on the main chain.

Trusted execution environment (TEE) for privacy enhancement. There exit efforts on utilizing TEE technology to improve the performance and security of smart contract [41, 42], which can be leveraged to implement exchange or provide privacy protection. However, this type of approach relies on the trust of the hardware and the vendor behind it.

9. Conclusion

Exchanging different types of digital assets with privacy protection on distributed ledger/ blockchain is a natural extension of payment transaction systems. In response to this demand, we developed a novel multi-asset and privacy-protected asset exchange framework on blockchain and applied zero-knowledge proof to enhance privacy protection for exchange operations. In addition, we discussed the implementation of the unified zero-knowledge proving system used to support both payment transactions and multi confidential asset exchanges with privacy; and a formal definition of oblivious and privacy-preserving exchange scheme on the public ledger. We extended the data formats and proving system of the existing privacy coin to support the designed protocol. We have implemented the design using xJsnark tool. Results of the circuit such as circuit complexity, QAP degree, and overall performance are reported in this paper.

References

[1] D. Ron, A. Shamir, Quantitative analysis of the full bitcoin transaction graph, IACR Cryptology ePrint Archive 2012 (2012) 584. URL `http://eprint.iacr.org/2012/584`

[2] R. Henry, A. Herzberg, A. Kate, Blockchain access privacy: Challenges and directions, IEEE Security Privacy 16 (4) (2018) 38–45.

[3] N. Van Saberhagen, Cryptonote v 2.0 (2013).

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

[5] S. Noether, Ring signature confidential transactions for monero., IACR Cryptology ePrint Archive 2015 (2015) 1098.

[6] B. Bünz, S. Agrawal, M. Zamani, D. Boneh, Zether: Towards privacy in a smart contract world, in: J. Bonneau, N. Heninger (Eds.), Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers, Vol. 12059 of Lecture Notes in Computer Science, Springer, 2020, pp. 423–443.

[7] M. C. Kus Khalilov, A. Levi, A survey on anonymity and privacy in bitcoin-like digital cash systems, IEEE Communications Surveys Tutorials 20 (3) (2018) 2543–2585.

[8] H. Al-Breiki, M. H. U. Rehman, K. Salah, D. Svetinovic, Trustworthy blockchain oracles: Review, comparison, and open research challenges, IEEE Access 8 (2020) 85675–85685.

[9] S. Gupta, J. Hellings, S. Rahnama, M. Sadoghi, Blockchain consensus unraveled: Virtues and limitations, in: Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems, DEBS '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 218–221.

[10] E. Ben-Sasson, A. Chiesa, E. Tromer, M. Virza, Succinct non-interactive zero knowledge for a von neumann architecture, in: 23rd USENIX Security Symposium (USENIX Security 14), USENIX Association, San Diego, CA, 2014, pp. 781–796.

[11] S. Goldwasser, S. Micali, C. Rackoff, The knowledge complexity of interactive proof-systems, in: Proceedings of the 7th annual ACM Symposium on Theory of Computing - STOC 1985, ACM, 1985, pp. 291–304.
[12] O. Goldreich, A. Kahan, How to construct constant-round zero-knowledge proof systems for np, Journal of Cryptology 9 (3) (1996) 167–189.
[13] M. Blum, P. Feldman, S. Micali, Non-interactive zero-knowledge and its applications, in: Proceedings of the twentieth annual ACM symposium on Theory of computing, ACM, 1988, pp. 103–112.
[14] J. Groth, R. Ostrovsky, A. Sahai, Perfect non-interactive zero knowledge for np, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2006, pp. 339–358.
[15] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, A. Sahai, Robust non-interactive zero knowledge, in: Annual International Cryptology Conference, Springer, 2001, pp. 566–598.
[16] R. Gennaro, C. Gentry, B. Parno, M. Raykova, Quadratic span programs and succinct nizks without pcps, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2013, pp. 626–645.
[17] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, M. Virza, Snarks for c: Verifying program executions succinctly and in zero knowledge, in: Advances in Cryptology–CRYPTO 2013, Springer, 2013, pp. 90–108.
[18] E. Ben-Sasson, A. Chiesa, E. Tromer, M. Virza, Succinct non-interactive zero knowledge for a von neumann architecture, in: 23rd USENIX Security Symposium, 2014, pp. 781–796.
[19] I. Miers, C. Garman, M. Green, A. D. Rubin, Zerocoin: Anonymous distributed e-cash from bitcoin, in: Security and Privacy (SP), 2013 IEEE Symposium on, IEEE, 2013, pp. 397–411.
[20] D. Hopwood, S. Bowe, T. Hornby, N. Wilcox, Zcash protocol specification, Tech. rep. 2016–1.10. Zerocoin Electric Coin Company, Tech. Rep.
[21] E. Ben-Sasson, I. Bentov, Y. Horesh, M. Riabzev, Scalable, transparent, and post-quantum secure computational integrity, Cryptology ePrint Archive, Report 2018/046, `https://eprint.iacr.org/2018/046` (2018).
[22] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, G. Maxwell, Bulletproofs: Short proofs for confidential transactions and more, in: 2018 IEEE Symposium on Security and Privacy, 2018, pp. 315–334.
[23] K. Kluczniak, M. H. Au, Fine-tuning decentralized anonymous payment systems based on arguments for arithmetic circuit satisfiability, Cryptology ePrint Archive, Report 2018/176, `https://eprint.iacr.org/2018/176` (2018).
[24] S. Bowe, A. Gabizon, M. D. Green, A multi-party protocol for constructing the public parameters of the pinocchio zk-snark, Cryptology ePrint Archive, Report 2017/602, `https://eprint.iacr.org/2017/602` (2017).
[25] S. Bowe, A. Gabizon, I. Miers, Scalable multi-party computation for zk-snark parameters in the random beacon model, Cryptology ePrint Archive, Report 2017/1050, `https://eprint.iacr.org/2017/1050` (2017).
[26] A. Kosba, A. Miller, E. Shi, Z. Wen, C. Papamanthou, Hawk: The blockchain model of cryptography and privacy-preserving smart contracts, University of Maryland and Cornell University.
[27] C. Garman, M. Green, I. Miers, Accountable privacy for decentralized anonymous payments, Cryptology ePrint Archive, Report 2016/061, `https://eprint.iacr.org/2016/061` (2016).
[28] B. Parno, J. Howell, C. Gentry, M. Raykova, Pinocchio: Nearly practical verifiable computation, in: Security and Privacy (SP), 2013 IEEE Symposium on, IEEE, 2013, pp. 238–252.
[29] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, S. Zahur, Geppetto: Versatile verifiable computation, in: 2015 IEEE Symposium on Security and Privacy, IEEE, 2015, pp. 253–270.
[30] A. Kosba, C. Papamanthou, E. Shi, xjsnark: A framework for efficient verifiable computation, in: 2018 IEEE Symposium on Security and Privacy (SP), IEEE Computer Society, Los Alamitos, CA, USA, 2018, pp. 944–961.
[31] V. Buterin, Quadratic arithmetic programs: from zero to hero (2016).
[32] S. Noether, Ring signature confidential transactions for monero, Cryptology ePrint Archive, Report 2015/1098, `https://eprint.iacr.org/2015/1098` (2015).
[33] ZkVM transaction specification, `https://github.com/stellar/slingshot/blob/main/zkvm/docs/zkvm-spec.md` (2019).
[34] A. Poelstra, A. Back, M. Friedenbach, G. Maxwell, P. Wuille, Confidential assets, in: A. Zohar, I. Eyal, V. Teague, J. Clark, A. Bracciali, F. Pintore, M. Sala (Eds.), Financial Cryptography and Data Security, Springer Berlin Heidelberg, Berlin, Heidelberg, 2019, pp. 43–63.
[35] S. Bai, G. Yang, C. Rong, G. Liu, H. Dai, Qhse: An efficient privacy-preserving scheme for blockchain-based transactions, Future Generation Computer Systems 112 (2020) 930–944.
[36] B. Gong, C. Cui, M. Hu, C. Guo, X. Li, Y. Ren, Anonymous traceability protocol based on group signature for blockchain, Future Generation Computer Systems 127 (2022) 160–167.
[37] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, H. Wu, Zexe: Enabling decentralized private computation, in: 2020 IEEE Symposium on Security and Privacy (SP), 2020, pp. 947–964.
[38] N. Narula, W. Vasquez, M. Virza, zkledger: Privacy-preserving auditing for distributed ledgers, in: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), USENIX Association, Renton, WA, 2018, pp. 65–80.
[39] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, S. Goldberg, Tumblebit: An untrusted bitcoin-compatible anonymous payment hub, in: 24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017, The Internet Society, 2017.
[40] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, M. Maffei, Anonymous multi-hop locks for blockchain scalability and interoperability, Cryptology ePrint Archive, Report 2018/472, `https://eprint.iacr.org/2018/472` (2018).
[41] R. Karanjai, L. Xu, L. Chen, F. Zhang, Z. Gao, W. Shi, Lessons learned from blockchain applications of trusted execution environments and implications for future research, arXiv preprint arXiv:2203.12724.
[42] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, A.-R. Sadeghi, Fastkitten: Practical smart contracts on bitcoin., IACR Cryptology ePrint Archive 2019 (2019) 154.