Felix Engelmann, Thomas Kerber, Markulf Kohlweiss, and Mikhail Volkhov*

Zswap: zk-SNARK Based Non-Interactive Multi-Asset Swaps

Abstract: Privacy-oriented cryptocurrencies, like Zcash or Monero, provide fair transaction anonymity and confidentiality, but lack important features compared to fully public systems, like Ethereum. Specifically, supporting assets of multiple types and providing a mechanism to atomically exchange them, which is critical for e.g. decentralized finance (DeFi), is challenging in the private setting. By combining insights and security properties from Zcash and SwapCT (PETS 21, an atomic swap system for Monero), we present a simple zk-SNARKs based transaction scheme, called Zswap, which is carefully malleable to allow the merging of transactions, while preserving anonymity. Our protocol enables multiple assets and atomic exchanges by making use of sparse homomorphic commitments with aggregated open randomness, together with Zcash friendly simulation-extractable non-interactive zero-knowledge (NIZK) proofs. This results in a provably secure privacy-preserving transaction protocol, with efficient swaps, and overall performance close to that of existing deployed private cryptocurrencies. It is similar to Zcash Sapling and benefits from existing code-bases and implementation expertise.

Keywords: NIZK, Cryptocurrency, Privacy, Multi-Asset, Exchange

DOI 10.56553/popets-2022-0120
Received 2022-02-28; revised 2022-06-15; accepted 2022-06-16.

1 Introduction

Cryptocurrencies are experiencing steady growth not only in terms of general popularity, security, and real-world applicability, but also in terms of diversity of financial instruments that can be realized with them. Decentralized finance (DeFi [23]), an umbrella term that covers such financial instruments in the cryptocurrency community, is one of the raison d'être of Ethereum, a cryptocurrency popularized by the wide applicability of its smart contract toolchain. One of the foundational pieces of DeFi is the ability (of the distributed ledger) to create user-defined tokens, and trade or exchange them directly on-chain. Ethereum, by providing Turing complete smart contracts and the ERC tokens standards (e.g. ERC20 or non-fungible ERC721), allows building tools such as automated exchanges [24], investment platforms[1][2], bidding platforms, insurance tools, NFT marketplaces, etc. An important limitation of current DeFi solutions is the public nature of these atomic on-chain exchanges.

While privacy-preserving cryptocurrencies, like Monero and Zcash, are undoubtedly practical, and ample academic research on the problem of private currencies is available [1, 4, 6, 12, 13], much less is known about the *private exchange* of assets [8, 11, 14] — hindering DeFi applications. This is unfortunate, as privacy on the blockchain, such as transaction anonymity or transfer amount secrecy, is not only interesting per se (for the end user), but also changes the financial landscape of the ecosystem. This can be advantageous, for example, if restricting the adversarial view prevents certain harmful behaviour that results from gaming the market (e.g. frontrunning, Miner Extractable Value). Consider a miner which observes a buy-order for an asset that is either from a notorious investor or has an exceptionally large transfer amount. The miner can buy that asset itself early and cheaply before the order drives up the price of the asset. Private swaps of assets can mitigate such attacks.

However, while practically attractive, combining privacy with DeFi tools is challenging. Solutions that try to tackle this question, such as private smart contracts [20] and privacy-friendly decentralized exchanges [3], often find it hard to balance practical applicability with the privacy guarantees they provide.

Felix Engelmann: IT University of Copenhagen, fe-research@nlogn.org
Thomas Kerber: University of Edinburgh, IOHK, thomas.kerber@iohk.io
Markulf Kohlweiss: University of Edinburgh, IOHK, markulf.kohlweiss@ed.ac.uk
***Corresponding Author: Mikhail Volkhov:** University of Edinburgh, mikhail.volkhov@ed.ac.uk

1 https://compound.finance/markets
2 https://polygon.market.xyz/

Flexibility, so desirable for DeFi, is at odds with privacy, since having both generally requires heavier cryptographic primitives, like general NIZKs, or multi-party computation (MPC).

In this work we do not support a full private DeFi solution, but instead focus on the foundational problem of constructing a private cryptocurrency mechanism that has both support for multiple assets, and an embedded functionality to perform non-interactive atomic assets swaps (as well as regular transfers).

An atomic swap is the exchange of different assets between multiple parties. It has to happen atomically such that all participants get their desired output or the transaction is aborted. A classical example is a foreign currency exchange, where a bank sells a foreign currency to a customer who pays in the local currency. There, atomicity is guaranteed by simultaneously handing over the assets.

The atomic swap protocol we suggest enables untrusted parties to merge transactions off-chain. The transactions themselves only reveal a map of the imbalance for each asset where the sum of its inputs is unequal to the sum of its outputs. Hence, a balanced transaction does not reveal any amounts or types. Especially, the mergers can neither deanonymize senders or receivers nor correlate a subsequent spending of an output by its precise amount and type. With such little trust required in mergers, this very basic functionality already allows creating local exchange markets, where users can send exchange offers (as transactions with a negative imbalance for the asked token and a positive imbalance for the offered), and community-selected participants can match them and merge them to then submit to the blockchain. The anonymity of the system is controlled by users: the system allows both private swaps between several parties, who agree on their exchange off-chain, and bigger exchange pools, as just mentioned. In both cases, the only information that the mergers and other users see is the one necessary to match the offers (total value for each unbalanced type), and it is erased as soon as the transaction is balanced and sent to the ledger. For partial merges, any type with an imbalance of zero is dropped. An open group of participants in a pool may provide sufficient liquidity and maintain a public order book, similar to classical exchanges.

An interesting open question is how to integrate Zswap with private smart contracts to support more elaborate private DeFi solutions. A first step in that direction would be to extend a public smart contract system with minting policies for private assets, to support, e.g. the private trading of NFTs. Finally, our mechanism is compatible with existing consensus protocols, such as proof-of-work or stake, and can be viewed as a full cryptocurrency.

1.1 Technical Overview

With respect to swaps, a major challenge of many existing solutions is that transaction data is not explicitly separated from the transaction signature, which binds inputs from multiple users together. This, often results in the need for (slow) MPC protocols to construct such a signature jointly. An important insight of our work is that the Zcash ecosystem already implement signature separation. It was first introduces by Zcash Sapling [16] to reduce the size of SNARK circuits for large transactions, and was inherited by the corresponding Shielded Assets[3], or similar MASP[4] multi-asset protocols. Another inspiration of our work is SwapCT, that realizes atomic swaps on Monero [11]. We continue this direction by designing and proving secure an extension of a Zcash Sapling like system that satisfies the required multi-asset and atomic swap properties.

Zcash Sapling, as opposed to the Zerocash paper, separates the validation of inputs and outputs (each input and output requires a separate NIZK) from the transaction balancing, which is done using homomorphic commitments and a Schnorr-like binding signature. The binding signature "seals" the inputs and outputs in place, forbidding adding or removing any extra inputs or outputs. Instead, in our work we use a sparse multi-value Pedersen commitments and relax the signature, allowing transactions to be non-interactively merged together. While inspired by SwapCT, we deal with different challenges specific to the zk-SNARK setting and take advantage of the existing signature separation in Zcash Sapling. The explicit signature separation opens more space for potential transaction malleability, and should be taken with care. To our knowledge our work is the first to extend the security analysis of [4] in that direction.

Our modelling relies on a one-time account (OTA) scheme to anonymously and confidentially create and store value in notes. This is an abstraction from existing protocols in Zcash and Monero. The spending of input

3 Previously "user-defined assets" (UDA) or UIT, see ZIP 220: https://github.com/zcash/zcash/issues/830

4 https://github.com/anoma/masp/blob/main/docs/multi-asset-shielded-pool.pdf

notes, stored in a Merkle tree, is authorized by a simulation extractable (SE) non-interactive zero-knowledge proof (NIZK). Double spends are prevented by proving the correctness of a deterministic nullifier, marking an input as spent. To connect inputs to newly created output notes, we use sparse homomorphic commitments to value-type pairs, which can be summed to check that the transaction is well-balanced. Importantly, the values and types remain hidden even when we publish their aggregated randomness, which is necessary to facilitate transaction merging.

Our protocol bears similarities to variants of Zcash and SwapCT which we compare in more detail:

Zcash Sapling: Compared to Sapling, we add multi-asset support and remove the authorization signature which is replaced by the SE NIZK. We also do not need a binding signature over all intermediate commitments — instead we directly publish the randomness. This opens room for controlled malleability, and enables non-interactive joint transaction generation (by merging) without MPC, which can be used to implement swaps mechanics. The multi-asset aspect of our scheme is very similar to Shielded Assets or MASP, both of which extend Sapling but do not provide a mechanism for atomic swaps. We also provide a rigorous theoretical formalization, which is lacking for both Sapling and its Shielded Asset and MASP variants, and can be of independent interest.

Conversely, the technical discussions and implementation effort that went into the Shielded Assets ZIP 220 and the MASP implementation are valuable starting points for the deployment of Zswap as part of a larger Zcash like blockchain. In particular, deployment of our new transactions into Zcash requires a hard fork and creates a new shielded pool holding typed notes. Transferring tokens from an existing shielded pool is possible by explicitly adding the `Zcash` type to the notes.

SwapCT: Our work is inspired by SwapCT which also provides atomic swaps and transaction merging, but on top of Monero (ring based anonymity). We simplified their scheme to unify their offers and transactions. With the use of SNARKs (requiring a trusted or transparent setup) our simpler construction no longer needs their anonymously aggregatable signatures. We achieve better confidentiality for unfinished transactions as we only reveal a total imbalance instead of per input and output values. Another important difference is that the use of SNARKS allows to efficiently support large rings. This motivates the use of a single global ring in our formalization.

Formalization: We formalize our protocol using game based definitions, which generalize Zerocash definitions, but are also compatible with Monero except for using a single global ring (as in Zcash and differing from SwapCT).

Zerocash [4] formalizes *Ledger indistinguishability*, *Transaction non-malleability* and *Balance*. Our balance definition is very similar with the addition of checking multiple assets. We replace indistinguishability with the privacy definition, as our transactions may be circulated off-chain. The non-malleability of merged transactions is inspired by Zerocash and SwapCT [11]. Instead of the strong non-malleability, it provides a relaxed property that controls malleability (transaction merging) and only requires theft prevention, namely that an adversary cannot steal from the intended recipients of a transaction, or split unbalanced transactions, rerouting honest funds. Modelling this malleability relies on a variant of hiding of the Pedersen commitment scheme, when the joint commitment randomness is revealed, which we call HID-OR[5]. This property does not require any additional security assumptions.

1.2 Our Contributions

In this work we present the following results:

- We specify a formal model for a multi-asset Zcash system with swaps that builds on top of a One-Time Account (OTA) System, abstracting a nullifier-like private UTXO mechanism. The OTA model and our proof techniques could be of independent interest for proving systems such as Zcash and Monero secure.
- We provide a minimal practical instantiation of non-interactive private atomic-swaps. It is based on a simplified version of Zcash that removes authorization and blinding signatures.
- We prove our construction secure under commonly used assumptions similar to the ones used in Zerocash. This validates the removal of the Zcash signatures and shows that the perfect hiding and binding properties of spend and output commitments is sufficient for security.
- We implement and evaluate our protocol: our merging mechanism is very effective, and all performance overheads of our construction, as compared to the basic single-asset protocol without swaps, are small.

5 HIDing with Open Randomness

1.3 Related Work

Surprisingly little academic work *directly* addresses exchange of multiple assets in a private ledger. Undoubtedly, such a functionality can be achieved through certain generic private smart contract solutions [17, 18, 22], but their flexibility comes at a cost of non-negligible performance overhead, since they often require heavy primitives like SNARKs over big contract code-dependant (or even universal) circuits. The performance of universal zero-knowledge based constructions such as ZEXE [5] requires minutes of proving time for ten times the constraints compared to our one second prover runtime. This is why we would like to consider systems with such a functionality embedded directly.

Several solutions take the route of extending the vanilla Zerocash. Ding et al. [10] propose a solution supporting multiple assets, but with no exchange mechanism, and with public asset types. Gao et al. [14] construct a transaction system specifically for exchanging assets which is based on storing debt in sibling notes which are only spendable if the debt is settled. Its inefficiency results from requiring multiple persisted transactions per swap.

On the other hand, Confidential Assets [21] uses a commitment construction that hashes an asset type descriptor to the bases of an extended Pedersen commitment such that the resulting commitments are additively homomorphic, which facilitates proving the balancing of amounts. We use this sparse commitment scheme as part of our construction. A similar solution by Zheng et al. [25] exists for the Mimblewimble private cryptocurrency. Both these works do not provide sender and receiver anonymity.

Sparse homomorphic commitments have the drawback that they can only store type-amount pairs. A more flexible approach is to use a hash-based commitment scheme for notes to store a vector of attributes. This is used in Zcash's multi-asset ZIP 220, Shielded Assets, or similarly MASP. To achieve balancing, these protocols prove equality between the type-value pair of a note and a sparse homomorphic commitment. We follow the same approach. Another system that *does not* rely on homomorphic commitments is Stellar[6], which instead uses shuffle proofs.

Finally, some works emphasize the exchange and offer matching functionality. Manta [8] describes a privacy-preserving decentralized exchange (DEX) based on an automated marked maker (AMM) scheme which works without a second party but does not hide the types, which is an inherent limitation of the AMM approach. Another idea is to privately exchange assets between different systems in cross-chain atomic swaps [9]. In contrast to both, we propose a more basic mechanism within a single blockchain, and leave leave it open to implement the concrete offer matching algorithm on top of Zswap. This allows, and will likely enable more powerful DeFi applications, since Zswapprovides a more flexible interface to the application layer.

2 Preliminaries

Let us introduce our notation. We write PPT for probabilistic polynomial-time. All our sets are multisets by default. The set minus operation $A \setminus B$ removes as many values in A as there are in B: if B has more values of the same type than in A, the number of elements in the resulting set is 0. The function $\mathsf{ToSet}(S)$ removes all duplicate entries in the multiset S and sorts the result according to some predefined ordering, essentially converting S to a "proper" set. We use $[n] := 1 \dots n$, and $|S|$ for the cardinality of the set S, $|\vec{x}|$ is the length if $\vec{x}$ is a vector, thus $[|S|]$ means $1 \dots |S|$.

As of pseudocode conventions, abortion by default means returning $\perp$ immediately, and assertions abort on failure. When the result of an assignment cannot be properly pattern-matched, e.g. $(a, b) \leftarrow \mathsf{foo}$ (function returns $\perp$ or a different type), the execution aborts. The symbol @ denotes temporary variable assignment in the pattern-matching cases, e.g. $(a@(x, y), z) \leftarrow \mathsf{Foo}()$ means $((x, y), z) \leftarrow \mathsf{Foo}(); a \leftarrow (x, y)$.

2.1 Sparse Homomorphic Commitments

Our first building block is a commitment scheme which is additively homomorphic for an exponential number of possible domains, also called types, however each commitment is sparsely populated, i.e. few domains have a non-zero value. Between domains, there exists no homomorphic property. Contructions were proposed by [21] and formalized by [7] which resemble vector Pedersen commitments with ad-hoc generators.

6 https://github.com/stellar/slingshot/blob/main/spacesuit/spec.md

Definition 2.1 (Sparse Homomorphic Commitment)**.** *A* SHC *scheme consists of the algorithms* ComSetup *and* Commit *defined as follows:*

$\mathsf{p} \leftarrow \mathsf{ComSetup}(1^\lambda)$ *takes the security parameter* λ *and outputs the public parameters* p *implicitly provided to* Commit*. It specifies a type space* $\mathbb{T}$

$\mathsf{com} \leftarrow \mathsf{Commit}(\{(\mathsf{ty}_i, a_i)\}_{i=1}^n, \mathsf{rc})$ *takes a set of distinct types* ty_i *and values* a_i*, and a randomness* rc*, and outputs a commitment* com*. We write* $\mathsf{Commit}(\mathsf{ty}, a, \mathsf{rc})$ *for* $\mathsf{Commit}(\{(\mathsf{ty}, a)\}, \mathsf{rc})$*.*

Definition 2.2 (Homomorphism)**.** *A* SHC *scheme is homomorphic, if there exists an efficient operation* $\oplus$*, such that for any* $\mathsf{ty} \in \mathbb{T}$ *it holds that*

$$\mathsf{Commit}(\{(\mathsf{ty}_i, a_i)\}_1^n, \mathsf{rc}) \oplus \mathsf{Commit}(\{(\mathsf{ty}_i, a'_i)\}_1^n, \mathsf{rc}') = \mathsf{Commit}(\{(\mathsf{ty}_i, a_i + a'_i)\}_1^n, \phi(\mathsf{rc}, \mathsf{rc}'))$$

for some function ϕ*, while zero values do not affect the input set:* $\mathsf{Commit}((\mathsf{ty}, 0), \mathsf{rc}) = \mathsf{Commit}(\emptyset, \mathsf{rc})$*.*

In other words, the commitment is additive on the same type, but acts like a vector commitment on distinct types.

We require standard binding and hiding properties.

Definition 2.3 (Commitment Binding)**.** *For any* $\lambda \in \mathbb{N}$ *with* $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$ *and any PPT adversary* $\mathcal{A}$*, it holds that*

$$\Pr\left[\begin{array}{l} \{(\mathsf{ty}_b, a_b, \mathsf{rc}_b)\}_{b=0}^1 \leftarrow \mathcal{A}(\mathsf{p}); \\ \mathsf{com}_0 = \mathsf{Commit}(\mathsf{ty}_0, a_0, \mathsf{rc}_0); \\ \mathsf{com}_1 = \mathsf{Commit}(\mathsf{ty}_1, a_1, \mathsf{rc}_1) \ : \\ \quad \mathsf{com}_0 = \mathsf{com}_1 \ \wedge \\ \quad (\mathsf{ty}_0, a_0, \mathsf{rc}_0) \neq (\mathsf{ty}_1, a_1, \mathsf{rc}_1) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Definition 2.4 (Commitment Hiding)**.** *For any* $\lambda \in \mathbb{N}$ *with* $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$ *and any stateful PPT adversary* $\mathcal{A}$*, it holds that*

$$\Pr\left[\begin{array}{l} \{(\mathsf{ty}_i, a_i)\}_{i=0}^1 \leftarrow \mathcal{A}(\mathsf{p}); \\ b \xleftarrow{\$} \{0,1\}, \mathsf{rc} \xleftarrow{\$} \mathbb{R}; \\ \mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{ty}_b, a_b, \mathsf{rc}); \\ b' \leftarrow \mathcal{A}(\mathsf{com}) \ : \\ \quad b' = b \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

To instantiate a SHC we use a well-known Pedersen-based construction derived from the [21], which we will simply call "sparse Pedersen commitment scheme". Let $\mathbb{G}$ be a cyclic group of order q with a generator G in which the discrete logarithm assumption holds. Additionally, we use a cryptographic hash function $H : \mathbb{T} \to \mathbb{G}$. Then $\mathsf{Commit}(\{(\mathsf{ty}_i, a_i)\}, \mathsf{rc}) := (\prod H(\mathsf{ty}_i)^a)\, G^{\mathsf{rc}}$ with $a_i, \mathsf{rc} \in \mathbb{Z}_q$.

$\underline{\text{HID-OR}_{\mathcal{A}}^b(1^\lambda)}$

$\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$
$\{(\mathsf{ty}_t, a_t), (\mathsf{ty}'_t, a'_t)\}_{t=0}^1 \leftarrow \mathcal{A}(\mathsf{p})$
$\Delta_{\mathsf{ty},t} := \{a_t \text{ if } \mathsf{ty}_t = \mathsf{ty} \text{ else } 0\} - \{a'_t \text{ if } \mathsf{ty}'_t = \mathsf{ty} \text{ else } 0\}$
assert $\forall \mathsf{ty} \in \bigcup \mathsf{ty}_t \cup \bigcup \mathsf{ty}'_t : \Delta_{\mathsf{ty},0} = \Delta_{\mathsf{ty},1}$
$r, r' \xleftarrow{\$} \mathbb{R}$
$\mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{ty}_b, a_b, r);\ \mathsf{com}' \leftarrow \mathsf{Commit}(\mathsf{ty}'_b, a'_b, r')$
$b' \leftarrow \mathcal{A}(\mathsf{com}, \mathsf{com}', \mathsf{rc} = r - r')$
return b'

Fig. 1. Hiding with Open Randomness Game

Finally, we observe the following property of the sparse Pedersen commitment scheme that guarantees that the adversary cannot distinguish between two pairs of commitments that sum to the same values (per type), even if we reveal their common randomness. This is the main property used in our swaps: we will use it to argue that transaction can be merged (to join two unbalanced swap offers), but not split apart (so swap offers cannot be adversarially modified).

Lemma 2.1 (Hiding with Open Randomness)**.** *The sparse Pedersen commitment scheme is* perfectly hiding with open randomness *in the RO model. By this we mean that if for all* $\lambda \in \mathbb{N}$ *and all PPT* $\mathcal{A}$*:*

$$\Pr[\mathit{HID\text{-}OR}_{\mathcal{A}}^1(1^\lambda) = 1] - \Pr[\mathit{HID\text{-}OR}_{\mathcal{A}}^0(1^\lambda) = 0] = 0$$

where the game is defined in Fig. 1.

The proof of this statement is presented in Appendix C and based on perfect hiding together with the equivocation of the Pedersen commitment scheme.

2.2 Proofs and Signatures of Knowledge

A *non-interactive zero-knowledge proof of knowledge* (or just NIZK) for the language $\mathcal{L}$ is a proving system consisting of algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$. The first algorithm allows to create a common reference string (CRS), $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ that is used as an input to Prove and Verify. To obtain a non-interactive proof that $\mathsf{stmt} \in \mathcal{L}$ with the corresponding witness w (s.t. $(\mathsf{stmt}, w) \in \mathcal{R}_{\mathcal{L}}$) one executes $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{stmt}, w)$. To then verify this prove one runs, correspondingly $\mathsf{Verify}(\mathsf{crs}, \mathsf{stmt}, \pi)$, which returns 0 or 1. We provide the standard definitions and security properties in Appendix A.

Signatures of Knowledge are NIZK proofs which are bound to a specific message m. For a simulation extractable (SE) NIZK protocol, this is done by including

m into the statement without asserting anything about m in the relation. Still, the SE property invalidates the proof, if the statement is changed.

3 One-Time-Account Scheme

To highlight our novel transaction mechanism, we first describe the *one-time account (OTA) scheme* that we use to model the mechanics of underlying accounts our transactions use. The OTA scheme may be seen as an anonymous version of an unspent transaction output (UTXO) system (e.g. the one used in Bitcoin); it generalises accounts of privacy-preserving transaction systems such as Zcash or Monero. Instead of creating transaction outputs including a long term identity as with plain UTXO, an OTA scheme generates a unique, anonymous, one-time account for each transaction output. To support the UTXO functionality to decide if a one-time-account is still valid, the OTA scheme allows generating a unique *nullifier* which anonymously marks it as spent. A duplicate nullifier indicates that the same one-time account is used.

The OTA scheme is used as follows. After a system Setup, each participant generates their credentials with KeyGen. Anyone with the public key can then non-interactively derive a one-time account, *also called "note"*, with Gen. Each note contains a set vector of attributes. Most importantly, one attribute is the amount the note represents. In our multi-type setting, a second attribute is the type of the amount. Other systems may make use of additional note attributes such as e.g. time locks — a timestamp identifier, allowing a note to be spendable only after a specified block number. To allow the intended recipient to recover the note, it is accompanied by an encryption C calculated by Enc. With the secret key, note and ciphertext, the original owner $\mathsf{Receives}$ the note and is then able to create the corresponding nullifier with $\mathsf{NulEval}$, a unique serial value characterizing the note that cannot be predicted by anyone else than the owner.

We emphasize that any party can create an OTA account for anyone knowing only their public key. The OTA itself does not allow any claims to the value stored inside. In our system, only the OTAs which are included as outputs of a valid, balanced and persisted transaction can be claimed by their owners in subsequent transactions. The transaction then enforces that sufficient inputs were consumed to cover for the output. To mint coins an OTA needs to be included on the ledger in an unbalanced transaction for the newly minted type. This transaction needs to be accepted by the ledger rules. This behaviour might as well be governed by a smart contract.

Definition 3.1. *A One-Time-Account (OTA) scheme consists of the five PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Receive}, \mathsf{NulEval})$ *defined as follows:*

$\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$**:** *takes the security parameter λ and outputs the public parameters p which are implicitly provided to the subsequent algorithms. This includes the note randomness space $\mathbb{S}$, a message space $\mathbb{M}$, and an encryption randomness space Ξ.*

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$**:** *generates a key pair $(\mathsf{sk}, \mathsf{pk})$. We assume a function $P(\mathsf{sk}) = \mathsf{pk}$ for generating the public key from a secret key.*

$\mathsf{note} \leftarrow \mathsf{Gen}(\mathsf{pk}, \vec{a}, r)$**:** *takes a public key pk, a vector of attributes $\vec{a} \in \mathbb{M}^{|\vec{a}|}$, where, by agreement, the first might be an amount, and randomness $r \in \mathbb{S}$. It outputs a one-time-account, known as note.*

$C \leftarrow \mathsf{Enc}(\mathsf{pk}, (\vec{a}, r), \xi)$**:** *encrypts the attributes $\vec{a}$ and the randomness r to the public key pk with additional randomness ξ. It outputs a ciphertext C.*

$(\vec{a}, r)/\perp \leftarrow \mathsf{Receive}(\mathsf{note}, C, \mathsf{sk})$**:** *if the note and ciphertext C belongs to the secret key sk, the algorithm outputs the vector of attributes $\vec{a}$ and the randomness r or fails otherwise.*

$\mathsf{nul} \leftarrow \mathsf{NulEval}(\mathsf{sk}, r)$**:** *Takes a secret key sk and a randomness r and outputs a nullifier nul.*

In addition, the algorithms Gen *and* $\mathsf{NulEval}$ *must be efficiently provable in zero-knowledge. More formally, the construction must provide a NIZK protocol for the following languages:*

$$\mathcal{L}^{\mathsf{nul}} = \{(\mathsf{note}, \mathsf{nul}) \mid \exists(\mathsf{sk}, \vec{a}, r) : \mathsf{note} = \mathsf{Gen}(P(\mathsf{sk}), \vec{a}, r) \wedge \mathsf{nul} = \mathsf{NulEval}(\mathsf{sk}, r)\}$$

$$\mathcal{L}^{\mathsf{open}} = \{\mathsf{note} \mid \exists(\mathsf{pk}, \vec{a}, r) : \mathsf{note} = \mathsf{Gen}(\mathsf{pk}, \vec{a}, r)\}$$

The language $\mathcal{L}^{\mathsf{open}}$ may optionally be extended, such that elements of $\vec{a}$ have relations to other commitments.

Definition 3.2 (OTA Correctness)**.** *An OTA scheme is correct if for any $\lambda \in \mathbb{N}$ with $\mathsf{p} \in \mathsf{Setup}(1^\lambda)$ it holds that any honestly generated note is receivable. Formally, for every $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{KeyGen}(\mathsf{p})$, every $(\vec{a}, r) \in \mathbb{M}^{|\vec{a}|} \times \mathbb{S}$ and $\xi \in \Xi$ it holds that* $\mathsf{Receive}(\mathsf{Gen}(\mathsf{pk}, \vec{a}, r), \mathsf{Enc}(\mathsf{pk}, (\vec{a}, r), \xi), \mathsf{sk}) = (\vec{a}, r)$

Once a note is created, it must bind the attributes and prevent opening the note to a different vector, even for the owner with the correct secret key.

Definition 3.3 (OTA Binding)**.** *An OTA scheme is binding with regard to the accounts created and the vector of attributes if for any* $\lambda \in \mathbb{N}$ *with* $\mathsf{p} \in \mathsf{Setup}(1^\lambda)$ *and any PPT adversary* $\mathcal{A}$*, it holds that*

$$\Pr\left[\begin{array}{l}(\mathsf{pk}_0, r_0, \vec{a}_0, \mathsf{pk}_1, r_1, \vec{a}_1) \leftarrow \mathcal{A}(\mathsf{p}) \\ \mathsf{note}_0 \leftarrow \mathsf{Gen}(\mathsf{pk}_0, \vec{a}_0, r_0) \\ \mathsf{note}_1 \leftarrow \mathsf{Gen}(\mathsf{pk}_1, \vec{a}_1, r_1) \ : \\ \quad \mathsf{note}_0 = \mathsf{note}_1 \wedge \vec{a}_0 \neq \vec{a}_1\end{array}\right] \leq \mathsf{negl}(\lambda)$$

The following privacy property assures that a note and its ciphertext do not leak who the note belongs to and what attributes it stores.

Definition 3.4 (OTA Privacy)**.** *Consider the following oracle:*

$$\underline{\mathcal{O}^{\mathsf{note}^*, C^*}_{\mathsf{Rcv}}(i, \mathsf{note}, C)}:$$
$$\text{assert } \mathsf{note} \neq \mathsf{note}^* \wedge C^* \neq C$$
$$\text{return } \mathsf{Receive}(\mathsf{note}, C, \mathsf{sk}_i)$$

An OTA scheme is private if for any $\lambda \in \mathbb{N}$ *with* $\mathsf{p} \in \mathsf{Setup}(1^\lambda)$ *and any stateful PPT adversary* $\mathcal{A}$*, it holds that*

$$\Pr\left[\begin{array}{l}(\mathsf{sk}_0, \mathsf{pk}_0) \xleftarrow{\$} \mathsf{KeyGen}() \\ (\mathsf{sk}_1, \mathsf{pk}_1) \xleftarrow{\$} \mathsf{KeyGen}() \\ (i_0, \vec{a}_0, i_1, \vec{a}_1) \leftarrow \\ \qquad \mathcal{A}^{\mathcal{O}^{\perp,\perp}_{\mathsf{Rcv}}}(\mathsf{p}, \mathsf{pk}_0, \mathsf{pk}_1) \\ b \xleftarrow{\$} \{0,1\}, r \xleftarrow{\$} \mathbb{S}, \xi \xleftarrow{\$} \Xi \\ \mathsf{note}^* \leftarrow \mathsf{Gen}(\mathsf{pk}_{i_b}, \vec{a}_b, r) \\ C^* \leftarrow \mathsf{Enc}(\mathsf{pk}_{i_b}, (\vec{a}_b, r), \xi) \\ b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{note}^*, C^*}_{\mathsf{Rcv}}}(\mathsf{note}^*, C^*) \ : \\ \quad b = b' \wedge |\vec{a}_0| = |\vec{a}_1|\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

The privacy game implicitly subsumes (1) note and ciphertext hiding, and (2) note and encryption anonymity (key privacy). In the first case, $\mathcal{A}$ cannot decide the content of the note or the ciphertext; in the second, it cannot decide which key was used to create it. E.g. if note is not hiding, $\mathcal{A}$ efficiently wins the game by first returning two different vectors $\vec{a}_0 \neq \vec{a}_1$ and then distinguishing the note note_b according to the attribute vector.

We require notes to be *unique* when generated with honest randomness:

Definition 3.5 (Note Uniqueness)**.** *A binding and private OTA scheme satisfies honestly generated note uniqueness: for all PPT* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{l}((\mathsf{pk}_0, \vec{a}_0), (\mathsf{pk}_1, \vec{a}_1)) \leftarrow \mathcal{A}(1^\lambda) \\ r_0, r_1 \xleftarrow{\$} \mathbb{S} \ : \\ \mathsf{Gen}(\mathsf{pk}_0, \vec{a}_0, r_0) = \mathsf{Gen}(\mathsf{pk}_1, \vec{a}_1, r_1)\end{array}\right] \leq \mathsf{negl}(\lambda)$$

The next property, similar to the tagging scheme of Omniring [19], captures the requirement that nullifiers produced with sk must be only "predictable" for the party that holds sk. This is achieved by requiring NulEval to behave like a pseudorandom function.

Definition 3.6 (Nullifier Pseudorandomness)**.** *Let* $\lambda \in \mathbb{N}$ *with* $\mathsf{p} \in \mathsf{Setup}(1^\lambda)$*,* $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{KeyGen}()$*, and* f *be randomly sampled function on the range* $\{0,1\}^{|\mathbb{S}|} \to \{0,1\}^{|\mathsf{NulEval}(\mathsf{sk},\cdot)|}$*. An OTA scheme nullifier is* pseudorandom *if for any PPT adversary* $\mathcal{A}$*, it holds that*

$$\Pr[\mathcal{A}^{\mathsf{Receive}(\cdot,\cdot,\mathsf{sk}), f(\cdot)}(\mathsf{pk}) = 1] - \Pr[\mathcal{A}^{\mathsf{Receive}(\cdot,\cdot,\mathsf{sk}), \mathsf{NulEval}(\mathsf{sk},\cdot)}(\mathsf{pk}) = 1] \leq \mathsf{negl}(\lambda)$$

Finally, to detect duplicate use of the same note, each is assigned a unique nullifier. Even with knowledge of the secret key, it is not possible to create two different nullifiers for the same note. The separate secret keys are important for constructions based on algebraic nullifiers. E.g. Omniring creates tags as $g^{\frac{1}{\mathsf{sk}+r}}$ for a generator g, so randomness may be "traded" for secret key. Allowing only one note with a single secret key would not capture the realistic setting where an adversary controls multiple correlated accounts.

Definition 3.7 (Nullifier Uniqueness)**.** *An OTA scheme satisfies nullifier uniqueness if for any* $\lambda \in \mathbb{N}$ *with* $\mathsf{p} \in \mathsf{Setup}(1^\lambda)$ *and any PPT adversary* $\mathcal{A}$*, it holds that*

$$\Pr\left[\begin{array}{l}(\mathsf{sk}_0, r_0, \vec{a}_0, \mathsf{sk}_1, r_1, \vec{a}_1) \leftarrow \mathcal{A}(\mathsf{p}); \\ \mathsf{note}_0 \leftarrow \mathsf{Gen}(P(\mathsf{sk}_0), \vec{a}_0, r_0); \\ \mathsf{note}_1 \leftarrow \mathsf{Gen}(P(\mathsf{sk}_1), \vec{a}_1, r_1) \ : \\ \ \mathsf{note}_0 = \mathsf{note}_1 \ \wedge \\ \ \mathsf{NulEval}(\mathsf{sk}_0, r_0) \neq \mathsf{NulEval}(\mathsf{sk}_1, r_1)\end{array}\right] \leq \mathsf{negl}(\lambda)$$

4 Zswap Scheme

A Zswap scheme is an extension of an OTA scheme which allows creating (SignTx), merging (MergeSig), and verifying (Verify) transactions that transfer coins between OTA accounts. SignTx takes a pre-transaction ptx as input and produces a transaction signature σ, MergeSig combines transaction signatures $\sigma_1, \ldots, \sigma_n$, and Verify verifies a signature σ for a transaction tx. A signature σ is viewed separately from its transaction tx (created by $\mathsf{tx} \leftarrow \mathsf{CompleteTx}(\mathsf{ptx})$ defined below), and not contained in it.

We first present auxiliary algorithms for Zswap in Fig. 2 for creating pre-transactions and transactions. These use the OTA scheme algorithms only in a black-box manner and simplify the exposition when defining the security properties. The first set of functions glues the Zswap and OTA schemes together.

- BuildPTx constructs inputs for SignTx from I, O instructions and a set of secret keys SK. Inputs I consist of a list of existing (note, C) notes, outputs O consist of a list of public key, value, and type triplets $(\mathsf{pk}^{\mathcal{T}}, a^{\mathcal{T}}, \mathsf{ty}^{\mathcal{T}})$ for creating the output notes. By receiving input notes and generating output notes it produces a pre-transaction information ptx.
- CompleteTx constructs a transaction tx for Verify from a pre-transaction ptx similar to SignTx.
- MergeTx creates a new transaction tx by combining a set of existing transactions $\mathsf{tx}_1, \ldots, \mathsf{tx}_n$. This is the non-cryptographic analogue of MergeSig.
- CheckPTx(ptx, st) checks whether pre-transaction is valid w.r.t. st, which is used in the correctness property. It is easy to see that for $\mathsf{ptx} = \mathsf{BuildPTx}(\cdot, \cdot, \mathsf{st}, \cdot)$ we have $\mathsf{CheckPTx}(\mathsf{ptx}, \mathsf{st}) = 1$.
- TryReceive attempts to "receive" a note note by decrypting its ciphertext C using any of the set SK of available secret keys: if an input can be received with one of the keys it also computes the note's nullifier.

Definition 4.1. *A Zswap transaction scheme, built on top of an OTA scheme, consists of a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{SignTx}, \mathsf{MergeSig}, \mathsf{Verify})$ *defined as follows:*

$\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$ *takes the security parameter* λ *and outputs public parameters* p *which are implicitly given to all the following algorithms. Setup is called once when a Zswap system is initialized.*

$\sigma \leftarrow \mathsf{SignTx}(\mathsf{ptx}, \mathsf{st})$ *takes a pre-transaction* $\mathsf{ptx} = (\mathcal{S}, \mathcal{T})$ *where*

- st *is the current state of valid previously issued notes.* st.MT *is the Merkle tree containing notes as leaves, and* st.NF *is the set of used nullifiers. If a state* st *is the evolution of a previous state* st′*, we define a function* $\mathsf{ev}(\mathsf{st}', \mathsf{st}) := 1$ *and* 0 *otherwise.*
- $\mathcal{S} = \{(\mathsf{sk}_i^{\mathcal{S}}, \mathsf{note}_i^{\mathcal{S}}, \mathsf{nul}_i, \mathsf{path}_i, (a_i^{\mathcal{S}}, \mathsf{ty}_i^{\mathcal{S}}), r_i^{\mathcal{S}})\}_{i=1}^{|\mathcal{T}|}$ *is a set of inputs with a nullifier* nul_i *corresponding to the* $\mathsf{note}_i^{\mathcal{S}}$*, and stored in the current state at the given path* $\mathsf{st.MT}[\mathsf{path}_i]$*, secret key* $\mathsf{sk}_i^{\mathcal{S}}$*, amount* $a_i^{\mathcal{S}}$ *and type* $\mathsf{ty}_i^{\mathcal{S}}$ *with input OTA notes' randomness* $r_i^{\mathcal{S}}$*.*
- $\mathcal{T} = \{(\mathsf{pk}_i^{\mathcal{T}}, \mathsf{note}_i^{\mathcal{T}}, (a_i^{\mathcal{T}}, \mathsf{ty}_i^{\mathcal{T}}), r_i^{\mathcal{T}})\}_{i=1}^{|\mathcal{T}|}$ *is a set of (output) notes* $\mathsf{note}_i^{\mathcal{T}}$ *with amount* $a_i^{\mathcal{T}}$*, type* $\mathsf{ty}_i^{\mathcal{T}}$ *and output OTA notes' randomness* $r_i^{\mathcal{T}}$*.*

It outputs a signature σ *as authorization to spend the inputs* $\mathcal{S}$ *on the given outputs* $\mathcal{T}$*.*

$b \leftarrow \mathsf{Verify}(\mathsf{st}, \mathsf{tx}, \sigma)$ *takes a transaction* tx*, a signature* σ *and returns a bit* b *representing the validity of the transaction w.r.t. the state* st*.*

$\sigma \leftarrow \mathsf{MergeSig}(\{\sigma_i\}_{i=1}^n)$ *takes* n *transaction signatures and generates a combined signature* σ *valid for the union of the transactions. To merge the corresponding transactions* $\mathsf{tx}_1, \ldots, \mathsf{tx}_n$ *together, we use the function* MergeTx *defined in Fig. 2. It just concatenates their input nullifiers, output notes, and sums their* Δ_{ty} *for each* ty*. As of the* st*, they must be equal in the transactions that need to be merged, but we can assume that* st *is only updated once an epoch, which is set to be a time interval long enough for transactions to be merged.*

4.1 Atomic Swap Example

After presenting all necessary algorithms, we show a small example of how they interact to create an atomic swap transaction between two parties, Alice and Bob. First, the system is created by Setup. Each participant joining, generates their key pair (sk, pk) with KeyGen. The creation of new assets is delegated to an external consensus mechanism which updates the global state of the system according to an agreed policy. At one point, Alice and Bob have to be the beneficiary of a transaction. They notice this by calling TryReceive with their secret key sk on every published transaction output (note, C). If they successfully receive a (note, C) to an amount a and type ty, they keep it for when they want to spend it. Let's assume Alice received a note of 10$ and Bob has a note of 10€. Alice now wants euros and Bob dollars. They assume an exchange rate of 7:5 and proceed to generate their pre-transactions ptx_i. Each party calls BuildPTx with their note as input instruction I. As output instructions O, Alice sends 5€ to her key pk. To make sure that someone fulfils the offer, she creates a change output to herself with only 2$, leaving 1$ as incentive. Bob performs the same. He inputs the 10€ note and generates outputs of 7$ and 5€ to himself.

The resulting pre-transactions are signed by both parties respectively with SignTx to get signatures σ_i. The final transactions tx_i are generated from the pre-transactions ptx_i by CompleteTx and can be verified against their signature σ_i with Verify. Note that both

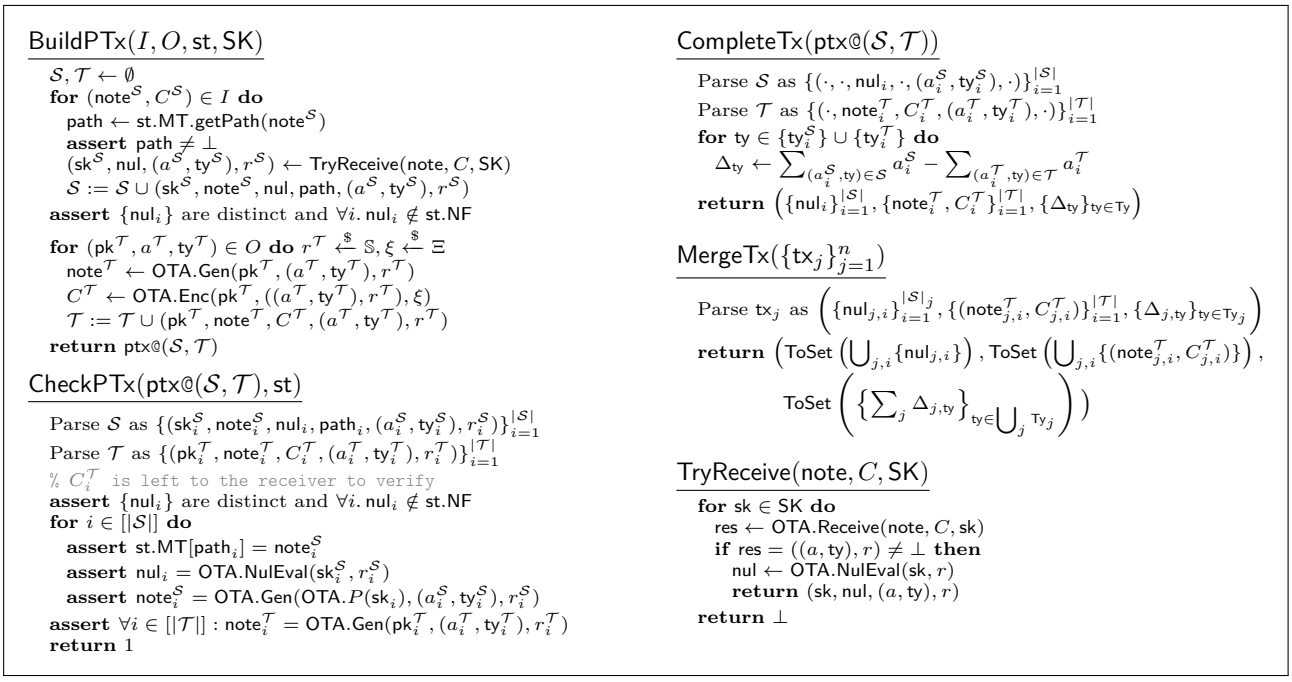

Fig. 2. Auxiliary algorithms for Zswap. These only depend on the OTA scheme and do not use Zswap methods such as Verify or SignTx.

transactions by themselves cannot be included in the public ledger. Both have a type with a negative balance. Alice's transaction has $\Delta_€ = -5$ and Bob's transaction has an imbalance of $\Delta_\$ = -7$. So far Alice and Bob have not communicated. The non-malleability of the transactions allow them to publish their transactions into an exchange pool. Exchange pools may be run globally or with limited access. The first party seeing both transactions recognizes that they are complementary and is able to merge both tx_i together with MergeTx and their signatures σ_i with MergeSig. As a prize, the merger is allowed to claim the surplus of 1$ paid by Alice. Technically there are now 3 transactions merged. The resulting merged transaction has no imbalance and can be included in the public ledger. Then all parties get their specified outputs and store them for future transactions.

4.2 Zswap Correctness

Definition 4.2 (Correctness)**.** *A Zswap scheme is correct if OTA is correct and if for all* $\lambda \in \mathbb{N}$ *and all* $\mathsf{p} \in \mathsf{Setup}(1^\lambda)$ *it holds that:*

(1) Honestly generated transactions are valid: *For any* $\mathcal{S}, \mathcal{T}, \mathsf{st}$ *such that* $\mathsf{CheckPTx}(\mathcal{S}, \mathcal{T}, \mathsf{st}) = 1$*, and* $\mathsf{tx} = \mathsf{CompleteTx}(\mathcal{S}, \mathcal{T})$*, and for any signature* $\sigma \leftarrow \mathsf{SignTx}(\mathcal{S}, \mathcal{T}, \mathsf{st})$*, it holds that* $\mathsf{Verify}(\mathsf{st}, \mathsf{tx}, \sigma) = 1$.

(2) Honestly merged valid transactions are again valid: *For any valid* st, $n \in \mathbb{N}$ *and each set of* $\{(\mathcal{S}_i, \mathcal{T}_i)\}_{i\in[n]}$ *s.t. for all* i, $\mathsf{CheckPTx}((\mathcal{S}_i, \mathcal{T}_i), \mathsf{st}) = 1$*, for each* $\sigma_i \in \mathsf{SignTx}(\mathcal{S}_i, \mathcal{T}_i, \mathsf{st})$ *and* $\sigma = \mathsf{MergeSig}(\{\sigma_i\}_{i=1}^n)$*, it holds that* $\mathsf{Verify}(\mathsf{st}, \mathsf{tx}, \sigma) = 1$ *where* $\mathsf{tx} \leftarrow \mathsf{MergeTx}(\{\mathsf{CompleteTx}(\mathcal{S}_i, \mathcal{T}_i)\}_{i=1}^n)$.

(3) Valid transactions are valid in a future state: *For any two linked states* $\mathsf{st}, \mathsf{st}'$ *(satisfying* $\mathsf{ev}(\mathsf{st}, \mathsf{st}') = 1$*) and any transaction* tx *and signature* σ *with* $\mathsf{Verify}(\mathsf{st}, \mathsf{tx}, \sigma) = 1$*, it holds that* $\mathsf{Verify}(\mathsf{st}', \mathsf{tx}, \sigma) = 1$*, unless* $\mathsf{st}'.\mathsf{Nf} \setminus \mathsf{st}.\mathsf{Nf}$ *contains any of the* tx *nullifiers.*

The security of our scheme is based on several top-level games, plus the games we define for the OTA scheme.

4.3 Support Oracles

To model both security and privacy we need to introduce the oracles which will model note spending that the adversary $\mathcal{A}$ sees, interactively. The relevant oracles, $\mathcal{O}_{\mathsf{KeyGen}}$, $\mathcal{O}_{\mathsf{Spend}}$ and $\mathcal{O}_{\mathsf{Insert}}$ are presented on Fig. 3. They are responsible for generating keys, honest transactions and inserting them into the ledger.

$\mathcal{O}_{\mathsf{KeyGen}}$ allows generating honest public and secret keys for further use in $\mathcal{O}_{\mathsf{Spend}}$.

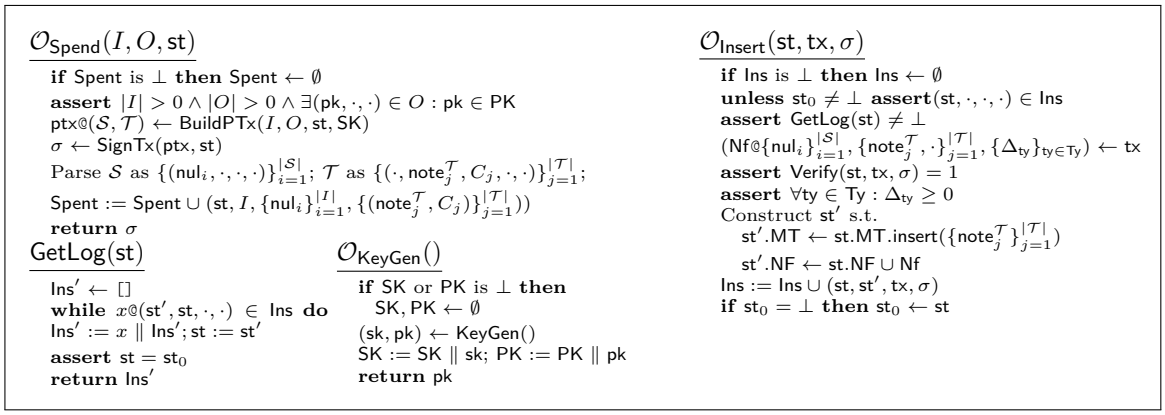

Fig. 3. Oracles for the security and privacy experiments.

$\mathcal{O}_{\mathsf{Spend}}$ models leakage of honestly generated transactions, including unbalanced "offer" transactions. Each $\mathcal{O}_{\mathsf{Spend}}$ query allows spending some notes, and successful spend logs are recorded in Spent. An adversary can specify any possible state it likes as long as the request is valid with respect to this state. The requirement that O has at least one honest output is a modelling artefact, and is explained further in the "anti-theft" section.

$\mathcal{O}_{\mathsf{Insert}}$ models "recording" balanced transaction in the ledger. Branching is possible inside $\mathcal{O}_{\mathsf{Insert}}$ — $\mathcal{A}$ can append an honest transaction to any of the recorded states, but still one can only spend notes through $\mathcal{O}_{\mathsf{Insert}}$ if their nullifier has not been used in the same branch. Moreover, the st_0 variable is shared between the oracle records as the first "root" state that $\mathcal{A}$ can initialize the oracles with. $\mathcal{O}_{\mathsf{Insert}}$ only accepts states that are eventually linked with this root st_0.

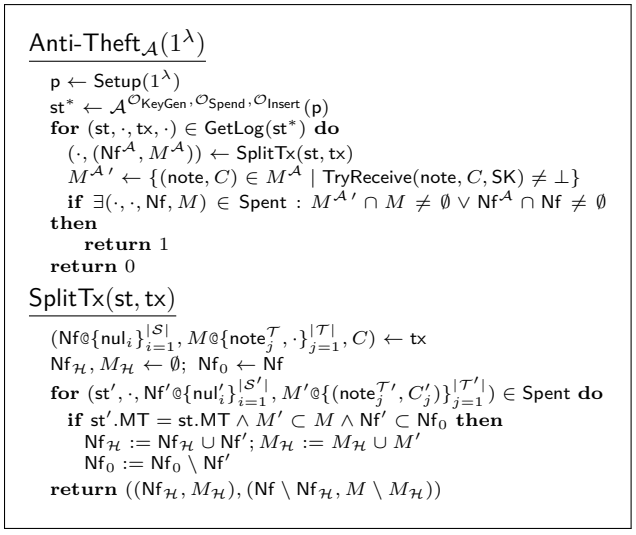

Fig. 4. The anti-theft experiment.

4.4 Anti-Theft and Non-Malleability

The notion of anti-theft we describe here is our main non-malleability notion. Intuitively it says that an adversary $\mathcal{A}$ can only merge transactions, but cannot break honest transactions apart, or modify them in any other way. More concretely, for any honest tx that $\mathcal{A}$ sees, $\mathcal{A}$ cannot submit tx^* which contains any of the (1) input nullifiers of tx; or (2) honest output notes of tx; without merging the whole tx into tx^*.

The anti-theft game is modelled by allowing $\mathcal{A}$ to interact with the oracles described before: $\mathcal{A}$ can generate new keys, produce unbalanced honest transactions (spend), and submit (insert) transactions (which we want to prove to be only honest, dishonest, or merges of the two types). The adversary then returns a state st^* which is examined. The challenger searches through the log of inserted transactions. The adversary wins if a transaction in the log contains an incomplete part of (an honest) transaction earlier returned by $\mathcal{O}_{\mathsf{Spend}}$ to $\mathcal{A}$, but *used* only partially. To do that, it calls a sub-function SplitTx which locates complete "honest subtransactions" returned by $\mathcal{O}_{\mathsf{Spend}}$ in tx, and returns these honest subtransactions as first argument, and the remaining parts of tx^*. If the challenger recognizes nullifiers or output notes in this remaining part, that were created in $\mathcal{O}_{\mathsf{Spend}}$, it means that $\mathcal{A}$ managed to deconstruct it, changing the output of $\mathcal{O}_{\mathsf{Spend}}$, or stealing an honest nullifier.

Definition 4.3 (Anti-Theft)**.** *A Zswap scheme is protecting against theft, if for any $\lambda \in \mathbb{N}$ and any PPT adversary $\mathcal{A}$,* $\Pr[\mathsf{Anti\text{-}Theft}_{\mathcal{A}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda)$ *where the* $\mathsf{Anti\text{-}Theft}_{\mathcal{A}}(1^\lambda)$ *game is defined in Fig. 4.*

Essentially, anti-theft captures non-malleability of output notes and their ciphertexts, which is in practice guaranteed by NIZK SE (that implies instance binding). If $\mathcal{A}$ can maul an output (e.g. change its value, or destination) and still submit the tx to $\mathcal{O}_{\mathsf{Insert}}$ successfully, it wins the game, since $\mathsf{SplitTx}$ will not locate tx as being in Spent, and thus the game challenger $\mathcal{C}$ will catch $\mathcal{A}$ spending a nullifier that was in Spent but not located by $\mathsf{SplitTx}$.

The requirement that O in $\mathcal{O}_{\mathsf{Spend}}$ contains at least one honest output is needed so that $\mathsf{SplitTx}$ can uniquely identify honest sub-transactions[7]. In other words, note uniqueness is necessary for the anti-theft game to make sense. This does not limit $\mathcal{A}$ from creating unbalanced input-only transactions, since $\mathcal{A}$ can still request to include a single zero-valued output.

The set Nf_0 in $\mathsf{SplitTx}$ is needed since without it honest sub-transactions can be counted twice. E.g. let $\mathsf{tx}_0, \mathsf{tx}_1$ be two honest transactions with the same input nullifiers Nf, but completely different outputs M_0, M_1 (each containing at least one honest output note). Without Nf_0, for tx containing Nf and $M_0 \cup M_1$, $\mathsf{SplitTx}$ will detect both $\mathsf{tx}_0, \mathsf{tx}_1$, and the winning condition will not be triggered. We, on the other hand, want anti-theft to prevent this: with Nf_0 one of tx_i will be considered honest, and the honest output note in tx_{1-i} will trigger the anti-theft winning condition. We do not need to similarly count M since output notes are unique — adding $M = M \setminus M'$ into $\mathsf{SplitTx}$ simply does not change the behaviour of the game.

Finally, it is critical to filter $M^{\mathcal{A}\prime}$ from $M^{\mathcal{A}}$. Otherwise, $\mathcal{A}$ would be able to trivially win the game by triggering $M^{\mathcal{A}} \cap M \neq 0$ in the following way:

1. $\mathcal{A}$, through $\mathcal{O}_{\mathsf{Spend}}$, requests an honest spend to an adversarial public key $\mathsf{pk}^{\mathcal{A}}$;
2. $\mathcal{A}$ obtains the proof and corresponding note^*, receives it, and creates a completely different, adversarial transaction tx^* on its own, where note^* is an output;
3. the game will not find any honest subtransaction in tx^* and thus $\mathsf{note}^* \in M \cap M^{\mathcal{A}}$.

7 Without this requirement, and without adding extra marker information to notes, identifying honest sub-transactions within $\mathsf{SplitTx}$ would take exponential time.

$\mathsf{Balance}_{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda)$

$\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{st}, I_0, I, \mathsf{SK}^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KeyGen}}, \mathcal{O}_{\mathsf{Spend}}, \mathcal{O}_{\mathsf{Insert}}}(\mathsf{p})$
assert $\{\mathsf{note}_{0,i}\} \in I_0$ are distinct, and $\mathsf{st}_0.\mathsf{MT}$ contains only these notes.
assert $\{\mathsf{note}_i\} \in I$ are distinct, in $\mathsf{st}.\mathsf{MT}$
for $(\mathsf{note}_i, C_i) \in I_0$ **do**
  $(\cdot, \mathsf{nul}, (a, \mathsf{ty}), \cdot) \leftarrow \mathsf{TryReceive}(\mathsf{note}_i, C_i, \mathsf{SK}^*)$
  assert $\mathsf{nul} \notin \mathsf{st}_0.\mathsf{NF}$
  $v_0[\mathsf{ty}] := v_0[\mathsf{ty}] + a$
for $(\mathsf{note}_i, C_i) \in I$ **do**
  $(\cdot, \mathsf{nul}, (a, \mathsf{ty}), \cdot) \leftarrow \mathsf{TryReceive}(\mathsf{note}_i, C_i, \mathsf{SK}^*)$
  assert $\mathsf{nul} \notin \mathsf{st}.\mathsf{NF}$
  $v_{\mathcal{A}}[\mathsf{ty}] := v_{\mathcal{A}}[\mathsf{ty}] + a$
$v_{\mathcal{H}-}, v_{\mathcal{H}+} \leftarrow (\mathsf{ty} \mapsto 0)$ % map with default value 0
$\mathsf{Ins}' \leftarrow \mathsf{GetLog}(\mathsf{st})$ % Also makes sure $\mathsf{ev}(\mathsf{st}_0, \mathsf{st})$
for all $(\mathsf{st}, \cdot, \mathsf{tx}, \sigma) \in \mathsf{Ins}'$ **do**
  $(\{\mathsf{nul}_i\}_{i=1}^{|S|}, \{\mathsf{note}_j^{\mathcal{T}}, \cdot\}_{j=1}^{|\mathcal{T}|}, \{\Delta_{\mathsf{ty}}\}_{\mathsf{ty}\in\mathsf{Ty}}) \leftarrow \mathsf{tx}$
  $(\mathsf{tx}_{\mathcal{H}}, \cdot) \leftarrow \mathsf{SplitTx}(\mathsf{st}, \mathsf{tx})$
  for $(\mathsf{Nf}, M) \in \mathsf{tx}_{\mathcal{H}}$ **do**
    Find $(\mathsf{st}, I, \mathsf{Nf}, M) \in \mathsf{Spent}$
    for $(\mathsf{note}', C') \in I$ **do**
      $(\cdot, (a, \mathsf{ty}), \cdot) \leftarrow \mathsf{TryReceive}(\mathsf{note}', C', \mathsf{SK})$
      $v_{\mathcal{H}-}[\mathsf{ty}] := v_{\mathcal{H}-}[\mathsf{ty}] + a$
  for $(\cdot, M) \in \mathsf{tx}$ **do**
    for $(\mathsf{note}', C') \in M$ **do**
      $\mathsf{res} \leftarrow \mathsf{TryReceive}(\mathsf{note}', C', \mathsf{SK})$
      if $\mathsf{res} = (\cdot, (a, \mathsf{ty}), \cdot)$ **then** $v_{\mathcal{H}+}[\mathsf{ty}] := v_{\mathcal{H}+}[\mathsf{ty}] + a$
if $\exists \mathsf{ty} \in T : v_{\mathcal{A}}[\mathsf{ty}] > v_0[\mathsf{ty}] + v_{\mathcal{H}-}[\mathsf{ty}] - v_{\mathcal{H}+}[\mathsf{ty}]$ **then**
  return 1
else return 0

Fig. 5. The balance experiment

4.5 Balance

The balance property presented next captures that transactions distribute underlying coins properly, that is, the adversarial balance per type only changes "predictably", by the adversary receiving or sending these coins. In particular, the balance game forbids malicious conversion between asset types, coin forging, and double spending.

Formally, the property is modelled as a game (Fig. 5) of $\mathcal{A}$ "against" the honest parties, in which $\mathcal{A}$ has to prove that it has more coins that it could have obtained honestly. It goes as follows. The challenger allows $\mathcal{A}$ to interact with the oracles, and asks $\mathcal{A}$ to present two sets of values: I_0 and I, together with a set of secret keys SK^*. Unlike in other games, st_0 in balance must contain only adversarial coins, and I_0 must be all the unspent corresponding notes (receivable with SK^*). The other set I is the set of unspent adversarial coins from the new state st (again receivable using SK^*) that $\mathcal{A}$ claims to have an illicitly produced coins, breaking the balance condition. This balance condition is computed next in the following manner. First, v_0 is set to the sum of all values in I_0, and $v_{\mathcal{A}}$ as a sum of all values in I. Then, by traversing the history of transactions from st_0 to st, the game computes the following two values:

1. $v_{\mathcal{H}-}$ is the sum of all honest inputs in transactions; and
2. $v_{\mathcal{H}+}$ is the total coins received by honest parties.

The final condition checks that $\mathcal{A}$ cannot show in st more coins than:

1. it had in st_0, and
2. that were sent by honest parties, minus received by honest parties.

This last difference may be both positive and negative, but importantly, the balance of $\mathcal{A}$, per type, is "tied" to the balance of honest parties, no extra coins can be produced.

Definition 4.4 (Balance)**.** *A Zswap scheme is balanced if for all PPT adversaries $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}_{\mathcal{A}}$ such that* $\Pr[\mathsf{Balance}_{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda)$ *with the game defined in Fig. 5.*

Note that this balance property does not prevent an external mechanism to change the state. This is useful for smar contract based minting of new types.

4.6 Privacy

The privacy game captures secrecy of coin transfers by means of an indistinguishability experiment. To model the game we will first introduce the notion of a transaction instruction tree T. Such a variable width tree has as its leaf i either

1. the instructions to construct an honest transaction $(\{\mathsf{note}_{i,j,C_{i,j}}\}_{j=1}^{|\mathcal{S}|_i}, \{(\mathsf{pk}_{i,j}, a^{\mathcal{T}}_{i,j}, \mathsf{ty}^{\mathcal{T}}_{i,j})\}_{j=1}^{|\mathcal{T}|_i})$, similarly to the input to $\mathcal{O}_{\mathsf{Spend}}$, or
2. a fully adversarial transaction $(\mathsf{tx}_i, \sigma_i)$.

Its intermediate leaves are empty, and merely represent how children transactions must be merged.

We will also need to decide what a transaction resulting from a merge according to T leaks. We formalize this notion by defining the tree equivalency relation, formally on Fig. 6 as follows: $\mathsf{EquivTree}(T_0, T_1) = 1$ if

1. The imbalance of amounts in each type is equal, B_1.
2. The number of input notes is equal (published nullifiers), and the number of output notes is equal. For honest $B_2^{\mathcal{S}} \wedge B_2^{\mathcal{T}}$ leaves, as it implicitly holds for adversarial leaves due to the next property B_3.
3. Malicious offers need to be the same in both trees, but maybe not at same positions, B_3.
4. For all the honest nullifiers Nf that $\mathcal{A}$ receives via $\mathcal{O}_{\mathsf{Spend}}$, if any of the related notes are included in the honest leaves of a tree, these notes must be included in both trees, B_4.
5. The adversarial output instructions of honest leaves are the same in both trees, B_5.

The privacy notion itself asks $\mathcal{A}$ to present two equivalent trees, and then builds a single transaction for each tree which both must not fail. It returns one merged transaction. The adversary wins the game if it can decide which tree was used. To illustrate the notion with a single example, imagine the tree to contain a single leaf node in both cases, in the first case spending a single note with X coins, sending 1 to Alice and $N-1$ to Bob, and in the second case spending a single note of Y coins, sending $Y-2$ to Alice and 2 to Bob. Such trees are equivalent according to our definition, and thus $\mathcal{A}$ should not be able to decide which transaction it sees.

Definition 4.5 (Transaction Privacy)**.** *A Zswap scheme has private transactions, if for all PPT adversaries $\mathcal{A}$ it holds that:*

$$\left|\Pr[\mathsf{Privacy}^0_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathsf{Privacy}^1_{\mathcal{A}}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

with $\mathsf{Privacy}^b_{\mathcal{A}}(1^\lambda)$ *defined in Fig. 6.*

5 Zswap Protocol

We present the Zswap protocol in Fig. 7. It extends the OTA scheme (constructed in Appendix B) and additionally utilizes a sparse homomorphic commitment scheme (sparse Pedersen): $\mathsf{SHC} = (\mathsf{ComSetup}, \mathsf{Commit})$. and Arguments of Knowledge: $\mathsf{AoK} = (\mathsf{AoK.Setup}, \mathsf{AoK.Prove}, \mathsf{AoK.Verify}, \mathsf{AoK.Sim})$.

The high-level idea is to create a transaction which has separate inputs and outputs handled by the OTA scheme. A transaction links them together through SHC commitments. Each input and output has a corresponding SHC commitment with an equal amount and type. This equivalency is assured by an AoK for each input and each output. The output AoK additionally assure non-malleability for the note ciphertext. The input AoKs enforce that the transaction creator possessed the secret key to authorize the spending and prove that the published nullifier is correct. This is captured by two NIZK languages.

The first one authenticates a valid spend — it says that the (rerandomized) SHC commitment $\mathsf{com}^{\mathcal{S}}$ is well-formed and contains the same value and type as a note

$\underline{\mathsf{Privacy}^b_{\mathcal{A}}(1^\lambda)}$

$\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathbb{O} = \{\mathcal{O}_{\mathsf{KeyGen}}, \mathcal{O}_{\mathsf{Spend}}, \mathcal{O}_{\mathsf{Insert}}\}$
$(\mathsf{st}, T_0, T_1) \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{p})$
assert $\mathsf{GetLog}(\mathsf{st}) \neq \bot$
$\mathsf{tx}_0 \leftarrow \mathsf{EvalTree}(\mathsf{st}, T_0)$
$\mathsf{tx}_1 \leftarrow \mathsf{EvalTree}(\mathsf{st}, T_1)$
assert $\mathsf{tx}_0 \neq \bot \wedge \mathsf{tx}_1 \neq \bot$
assert $\mathsf{EquivTree}(T_0, T_1, \mathsf{st}) = 1$
$b' \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{tx}_b)$
return b'

$\underline{\mathsf{EvalTree}(\mathsf{st}, T)}$

for all $\mathsf{leaf}_i \in T$ **do**
  if $\mathsf{leaf}_i = (\mathsf{tx}, \sigma)$ **then** % Validate adversarial txs
    assert $\mathsf{Verify}(\mathsf{st}, \mathsf{tx}, \sigma) = 1$
  % Replace instruction leaves with real txs
  if $\mathsf{leaf}_i = (I, O)$ **then**
    $\mathsf{ptx} \leftarrow \mathsf{BuildPTx}(I, O, \mathsf{st}, \mathsf{SK})$
    $\mathsf{tx}_i \leftarrow \mathsf{CompleteTx}(\mathsf{ptx})$
    $\sigma_i \leftarrow \mathsf{SignTx}(\mathsf{ptx}, \mathsf{st})$
    Replace leaf_i by $(\mathsf{tx}_i, \sigma_i)$ in T
% Fold T into a single root node by merging
while $\exists$ node N in T with only $\{(\mathsf{tx}_i, \sigma_i)\}_{i=1}^{c}$ as children **do**
  Replace node with merged transactions of its children:
  $N \leftarrow (\mathsf{MergeTx}(\{\mathsf{tx}_i\}_{i=1}^{c}, \mathsf{MergeSig}(\{\sigma_i\}_{i=1}^{c}))$
  Remove its children
return T

$\underline{\mathsf{EquivTree}(T_0, T_1, \mathsf{st})}$

Parse each $\mathsf{leaf}_{b,i}$ of each T_b as either an adversarial leaf $\mathsf{leaf}^{\mathcal{A}}$
  $(\mathsf{tx}_{b,i}@(\{\mathsf{nul}_{b,i,j}\}_{j=1}^{|\mathcal{S}|_{b,i}}, \{(\mathsf{note}^{\mathcal{T}}_{b,i,j}, C^{\mathcal{T}}_{b,i,j})\}_{j=1}^{|\mathcal{T}|_{b,i}},$
    $\{\Delta_{b,i,\mathsf{ty}}\}_{\mathsf{ty}\in\mathsf{Ty}_{b,i}}), \sigma_{b,i}) \leftarrow \mathsf{leaf}^{\mathcal{A}}_{b,i}$
  or an honest leaf $\mathsf{leaf}^{\mathcal{H}}$
  $(\{(\mathsf{note}^{\mathcal{S}}_{b,i,j}, C^{\mathcal{S}}_{b,i,j})\}_{j=1}^{|\mathcal{S}|_{b,i}}, \{(\mathsf{pk}_{b,i,j}, a^{\mathcal{T}}_{b,i,j}, \mathsf{ty}^{\mathcal{T}}_{b,i,j})\}_{j=1}^{|\mathcal{T}|_{b,i}}) \leftarrow \mathsf{leaf}^{\mathcal{H}}_{b,i}$
for $b \in \{0,1\}, \mathsf{leaf}^{\mathcal{H}}_i \in T_b$ **do**
  for $j \in [|\mathcal{T}_{b,i}|]$ **do**
    $(\mathsf{nul}_{b,i,j}, (a^{\mathcal{S}}_{b,i,j}, \mathsf{ty}^{\mathcal{S}}_{b,i,j}), r_{b,i,j}) \leftarrow \mathsf{TryReceive}(\mathsf{note}^{\mathcal{S}}_{b,i,j}, C^{\mathcal{S}}_{b,i,j}, \mathsf{SK})$
  for $\mathsf{ty} \in \{\mathsf{ty}^{\mathcal{S}}_{b,i,j}\}_{j=1}^{|\mathcal{T}|_{b,i}}$ **do**
    $\Delta_{b,i,\mathsf{ty}} \leftarrow \sum_{j:\mathsf{ty}_{b,i,j}=\mathsf{ty}} a^{\mathcal{S}}_{b,i,j} - \sum_{j:\mathsf{ty}_{0,i,j}=\mathsf{ty}} a^{\mathcal{T}}_{0,i,j}$
$B_1 \leftarrow \forall \mathsf{ty} : \sum_{\mathsf{leaf}^{\mathcal{H}}_i \in T_0} \Delta_{0,i,\mathsf{ty}} = \sum_{\mathsf{leaf}^{\mathcal{H}}_i \in T_1} \Delta_{1,i,\mathsf{ty}}$
$B^{\mathcal{S}}_2 \leftarrow \sum_{\mathsf{leaf}^{\mathcal{H}}_i \in T_0} |\mathcal{S}|_{0,i} = \sum_{\mathsf{leaf}^{\mathcal{H}}_i \in T_1} |\mathcal{S}|_{1,i}$
$B^{\mathcal{T}}_2 \leftarrow \sum_{\mathsf{leaf}^{\mathcal{H}}_i \in T_0} |\mathcal{T}|_{0,i} = \sum_{\mathsf{leaf}^{\mathcal{H}}_i \in T_1} |\mathcal{T}|_{1,i}$
$B_3 \leftarrow \bigcup_{\mathsf{leaf}^{\mathcal{A}}_i \in T_0} \{(\mathsf{tx}_{0,i}, \sigma_{0,i})\} = \bigcup_{\mathsf{leaf}^{\mathcal{A}}_i \in T_1} \{(\mathsf{tx}_{1,i}, \sigma_{1,i})\}$
$\mathsf{Nf} \leftarrow \bigcup_{(\cdot,\cdot,\mathsf{Nf}_i,\cdot)\in\mathsf{Spent}} \mathsf{Nf}_i$
$B_4 \leftarrow \mathsf{Nf} \cap (\bigcup_{\mathsf{leaf}^{\mathcal{H}}_i \in T_{0,j}} \mathsf{nul}_{0,i,j}) = \mathsf{Nf} \cap (\bigcup_{\mathsf{leaf}^{\mathcal{H}}_i \in T_{1,j}} \mathsf{nul}_{1,i,j})$
for $b \in \{0,1\}, \mathsf{leaf}^{\mathcal{H}}_i \in T_b$ **do** $O^{\mathcal{A}}_b \leftarrow \bigcup_{\mathsf{pk}_{b,i,j} \notin \mathsf{PK}} (\mathsf{pk}_{b,i,j}, a^{\mathcal{S}}_{b,i,j}, \mathsf{ty}^{\mathcal{S}}_{b,i,j})$
$B_5 \leftarrow O^{\mathcal{A}}_0 = O^{\mathcal{A}}_1$
return $B_1 \wedge B^{\mathcal{S}}_2 \wedge B^{\mathcal{T}}_2 \wedge B_3 \wedge B_4 \wedge B_5$

Fig. 6. Transaction privacy experiment

in the Merkle tree of state st with the given nullifier nul.

$$\mathcal{L}^{\mathsf{spend}} = \Big\{ \big(\mathsf{st}, \mathsf{nul}, \mathsf{com}^{\mathcal{S}}\big) \mid \exists (\mathsf{path}, \mathsf{sk}^{\mathcal{S}}, a^{\mathcal{S}}, \mathsf{ty}^{\mathcal{S}}, r^{\mathcal{S}}, \mathsf{rc}^{\mathcal{S}}) : (\mathsf{st.MT}[\mathsf{path}], \mathsf{nul}; \mathsf{sk}^{\mathcal{S}}, (a^{\mathcal{S}}, \mathsf{ty}^{\mathcal{S}}), r^{\mathcal{S}}) \in \mathcal{L}^{\mathsf{nul}} \wedge (\mathsf{st.MT}[\mathsf{path}]; \mathsf{OTA}.P(\mathsf{sk}^{\mathcal{S}}), (a^{\mathcal{S}}, \mathsf{ty}^{\mathcal{S}}), r^{\mathcal{S}}) \in \mathcal{L}^{\mathsf{open}} \wedge \mathsf{com}^{\mathcal{S}} = \mathsf{Commit}(\mathsf{ty}^{\mathcal{S}}, a^{\mathcal{S}}; \mathsf{rc}^{\mathcal{S}}) \Big\}$$

The second language $\mathcal{L}^{\mathsf{output}}$ is even simpler. It claims that the two output commitments, the real (which is contained inside the output note) and the randomized one, contain the same value of the same type. To prevent overflows in the homomorphic commitments, we include a range proof where β is chosen small enough in relation to the group order (maximum inputs and outputs times $2^\beta < |\mathbb{G}|$). Thereby we assume integer amounts in subsequent arguments.

$$\mathcal{L}^{\mathsf{output}} = \Big\{ (\mathsf{note}^{\mathcal{T}}, C^{\mathcal{T}}, \mathsf{com}^{\mathcal{T}}) \mid \exists (\mathsf{pk}^{\mathcal{T}}, a^{\mathcal{T}}, \mathsf{ty}^{\mathcal{T}}, r^{\mathcal{T}}, \mathsf{rc}^{\mathcal{T}}) : (\mathsf{st.MT}[\mathsf{path}]; \mathsf{pk}, (a^{\mathcal{T}}, \mathsf{ty}^{\mathcal{T}}), r^{\mathcal{T}}) \in \mathcal{L}^{\mathsf{open}} \wedge \mathsf{com}^{\mathcal{T}} = \mathsf{Commit}(\mathsf{ty}^{\mathcal{T}}, a^{\mathcal{T}}; \mathsf{rc}^{\mathcal{T}}) \; a^{\mathcal{T}} \in \{0, \ldots, 2^\beta - 1\} \Big\}$$

Note that the ciphertext $C^{\mathcal{T}}$ is not referred to in the relation, but when used with a simulation extractable (SE) NIZK, realizes a Signature of Knowledge. I.e. every proof is bound to a specific ciphertext and is invalid for any other ciphertext. The transaction signature σ then contains a proof $\pi^{\mathcal{S}}_i$ for each input together with the SHC $\mathsf{com}^{\mathcal{S}}_i$ and for each output a signature $\pi^{\mathcal{T}}_i$ and the SHC $\mathsf{com}^{\mathcal{T}}_i$. The last component of the signature is the aggregated randomness of the commitments which, together with the Δ_{ty} imbalance, allows verification.

For a full transaction verification, all proofs and signatures contained in σ must be valid and the published nullifiers must be unique regarding the set of nullifiers in state st.

With the transaction signature σ having a separate proof for each input and output, it is possible to merge transactions by calculating the union of their proofs. To maintain the verifiability of the commitments, the randomness of the merged transactions is added. The irreversible addition operation then prevents future parties to unmerge a transaction if they have not seen the separate parts beforehand. To maintain the anonymity, we order inputs and outputs canonically after each merge.

As a remark, the aggregated randomness in a transaction may be replaced by a proof of knowledge. Like the binding signature of Sapling, this finalizes a transaction such that it can no longer be merged with others.

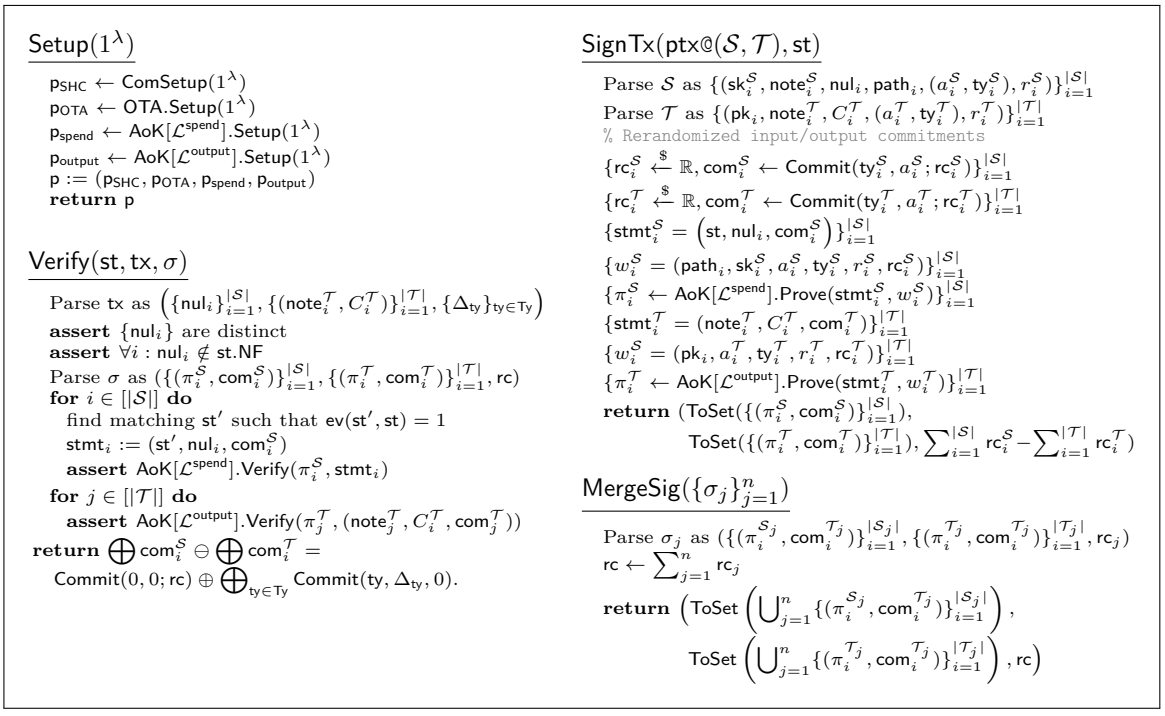

Fig. 7. The Zswap Construction

6 Security Proof

In this section we prove the main three security properties of Zswap construction on Fig. 7 we introduced in Section 4.

Theorem 6.1 (Anti-Theft)**.** *The Zswap protocol prevents theft (Definition 4.3), assuming OTA security, NIZK zero-knowledge, NIZK simulation-extractability, and SHC binding and HID-OR.*

Proof (Sketch). When $\mathcal{A}$ triggers the winning condition of the anti-theft game with an adversarial transaction tx^*, there exists a note or a nullifier taken from some honest $\mathcal{O}_{\mathsf{Spend}}$ query $\hat{E}$, such that not all nullifiers and notes from tx were included in tx^*. By NIZK simulation-extractability, the proofs that were produced in $\hat{E}$ "bind" together notes and nullifiers with the corresponding input and output commitments. This means $\mathcal{A}$ uses some commitments from tx, but avoids some other. Given commitment binding, the values we extract for commitments of tx^* (we can extract them from NIZKs) are the same as the values committed in tx.

From this point we can build a reduction $\mathcal{B}$ to HID-OR. $\mathcal{B}$ guesses a commitment C_X that is present in both $\mathsf{tx}, \mathsf{tx}^*$, and C_Y that is only present in tx. It asks the HID-OR challenger $\mathcal{C}$ for either

1. two commitments corresponding to the values $(a_1, \mathsf{ty}_1), (a_2, \mathsf{ty}_2)$ as they should be honestly; or
2. the values swapped as $(a_2, \mathsf{ty}_1), (a_1, \mathsf{ty}_2)$ if both indices correspond to inputs or both to outputs, and
3. swapped with negation $(-a_2, \mathsf{ty}_2), (-a_1, \mathsf{ty}_1)$ otherwise.

This way of embedding guarantees the HID-OR requirement that $\Delta_{\mathsf{ty},0} = \Delta_{\mathsf{ty},1}$. Then $\mathcal{B}$ simulates the two NIZKs corresponding to C_X, C_Y. When $\mathcal{A}$ presents tx^*, $\mathcal{B}$ will extract the randomness rc_i for all commitments except for C_X, and thus because tx^* also includes the joint randomness rc^*, $\mathcal{B}$ can compute the randomness rc_X for C_X (as $\mathsf{rc}^* - \sum \mathsf{rc}_i$). Given rc_X, $\mathcal{B}$ can check whether C_X contains (a_1, ty_1) or the "swapped" value, and thus wins HID-OR. □

We present the detailed proof of anti-theft in the full version of this paper.

Theorem 6.2 (Privacy)**.** *The Zswap protocol is private (Definition 4.5), if NIZK is zero-knowledge, Pedersen commitments are hiding, and OTA is private and satisfies nullifier pseudorandomness.*

Proof. First, we note that changing the order of merge in the tree T does not affect the resulting merge transaction. Let T' be a variant of T where any two transactions are swapped — then $\mathsf{EquivTree}(T, T') = 1$ and

$\mathsf{EvalTree}(\mathsf{st}, T) = \mathsf{EvalTree}(\mathsf{st}, T')$. The first statement can be verified by manually checking all the predicates and making sure they are indifferent to the order of the leaves. As for the evaluation equality statement, first note that all the leaves will be processed in the same way and with respect to the same st irrespectively of their order. So we only need to argue that $\mathsf{MergeTx}$ and $\mathsf{MergeSig}$ are commutative — which is trivial since they are only uniting sets and taking sums (which are commutative operations on their own).

Hence we can represent T as a merge of two transactions, coming from subtrees $T_{\mathcal{A}}$ and $T_{\mathcal{H}}$. $T_{\mathcal{A}}$ contains exactly the same leaves for both T_0 and T_1 if they are equivalent. Therefore, we only need to show the indistinguishability of transactions $\mathsf{tx}_{\mathcal{H},0}$ and $\mathsf{tx}_{\mathcal{H},1}$, created from $T_{\mathcal{H},0}, T_{\mathcal{H},1}$ correspondingly. If $T_{\mathcal{H}}$ is empty in one case (contains no leaves), it must be empty in the other case, due to the restriction on input and output size, B_2, in $\mathsf{EquivTree}$. And thus if $\mathsf{tx}_{\mathcal{H},b}$ is empty, $\mathsf{tx}_{\mathcal{H},1-b}$ should be empty too for each $b \in \{0,1\}$, in which case the privacy proof is trivial, since adversarial transactions are constructed in exactly the same manner in both worlds. Therefore, assume that there is at least one input or output in an honest transaction in both cases. Next, observe that:

$$\mathsf{MergeSig}(\{\mathsf{SignTx}(\mathsf{BuildPTx}(I_i, O_i, \mathsf{st}, \mathsf{SK}), \mathsf{st})\}_i) =$$
$$\mathsf{SignTx}(\mathsf{BuildPTx}(I' = \bigcup I_i, O' = \bigcup O_i, \mathsf{st}, \mathsf{SK}), \mathsf{st})$$

Because of this homomorphic property (and a similar one for $\mathsf{MergeTx}$), we can assume that both $\mathsf{tx}_{\mathcal{H},b}$ and their signatures are just a direct output of $\mathsf{CompleteTx}$ and SignTx (w.r.t. (I', O')), and no merging is involved. The form of $\mathsf{tx}_{\mathcal{H},b}$ that $\mathcal{A}$ receives is

$$(\{\mathsf{nul}_i\}^{|\mathcal{S}|}, \{(\mathsf{note}_i, C_i)\}^{|\mathcal{T}|}, \{\Delta_{\mathsf{ty}}\},$$
$$\sigma_i@(\{(\pi_i^{\mathcal{S}}, \mathsf{com}_i^{\mathcal{S}})\}^{|\mathcal{S}|}, \{(\pi_i^{\mathcal{T}}, \mathsf{com}_i^{\mathcal{T}})\}^{|\mathcal{T}|}, \mathsf{rc}))$$

We first argue informally why this transaction looks the same for both trees.

1. Set sizes: number of inputs and outputs $|\mathcal{S}|$ and $|\mathcal{T}|$ are the same, guaranteed by B_2, and by the fact these dimensions are just summed when transactions are merged.
2. Size and content of the $\{\Delta_{\mathsf{ty}}\}$ set is the same by B_1.
3. The set of (honest) input nullifiers $\{\mathsf{nul}_i\}^{|\mathcal{S}|}$ contains: nullifiers that $\mathcal{A}$ received from $\mathcal{O}_{\mathsf{Spend}}$ — these are the same by B_4; and nullifiers unknown previously to $\mathcal{A}$ — deterministic but indistinguishable by nullifier pseudorandomness.
4. $\{(\mathsf{note}_i, C_i)\}^{|\mathcal{T}|}$ are either: belonging to adversarial keys in which case both the notes and ciphertexts have the same distribution, as guaranteed by B_5 — these notes contain the same values, but created in both trees with uniformly sampled randomness, so are indistinguishable; the same applies to the adversarial ciphertexts. Or the notes belong to the honest keys, in which case they are (together with the ciphertexts) indistinguishable by the OTA privacy.
5. In the signatures σ_i:
   (5.1) By zero-knowledge, proofs $\pi_i^{\mathcal{S}}$ and $\pi_i^{\mathcal{T}}$ can be simulated in both worlds, so they are indistinguishable.
   (5.2) Intermediary commitments $\mathsf{com}_i^{\mathcal{S}}, \mathsf{com}_i^{\mathcal{T}}$ can contain just zero (except for one chosen commitment for each type which must contain Δ_{ty} to balance out the public imbalance), and this is indistinguishable by HID-OR — commitment hiding with open randomness (by a variation with n elements involved simultaneously, as described in Lemma 2.1).

The formal reduction is described in Appendix D. □

Theorem 6.3 (Balance)**.** *The Zswap protocol satisfies the Balance property (Definition 4.4) by anti-theft, NIZK ZK, NIZK SE, and OTA Security.*

Proof (Sketch). Intuitively, the property reduces to:

1. NIZK SE (implying knowledge soundness (KS)) and commitment homomorphism (every valid transaction is balanced);
2. Anti-theft, since $\mathcal{A}$ can use only its own notes; and binding, since $\mathcal{A}$ can only open output commitments in a single way when spending them.

The first guarantees that transactions do not break per-type balance and do not introduce any coins out of thin air etc. The second proves that in the long-term $\mathcal{A}$ can only move funds through such honest transactions. The full proof is presented in Appendix E. □

7 Implementation

To show that our construction is practical, we developed a prototype implementation available online[8]. We use the rust framework ark-works[9] and their Groth16 SNARK library. For an efficient hashing circuit, we use Poseidon [15]. As the construction is similar to the Zcash Sapling version, we use this as performance com-

8 https://anonymous.4open.science/r/zswap-E331
9 https://arkworks.rs

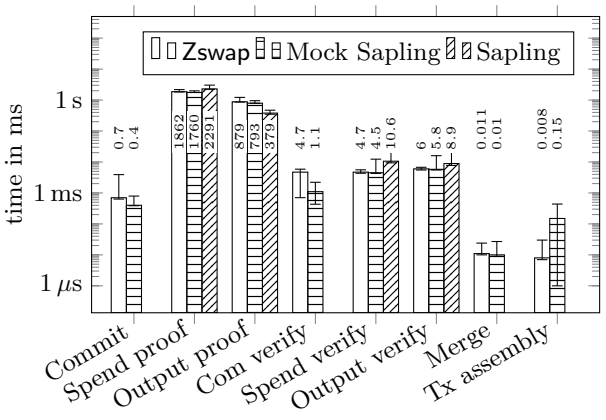

Fig. 8. Comparison of our protocol to our Mock Sapling implementation with the same hash function and the original Zcash.

parison. Our implementation differs in various aspects that aren't material to the core protocol, including the SNARK implementation, hash functions used, and commitments used in the coin commitment Merkle tree. The production Zcash Sapling implementation[10] uses less performant variants for backwards compatibility and integration purposes. A direct performance comparison to Zcash is therefore misleading, and we introduce an artificial "Mock Sapling" code base, that re-implements the Zcash Sapling cryptographic protocol but uses the same primitives as we do. We achieve this by removing the types and type commitments from our construction. The comparison then allows us to detect the performance impact of our changes to the cryptographic protocol compared to Sapling without the measurement error caused by different implementations.

For spend proofs, our protocol requires 25,416 gates, 10% more than Mock Sapling (23,084). For output proofs, we use 14,852 gates, 18% more (12,520). In both cases the difference of 2,332 gates consists of a hash-to-curve operation to associate the type with a curve point. The rest of the constraints further break down into two dominant groups: Symmetric cryptography (hashes, commitments, and Merkle tree verification), and group operations related to the binding signature. For spend proofs, symmetric operations dominate with 66% of constraints (16,935), primarily from the Merkle tree verification (using a tree of height 32). The group operations cover almost all of the remaining 24% of constraints (with 6,114). For output proofs, group operations make up 41% of the constraints (with the same 6,114 constraints), and symmetric operations 43% of constraints (6,403).

10 https://github.com/zcash/zcash

The median, min and max runtimes of 30 executions on a 6$^{\text{th}}$ generation i7 CPU@2.7GHz are summarized in Figure 8. The times are for creating Commitments, Spend proofs and Output proofs. For verification, we measure the homomorphic commitment comparison, the Spend proof verification and the Output proof verification. Without comparison to Sapling are Merging transaction and transaction assembly. The SNARKs proving time takes around two seconds for each input and one second for each output, dominating the transaction generation. The verification is noticeably slower with support for types but approximately equal to Zcash. Overall, we notice that the impact of the additional constraints required for our protocol are minimal while providing additional functionality.

Acknowledgements

We thank the anonymous reviewers and our shepherd, Sherman Chow, for their helpful comments in improving this work. This work was partially funded by the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM), the Sapere Aude: DFF-Starting Grant number 0165-00079B "Foundations of Privacy Preserving and Accountable Decentralized Protocols" and by Input Output (iohk.io) through their funding of the Edinburgh Blockchain Technology Lab.

References

[1] Kurt M. Alonso and Jordi Herrera Joancomartí. Monero - privacy in the blockchain. Cryptology ePrint Archive, Report 2018/535, 2018. https://eprint.iacr.org/2018/535.

[2] Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction and randomization of groth's zk-SNARK. Cryptology ePrint Archive, Report 2020/811, 2020. https://eprint.iacr.org/2020/811.

[3] Carsten Baum, Bernardo David, and Tore Kasper Frederiksen. P2dex: Privacy-preserving decentralized cryptocurrency exchange. In Kazue Sako and Nils Ole Tippenhauer, editors, *Applied Cryptography and Network Security*, pages 163–194, Cham, 2021. Springer International Publishing.

[4] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014. 10.1109/SP.2014.36.

[5] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: Enabling decentralized private computation. In *2020 IEEE Symposium*

on Security and Privacy, pages 947–964. IEEE Computer Society Press, May 2020. 10.1109/SP40000.2020.00050.

[6] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020*, volume 12059 of *LNCS*, pages 423–443. Springer, Heidelberg, February 2020. 10.1007/978-3-030-51280-4_23.

[7] Matteo Campanelli, Felix Engelmann, and Claudio Orlandi. Zero-knowledge for homomorphic key-value commitments with applications to privacy-preserving ledgers. Cryptology ePrint Archive, Report 2021/1678, 2021. https://eprint.iacr.org/2021/1678.

[8] Shumo Chu, Qiudong Xia, and Zhenfei Zhang. Manta: Privacy preserving decentralized exchange. Cryptology ePrint Archive, Report 2020/1607, 2020. https://ia.cr/2020/1607.

[9] Apoorvaa Deshpande and Maurice Herlihy. Privacy-preserving cross-chain atomic swaps. In Matthew Bernhard, Andrea Bracciali, L. Jean Camp, Shin'ichiro Matsuo, Alana Maurushat, Peter B. Rønne, and Massimiliano Sala, editors, *FC 2020 Workshops*, volume 12063 of *LNCS*, pages 540–549. Springer, Heidelberg, February 2020. 10.1007/978-3-030-54455-3_38.

[10] Donghui Ding, Kang Li, Linpeng Jia, Zhongcheng Li, Jun Li, and Yi Sun. Privacy protection for blockchains with account and multi-asset model. *China Communications*, 16(6):69–79, 2019.

[11] Felix Engelmann, Lukas Müller, Andreas Peter, Frank Kargl, and Christoph Bösch. SwapCT: Swap confidential transactions for privacy-preserving multi-token exchanges. *PoPETs*, 2021(4):270–290, October 2021. 10.2478/popets-2021-0070.

[12] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 649–678. Springer, Heidelberg, December 2019. 10.1007/978-3-030-34578-5_23.

[13] Georg Fuchsbauer, Michele Orrù, and Yannick Seurin. Aggregate cash systems: A cryptographic investigation of Mimblewimble. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 657–689. Springer, Heidelberg, May 2019. 10.1007/978-3-030-17653-2_22.

[14] Zhimin Gao, Lei Xu, Keshav Kasichainula, Lin Chen, Bogdan Carbunar, and Weidong Shi. Private and atomic exchange of assets over zero knowledge based payment ledger. *arXiv preprint arXiv:1909.06535*, 2019.

[15] Lorenzo Grassi, Daniel Kales, Dmitry Khovratovich, Arnab Roy, Christian Rechberger, and Markus Schofnegger. Starkad and Poseidon: New hash functions for zero knowledge proof systems. Cryptology ePrint Archive, Report 2019/458, 2019. https://eprint.iacr.org/2019/458.

[16] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. *GitHub: San Francisco, CA, USA*, 2016.

[17] Thomas Kerber, Aggelos Kiayias, and Markulf Kohlweiss. Kachina–foundations of private smart contracts. In *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, pages 1–16. IEEE, 2021.

[18] Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy*, pages 839–858. IEEE Computer Society Press, May 2016. 10.1109/SP.2016.55.

[19] Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. Omniring: Scaling private payments without trusted setup. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 31–48. ACM Press, November 2019. 10.1145/3319535.3345655.

[20] Sarah Meiklejohn and Rebekah Mercer. Möbius: Trustless tumbling for transaction privacy. *PoPETs*, 2018(2):105–121, April 2018. 10.1515/popets-2018-0015.

[21] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *FC 2018 Workshops*, volume 10958 of *LNCS*, pages 43–63. Springer, Heidelberg, March 2019. 10.1007/978-3-662-58820-8_4.

[22] Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin T. Vechev. zkay: Specifying and enforcing data privacy in smart contracts. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1759–1776. ACM Press, November 2019. 10.1145/3319535.3363222.

[23] Sam M Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J Knottenbelt. Sok: Decentralized finance (defi). *arXiv preprint arXiv:2101.08778*, 2021.

[24] Jiahua Xu, Nazariy Vavryk, Krzysztof Paruch, and Simon Cousaert. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *arXiv preprint arXiv:2103.12732*, 2021.

[25] Zheng Yi, Howard Ye, Patrick Dai, Sun Tongcheng, and Vladislav Gelfer. Confidential assets on MimbleWimble. Cryptology ePrint Archive, Report 2019/1435, 2019. https://eprint.iacr.org/2019/1435.

A Proofs of Knowledge

We quickly remind the reader of the classical security properties of a NIZK: The scheme is said to be *complete*, if $\forall(\mathsf{stmt}, w) \in \mathcal{R}_{\mathcal{L}}$ we have

$$\Pr\left[\begin{array}{l}(\mathsf{crs}, \cdot) \leftarrow \mathsf{Setup}(\mathcal{R}_{\mathcal{L}});\\ \mathsf{Verify}(\mathsf{crs}, \mathsf{stmt}, \mathsf{Prove}(\mathsf{crs}, \mathsf{stmt}, w)) = 1\end{array}\right] = 1$$

The scheme is *knowledge sound* (KS) if for all PPT $\mathcal{A}$ there exists an extractor $\mathcal{E}$ such that

$$\Pr\left[\begin{array}{l}(\mathsf{crs},\cdot)\leftarrow\mathsf{Setup}(\mathcal{R}_{\mathcal{L}});\\(\mathsf{stmt},\pi)\leftarrow\mathcal{A}(\mathsf{crs});\ w\leftarrow\mathcal{E}(\mathsf{stmt},\pi):\\\quad\mathsf{Verify}(\mathsf{crs},\mathsf{stmt},\pi)=1\ \wedge\\\quad(\mathsf{stmt},w)\notin\mathcal{R}_{\mathcal{L}}\end{array}\right]\leq\mathsf{negl}(\lambda)$$

Finally, we say that the scheme is (computational) *zero-knowledge* if for any PPT adversary $\mathcal{A}$ and $\mathcal{R}_\lambda$, $|\varepsilon_0 - \varepsilon_1| \leq \mathsf{negl}(\lambda)$, where

$$\varepsilon_b = \Pr\left[\ (\mathsf{crs},\tau)\leftarrow\mathsf{Setup}(\mathcal{R}_{\mathcal{L}}) : \mathcal{A}^{\mathcal{S}_{b,\mathsf{crs},\tau}}(\mathsf{crs})=1\ \right]$$

The oracle $\mathcal{S}_{b,\mathsf{crs},\tau}$ on input (stmt,w) asserts that $(\mathsf{stmt},w)\in\mathcal{R}_\lambda$ and then returns $\pi=\mathsf{Prove}(\mathsf{crs},\mathsf{stmt},w)$ if $b=0$, and $\pi=\mathsf{Sim}(\mathsf{crs},\tau,\mathsf{stmt})$ if $b=1$.

We additionally require a stronger variant of the KS property called simulation-extractability, where extraction is possible in the presence of simulated proofs.

Definition A.1 (Simulation-Extractability [2])**.** *We say that a non-interactive argument is weakly simulation-extractable (SE) if for any PPT adversary $\mathcal{A}$ there exists a PT extractor $\mathcal{E}_\mathcal{A}$ such that for $\mathcal{R}_\lambda$,*

$$\Pr\left[\begin{array}{l}(\mathsf{crs},\tau)\leftarrow\mathsf{Setup}(\mathcal{R}_{\mathcal{L}});\\(\mathsf{stmt},\pi)\leftarrow\mathcal{A}^{\mathcal{S}_{\mathsf{crs},\tau}}(\mathsf{crs});\\w\leftarrow\mathcal{E}_\mathcal{A}(\mathsf{stmt},\pi):\\\mathsf{Verify}(\mathsf{crs},\mathsf{stmt},\pi)=1\ \wedge\\\quad(\mathsf{stmt},w)\notin\mathcal{R}_\lambda\wedge\mathsf{stmt}\notin Q\end{array}\right]\leq\mathsf{negl}(\lambda)$$

where $\mathcal{S}_{\mathsf{crs},\tau}(\mathsf{stmt})$ is a simulator oracle that calls $\mathsf{Sim}(\mathsf{crs},\tau,\mathsf{stmt})$ internally, and also records stmt into Q.

The SE property guarantees that a NIZK in a certain sense "instance-binding". E.g. if a certain game produces proofs $\{\pi_i\}$ for a language with a hard sublanguage (e.g. $\mathsf{stmt}=(\mathsf{stmt}_1,\mathsf{stmt}_2)$ and $\mathsf{stmt}_1=f(w_1)$ where f is a one-way function), we can simulate these proofs, and by SE $\mathcal{A}$ can only return either these simulated proofs for the *same* instance, or we can extract w from them (in which case the reduction essentially inverts f). This means that $\mathcal{A}$ cannot change the stmt_2 part in $\{\pi_i\}$. In practice this means that if the honest π hides certain secrets (a key, or a secret random value), then $\mathcal{A}$ cannot maul π into proofs for a "slightly different" instance.

B Zerocash-Style OTA

To use Zerocash style one-time-accounts, we provide an instantiation which is compatible with Zerocash.

Given a key anonymous public key encryption scheme PKE, a labeled PRF and a commitment scheme (Commit). The randomness space is $\mathbb{S}:=(\{0,1\}^\lambda)^3$, the key space is $\mathbb{K}:=\{0,1\}^\lambda$ and the message space consists of two λ-bit integers $\mathbb{M}:=\mathbb{Z}_{2^\lambda}^2$ the first representing an amount and the second identifying a type. The randomness space Ξ is specified by PKE.

$\mathsf{p}\leftarrow\mathsf{Setup}(1^\lambda)$**:** Initializes the SNARK parameters p.

$(\mathsf{sk},\mathsf{pk})\leftarrow\mathsf{KeyGen}()$**:** Samples a_{sk} randomly from $\mathbb{K}$ and defines $a_{\mathsf{pk}}\leftarrow P(\mathsf{sk}):=\mathsf{PRF}^{\mathrm{addr}}_{a_{\mathsf{sk}}}(0)$; samples an encryption key pair $\mathsf{sk}_{\mathrm{enc}},\mathsf{pk}_{\mathrm{enc}}$. Return $\mathsf{sk}=(a_{\mathsf{sk}},\mathsf{sk}_{\mathrm{enc}}),\mathsf{pk}=(a_{\mathsf{pk}},\mathsf{pk}_{\mathrm{enc}})$. The relationship between secret and public keys of the encryption keys is defined in the encryption scheme.

$\mathsf{note}\leftarrow\mathsf{Gen}(\mathsf{pk},\vec{a},r)$**:** Parse $(\mathsf{rk},\mathsf{rc},\mathsf{rn})\leftarrow r$ and $(a_{\mathsf{pk}},\mathsf{pk}_{\mathrm{enc}})\leftarrow\mathsf{pk}$. Commit to $(a_{\mathsf{pk}},\mathsf{rn})$ with randomness rk as commitment com and then commit to $\mathsf{com},\vec{a}$ with randomness rc. So $\mathsf{note}=\mathsf{Commit}(\mathsf{Commit}(a_{\mathsf{pk}},\mathsf{rn};\mathsf{rk}),\vec{a};\mathsf{rc})$.

$C\leftarrow\mathsf{Enc}(\mathsf{pk},(\vec{a},r),\xi)$**:** Parse $(a_{\mathsf{pk}},\mathsf{pk}_{\mathrm{enc}})\leftarrow\mathsf{pk}$. Encrypt $(\vec{a},r)$ with $\mathsf{pk}_{\mathrm{enc}}$ to ciphertext C and return C.

$(\vec{a}',r')/\bot\leftarrow\mathsf{Receive}(\mathsf{note},C,\mathsf{sk})$**:** Parse $(a_{\mathsf{sk}},\mathsf{sk}_{\mathrm{enc}})\leftarrow\mathsf{sk}$. Decrypt C with $\mathsf{sk}_{\mathrm{enc}}$ to $(\vec{a}',r')$. Parse $(\mathsf{rk}',\mathsf{rc}',\mathsf{rn}')\leftarrow r'$ and verify that these values recreate the commitment $\mathsf{note}=\mathsf{Commit}(\mathsf{Commit}(P(a_{\mathsf{sk}}),\mathsf{rn}';\mathsf{rk}'),\vec{a}';\mathsf{rc}')$. If decryption fails or the commitment does not match the note, return fail.

$\mathsf{nul}\leftarrow\mathsf{NulEval}(\mathsf{sk},r)$**:** Parse $(a_{\mathsf{sk}},\mathsf{sk}_{\mathrm{enc}}):=\mathsf{sk}$ and $(\mathsf{rk},\mathsf{rc},\mathsf{rn}):=r$. Evaluate $\mathsf{nul}=\mathsf{PRF}^{\mathrm{sn}}_{a_{\mathsf{sk}}}(\mathsf{rn})$ and return nul.

Theorem B.1 (OTA Binding)**.** *When using a binding commitment scheme* Commit*, the Zerocash commitment scheme is binding according to Definition 3.3*

Proof. Let $\mathcal{A}$ return $\mathsf{pk}_0,\vec{a}_0,r_0,\mathsf{pk}_1,\vec{a}_1,r_1$ such that $\mathsf{Gen}(\mathsf{pk}_0,\vec{a}_0,r_0)=\mathsf{Gen}(\mathsf{pk}_1,\vec{a}_1,r_1)$ and $\vec{a}_0\neq\vec{a}_1$. Parsing $\forall i\in\{0,1\}:(\mathsf{rk}_i,\mathsf{rc}_i,\mathsf{rn}_i)\leftarrow r_i$, the first condition implies the equality $\mathsf{Commit}(\mathsf{Commit}(\mathsf{pk}_0,\mathsf{rn}_0;\mathsf{rk}_0),\vec{a}_0;\mathsf{rc}_0)=\mathsf{Commit}(\mathsf{Commit}(\mathsf{pk}_1,\mathsf{rn}_1;\mathsf{rk}_1),\vec{a}_1;\mathsf{rc}_1)$ but the commitments have different values for $\vec{a}$. This breaks the binding property of the commitment scheme. □

Theorem B.2 (Note Uniqueness)**.** *Zerocash OTA style notes are unique according to Definition 3.5*

Proof. Trivially follows from binding of the underlying commitment scheme – since both pk and $\vec{a}$ are commitment messages, finding a collision (even with adversar-

ially chosen randomness) amounts to breaking commitment binding. □

Theorem B.3 (Nullifier Uniqueness)**.** *The Zerocash OTA scheme has unique nullifiers (Definition 3.7), if the PRF is secure and the commitment scheme is binding.*

Proof. Let $\mathcal{A}$ return $(\mathsf{sk}_0, r_0, \vec{a}_0, \mathsf{sk}_1, r_1, \vec{a}_1)$ which generates the same note but different nullifiers nul_0 and nul_1. The binding commitment scheme ensures that $(\mathsf{sk}_0, r_0, \vec{a}_0) = (\mathsf{sk}_1, r_1, \vec{a}_1)$. As $\mathsf{NulEval}$ is deterministic, the nullifiers are equal, contradicting our assumption. □

Theorem B.4 (Nullifier Pseudorandomness)**.** *The Zerocash OTA nullifiers are pseudorandom (Definition 3.6), if the PRF is secure and the commitment scheme is hiding.*

Proof. The pseudorandomness experiment is exactly the same as standard pseudorandomness, except $\mathcal{A}$ is given (1) a public key, (2) an oracle access to $\mathsf{Receive}(\cdot, \cdot, \mathsf{sk})$. The public key is computed as $\mathsf{PRF}^{\mathrm{addr}}_{a_{\mathsf{sk}}}(0)$, and so by pseudorandomness of this PRF (with a different domain) can be replaced by a random value ψ, so knowledge of an extra random value does not help $\mathcal{A}$ to distinguish. Regarding the $\mathsf{Receive}$ oracle, it performs decryption with an unrelated $\mathsf{sk}_{\mathrm{enc}}$, so it does not interfere with the main pseudorandomness reduction, since this oracle does not use a_{sk} (it uses ψ which we already argued to be random and thus irrelevant). □

Theorem B.5 (OTA Privacy)**.** *Using a hiding commitment scheme and a IND-CCA, key anonymous (IK-IND) encryption scheme, the Zerocash-style OTA is private according to Definition 3.4*

Proof. This property trivially reduces to the assumptions mentioned, since its game is essentially a merger of IND-CCA, IK-IND, and a commitment hiding, where all the data is showed simultaneously, depending on b. Formally, we

1. switch nullifiers to random values,
2. switch the commitments from $b = 0$ to $b = 1$, and then
3. switch messages by IND-CCA, and (4) keys by IK-IND.

In the steps (1) and (2) we can still simulate $\mathcal{O}_{\mathsf{Rcv}}$ since in both cases it uses a keypair that is unrelated to the PRF keypair. In (3) and (4) we just wire the challenge oracles through to $\mathcal{O}_{\mathsf{Rcv}}$, since $\mathcal{O}_{\mathsf{Rcv}}$ is essentially a CCA-like oracle. □

C Proof of HID-OR for Sparse Pedersen Commitment

Proof of Lemma 2.1. Let the hash function be modelled by the random oracle, and $H(\mathsf{ty}_i) = G^{t_i}$, and thus the challenger knows all t_i. The form of commitments that are given to the adversary is thus:

$$[t_b a_b + r], [t'_b a'_b + r'], \mathsf{rc} = r - r'$$

The initial game is HID-OR^0, in which $\mathcal{A}$ sees the previous equation for $b = 0$. In Game_1 we sample $\hat{r}$ and instead give $\mathcal{A}$ the following:

$$[\hat{r}], [t'_0 a'_0 + r'], \mathsf{rc} = (\hat{r} - t_0 a_0) - r'$$

Note that $t_0 a_0 + r$ and $\hat{r}$ are both uniform, so $\mathcal{A}$ does not see the difference, the transition is perfect. Similarly we equivocate the second commitment:

$$[\hat{r}], [\hat{r}'], \mathsf{rc} = (\hat{r} - \hat{r}') - (t_0 a_0 - t'_0 a'_0)$$

And now we "switch" the elements to the case of $b = 1$. Let $T = \{t_0, t'_0\} = \{t_1, t'_1\}$. Given $\Delta_{\mathsf{ty},0} = \Delta_{\mathsf{ty},1}$ for all $\mathsf{ty} \in T$ we have:

$$t_0 a_0 - t'_0 a'_0 = \sum_{t \in T} t \Delta_{\mathsf{ty},0} = \sum_{t \in T} t \Delta_{\mathsf{ty},1} = t_1 a_1 - t'_1 a'_1$$

Therefore what $\mathcal{A}$ sees now is:

$$[\hat{r}], [\hat{r}'], \mathsf{rc} = (\hat{r} - \hat{r}') - (t_1 a_1 - t'_1 a'_1)$$

So we can proceed with replacing $\hat{r}$ and $\hat{r}'$ "back" into real commitments to the $b = 1$ values similarly to how we abstracted them in the first steps of the proof. The end result is a distribution that $\mathcal{A}$ sees for $b = 1$ (HID-OR^1): $[t_1 a_1 + r], [t'_1 a'_1 + r'], \mathsf{rc} = r - r'$ so our hiding holds with probability 1 (is perfect). □

Corollary C.1. *Assuming commitment hiding, Lemma 2.1 holds even if $\mathcal{A}$ provides two sets of size n (not just pairs) of input and output commitments, $\{(\mathsf{ty}_{i,t}, a_{i,t}), (\mathsf{ty}'_{i,t}, a'_{i,t})\}^{n,1}_{i,t=0,0}$, as long as they still jointly sum to the same values per type.*

Proof. The proof is exactly the same as the previous one, except that we first "idealise" $2n$ commitments (and not just two) from $b = 0$, and then de-idealise them all back. The transition logic in the middle holds similarly because $\Delta_{\mathsf{ty},0} = \Delta_{\mathsf{ty},1}$. □

D Privacy Proof

Proof of Theorem 6.2, deferred reduction. Formally, we construct a series of games transforming $\mathsf{Privacy}^0_{\mathcal{A}}(1^\lambda)$ into $\mathsf{Privacy}^1_{\mathcal{A}}(1^\lambda)$.

$\underline{\mathsf{Game}_1}$: Replace all honest NIZKs (created inside $\mathcal{O}_{\mathsf{Spend}}$ and SignTx) to simulated NIZKs.

$\underline{\mathsf{Game}_2}$: By nullifier pseudorandomness, replace all the honest nullifiers $\mathcal{A}$ has not seen through $\mathcal{O}_{\mathsf{Spend}}$ by the nullifiers over a different set of secret keys.

To switch the $\mathsf{NulEval}(\mathsf{sk}_0, r_0)$ to $\mathsf{NulEval}(\mathsf{sk}_1, r_1)$, we first pick a random function f and switch *all* evaluations of $\mathsf{NulEval}(\mathsf{sk}_0, r)$ to $f(r)$. Since the nullifiers in $\mathsf{tx}^{\mathcal{H}}$ have not been queried previously in $\mathcal{O}_{\mathsf{Spend}}$, the evaluation of $f(r_0)$ is the first one in the game. Hence, it is equivalent to returning a random value ψ instead of $f(r_0)$. Then, again by pseudorandomness, we return all the other (not related to $\mathsf{tx}^{\mathcal{H}}$) evaluations of $f(\cdot)$ back to $\mathsf{NulEval}(\mathsf{sk}_0, \cdot)$. We now perform exactly the same steps then for the second key, de-idealizing ψ into $\mathsf{NulEval}(\mathsf{sk}_1, r_1)$. First we replace all evaluations of $\mathsf{NulEval}(\mathsf{sk}_1, r)$ to $f(r)$, then we observe that r_1 has not yet been queried to $f(\cdot)$, so it is equivalent to return $f(r_1)$ instead of ψ. Then we replace all $f(r)$ back to $\mathsf{NulEval}(\mathsf{sk}_1, r)$, *including* the target nullifier in $\mathsf{tx}^{\mathcal{H}}$. This is how we have just replaced $\mathsf{NulEval}(\mathsf{sk}_0, r_0)$ by $\mathsf{NulEval}(\mathsf{sk}_1, r_1)$. Repeat the procedure for all pairs of nullifiers in $\mathsf{tx}^{\mathcal{H}}$ in any order (i.e. it does not matter what the source and target of the replacement is since all values are pseudorandom).

$\underline{\mathsf{Game}_3}$: Replace output notes and related ciphertexts to honest secret keys by the notes and ciphertexts for T_1. Recall that output notes to $\mathcal{A}$ keys must have exactly the same inputs in both cases, so their distribution is exactly equal. This step relies on the OTA privacy and again on nullifier pseudorandomness.

By OTA privacy one can replace $\mathsf{OTA.Gen}(\mathsf{pk}_b, (\mathsf{ty}_b, v_b), r)$ and $\mathsf{OTA.Enc}(\mathsf{pk}_b, (\mathsf{ty}_b, v_b, r), \xi)$ from $b = 0$ to $b = 1$ even if there are $\mathsf{Receive}$ calls in the game before and after the replacement. (After, we cannot query the challenge note in a CCA fashion, which is irrelevant since we replace it in the very end of the game.)

The only detail left now is that there are also nullifier evaluations that use the challenge sk_b, but by nullifier pseudorandomness we can replace these nullifiers by consistent random values before the OTA privacy switch (we can do that in the presence of $\mathsf{Receive}$ evaluations because of the $\mathsf{Receive}$ oracle in the pseudorandomness definition), perform the privacy switch, and then return the real $\mathsf{NulEval}$ evaluations back.

$\underline{\mathsf{Game}_4}$: Replace intermediary commitments com with the values from T_1. Here we apply HID-OR for vectors, as described in Corollary C.1: replacing one set of typed commitments with another, given that $\Delta_{\mathsf{ty},0} = \Delta_{\mathsf{ty},1}$ for each type. This transition is perfect.

$\underline{\mathsf{Game}_5}$: Remove the simulation, create real proofs according to the new data in T_1. From the first game, we have not modified anything outside of the scope of $\mathsf{EvalTree}$, so all NIZKs in $\mathcal{O}_{\mathsf{Spend}}$ queries are "restored" from simulations without any issue, since they are for exactly the same data. Now it is a matter of a completeness check to make sure that the new transaction constructed in $\mathsf{EvalTree}$ has the same distribution as $\mathsf{tx}_{\mathcal{H},1}$ except for the (yet simulated) proofs, and in particular it satisfies $\mathcal{L}^{\mathsf{spend}}$ and $\mathcal{L}^{\mathsf{output}}$. So now we enable real NIZKs back as well, and by zero-knowledge we obtain honest proofs for T_1. Now Game_5 is equivalent to $\mathsf{Privacy}^1_{\mathcal{A}}(1^\lambda)$, which concludes the privacy proof.

E Balance Proof

Proof of Theorem 6.3. We start with the inductive step first, proving the single-tx $\mathsf{Balance\text{-}Single}$ property, presented next, which asserts that each separate transaction is well-formed (with respect to the data extracted from NIZKs), and in particular, balanced. Additionally, it allows introducing the extractor into the balance game, which we will use later.

$\underline{\mathsf{Balance\text{-}Single}_{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda)}$

$\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{st}, \mathsf{tx}, \sigma) \leftarrow \mathcal{A}(\mathsf{p})$
Parse tx as $\left(\{\mathsf{nul}_i^*\}_{i=1}^{|\mathcal{S}|}\{(\mathsf{note}_i^{\mathcal{T}*}, C_i)\}_{i=1}^{|\mathcal{T}|}, \{\Delta_{a:\mathsf{ty}}\}_{\mathsf{ty}\in\mathsf{Ty}}\right)$
$\mathsf{ptx}_{\mathcal{E}}@(\mathcal{S}_{\mathcal{E}}, \mathcal{T}_{\mathcal{E}}) \leftarrow \mathcal{E}_{\mathcal{A}}(\mathsf{p}, \mathsf{tx}, \sigma)$
Parse $\mathcal{S}_{\mathcal{E}}$ as $\{(\mathsf{sk}_i^{\mathcal{S}}, \mathsf{note}_i^{\mathcal{S}}, \mathsf{nul}_i, \mathsf{path}_i, (a_i^{\mathcal{S}}, \mathsf{ty}_i^{\mathcal{S}}), r_i^{\mathcal{S}})\}_{i=1}^{|\mathcal{S}|}$
Parse $\mathcal{T}_{\mathcal{E}}$ as $\{(\mathsf{pk}_i^{\mathcal{T}}, \mathsf{note}_i^{\mathcal{T}}, C_i^{\mathcal{T}}, (a_i^{\mathcal{T}}, \mathsf{ty}_i^{\mathcal{T}}), r_i^{\mathcal{T}})\}_{i=1}^{|\mathcal{T}|}$
return $\mathsf{Verify}(\mathsf{st}, \mathsf{tx}, \sigma) = 1 \wedge$
$(\mathsf{CheckPTx}(\mathsf{ptx}_{\mathcal{E}}, \mathsf{st}) \neq 1 \vee \mathsf{CompleteTx}(\mathsf{ptx}_{\mathcal{E}}) \neq \mathsf{tx})$

Formally, we claim that for every $\mathcal{A}$ exists $\mathcal{E}_{\mathcal{A}}$ s.t. $\Pr[\mathsf{Balance\text{-}Single}_{\mathcal{A},\mathcal{E}_{\mathcal{A}}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

Proof (Balance-Single). We follow the $\mathsf{CheckPTx}$ code and show why it succeeds. The check that $\mathsf{nul}_i \notin \mathsf{st.MT}$ is directly checked in Verify. All except the last (balance) checks reduce to the NIZK knowledge-soundness.

Looking at every $\mathcal{L}^{\mathsf{spend}}$ NIZK first, the fact it verifies implies that we extract $w = (\mathsf{path}_i, \mathsf{sk}_i^{\mathcal{S}}, a_i^{\mathcal{S}}, \mathsf{ty}_i^{\mathcal{S}}, r_i^{\mathcal{S}}, \mathsf{rc}_i^{\mathcal{S}})$, which by NIZK soundness guarantees for all $i \in [|\mathcal{S}|]$:

$$\mathsf{st.MT}[\mathsf{path}_i] = \mathsf{note}_i^{\mathcal{S}} \wedge$$
$$\mathsf{nul}_i = \mathsf{OTA.NulEval}(\mathsf{sk}_i^{\mathcal{S}}, r_i^{\mathcal{S}}) \wedge$$
$$\mathsf{note}_i^{\mathcal{S}} = \mathsf{OTA.Gen}(\mathsf{OTA}.P(\mathsf{sk}_i), (a_i^{\mathcal{S}}, \mathsf{ty}_i^{\mathcal{S}}), r_i^{\mathcal{S}})$$

The (1) and (3) are by $\mathcal{L}^{\mathsf{open}}$, and (2) is by $\mathcal{L}^{\mathsf{nul}}$ (see $\mathcal{L}^{\mathsf{spend}}$ structure); Similarly, the output NIZKs guarantee (through $\mathcal{L}^{\mathsf{open}}$) the existence of $(\mathsf{pk}^{\mathcal{T}}, a^{\mathcal{T}}, \mathsf{ty}^{\mathcal{T}}, r^{\mathcal{T}}, \mathsf{rc}^{\mathcal{T}})$ s.t.:

$$\forall i \in [|\mathcal{T}|] : \mathsf{note}_i^{\mathcal{T}} = \mathsf{OTA.Gen}(\mathsf{pk}_i^{\mathcal{T}}, (a_i^{\mathcal{T}}, \mathsf{ty}_i^{\mathcal{T}}), r_i^{\mathcal{T}})$$

Which is again by $\mathcal{L}^{\mathsf{open}}$. The last (balance) check that $\forall \mathsf{ty} \in \{\mathsf{ty}_i^{\mathcal{S}}\}_{i=1}^{|\mathcal{S}|} \cup \{\mathsf{ty}_i^{\mathcal{T}}\}_{i=1}^{|\mathcal{T}|}$: $\sum_{a \in \{a_i^{\mathcal{S}} | \mathsf{ty}_i^{\mathcal{S}} = \mathsf{ty}\}_{i=1}^{|\mathcal{S}|}} a - \sum_{a \in \{a_i^{\mathcal{T}} | \mathsf{ty}_i^{\mathcal{T}} = \mathsf{ty}\}_{i=1}^{|\mathcal{T}|}} a = \Delta_{\mathsf{ty}}$ succeeds by the commitment homomorphic property in $\mathsf{Verify}(\cdot,\cdot,\cdot)$. We know by the previous steps that a_i are the same values that are committed into com_i, and the commitments sum to the identity, which means that the committed values a_i sum to 0. □

The main balance property follows inductively by following the $(\mathsf{st}, \mathsf{st}', \mathsf{tx}, \sigma) \in \mathsf{Ins}'$ loop step-by-step. But before we present the main proof reasoning, we must introduce several auxiliary constructions into the game. At each step we will trace the notes that $\mathcal{A}$ can spend, and will argue that both during the game, and in the end of it, the sum of these is not enough to break the balance predicate.

- Instead of computing $v_{\mathcal{A}}$ in a separate loop before the main loop, we will move it *into* the main loop. Now, $v_{\mathcal{A}}$ is updated whenever the balance game locates a $\mathsf{note} \in I$ in the outputs of tx (this note is still attempted to be received by SK^*). This allows us to track the successive stake accumulation of $\mathcal{A}$, but only on those coins that it claims in I.
- We add a set[11] $N_{\overline{\mathcal{H}}}$, initially populated with notes in I_0. Each loop iteration will
  1. remove adversarial input notes from it, and
  2. add to it all the output notes that cannot be received by SK.

  For both actions we need the data extracted by Balance-Single, that "reveals" the notes behind the nullifiers, including adversarial ones. The new variable $v_{\overline{\mathcal{H}}}$ will track balances inside $N_{\overline{\mathcal{H}}}$, and is updated whenever $N_{\overline{\mathcal{H}}}$ is changed. $N_{\overline{\mathcal{H}}}$ and $v_{\overline{\mathcal{H}}}$ track *all* the transient adversarial transactions, not counted in the final $v_{\mathcal{A}}$ (which only sums up the "resulting" assets of $\mathcal{A}$). E.g. $\mathcal{A}$ can "refresh" the note $\mathsf{note} \in I_0$ by moving its funds to another adversarial $\mathsf{note}' \notin I$, and without $N_{\overline{\mathcal{H}}}$ this will not be reflected anywhere.
- We insert the assertion $\exists \mathsf{ty} \in T : v_{\overline{\mathcal{H}}}[\mathsf{ty}] \leq v_0[\mathsf{ty}] + v_{\mathcal{H}-}[\mathsf{ty}] - v_{\mathcal{H}+}[\mathsf{ty}]$ into the end of each loop. Clearly this only increases adversarial chances to win by providing an extra winning condition.
  For example, if this predicate fails in the middle of the loop w.r.t. some intermediate st', $\mathcal{A}$ could as well choose st' for its challenge response, and win the original game with the assertion in the end. And also clearly using $v_{\overline{\mathcal{H}}}$ instead of $v_{\mathcal{A}}$ in the predicate only gives $\mathcal{A}$ more advantage, since in the end $v_{\mathcal{A}} \leq v_{\overline{\mathcal{H}}}$: by anti-theft, $\mathcal{A}$ cannot spend honest nullifiers, and thus it can only claim notes from $N_{\overline{\mathcal{H}}}$ (and can only do it once by nullifier uniqueness and OTA binding), but $v_{\mathcal{A}}$ is calculated w.r.t. $I \subset N_{\overline{\mathcal{H}}}$.

In the base case of the induction, before the first loop (before any transaction is processed), the predicate holds, since $v_{\overline{\mathcal{H}}}[\mathsf{ty}] = v_0[\mathsf{ty}]$ and $v_{\mathcal{H}+} = v_{\mathcal{H}-} = 0$.

Assume now that the predicate holds in the beginning of the loop, we will show it persists through the loop execution. Fix a type ty, and apply the following reasoning to each type in the transaction. Compute the local values $v^{\Delta}_{\mathcal{H}-}, v^{\Delta}_{\mathcal{H}+}$ as prescribed by the game. Compute $v^{\Delta}_{\mathcal{A}}$ by looking at all the outputs in $\mathsf{tx}_{\mathcal{A}}$, and summing the corresponding values that can be received using SK^*. These values are the same as the values extracted by a single-step definition, by OTA output note binding.

Compute $v^{\Delta}_{\overline{\mathcal{H}}}$ from the extracted (input and output) NIZKs by adding all the adversarial output values and subtracting all the adversarial input values.

Now we need to prove that $v^{\Delta}_{\overline{\mathcal{H}}} \leq v^{\Delta}_{\mathcal{H}-} - v^{\Delta}_{\mathcal{H}+}$ (v_0 does not ever change). But this reduces to the commitment balancing condition: $v^{\Delta}_{\overline{\mathcal{H}}} = \sum a_i^{\mathcal{T}\mathcal{A}} - \sum a_i^{\mathcal{S}\mathcal{A}}$, while $v_{\mathcal{H}-} - v_{\mathcal{H}+} = \sum a_i^{\mathcal{S}\mathcal{H}} - \sum a_i^{\mathcal{T}\mathcal{H}}$, and by the last (balancing) check in CheckPTx, guaranteed by Balance-Single, we have

$$\sum a_i^{\mathcal{S}\mathcal{A}} + \sum a_i^{\mathcal{S}\mathcal{H}} - \sum a_i^{\mathcal{T}\mathcal{A}} - \sum a_i^{\mathcal{T}\mathcal{H}} \geq 0$$

(Greater than equal, since Δ_{ty} can be positive.) Which translates into $-v^{\Delta}_{\overline{\mathcal{H}}} + v_{\mathcal{H}-} - v_{\mathcal{H}+} \geq 0$, equivalent to $v^{\Delta}_{\overline{\mathcal{H}}} \leq v_{\mathcal{H}-} - v_{\mathcal{H}+}$ as we need. So the predicate persists through the loop iteration.

Since at each point $v_{\overline{\mathcal{H}}} \geq v_{\mathcal{A}}$, and the predicate $v_{\overline{\mathcal{H}}} \leq \ldots$ holds in the end of each loop, the predicate in the end with $v_{\mathcal{A}} \leq \ldots$ will also hold, and thus $\mathcal{A}$ cannot win the balance game.

11 "Not honest" in the context of notes means adversarial notes and also notes that are "burned" — that cannot be accessed by anyone.