

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

Felix Engelmann¹, Thomas Kerber², Markulf Kohlweiss^{2,3}, and Mikhail Volkhov^{3*}

¹ IT University of Copenhagen, Denmark

`fe-research@nlogn.org`

² IOHK

`thomas.kerber@iohk.io`

³ University of Edinburgh

`{markulf.kohlweiss,mikhail.volkhov}@ed.ac.uk`

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

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 [24]), 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 [25], 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 [1,5,13,7,14], much

* Corresponding Author

¹ <https://compound.finance/markets>

² <https://polygon.market.xyz/>

less is known about the *private exchange* of assets [12,9,15] — 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 [21] and privacy-friendly decentralized exchanges [3], often find it hard to balance practical applicability with the privacy guarantees they provide. 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 introduced by Zcash Sapling [17] to reduce the size of SNARK circuits for large transactions, and was inherited by the corresponding Shielded Assets³, or similar MASP⁴ multi-asset protocols. Another inspiration of our work is SwapCT, that realizes atomic swaps on Monero [12]. 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 [5] 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 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

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. 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 [5] 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 [12]. 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-OR⁵. 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 private non-interactive 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.

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 [19,18,23], 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

⁵ HIDing with Open Randomness

as ZEXE [6] 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. [11] propose a solution supporting multiple assets, but with no exchange mechanism, and with public asset types. Gao et al. [15] 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 [22] 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. [26] exists for the Mumblewimble 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 Stellar⁶, which instead uses shuffle proofs.

Finally, some works emphasize the exchange and offer matching functionality. Manta [9] describes a privacy-preserving decentralized exchange (DEX) based on an automated market 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 [10]. In contrast to both, we propose a more basic mechanism within a single blockchain, and 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 Zswap provides 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 $\text{ToSet}(S)$ 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. $(a, b) \leftarrow \text{foo}$ (function returns \perp or a different type), the execution aborts. The symbol $@$ denotes temporary variable assignment in the pattern-matching cases, e.g. $(a@(x, y), z) \leftarrow \text{Foo}()$ means $((x, y), z) \leftarrow \text{Foo}(); a \leftarrow (x, y)$.

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

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

populated, i.e. few domains have a non-zero value. Between domains, there exists no homomorphic property. Constructions were proposed by [22] and formalized by [8] which resemble vector Pedersen commitments with ad-hoc generators.

Definition 1 (Sparse Homomorphic Commitment). A SHC scheme consists of the algorithms ComSetup and Commit defined as follows:

$\mathbf{p} \leftarrow \text{ComSetup}(1^\lambda)$ takes the security parameter λ and outputs the public parameters \mathbf{p} implicitly provided to Commit . It specifies a type space \mathbb{T}
 $\text{com} \leftarrow \text{Commit}(\{\{\mathbf{ty}_i, a_i\}_{i=1}^n, \text{rc}\})$ takes a set of distinct types \mathbf{ty}_i and values a_i , and a randomness rc , and outputs a commitment com . We write $\text{Commit}(\mathbf{ty}, a, \text{rc})$ for $\text{Commit}(\{\{\mathbf{ty}, a\}, \text{rc}\})$.

Definition 2 (Homomorphism). A SHC scheme is homomorphic, if there exists an efficient operation \oplus , such that for any $\mathbf{ty} \in \mathbb{T}$ it holds that

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

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

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

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

Definition 4 (Commitment Hiding). For any $\lambda \in \mathbb{N}$ with $\mathbf{p} \leftarrow \text{Setup}(1^\lambda)$ and any stateful PPT adversary \mathcal{A} , it holds that

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

To instantiate a SHC we use a well-known Pedersen-based construction derived from the [22], 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 $H : \mathbb{T} \rightarrow \mathbb{G}$. Then $\text{Commit}(\{\{\mathbf{ty}_i, a_i\}, \text{rc}\}) := (\prod H(\mathbf{ty}_i)^{a_i}) G^{\text{rc}}$ with $a_i, \text{rc} \in \mathbb{Z}_q$.

To avoid value overflows in our protocol, we will upper-bound the committed values by $2^\alpha - 1$, and the total number of commitments that we homomorphically combine to β — in this way, the maximum sum in the exponent of a single base will be $\alpha + \beta$ bits, which must be below certain B which in turn must be reasonably smaller than the order of any base $H(\mathbf{ty})$ (thus the message space). We assume that this upper bound $B(\lambda)$ and the parameters $(\alpha(\lambda), \beta(\lambda))$, for simplicity, is given by the commitment scheme.

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).

<p style="margin: 0;">HID-OR$_{\mathcal{A}}^b(1^\lambda)$</p> <hr style="border: 0.5px solid black; margin: 2px 0;"/> <p style="margin: 0;">$\mathbf{p} \leftarrow \text{Setup}(1^\lambda)$</p> <p style="margin: 0;">$\{(\mathbf{ty}_t, a_t), (\mathbf{ty}'_t, a'_t)\}_{t=0}^1 \leftarrow \mathcal{A}(\mathbf{p})$</p> <p style="margin: 0;">$\Delta_{\mathbf{ty},t} := \{a_t \text{ if } \mathbf{ty}_t = \mathbf{ty} \text{ else } 0\} - \{a'_t \text{ if } \mathbf{ty}'_t = \mathbf{ty} \text{ else } 0\}$</p> <p style="margin: 0;">assert $\forall \mathbf{ty} \in \bigcup \mathbf{ty}_t \cup \bigcup \mathbf{ty}'_t : \Delta_{\mathbf{ty},0} = \Delta_{\mathbf{ty},1}$</p> <p style="margin: 0;">$r, r' \xleftarrow{\\$} \mathbb{R}$</p> <p style="margin: 0;">$\text{com} \leftarrow \text{Commit}(\mathbf{ty}_b, a_b, r); \text{com}' \leftarrow \text{Commit}(\mathbf{ty}'_b, a'_b, r')$</p> <p style="margin: 0;">$b' \leftarrow \mathcal{A}(\text{com}, \text{com}', \text{rc} = r - r')$</p> <p style="margin: 0;">return b'</p>

Fig. 1. Hiding with Open Randomness Game

Lemma 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} :*

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

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 (Setup, Prove, Verify, Sim). The first algorithm allows to create a common reference string (CRS), $\text{crs} \leftarrow \text{Setup}(1^\lambda)$ that is used as an input to Prove and Verify. To obtain a non-interactive proof that $\text{stmt} \in \mathcal{L}$ with the corresponding witness w (s.t. $(\text{stmt}, w) \in \mathcal{R}_{\mathcal{L}}$) one executes $\pi \leftarrow \text{NIZK.Prove}(\text{crs}, \text{stmt}, w)$. To then verify this prove one runs, correspondingly $\text{NIZK.Verify}(\text{crs}, \text{stmt}, \pi)$, which returns 0 or 1. To simulate the proof, one uses the the trapdoor τ created with Setup (but which is not used in the honest case), additionally to the CRS, and then passes this τ to $\text{Sim}(\text{crs}, \text{stmt}, \tau)$ to obtain a simulated proof. 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”*⁷, 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 **Receives** the note and is then able to create the corresponding nullifier with **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 5. *A One-Time-Account (OTA) scheme consists of the PPT algorithms (**Setup**, **KeyGen**, **Enc**, **Gen**, **Receive**, **NulEval**) defined as follows:*

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

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

$\text{note} \leftarrow \mathbf{Gen}(\mathfrak{pk}, \vec{a}, r)$: *takes a public key \mathfrak{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.*

$C \leftarrow \mathbf{Enc}(\mathfrak{pk}, (\vec{a}, r), \xi)$: *encrypts the attributes \vec{a} and the randomness $r \in \mathbb{S}$ to the public key \mathfrak{pk} with additional randomness $\xi \in \Xi$. It outputs a ciphertext C .*

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

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

*In addition, the algorithms **Gen** and **NulEval** must be efficiently provable in zero-knowledge. More formally, the construction must provide NIZK-friendly circuits for the following languages:*

$$\begin{aligned} \mathcal{L}^{\text{nul}} &= \{(\text{note}, \text{nul}) \mid \exists(\mathfrak{sk}, \vec{a}, r) : \text{note} = \mathbf{Gen}(P(\mathfrak{sk}), \vec{a}, r) \wedge \text{nul} = \mathbf{NulEval}(\mathfrak{sk}, r)\} \\ \mathcal{L}^{\text{open}} &= \{\text{note} \mid \exists(\mathfrak{pk}, \vec{a}, r) : \text{note} = \mathbf{Gen}(\mathfrak{pk}, \vec{a}, r)\} \end{aligned}$$

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

We first present the basic correctness and soundness properties of the OTA scheme, which primarily dictate how **Receive** should be implemented.

⁷ Unlike in some other works, “note” here means the final, hidden account; and not the plaintext coin.

Definition 6 (OTA Correctness). An OTA scheme is correct if for any $\lambda \in \mathbb{N}$ with $\mathbf{p} \in \text{Setup}(1^\lambda)$ it holds that any honestly generated note is receivable. Formally, for every $(\text{sk}, \text{pk}) \in \text{KeyGen}(\mathbf{p})$, every $(\vec{a}, r) \in \mathbb{M}^{|\vec{a}|} \times \mathbb{S}$ and $\xi \in \Xi$ it holds that

$$\text{Receive}(\text{Gen}(\text{pk}, \vec{a}, r), \text{Enc}(\text{pk}, (\vec{a}, r), \xi), \text{sk}) = (\vec{a}, r)$$

Definition 7 (OTA Soundness). An OTA scheme is sound if any non- \perp output of `Receive` reconstructs the note that was given to `Receive` as an input. Formally, for any $\lambda \in \mathbb{N}$ with $\mathbf{p} \in \text{Setup}(1^\lambda)$, every $(\text{sk}, \text{pk}) \in \text{KeyGen}(\mathbf{p})$, every $(\vec{a}, r) \in \mathbb{M}^{|\vec{a}|} \times \mathbb{S}$, $\xi \in \Xi$ it holds that

$$\text{Receive}(\text{note}, C, \text{sk}) = (\vec{a}, r) \implies \text{OTA.Gen}(P(\text{sk}), \vec{a}, r) = \text{note}$$

If correctness is present, soundness can be easily achieved by appending a condition to the realisation of `Receive` that asserts that extracted (\vec{a}, r) is valid.

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 8 (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 $\mathbf{p} \in \text{Setup}(1^\lambda)$ and any PPT adversary \mathcal{A} , it holds that

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

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 9 (OTA Privacy). Consider the following oracle, modelling OTA note receiving:

$$\begin{array}{l} \mathcal{O}_{\text{Rcv}}^{\text{note}^*, C^*}(i, \text{note}, C) : \\ \quad \text{assert } \text{note} \neq \text{note}^* \wedge C^* \neq C \\ \quad \text{return } \text{Receive}(\text{note}, C, \text{sk}_i) \end{array}$$

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

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

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 `noteb` according to the attribute vector.

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

Definition 10 (Note Uniqueness). *A binding and private OTA scheme satisfies honestly generated note uniqueness: for all PPT \mathcal{A} ,*

$$\Pr \left[\begin{array}{l} ((pk_0, \vec{a}_0), (pk_1, \vec{a}_1)) \leftarrow \mathcal{A}(1^\lambda) \\ r_0, r_1 \xleftarrow{\$} \mathbb{S} : \\ \text{Gen}(pk_0, \vec{a}_0, r_0) = \text{Gen}(pk_1, \vec{a}_1, r_1) \end{array} \right] \leq \text{negl}(\lambda)$$

The next property, similar to the tagging scheme of Omniring [20], 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 11 (Nullifier Pseudorandomness). *Let $\lambda \in \mathbb{N}$ with $p \in \text{Setup}(1^\lambda)$, $(sk, pk) \xleftarrow{\$} \text{KeyGen}()$, and f be randomly sampled function on the range $\{0, 1\}^{|\mathbb{S}|} \rightarrow \{0, 1\}^{|\text{NulEval}(sk, \cdot)|}$. An OTA scheme nullifier is pseudorandom if for any PPT adversary \mathcal{A} , it holds that*

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

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 $g^{\frac{1}{sk+r}}$ 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 12 (Nullifier Uniqueness). *An OTA scheme satisfies nullifier uniqueness if for any $\lambda \in \mathbb{N}$ with $p \in \text{Setup}(1^\lambda)$ and any PPT adversary \mathcal{A} , it holds that*

$$\Pr \left[\begin{array}{l} (sk_0, r_0, \vec{a}_0, sk_1, r_1, \vec{a}_1) \leftarrow \mathcal{A}(p); \\ \text{note}_0 \leftarrow \text{Gen}(P(sk_0), \vec{a}_0, r_0); \\ \text{note}_1 \leftarrow \text{Gen}(P(sk_1), \vec{a}_1, r_1) \end{array} : \begin{array}{l} \text{note}_0 = \text{note}_1 \wedge \\ \text{NulEval}(sk_0, r_0) \neq \text{NulEval}(sk_1, r_1) \end{array} \right] \leq \text{negl}(\lambda)$$

Finally, the generated nullifiers must be also collision-resistant:

Definition 13 (Nullifier Collision-Resistance). *An OTA scheme satisfies nullifier collision-resistance if for any $p \in \text{Setup}(1^\lambda)$ and PPT \mathcal{A} , it holds that*

$$\Pr \left[\begin{array}{l} (sk_0, r_0, sk_1, r_1) \leftarrow \mathcal{A}(p) : \\ \text{NulEval}(sk_0, r_0) = \text{NulEval}(sk_1, r_1) \end{array} \right] \leq \text{negl}(\lambda)$$

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 $\sigma_1, \dots, \sigma_n$, and Verify verifies a signature σ for a transaction tx . A signature σ is viewed separately from its transaction tx (created by $\text{tx} \leftarrow \text{CompleteTx}(\text{ptx})$ defined below), and not contained in it.

Many algorithms will make use of st – the current state of valid previously issued notes. It consists of two parts: st.MT is the Merkle tree containing notes as leaves, and st.NF is the set of used nullifiers. Intuitively, when the note is spendable, its commitment should be in st.MT ; when the note is spent, its (unique, unlinkable) nullifier goes into st.NF .

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 (pk^T, a^T, ty^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 tx_1, \dots, 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 $ptx = \text{BuildPTx}(st, \cdot, \cdot, \cdot)$ we have $\text{CheckPTx}(ptx, st) = 1$.
- **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.
- **CheckBalance** is merely an alias that checks that for each type the pre-transaction is balanced, and all the values are within bounds.

Definition 14. *A Zswap transaction scheme, built on top of an OTA scheme, consists of a tuple of PPT algorithms (**Setup**, **SignTx**, **MergeSig**, **Verify**) defined as follows:*

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

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

- $\mathcal{S} = \{(sk_i^S, note_i^S, nul_i, path_i, (a_i^S, ty_i^S), r_i^S)\}_{i=1}^{|\mathcal{T}|}$ is a set of inputs with a nullifier nul_i corresponding to the $note_i^S$, and stored in the current state at the given path $st.MT[path_i]$, secret key sk_i^S , amount a_i^S and type ty_i^S with input OTA notes’ randomness r_i^S .
- $\mathcal{T} = \{(pk_i^T, note_i^T, (a_i^T, ty_i^T), r_i^T)\}_{i=1}^{|\mathcal{T}|}$ is a set of (output) notes $note_i^T$ with amount a_i^T , type ty_i^T and output OTA notes’ randomness r_i^T .

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

$b \leftarrow \text{Verify}(\vec{st}, tx, \sigma)$ takes a transaction tx , a signature σ and returns a bit b representing the validity of the transaction w.r.t. the valid history of states \vec{st} (this history may be partial, or contain just one last element⁸).

$\sigma \leftarrow \text{MergeSig}(\{\sigma_i\}_{i=1}^n)$ takes n transaction signatures and generates a combined signature σ valid for the union of the transactions. To merge the corresponding transactions tx_1, \dots, 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

⁸ See the correctness definition in the next sections.

<p>BuildPTx(st, I, O, SK)</p> <pre> $\mathcal{S}, \mathcal{T} \leftarrow \emptyset$ for $(\text{note}^S, C^S) \in I$ do $\text{path} \leftarrow \text{st.MT.getPath}(\text{note}^S)$ assert $\text{path} \neq \perp$ $(\text{sk}^S, \text{nul}, (a^S, \text{ty}^S), r^S) \leftarrow \text{TryReceive}(\text{note}, C, \text{SK})$ $\mathcal{S} := \mathcal{S} \cup (\text{sk}^S, \text{note}^S, \text{nul}, \text{path}, (a^S, \text{ty}^S), r^S)$ assert $\{\text{nul}_i\}$ are distinct and $\forall i. \text{nul}_i \notin \text{st.NF}$ for $(\text{pk}^T, a^T, \text{ty}^T) \in O$ do $r^T \xleftarrow{\\$} \mathcal{S}, \xi \xleftarrow{\\$} \Xi$ $\text{note}^T \leftarrow \text{OTA.Gen}(\text{pk}^T, (a^T, \text{ty}^T), r^T)$ $C^T \leftarrow \text{OTA.Enc}(\text{pk}^T, ((a^T, \text{ty}^T), r^T), \xi)$ $\mathcal{T} := \mathcal{T} \cup (\text{pk}^T, \text{note}^T, C^T, (a^T, \text{ty}^T), r^T)$ $\text{ptx} \leftarrow (\mathcal{S}, \mathcal{T})$ assert $\text{CheckBalance}(\text{ptx}) = 1$ return ptx </pre> <p>CheckPTx(ptx@(\mathcal{S}, \mathcal{T}), st)</p> <pre> Parse \mathcal{S} as $\{(\text{sk}_i^S, \text{note}_i^S, \text{nul}_i, \text{path}_i, (a_i^S, \text{ty}_i^S), r_i^S)\}_{i=1}^{ \mathcal{S} }$ Parse \mathcal{T} as $\{(\text{pk}_i^T, \text{note}_i^T, C_i^T, (a_i^T, \text{ty}_i^T), r_i^T)\}_{i=1}^{ \mathcal{T} }$ % C_i^T is left to the receiver to verify assert $\{\text{nul}_i\}$ are distinct and $\forall i. \text{nul}_i \notin \text{st.NF}$ for $i \in [\mathcal{S}]$ do assert $\text{st.MT}[\text{path}_i] = \text{note}_i^S$ assert $\text{nul}_i = \text{OTA.NulEval}(\text{sk}_i^S, r_i^S)$ assert $\text{note}_i^S = \text{OTA.Gen}(\text{OTA.P}(\text{sk}_i^S), (a_i^S, \text{ty}_i^S), r_i^S)$ assert $\forall i \in [\mathcal{T}] :$ $\text{note}_i^T = \text{OTA.Gen}(\text{pk}_i^T, (a_i^T, \text{ty}_i^T), r_i^T)$ assert $\text{CheckBalance}(\text{ptx}) = 1$ return 1 </pre>	<p>CompleteTx(ptx@(\mathcal{S}, \mathcal{T}))</p> <pre> Parse \mathcal{S} as $\{(\cdot, \cdot, \text{nul}_i, \cdot, (a_i^S, \text{ty}_i^S), \cdot)\}_{i=1}^{ \mathcal{S} }$ Parse \mathcal{T} as $\{(\cdot, \text{note}_i^T, C_i^T, (a_i^T, \text{ty}_i^T), \cdot)\}_{i=1}^{ \mathcal{T} }$ for $\text{ty} \in \text{Ty} \ominus (\{\text{ty}_i^S\} \cup \{\text{ty}_i^T\})$ do $\Delta_{\text{ty}} \leftarrow \sum_{(a_i^S, \text{ty}) \in \mathcal{S}} a_i^S - \sum_{(a_i^T, \text{ty}) \in \mathcal{T}} a_i^T$ return $(\{\text{nul}_i\}_{i=1}^{ \mathcal{S} }, \{\text{note}_i^T, C_i^T\}_{i=1}^{ \mathcal{T} }, \{\Delta_{\text{ty}}\}_{\text{ty} \in \text{Ty}})$ </pre> <p>MergeTx($\{\text{tx}_j\}_{j=1}^n$)</p> <pre> Parse tx_j as $(\{\text{nul}_{j,i}\}_{i=1}^{ \mathcal{S} _j}, \{(\text{note}_{j,i}^T, C_{j,i}^T)\}_{i=1}^{ \mathcal{T} _j}, \{\Delta_{j,\text{ty}}\}_{\text{ty} \in \text{Ty}_j})$ assert $\sum \mathcal{S}_i + \sum \mathcal{T}_i \leq \beta$ return $(\text{ToSet}(\bigcup_{j,i} \{\text{nul}_{j,i}\}),$ $\text{ToSet}(\bigcup_{j,i} \{(\text{note}_{j,i}^T, C_{j,i}^T)\}),$ $\text{ToSet}(\{\sum_j \Delta_{j,\text{ty}}\}_{\text{ty} \in \bigcup_j \text{Ty}_j}))$ </pre> <p>TryReceive(note, C, SK)</p> <pre> for $\text{sk} \in \text{SK}$ do $\text{res} \leftarrow \text{OTA.Receive}(\text{note}, C, \text{sk})$ if $\text{res} = ((a, \text{ty}), r) \neq \perp$ then $\text{nul} \leftarrow \text{OTA.NulEval}(\text{sk}, r)$ return $(\text{sk}, \text{nul}, (a, \text{ty}), r)$ return \perp </pre> <p>CheckBalance(ptx@(\mathcal{S}, \mathcal{T}))</p> <pre> assert $\sum \mathcal{S} + \sum \mathcal{T} \leq \beta$ assert $\forall \text{ty} \in \{\text{ty}_i^S\} \cup \{\text{ty}_i^T\}.$ $\sum_{(a_i^S, \text{ty}) \in \mathcal{S}} a_i^S - \sum_{(a_i^T, \text{ty}) \in \mathcal{T}} a_i^T \geq 0 \wedge$ $(\forall i. a_i^S < 2^\alpha) \wedge (\forall i. a_i^T < 2^\alpha)$ return 1 </pre>
---	---

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

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 fulfills 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 transactions by themselves cannot be included in the public ledger. Both have a type with a negative balance. Alice's

transaction has $\Delta_\epsilon = -5$ 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.

<pre> $\mathcal{O}_{\text{Spend}}(\text{st}, I, O)$ if Spent is \perp then Spent $\leftarrow \emptyset$ assert $O > 0 \wedge \exists (\text{pk}, \cdot, \cdot) \in O : \text{pk} \in \text{PK}$ $\text{ptx} \in (\mathcal{S}, \mathcal{T}) \leftarrow \text{BuildPTx}(\text{st}, I, O, \text{SK})$ $\sigma \leftarrow \text{SignTx}(\text{st}, \text{ptx})$ Parse \mathcal{S} as $\{(\text{nul}_i, \cdot, \cdot)\}_{i=1}^{ \mathcal{S} }$; \mathcal{T} as $\{(\cdot, \text{note}_j^{\mathcal{T}}, C_j, \cdot, \cdot)\}_{j=1}^{ \mathcal{T} }$; Spent := Spent $\cup (\text{st}, I, \{\text{nul}_i\}_{i=1}^{ \mathcal{S} }, \{\text{note}_j^{\mathcal{T}}, C_j\}_{j=1}^{ \mathcal{T} })$ return σ </pre>	<pre> $\mathcal{O}_{\text{Insert}}(\text{st}, \text{tx}, \sigma)$ if Ins is \perp then $\text{Ins} \leftarrow \emptyset$ if $\text{st}_0 = \perp$ then assert $\text{st.NF} = \perp$ $\vec{\text{st}} \leftarrow \{\text{st}\}$ else assert $(\text{st}, \cdot, \cdot, \cdot) \in \text{Ins}$ $(\cdot, \vec{\text{st}}) \leftarrow \text{GetLog}(\text{st})$ $(\text{Nf} \{ \text{nul}_i \}_{i=1}^{ \mathcal{S} }, \{ \text{note}_j^{\mathcal{T}}, \cdot \}_{j=1}^{ \mathcal{T} }, \{ \Delta_{\text{ty}} \}_{\text{ty} \in \text{Ty}}) \leftarrow \text{tx}$ % Transaction verifies assert $\text{Verify}(\vec{\text{st}}, \text{tx}, \sigma) = 1$ % And is not an offer assert $\forall \text{ty} \in \text{Ty} : \Delta_{\text{ty}} \geq 0$ Construct st' s.t. $\text{st}'.\text{MT} \leftarrow \text{st}.\text{MT}.\text{insert}(\{ \text{note}_j^{\mathcal{T}} \}_{j=1}^{ \mathcal{T} })$ $\text{st}'.\text{NF} \leftarrow \text{st}.\text{NF} \cup \text{Nf}$ $\text{Ins} := \text{Ins} \cup (\text{st}, \text{st}', \text{tx}, \sigma)$ if $\text{st}_0 = \perp$ then $\text{st}_0 \leftarrow \text{st}$ </pre>
<pre> $\mathcal{O}_{\text{KeyGen}}()$ if SK or PK is \perp then SK, PK $\leftarrow \emptyset$ $(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}()$ SK := SK \parallel sk PK := PK \parallel pk return pk </pre>	<pre> GetLog(st) $\text{Ins}' \leftarrow \square$; $\vec{\text{st}} \leftarrow (\text{st})$ while $x \in (\text{st}', \text{st}, \cdot, \cdot) \in \text{Ins}$ do $\text{Ins}' := x \parallel \text{Ins}'$; $\text{st} := \text{st}'$ $\vec{\text{st}} = \vec{\text{st}} \parallel \text{st}'$ assert $\text{st} = \text{st}_0$ return $(\text{Ins}', \vec{\text{st}})$ </pre>

Fig. 3. Oracles for the security experiments.

4.2 Security Modelling with Support Oracles

The security of our scheme is based on several top-level games, plus the games we define for the OTA scheme. To model potential blockchain executions we need to introduce the oracles which will model note spending that the adversary \mathcal{A} sees, interactively. The relevant oracles, $\mathcal{O}_{\text{KeyGen}}$, $\mathcal{O}_{\text{Spend}}$ and $\mathcal{O}_{\text{Insert}}$ are presented on Fig. 3. They are responsible for generating keys, honest transactions and inserting them into the ledger.

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

$\mathcal{O}_{\text{Spend}}$ models leakage of honestly generated transactions, including unbalanced “offer” transactions. Each $\mathcal{O}_{\text{Spend}}$ 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.

$\mathcal{O}_{\text{Insert}}$ models “recording” balanced transaction in the ledger. Branching is possible inside $\mathcal{O}_{\text{Insert}}$ — \mathcal{A} can append an honest transaction to any of the recorded states, but still one can only spend notes through $\mathcal{O}_{\text{Insert}}$ 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. $\mathcal{O}_{\text{Insert}}$ only accepts states that are eventually linked with this root st_0 .

GetLog function is a state maintenance helper. When called on the state st it makes sure that st is a valid progression of the initial state st_0 , and returns two values: Ins' , which is a log of

this state progression (containing transaction history), and \vec{st} – subset of Ins' containing states only (only state progression history). When `GetLog` is called in security experiments on st , it implicitly asserts that st is a valid state that was created through the oracles.

4.3 Correctness

We start from the basic correctness definition, covering interactions between honest users.

Definition 15 (Correctness). *A Zswap scheme is correct if OTA is correct and if for all $\lambda \in \mathbb{N}$, $p \in \text{Setup}(1^\lambda)$, and all PPT \mathcal{B} (setup algorithm) it holds that:*

- (1) **Honestly generated transactions are immediately valid:** *For any $\mathcal{S}, \mathcal{T}, st$ such that $\text{CheckPTx}(\mathcal{S}, \mathcal{T}, st) = 1$, and $tx = \text{CompleteTx}(\mathcal{S}, \mathcal{T})$, and for any signature $\sigma \leftarrow \text{SignTx}(st, (\mathcal{S}, \mathcal{T}))$, it holds that $\text{Verify}([st], tx, \sigma) = 1$.*
- (2) **Honestly generated transactions are valid in any future state:** *Let st_1, st_2 be any pair of states returned by $\mathcal{B}^{\mathcal{O}_{\text{KeyGen}}, \mathcal{O}_{\text{Spend}}, \mathcal{O}_{\text{Insert}}}(p)$, such that $(\cdot, \vec{st}) \leftarrow \text{GetLog}(st_2)$ and $st_1 \in \vec{st}$ (this implies `GetLog` does not fail, and st_2 is a valid progression of st_1 ; st_1 may be equal to st_2). Let \mathcal{S}, \mathcal{T} be such that $\text{CheckPTx}(\mathcal{S}, \mathcal{T}, st_1) = 1$; let $tx = \text{CompleteTx}(\mathcal{S}, \mathcal{T})$ and $\sigma \leftarrow \text{SignTx}(st_1, (\mathcal{S}, \mathcal{T}))$. Then $\text{Verify}(\vec{st}, tx, \sigma) = 1$.*
- (3) **Honestly merged valid transactions are again valid:** *Let $n \in \mathbb{N}$. Let $\{st_i\}_{i=1}^n, st \leftarrow \mathcal{B}^{\mathcal{O}_{\text{KeyGen}}, \mathcal{O}_{\text{Spend}}, \mathcal{O}_{\text{Insert}}}(p, n)$ be a set of states such that (st_i, st) are valid progressions in the same history: $\vec{st} \leftarrow \text{GetLog}(st)$ and $\forall i. st_i \in \vec{st}$. Let there be a set of $\{(\mathcal{S}_i, \mathcal{T}_i)\}_{i \in [n]}$ s.t. for all i , $\text{CheckPTx}((\mathcal{S}_i, \mathcal{T}_i), st_i) = 1$, for each $\sigma_i \in \text{SignTx}(st_i, (\mathcal{S}_i, \mathcal{T}_i))$ and $\sigma = \text{MergeSig}(\{\sigma_i\}_{i=1}^n)$, and $tx \leftarrow \text{MergeTx}(\{\text{CompleteTx}(\mathcal{S}_i, \mathcal{T}_i)\}_{i=1}^n)$, it holds that $\text{Verify}(\vec{st}, tx, \sigma) = 1$.*

Regarding the ability to “receive” coins that were sent to the party, the formal completeness definition is part of the OTA scheme, and the fact we receive properly send coins is guaranteed by `BuildPTx`, which is not part of zswap scheme. At the same time, correctness of ciphertexts is not guaranteed, so it is possible to create an output with an invalid ciphertext, undecryptable by the receiver. In this case, receiver of the coin must consider this transaction factually invalid, as burning the coin that was designated to them.

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 split 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^* . Anti-theft subsumes the even more basic property that \mathcal{A} cannot spend honest notes without doing it honestly (asking honest parties through $\mathcal{O}_{\text{Spend}}$).

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 $\mathcal{O}_{\text{Spend}}$ to \mathcal{A} , but *used* only partially. To do that, it calls a sub-function `SplitTx` which locates complete “honest subtransactions” returned by $\mathcal{O}_{\text{Spend}}$ in tx , and returns these honest sub-transactions 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 $\mathcal{O}_{\text{Spend}}$, it means that \mathcal{A} managed to deconstruct it, changing the output of $\mathcal{O}_{\text{Spend}}$, or stealing an honest nullifier.

Anti-Theft $_{\mathcal{A}}(1^\lambda)$

```

p ← Setup(1λ)
st* ←  $\mathcal{A}^{\mathcal{O}_{\text{KeyGen}}, \mathcal{O}_{\text{Spend}}, \mathcal{O}_{\text{Insert}}}$ (p)
(Ins', ·) ← GetLog(st*)
for (st, ·, tx, ·) ∈ Ins' do
  (·, (NfA, MA)) ← SplitTx(st, tx)
  MA' ← {(note, C) ∈ MA | TryReceive(note, C, SK) ≠ ⊥}
  if ∃(·, ·, Nf, M) ∈ Spent : MA' ∩ M ≠ ∅ ∨ NfA ∩ Nf ≠ ∅ then
    return 1
return 0

```

SplitTx(st, tx)

```

(Nf@{nuli}i=1|S|, M@{notejT, ·}j=1|T|, C) ← tx
txH ← []
Nf0 ← Nf
for (st', ·, Nf'@{nul'i}i=1|S'|, M'@{(notejT', C'j)j=1|T'|}) ∈ Spent do
  if st'.MT = st.MT ∧ M' ⊂ M ∧ Nf' ⊂ Nf0 then
    txH = txH ∥ (Nf', M')
    Nf0 := Nf0 \ Nf'
txA = (Nf \ {NfH | (NfH, ·) ∈ txH}, M \ {MH} | (·, MH) ∈ txH})
return (txH, txA)

```

Fig. 4. The anti-theft experiment.

Definition 16 (Anti-Theft). A Zswap scheme is protecting against theft, if for any $\lambda \in \mathbb{N}$ and any PPT adversary \mathcal{A} , $\Pr[\text{Anti-Theft}_{\mathcal{A}}(1^\lambda) = 1] \leq \text{negl}(\lambda)$ where the $\text{Anti-Theft}_{\mathcal{A}}(1^\lambda)$ 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 $\mathcal{O}_{\text{Insert}}$ successfully, it wins the game, since 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 SplitTx .

On the more basic non-malleability side, \mathcal{A} cannot construct transactions spending honest notes. Assume that \mathcal{A} constructs tx₀ sending a note to an honest party ($\text{pk} \in \text{PK}$), and then succeeds to spend it using tx*, without using $\mathcal{O}_{\text{Spend}}$. If this is possible, it means \mathcal{A} presented a nullifier inside tx* corresponding to the note — then \mathcal{A} could also make a single query to $\mathcal{O}_{\text{Spend}}$ instructing to spend the same note to elsewhere, and ignore the result of $\mathcal{O}_{\text{Spend}}$, submitting tx* as planned. This would “mark” the nullifier, and trigger winning condition of the anti-theft game.

Modelling details. The requirement that O in $\mathcal{O}_{\text{Spend}}$ contains at least one honest output is needed so that SplitTx can uniquely identify honest sub-transactions⁹. In other words, note uniqueness is necessary for the anti-theft game to make sense. Without it the game would produce false positives: \mathcal{A} could trick SplitTx into not recognising some honest sub-transactions, even though the tx* submitted by \mathcal{A} is perfectly normal. This modelling artefact does not limit \mathcal{A} from creating unbalanced input-only transactions, since \mathcal{A} can still request to include a single zero-valued output.

No similar restrictions are put on the inputs, and they can be null; in other words, \mathcal{A} can instruct to generate an offer with a single honest output. However, it is easy to see that \mathcal{A}

⁹ Without this requirement, and without adding extra marker information to notes, identifying honest sub-transactions within SplitTx would take exponential time.

cannot trigger the winning condition with this input — since the output notes are unique, it is guaranteed that this output note will be detected in `SplitTx`. This does not rule out the possibility of \mathcal{A} using this output in other way to break the property.

The set Nf_0 in `SplitTx` is needed since without it honest sub-transactions can be counted twice. E.g. let tx_0, 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 $M_0 \cup M_1$, `SplitTx` will detect both tx_0, 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 `SplitTx` simply does not change the behaviour of the game.

Finally, it is critical to filter $M^{\mathcal{A}'}$ 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 $\mathcal{O}_{\text{Spend}}$, requests an honest spend to an adversarial public key $\text{pk}^{\mathcal{A}}$;
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 $\text{note}^* \in M \cap M^{\mathcal{A}}$.

4.5 Balance

The balance property says 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 — both within a single transaction, and for any history of adversarial interaction with the ledger.

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 than it could have obtained honestly. The challenger lets \mathcal{A} interact with the oracles, and asks \mathcal{A} to present two sets of values: I_0 and I containing notes together with their secrets. Unlike in other games, st_0 in balance must contain only adversarial coins, and I_0 must be all the unspent corresponding notes. The other set I is the set of unspent adversarial coins from the new state st^* that (supposedly, as \mathcal{A} claims) contain illicitly produced coins, breaking the global balance condition. This balance condition is computed next in the following manner. Initially, 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 (how much per type \mathcal{A} owned in the beginning of the game, and how much it owns in the end). 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 (located in transaction outputs).

The final condition checks that \mathcal{A} cannot show in st^* more coins than: (1) it had in st_0 , plus (2) what was sent by honest parties, minus what was received by honest parties. This last difference is non-positive (because I_0 is all the system coins), and importantly the balance of \mathcal{A} per type is “tied” to the balance of honest parties, so no extra coins can be produced.

Note that balance is defined in conjunction with anti-theft, since it uses `SplitTx` which it assumes to work correctly (according to the anti-theft definition). In practice it also makes sense to see them together: balance and anti-theft jointly guarantee that adversary cannot receive more than honestly, *given* that it can combine its transactions with dishonest ones. Theft guarantees that \mathcal{A} can only merge transactions; balance guarantees that these transactions never break the total balance of coins in the system.

In particular, the balance property guarantees the following:

```

Balance $\mathcal{A}, \mathcal{E}_{\mathcal{A}}$ ( $1^\lambda$ )
p  $\leftarrow$  Setup( $1^\lambda$ )
(st*, I0, I)  $\leftarrow$   $\mathcal{A}^{\mathcal{O}_{\text{KeyGen}}, \mathcal{O}_{\text{Spend}}, \mathcal{O}_{\text{Insert}}}$ (p)
assert {notei}  $\in$  I0 are distinct, and
    st0.MT contains only these notes.
assert {notei}  $\in$  I are distinct, and are in st*.MT
for (notei, nul, (a, ty), sk, r)  $\in$  I  $\cup$  I0 do
    assert notei = Gen(P(sk), (a, ty), r)  $\wedge$  nul = NulEval(sk, r)
for ( $\cdot$ , nul, (a, ty),  $\cdot$ ,  $\cdot$ )  $\in$  I0 do
    assert nul  $\notin$  st0.NF
    v0[ty] := v0[ty] + a
for ( $\cdot$ , nul, (a, ty),  $\cdot$ ,  $\cdot$ )  $\in$  I do
    assert nul  $\notin$  st*.NF
    v $\mathcal{A}$ [ty] := v $\mathcal{A}$ [ty] + a
v $\mathcal{H}_-$ , v $\mathcal{H}_+$   $\leftarrow$  (ty  $\mapsto$  0) % map with default value 0
(Ins',  $\cdot$ )  $\leftarrow$  GetLog(st*)
for all (st,  $\cdot$ , tx,  $\sigma$ )  $\in$  Ins' do
    ({nuli}i=1|S|, {notejT,  $\cdot$ }j=1|T|,  $\emptyset$ )  $\leftarrow$  tx
    (tx $\mathcal{H}$ ,  $\cdot$ )  $\leftarrow$  SplitTx(st, tx)
    for (Nf, M)  $\in$  tx $\mathcal{H}$  do
        Find( $\cdot$ , I, Nf, M)  $\in$  Spent
        for (note', C')  $\in$  I do
            ( $\cdot$ , (a, ty),  $\cdot$ )  $\leftarrow$  TryReceive(note', C', SK)
            % Honestly spent inputs
            v $\mathcal{H}_-$ [ty] := v $\mathcal{H}_-$ [ty] + a
        for (note', C')  $\in$  M where ( $\cdot$ , M)  $\in$  tx do
            res  $\leftarrow$  TryReceive(note', C', SK)
            if res = ( $\cdot$ , (a, ty),  $\cdot$ )  $\neq \perp$  then
                % Outputs to honest parties
                v $\mathcal{H}_+$ [ty] := v $\mathcal{H}_+$ [ty] + a
if  $\exists$  ty : v $\mathcal{A}$ [ty] > v0[ty] + v $\mathcal{H}_-$ [ty] - v $\mathcal{H}_+$ [ty] then
    return 1
else return 0

```

Fig. 5. The balance experiment

1. Total spendable amount of coins per type is constant in the system: if there are more coins in the system than v_0 , and they are spendable, \mathcal{A} can transfer them to itself, and present them in I , thus breaking $v_{\mathcal{A}} > v_0$. Even if these coins belong to the honest users, and thus $v_{\mathcal{H}_-} - v_{\mathcal{H}_+} > 0$ (which should not happen), \mathcal{A} can always create a transaction transferring these coins to itself using $\mathcal{O}_{\text{Spend}}$.
2. Every transaction is balanced per type, that is it does not produce more coins than it consumes. This means no conversion between types, and no coin forging. It follows from (1), since an unbalanced transaction would create more funds in the state st' that follows this transaction. So \mathcal{A} could just present state st' and win the balance game.
3. Double spending is forbidden. Because we have balancing, it is not possible to spend a note in such a way so that its funds disappear from the counter of the total coins. This means that double spending would produce total coin imbalance, which is forbidden.

Definition 17 (Balance). A Zswap scheme is balanced if for all PPT adversaries \mathcal{A} there exists a PPT extractor $\mathcal{E}_{\mathcal{A}}$ such that $\Pr[\text{Balance}_{\mathcal{A}, \mathcal{E}_{\mathcal{A}}}(1^\lambda) = 1] \leq \text{negl}(\lambda)$ with the game defined in Fig. 5.

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 ($\{\text{note}_{i,j,C_{i,j}}\}_{j=1}^{|\mathcal{S}|_i}$, $\{(\text{pk}_{i,j}, a_{i,j}^{\mathcal{T}}, \text{ty}_{i,j}^{\mathcal{T}})\}_{j=1}^{|\mathcal{T}|_i}$), similarly to the input to $\mathcal{O}_{\text{Spend}}$, or
2. a fully adversarial transaction (tx_i, σ_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: $\text{EquivTree}(T_0, T_1) = 1$ if the following conditions hold:

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. This applies to honest ($B_2^S \wedge B_2^{\mathcal{T}}$) leaves, as the same property implicitly holds for adversarial leaves due to the next check (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 $\mathcal{O}_{\text{Spend}}$, 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).

<p>Privacy$_{\mathcal{A}}^b(1^\lambda)$</p> <pre> p ← Setup(1^λ) (st, T₀, T₁) ← A^{KeyGen · O_{Spend} · O_{Insert}}(p) assert (·, st̄) ← GetLog(st) % st̄[1] = st tx₀ ← EvalTree(st̄, T₀) tx₁ ← EvalTree(st̄, T₁) assert tx₀ ≠ ⊥ ∧ tx₁ ≠ ⊥ assert EquivTree(st, T₀, T₁) = 1 b' ← A(tx₀) return b' </pre> <p>EvalTree(st̄, T)</p> <pre> for all leaf_i ∈ T do % Validate adversarial txs if leaf_i = (tx, σ) then assert Verify(st̄, tx, σ) = 1 % Replace instruction leaves with real txs if leaf_i = (I, O) then ptx ← BuildPTx(st̄[1], I, O, SK) tx_i ← CompleteTx(ptx) σ_i ← SignTx(st̄[1], ptx) Replace leaf_i by (tx_i, σ_i) in T % Fold T into a single root node by merging while ∃ node N in T with only {(tx_i, σ_i)}^c_{i=1} as children do Replace node with merged transactions of its children: N ← (MergeTx({tx_i}^c_{i=1}, MergeSig({σ_i}^c_{i=1}))) Remove its children return T </pre>	<p>EquivTree(st, T₀, T₁)</p> <pre> Parse each leaf_{b,i} of each T_b as either an adv. leaf leaf^A: (tx_{b,i}, @({nul_{b,i,j}}^S_{j=1}^{S _{b,i}}, {(note_{b,i,j}^T, C_{b,i,j}^T)_{j=1}^{T _{b,i}}}, {Δ_{b,i,ty}}_{ty ∈ Ty_{b,i}}, σ_{b,i}) ← leaf^A_{b,i} or an honest leaf leaf^H: ({(note_{b,i,j}^S, C_{b,i,j}^S)_{j=1}^{S _{b,i}}, {(pk_{b,i,j}, a_{b,i,j}^T, ty_{b,i,j}^T)_{j=1}^{T _{b,i}}} ← leaf^H_{b,i} for b ∈ {0, 1}, leaf^H_i ∈ T_b do for j ∈ [T_{b,i}] do (nul_{b,i,j}, (a_{b,i,j}^S, ty_{b,i,j}^S), r_{b,i,j}) ← TryReceive(note_{b,i,j}^S, C_{b,i,j}^S, SK) for ty ∈ {ty_{b,i,j}^S}^{T _{b,i}} do Δ_{b,i,ty} ← ∑_{j:ty_{b,i,j}^S=ty} a_{b,i,j}^S − ∑_{j:ty_{0,i,j}^S=ty} a_{0,i,j}^T B₁ ← ∀ty : ∑_{leaf^H_i ∈ T₀} Δ_{0,i,ty} = ∑_{leaf^H_i ∈ T₁} Δ_{1,i,ty} B₂^S ← ∑_{leaf^H_i ∈ T₀} S _{0,i} = ∑_{leaf^H_i ∈ T₁} S _{1,i} B₂^T ← ∑_{leaf^H_i ∈ T₀} T _{0,i} = ∑_{leaf^H_i ∈ T₁} T _{1,i} B₃ ← ∪_{leaf^A_i ∈ T₀} {(tx_{0,i}, σ_{0,i})} = ∪_{leaf^A_i ∈ T₁} {(tx_{1,i}, σ_{1,i})} Nf ← ∪_{(·, ·, Nf_i, ·) ∈ Spent} Nf_i B₄ ← Nf ∩ (∪_{leaf^H_i ∈ T₀} nul_{0,i,j}) = Nf ∩ (∪_{leaf^H_i ∈ T₁} nul_{1,i,j}) for b ∈ {0, 1}, leaf^H_i ∈ T_b do O_b^A ← ∪_{pk_{b,i,j} ∈ PK} (pk_{b,i,j}, a_{b,i,j}^S, ty_{b,i,j}^S) B₅ ← O₀^A = O₁^A return B₁ ∧ B₂^S ∧ B₂^T ∧ B₃ ∧ B₄ ∧ B₅ </pre>
---	--

Fig. 6. Transaction privacy experiment

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 trees that contains a single leaf node, in the first case spending a single note with X coins, sending 1 to Alice and $X - 1$ 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 is produced by which tree.

Definition 18 (Transaction Privacy). *A Zswap scheme has private transactions, if for all PPT adversaries \mathcal{A} it holds that:*

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

with $\text{Privacy}_{\mathcal{A}}^b(1^\lambda)$ defined in Fig. 6.

5 The 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): SHC = (ComSetup, Commit). and Arguments of Knowledge: AoK = (AoK.Setup, AoK.Prove, AoK.Verify, AoK.Sim).

<p>Setup(1^λ)</p> <pre> P_{SHC} ← ComSetup(1^λ) P_{OTA} ← OTA.Setup(1^λ) P_{spend} ← AoK[$\mathcal{L}^{\text{spend}}$].Setup($1^\lambda$) P_{output} ← AoK[$\mathcal{L}^{\text{output}}$].Setup($1^\lambda$) P := (P_{SHC}, P_{OTA}, P_{spend}, P_{output}) return p </pre> <p>SignTx(st, ptx@(S, T))</p> <pre> Parse S as {(sk_i^S, note_i^S, nul_i, path_i, (a_i^S, ty_i^S), r_i^{S})}_{i=1}^{ S } Parse T as {(pk_i, note_i^T, C_i^T, (a_i^T, ty_i^T), r_i^{T})}_{i=1}^{ \mathcal{T} } % Rerandomized input/output commitments {rc_i^S ←^{\\$} \mathbb{R}, com_i^S ← Commit(ty_i^S, a_i^S; rc_i^S)}_{i=1}^{ S } {rc_i^T ←^{\\$} \mathbb{R}, com_i^T ← Commit(ty_i^T, a_i^T; rc_i^T)}_{i=1}^{ \mathcal{T} } {stmt_i^S = (st, nul_i, com_i^S)}_{i=1}^{ S } {w_i^S = (path_i, sk_i^S, a_i^S, ty_i^S, r_i^S, rc_i^S)}_{i=1}^{ S } {\pi_i^S ← AoK[$\mathcal{L}^{\text{spend}}$].Prove(stmt_i^S, w_i^S)}_{i=1}^{ S } {w_i^T = (pk_i, a_i^T, ty_i^T, r_i^T, rc_i^T)}_{i=1}^{ \mathcal{T} } {\pi_i^T ← AoK[$\mathcal{L}^{\text{output}}$].Prove(stmt_i^T, w_i^T)}_{i=1}^{ \mathcal{T} } return (ToSet({(\pi_i^S, com_i^S)}_{i=1}^{ S }), ToSet({(\pi_i^T, com_i^T)}_{i=1}^{ \mathcal{T} }), \sum_{i=1}^{ S } rc_i^S - \sum_{i=1}^{ \mathcal{T} } rc_i^T)}}</pre>	<p>Verify($\vec{\text{st}}$, tx, σ)</p> <pre> Parse tx as ((nul_i)}_{i=1}^{ S }, {(note_i^T, C_i^T)}_{i=1}^{ \mathcal{T} }, {\Delta_{ty}}_{ty \in \text{Ty}}) assert S + \mathcal{T} ≤ β assert {nul_i} are distinct assert \forall i : nul_i \notin \vec{\text{st}}[1].NF Parse \sigma as (({\pi_i^S, com_i^S)}_{i=1}^{ S }, {({\pi_i^T, com_i^T)}_{i=1}^{ \mathcal{T} }, rc) for i \in [S] do Find \hat{i} such that tx was created w.r.t. \vec{\text{st}}[\hat{i}]. stmt_i := (\vec{\text{st}}[\hat{i}], nul_i, com_i^S) assert AoK[$\mathcal{L}^{\text{spend}}$].Verify(\pi_i^S, stmt_i) for j \in [\mathcal{T}] do assert AoK[$\mathcal{L}^{\text{output}}$].Verify(\pi_j^T, (note_j^T, C_j^T, com_j^T)) return \oplus_{i=1}^{ S } com_i^S \oplus \oplus_{i=1}^{ \mathcal{T} } com_i^T = Commit(0, 0; rc) \oplus \oplus_{ty \in \text{Ty}} Commit(ty, \Delta_{ty}, 0). </pre> <p>MergeSig({\sigma_j}_{j=1}^n)</p> <pre> Parse \sigma_j as (({\pi_i^{S_j}, com_i^{T_j})}_{i=1}^{ S_j }, {({\pi_i^{T_j}, com_i^{T_j})}_{i=1}^{ \mathcal{T}_j }, rc_j) assert \sum S_i + \sum \mathcal{T}_i ≤ β rc ← \sum_{j=1}^n rc_j return (ToSet(\bigcup_{j=1}^n {(\pi_i^{S_j}, com_i^{T_j})}_{i=1}^{ S_j }), ToSet(\bigcup_{j=1}^n {(\pi_i^{T_j}, com_i^{T_j})}_{i=1}^{ \mathcal{T}_j }), rc) </pre>
---	--

Fig. 7. The Zswap Construction

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 com^S is well-formed and contains the same value and type as a note in the Merkle tree of state st with the given nullifier nul . To prevent overflows in the homomorphic commitments, we include a range proof where α is chosen small enough in relation to the group order (maximum inputs and outputs times $2^\alpha < |\mathbb{G}|$). Thereby we assume integer amounts in subsequent arguments.

$$\begin{aligned} \mathcal{L}^{\text{spend}} = \{ & (\text{st}, \text{nul}, \text{com}^S) \mid \exists(\text{path}, \text{note}, \text{sk}^S, a^S, \text{ty}^S, r^S, \text{rc}^S) : \\ & \text{st.MT}[\text{path}] = \text{note} \wedge \\ & (\text{note}, \text{nul}; \text{sk}^S, (a^S, \text{ty}^S), r^S) \in \mathcal{L}^{\text{nul}} \wedge \\ & (\text{note}; \text{OTA}.P(\text{sk}^S), (a^S, \text{ty}^S), r^S) \in \mathcal{L}^{\text{open}} \wedge \\ & \text{com}^S = \text{Commit}(\text{ty}^S, a^S; \text{rc}^S) \wedge a^S \in [2^\alpha] \} \end{aligned}$$

The second language $\mathcal{L}^{\text{output}}$ 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.

$$\begin{aligned} \mathcal{L}^{\text{output}} = \{ & (\text{note}^T, C^T, \text{com}^T) \mid \exists(\text{note}, \text{pk}^T, a^T, \text{ty}^T, r^T, \text{rc}^T) : \\ & (\text{note}; \text{pk}, (a^T, \text{ty}^T), r^T) \in \mathcal{L}^{\text{open}} \wedge \\ & \text{com}^T = \text{Commit}(\text{ty}^T, a^T; \text{rc}^T) \wedge a^T \in [2^\alpha] \} \end{aligned}$$

Note that the ciphertext C^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 π_i^S for each input together with the SHC com_i^S and for each output a signature π_i^T and the SHC com_i^T . 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.

6 Security Proof

In this section we prove the main three security properties of Zswap construction on Fig. 7 we introduced in Section 4. The full proofs are deferred to Appendices D, E, and F.

Theorem 1 (Anti-Theft). *The Zswap protocol prevents theft (Definition 16), 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 $\mathcal{O}_{\text{Spend}}$ query \hat{E} , producing tx , such that not all nullifiers and notes from tx were included in tx^* . This is the query that triggers the winning condition. 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 drops some other. Assuming 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 tx, 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 $(a_1, \text{ty}_1), (a_2, \text{ty}_2)$ as they should be honestly; or
2. the values swapped as $(a_2, \text{ty}_1), (a_1, \text{ty}_2)$ if both indices correspond to inputs or both to outputs, and
3. swapped with negation $(-a_2, \text{ty}_2), (-a_1, \text{ty}_1)$ otherwise.

This way of embedding guarantees the HID-OR requirement that $\Delta_{\text{ty},1} = \Delta_{\text{ty},2}$. 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 $\text{rc}^* - \sum \text{rc}_i$). 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 Appendix D. □

Theorem 2 (Balance). *The Zswap protocol satisfies the Balance property (Definition 17) by anti-theft, NIZK SE, OTA Security, commitment binding, and Merkle tree binding.*

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. □

Theorem 3 (Privacy). *The Zswap protocol is private (Definition 18), 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 $\text{EquivTree}(T, T') = 1$ and $\text{EvalTree}(\text{st}, T) = \text{EvalTree}(\text{st}, T')$. 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 MergeTx and 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 $\text{tx}_{\mathcal{H},0}$ and $\text{tx}_{\mathcal{H},1}$, 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 **EquivTree**. And thus if $\text{tx}_{\mathcal{H},b}$ is empty, $\text{tx}_{\mathcal{H},1-b}$ 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:

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

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

$$\left(\{\text{nul}_i\}^{|\mathcal{S}|}, \{(\text{note}_i, C_i)\}^{|\mathcal{T}|}, \{\Delta_{\text{ty}}\}, \sigma_i \otimes \left(\{(\pi_i^{\mathcal{S}}, \text{com}_i^{\mathcal{S}})\}^{|\mathcal{S}|}, \{(\pi_i^{\mathcal{T}}, \text{com}_i^{\mathcal{T}})\}^{|\mathcal{T}|}, \text{rc} \right) \right)$$

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 $\{\Delta_{\text{ty}}\}$ set is the same by B_1 ,
3. the set of (honest) input nullifiers $\{\text{nul}_i\}^{|\mathcal{S}|}$ contains: nullifiers that \mathcal{A} received from $\mathcal{O}_{\text{Spend}}$ — these are the same by B_4 ; and nullifiers unknown previously to \mathcal{A} — deterministic but indistinguishable by nullifier pseudorandomness,
4. $\{(\text{note}_i, C_i)\}^{|\mathcal{T}|}$ 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 $\text{com}_i^{\mathcal{S}}, \text{com}_i^{\mathcal{T}}$ 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 1).
 - (5.3) joint randomness rc — the total randomness is always the sum of the individual rc_i of internal nodes. If at least one node in the tree is honest, the final rc is uniform in both worlds, so perfectly indistinguishable. If there are no honest nodes, then the final rc is exactly equal in both worlds (by B_3 malicious leaves must be the same).

The formal reduction is described in Appendix F. □

7 Implementation

To show that our construction is practical, we developed a prototype implementation available online¹⁰. We use the rust framework **ark-works**¹¹ and their **Groth16** SNARK library. For an ef-

¹⁰ <https://github.com/felix-engelmann/zswap-code>

¹¹ <https://arkworks.rs>

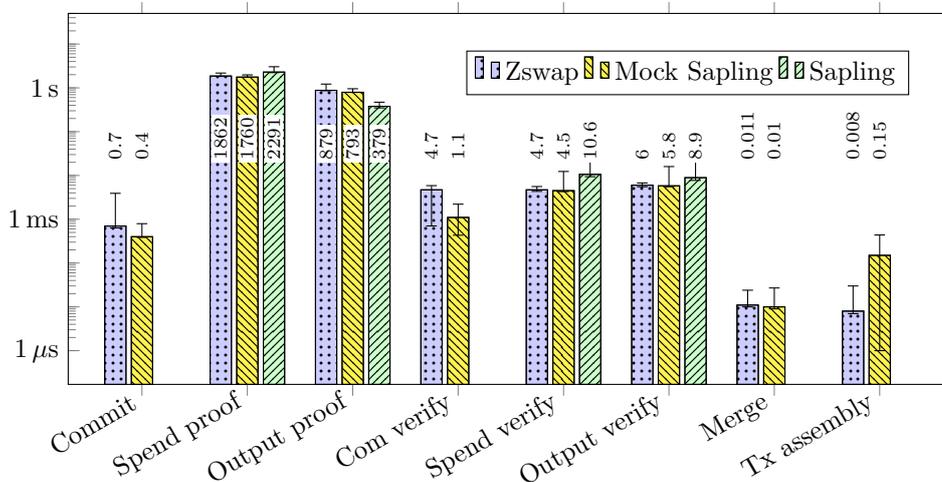


Fig. 8. Comparison of our protocol to our Mock Sapling implementation with the same hash function and the original Zcash. The measurements of procedures is in milliseconds.

efficient hashing circuit, we use Poseidon [16]. As the construction is similar to the Zcash Sapling version, we use this as performance comparison. 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 implementation¹² 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).

The median, min and max runtimes of 30 executions on a 6th 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

¹² <https://github.com/zcash/zcash>

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. Alonso, K.M., Joancomartí, J.H.: Monero - privacy in the blockchain. Cryptology ePrint Archive, Report 2018/535 (2018), <https://eprint.iacr.org/2018/535>
2. Baghery, K., Kohlweiss, M., Siim, J., Volkhov, M.: 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. Baum, C., David, B., Frederiksen, T.K.: P2dex: Privacy-preserving decentralized cryptocurrency exchange. In: Sako, K., Tippenhauer, N.O. (eds.) Applied Cryptography and Network Security. pp. 163–194. Springer International Publishing, Cham (2021)
4. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 566–582. Springer, Heidelberg (Dec 2001). https://doi.org/10.1007/3-540-45682-1_33
5. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE Computer Society Press (May 2014). <https://doi.org/10.1109/SP.2014.36>
6. Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: ZEXE: Enabling decentralized private computation. In: 2020 IEEE Symposium on Security and Privacy. pp. 947–964. IEEE Computer Society Press (May 2020). <https://doi.org/10.1109/SP40000.2020.00050>
7. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 423–443. Springer, Heidelberg (Feb 2020). https://doi.org/10.1007/978-3-030-51280-4_23
8. Campanelli, M., Engelmann, F., Orlandi, C.: 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>
9. Chu, S., Xia, Q., Zhang, Z.: Manta: Privacy preserving decentralized exchange. Cryptology ePrint Archive, Report 2020/1607 (2020), <https://ia.cr/2020/1607>
10. Deshpande, A., Herlihy, M.: Privacy-preserving cross-chain atomic swaps. In: Bernhard, M., Bracciali, A., Camp, L.J., Matsuo, S., Maurushat, A., Rønne, P.B., Sala, M. (eds.) FC 2020 Workshops. LNCS, vol. 12063, pp. 540–549. Springer, Heidelberg (Feb 2020). https://doi.org/10.1007/978-3-030-54455-3_38
11. Ding, D., Li, K., Jia, L., Li, Z., Li, J., Sun, Y.: Privacy protection for blockchains with account and multi-asset model. China Communications **16**(6), 69–79 (2019)
12. Engelmann, F., Müller, L., Peter, A., Kargl, F., Bösch, C.: SwapCT: Swap confidential transactions for privacy-preserving multi-token exchanges. PoPETs **2021**(4), 270–290 (Oct 2021). <https://doi.org/10.2478/popets-2021-0070>

13. Fauzi, P., Meiklejohn, S., Mercer, R., Orlandi, C.: Quisquis: A new design for anonymous cryptocurrencies. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part I. LNCS, vol. 11921, pp. 649–678. Springer, Heidelberg (Dec 2019). https://doi.org/10.1007/978-3-030-34578-5_23
14. Fuchsbauer, G., Orrù, M., Seurin, Y.: Aggregate cash systems: A cryptographic investigation of Mumblewimble. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 657–689. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17653-2_22
15. Gao, Z., Xu, L., Kasichainula, K., Chen, L., Carbutar, B., Shi, W.: Private and atomic exchange of assets over zero knowledge based payment ledger. arXiv preprint arXiv:1909.06535 (2019)
16. Grassi, L., Kales, D., Khovratovich, D., Roy, A., Rechberger, C., Schafneger, M.: Starkad and Poseidon: New hash functions for zero knowledge proof systems. Cryptology ePrint Archive, Report 2019/458 (2019), <https://eprint.iacr.org/2019/458>
17. Hopwood, D., Bove, S., Hornby, T., Wilcox, N.: Zcash protocol specification. GitHub: San Francisco, CA, USA (2016)
18. Kerber, T., Kiayias, A., Kohlweiss, M.: Kachina—foundations of private smart contracts. In: 2021 IEEE 34th Computer Security Foundations Symposium (CSF). pp. 1–16. IEEE (2021)
19. Kosba, A.E., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE Symposium on Security and Privacy. pp. 839–858. IEEE Computer Society Press (May 2016). <https://doi.org/10.1109/SP.2016.55>
20. Lai, R.W.F., Rong, V., Ruffing, T., Schröder, D., Thyagarajan, S.A.K., Wang, J.: Omniring: Scaling private payments without trusted setup. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 31–48. ACM Press (Nov 2019). <https://doi.org/10.1145/3319535.3345655>
21. Meiklejohn, S., Mercer, R.: Möbius: Trustless tumbling for transaction privacy. PoPETs **2018**(2), 105–121 (Apr 2018). <https://doi.org/10.1515/popets-2018-0015>
22. Poelstra, A., Back, A., Friedenbach, M., Maxwell, G., Wuille, P.: Confidential assets. In: Zohar, A., Eyal, I., Teague, V., Clark, J., Bracciali, A., Pintore, F., Sala, M. (eds.) FC 2018 Workshops. LNCS, vol. 10958, pp. 43–63. Springer, Heidelberg (Mar 2019). https://doi.org/10.1007/978-3-662-58820-8_4
23. Steffen, S., Bichsel, B., Gersbach, M., Melchior, N., Tsankov, P., Vechev, M.T.: zkay: Specifying and enforcing data privacy in smart contracts. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 1759–1776. ACM Press (Nov 2019). <https://doi.org/10.1145/3319535.3363222>
24. Werner, S.M., Perez, D., Gudgeon, L., Klages-Mundt, A., Harz, D., Knottenbelt, W.J.: Sok: Decentralized finance (defi). arXiv preprint arXiv:2101.08778 (2021)
25. Xu, J., Vavryk, N., Paruch, K., Cousaert, S.: Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. arXiv preprint arXiv:2103.12732 (2021)
26. Yi, Z., Ye, H., Dai, P., Tongcheng, S., Gelfer, V.: Confidential assets on MumbleWimble. 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 $\forall(\text{stmt}, w) \in \mathcal{R}_{\mathcal{L}}$ we have

$$\Pr \left[(\text{crs}, \cdot) \leftarrow \text{Setup}(\mathcal{R}_{\mathcal{L}}) : \text{Verify}(\text{crs}, \text{stmt}, \text{Prove}(\text{crs}, \text{stmt}, w)) = 1 \right] = 1$$

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

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

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

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

The oracle $\mathcal{S}_{b,\text{crs},\tau}$ on input (stmt, w) asserts that $(\text{stmt}, w) \in \mathcal{R}_\lambda$ and then returns $\pi = \text{Prove}(\text{crs}, \text{stmt}, w)$ if $b = 0$, and $\pi = \text{Sim}(\text{crs}, \tau, \text{stmt})$ 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 19 (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}_λ ,*

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

where $\mathcal{S}_{\text{crs},\tau}(\text{stmt})$ is a simulator oracle that calls $\text{Sim}(\text{crs}, \tau, \text{stmt})$ 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. $\text{stmt} = (\text{stmt}_1, \text{stmt}_2)$ and $\text{stmt}_1 = f(w_1)$ 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 PRF of type PRF^{addr} must in addition be collision-resistant. The randomness space is $\mathbb{S} := (\{0, 1\}^\lambda)^3$, the key space is 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.

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

$(\text{sk}, \text{pk}) \leftarrow \text{KeyGen}()$: Samples a_{sk} randomly from $\{0, 1\}^\lambda$ and defines $a_{\text{pk}} \leftarrow \text{PRF}_{a_{\text{sk}}}^{\text{addr}}(0)$; samples an encryption key pair $\text{sk}_{\text{enc}}, \text{pk}_{\text{enc}}$ using the corresponding PKE algorithm. Return $\text{sk} = (a_{\text{sk}}, \text{sk}_{\text{enc}})$, $\text{pk} = (a_{\text{pk}}, \text{pk}_{\text{enc}})$.

$P(\text{sk})$: is defined as $P(a_{\text{sk}}, \text{sk}_{\text{enc}}) := (\text{PRF}_{a_{\text{sk}}}^{\text{addr}}(0), P_{\text{enc}}(\text{sk}_{\text{enc}}))$ where P_{enc} is assumed to be defined in the encryption scheme.

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

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

$(\vec{a}', r')/\perp \leftarrow \text{Receive}(\text{note}, C, \text{sk})$: Parse $(a_{\text{sk}}, \text{sk}_{\text{enc}}) \leftarrow \text{sk}$. Decrypt C with sk_{enc} to (\vec{a}', r') . Parse $(\text{rk}', \text{rc}', \text{rn}') \leftarrow r'$ and verify that these values recreate the commitment

$$\text{note} = \text{Commit}(\text{Commit}(P(a_{\text{sk}}), \text{rn}'; \text{rk}'), \vec{a}'; \text{rc}')$$

If decryption fails or the commitment does not match the note, return \perp .

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

The completeness and soundness proofs are straightforward: completeness follows by completeness of the PKE, and soundness is ensured by the verification check in the end of Receive.

Theorem 4 (OTA Binding). *When using a binding commitment scheme Commit, the Zerocash commitment scheme is binding according to Definition 8*

Proof. Let \mathcal{A} return $\text{pk}_0, \vec{a}_0, r_0, \text{pk}_1, \vec{a}_1, r_1$ such that $\text{Gen}(\text{pk}_0, \vec{a}_0, r_0) = \text{Gen}(\text{pk}_1, \vec{a}_1, r_1)$ and $\vec{a}_0 \neq \vec{a}_1$. Parsing $\forall i \in \{0, 1\} : (\text{rk}_i, \text{rc}_i, \text{rn}_i) \leftarrow r_i$, the first condition implies the equality

$$\text{Commit}(\text{Commit}(\text{pk}_0, \text{rn}_0; \text{rk}_0), \vec{a}_0; \text{rc}_0) = \text{Commit}(\text{Commit}(\text{pk}_1, \text{rn}_1; \text{rk}_1), \vec{a}_1; \text{rc}_1)$$

but the commitments have different values for \vec{a} . This breaks the binding property of the commitment scheme. \square

Theorem 5 (Note Uniqueness). *Zerocash OTA style notes are unique according to Definition 10*

Proof. Trivially follows from binding of the underlying commitment scheme – since both pk and \vec{a} are commitment messages, finding a collision (even with adversarially chosen randomness) amounts to breaking commitment binding.

Theorem 6 (Nullifier Uniqueness). *The Zerocash OTA scheme has unique nullifiers (Definition 12), if the PRF is secure and the commitment scheme is binding.*

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

Theorem 7 (Nullifier Pseudorandomness). *The Zerocash OTA nullifiers are pseudorandom (Definition 11), 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 $\text{Receive}(\cdot, \cdot, \text{sk})$. The public key is computed as $\text{PRF}_{a_{\text{sk}}}^{\text{addr}}(0)$, 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 Receive oracle, it performs decryption with an unrelated sk_{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). \square

Theorem 8 (Nullifier Collision-Resistance). *Zerocash OTA nullifiers are collision-resistant.*

Proof. Follows directly from the collision-resistance of PRF^{sn} which in practice is instantiated using a collision-resistant hash function.

Theorem 9 (OTA Privacy). *Using a hiding commitment scheme and a IND-CCA, key anonymous¹³ (IK-CCA) encryption scheme, the Zerocash-style OTA is private according to Definition 9.*

Proof. The proof proceeds in three hops:

¹³ See [4], Definition 1.

1. By commitment hiding we replace $\text{note}^* \leftarrow \text{Commit}(\text{Commit}(a_{\text{pk}_{i_b}}, \text{rn}; \text{rk}), \vec{a}; \text{rc})$ by the commitment to zero: $\text{note}^* \leftarrow \text{Commit}(\text{Commit}(0, 0; \text{rk}), 0; \text{rc})$. A reduction to commitment hiding is straightforward: we do not need anything else to simulate \mathcal{O}_{Rcv} to \mathcal{A} when building \mathcal{B} against hiding, because \mathcal{O}_{Rcv} does not reply to the challenge notes.
2. By IK-CCA we can replace the encryption under pk_{i_b} to encryption under pk_0 always and just ignore i_1, i_0 . Note that in the first step we already remove note dependency on pk_{i_b} , so now public keys are only used in construction of C^* . In the reduction to IK-CCA we simulate \mathcal{O}_{Rcv} (which requires decrypting non-challenge C^*) to \mathcal{A} using IK-CCA decryption oracles. By the end of this game \mathcal{A} always receives $C^* = \text{Enc}(\text{pk}_0, (\vec{a}_b, r), \xi)$.
3. Finally, by IND-CCA we replace the encryption of a_b by encryption of 0. To simulate the \mathcal{O}_{Rcv} oracle to \mathcal{A} we again use the IND-CCA decryption oracle (and we just need one, since we always use the same encryption key pk_0).

After the three games what \mathcal{A} sees is $\text{Commit}(\text{Commit}(0, 0; \text{rk}), 0; \text{rc})$ as a note, and $\text{Enc}_{\text{pk}_0}(0; \xi)$ as a ciphertext. Both do not depend on b , and therefore \mathcal{A} wins the final game with probability exactly $1/2$. \square

C Proof of HID-OR for Sparse Pedersen Commitment

Proof (Lemma 1). Let the hash function be modelled by the random oracle, and $H(\text{ty}_i) = G^{t_i}$, and thus the challenger knows all t_i . The form of commitments that are given to the adversary is thus:

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

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:

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

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:

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

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

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

Therefore what \mathcal{A} sees now is:

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

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): $[t_1 a_1 + r], [t'_1 a'_1 + r'], \text{rc} = r - r'$ so our hiding holds with probability 1 (is perfect). \square

Corollary 1. *Assuming commitment hiding, Lemma 1 holds even if \mathcal{A} provides two sets of size n (not just pairs) of input and output commitments, $\{(\text{ty}_{i,t}, a_{i,t}), (\text{ty}'_{i,t}, a'_{i,t})\}_{i,t=0,1}^{n,1}$, 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 $\Delta_{\text{ty},0} = \Delta_{\text{ty},1}$. \square

D Anti-Theft Proof

Proof (Theorem 1). The proof starts by assuming \mathcal{A} wins the game, and finishes with breaking the HID-OR assumption, while using other assumptions in the process.

Assume \mathcal{A} wins the anti-theft game. The challenger finds valid $\text{st}, \text{tx}^*, \sigma^*$ in the log of $\mathcal{O}_{\text{Insert}}$ such that for $(\text{Nf}^A, M^{A'})$ (obtained through SplitTx and filtering M^A) there exists an entry $E = (\cdot, \cdot, \text{Nf}, M)$ in Spent such that $M^{A'} \cap M \neq \emptyset \vee \text{Nf}^A \cap \text{Nf} \neq \emptyset$.

We will now argue that whenever \mathcal{A} uses (in tx^*) an output note note or input nullifier nul from E , it also uses the corresponding com^T or com^S . And vice versa — including a commitment in tx^* from E forces \mathcal{A} to also include the same note or nul as in E .

Let us first assume that $M^{A'} \cap M \neq \emptyset$, let $\text{note} \in M^{A'} \cap M$. note is honestly owned (by SK), by construction of $M^{A'}$. Since honestly produced notes are unique (see Section 3), all *output* notes produced in $\mathcal{O}_{\text{Spend}}$ are unique too, and thus we can determine in which “critical” Spent call \hat{E} this note was created; thus \hat{E} is uniquely defined. Locate the corresponding $\mathcal{O}_{\text{Spend}}$ call, and the related “output” proof π^T together with com^T . Say that in tx^* the proof $\pi^{T'}$ corresponding to note is for the statement $\text{stmt}^{T'} = (\text{note}, C^{T'}, \text{com}^{T'})$. We claim that under NIZK SE and OTA anonymity, $\text{com}^{T'} = \text{com}^T$ (and, more generally, $\text{stmt}^{T'} = \text{stmt}^T$, where $\text{stmt}^T = (\text{note}, C^T, \text{com}^T)$ as produced in $\mathcal{O}_{\text{Spend}}$). In other words, π^T binds together a unique note and com^T , so \mathcal{A} cannot produce a proof for note with a different $\text{com}^{T'} \neq \text{com}^T$.

Proof (Anti-Theft Claim 1). The gist of the reduction is that it will embed OTA anonymity challenge (note^c, C^c) in \hat{E} into tx instead of (note, C) , and simulate π^T ; later, on seeing $\pi^{T'}$ from tx^* , if $\text{com}^{T'} \neq \text{com}^T$, the reduction extracts note^c message from $\pi^{T'}$ by NIZK SE.

The reduction \mathcal{B} starts by guessing the $\mathcal{O}_{\text{KeyGen}}$ query to use for embedding the public key pk^c provided by the anonymity game. (Formally the game will give \mathcal{B} two public keys — pk_0, pk_1 , but \mathcal{B} will only use one of them, say pk_0 , and also for the challenge later, sending $i_0 = i_1 = 0$). Since the number of queries to $\mathcal{O}_{\text{KeyGen}}$ is poly-limited, the reduction can decide from the beginning which $\mathcal{O}_{\text{KeyGen}}$ query it will use for this. In that query \mathcal{B} will return pk^c to \mathcal{A} .

Immediately \mathcal{B} has the following issue with simulating $\mathcal{O}_{\text{Spend}}$: when asked to spend coins from pk^c it cannot produce a proper π^S since for that it needs to know the corresponding sk^c . Luckily, this can be overcome. When the challenger gets (note, C) as part of \mathcal{A} 's input in I , it will first, as before, try to see whether this note can be received using SK (in which case it is an honest spend request). If not, it might be that this is a request to spend from pk^c : this \mathcal{B} will verify by sending (note, C) to \mathcal{O}_{Rcv} . In this case it will obtain $((a, \text{ty}), r)$ and derive nul from this information. Then, \mathcal{B} will simulate the corresponding π^S .

\mathcal{B} also pre-guesses a “critical” $\mathcal{O}_{\text{Spend}}$ query, in which it embeds the anonymity challenge — again \mathcal{B} can do it assuming poly-limited number of queries \mathcal{A} can do. This critical query must have an adversarial instruction to create an output note for pk^c with some (ty, a) . But instead of doing that, \mathcal{B} will give \mathcal{C} two different sets of attributes (e.g. $(\text{ty}, 0), (\text{ty}, 1)$) for the same anonymity game key ($i_0 = i_1 = 0$), receive the challenge note and ciphertext note^c, C^c , and embed them into the $\mathcal{O}_{\text{Spend}}$ reply. The corresponding π^T must be simulated¹⁴ Note that this embedding strategy only makes sense when note is honest.

After embedding, \mathcal{B} will continue to simulate $\mathcal{O}_{\text{Spend}}$ as before, with the only difference. In all the following $\mathcal{O}_{\text{Spend}}$ queries (if they happen at all) where \mathcal{A} attempts to spend from note^c , \mathcal{B} will use the original (ty, a) instead of properly receiving the note (which it cannot do since sk^c is owned by \mathcal{C} only).

¹⁴ Formally, to apply SE, we must simulate all the NIZKs in the game (and we can do that). However, by ZK \mathcal{A} cannot distinguish between simulated and non-simulated proofs. Therefore, to simplify the proof, we do not simulate all the proofs produced by $\mathcal{O}_{\text{Spend}}$, but only this critical one, which is equivalent.

In the end of the anti-theft game, on detecting tx^* triggering a winning condition, assuming $\text{com}^{\mathcal{T}'} \neq \text{com}^{\mathcal{T}}$ we know that $\text{stmt}^{\mathcal{T}'} \neq \text{stmt}^{\mathcal{T}}$. Hence, by NIZK SE \mathcal{B} can extract from $\pi^{\mathcal{T}}$ included with the challenge note of tx^* (extraction is possible whenever proof verifies, and statement is different from statements of all (simulated) proofs \mathcal{A} sees), and obtain the randomness r for this note. Using r , \mathcal{B} can decide which note, note_0 or note_1 it was given by anonymity challenger \mathcal{C} , and thus break OTA anonymity. \square

Second, assume that $\text{Nf}^{\mathcal{A}} \cap \text{Nf} \neq \emptyset$, and take any nullifier nul from this intersection. This case is a bit more tricky: since nullifiers are not unique, and nul can appear in many Spent entries (call their set $\text{Spent}_{\text{nul}}$), it is not immediately clear which (critical) query $\hat{E} \in \text{Spent}_{\text{nul}}$ the nullifier was “taken from”¹⁵.

In tx^* locate the corresponding $\mathcal{L}^{\text{spend}}$ proof $\pi^{\mathcal{S}'}$ and the commitment $\text{com}^{\mathcal{S}'}$, such that the proof verifies for $\text{stmt}^{\mathcal{S}'} = (\text{st}^*, \text{nul}, \text{com}^{\mathcal{S}'})$. In the queries from $E \in \text{Spent}_{\text{nul}}$, $\mathcal{O}_{\text{Spend}}$ produces proofs π_i for statements $\text{stmt}_i^{\mathcal{S}} = (\text{st}_i, \text{nul}, \text{com}_i^{\mathcal{S}})$. We claim (by nullifier pseudorandomness, collision-resistance, and weak SE of the NIZK) that for some $E_j \in \text{Spent}_{\text{nul}}$ it holds that $\text{stmt}_j^{\mathcal{S}} = \text{stmt}^{\mathcal{S}'}$, and as a result, $\text{com}^{\mathcal{S}'} = \text{com}_j^{\mathcal{S}}$. As a side result of this claim, \mathcal{B} can now identify the critical query $E_j = \hat{E}$ by looking for $\text{com}^{\mathcal{S}'}$ among $\text{com}_j^{\mathcal{S}}$.

Proof (Anti-Theft Claim 2). The proof is similar to the first claim, but by NIZK SE extraction we obtain the preimage of the nullifier (which is supposed to be pseudorandom).

Recall that the pseudorandomness game gives us a challenge public key pk^c , a $\text{Receive}(\cdot, \cdot, \text{sk}^c)$ oracle \mathcal{O}_{Rcv} , and the challenge oracle \mathcal{O}_{PRF} that returns evaluations of either a real $\text{NulEval}(\text{sk}^c, \cdot)$ or a randomly chosen $f(\cdot)$.

The reduction \mathcal{B} first selects a target $\mathcal{O}_{\text{KeyGen}}$ query as in Claim 1, and in this query, simulating $\mathcal{O}_{\text{KeyGen}}$ to \mathcal{A} , it will return pk^c (without knowing sk^c). Whenever \mathcal{B} needs to receive a note sent to pk^c (e.g. sent by \mathcal{A} through $\mathcal{O}_{\text{Insert}}$), \mathcal{B} will use the \mathcal{O}_{Rcv} .

As the game proceeds, \mathcal{B} uses \mathcal{O}_{PRF} to generate a value for *each* $\text{NulEval}(\text{sk}^c, \cdot)$ (in $\mathcal{O}_{\text{Spend}}$ queries if such appear). It simulates all the corresponding $\mathcal{L}^{\text{spend}}$ NIZKs. Call the logs of all these $\mathcal{O}_{\text{Spend}}$ queries $\text{Spent}_{\text{nul}}$. In other words, \mathcal{B} embeds \mathcal{O}_{PRF} responses into all relevant queries $\text{Spent}_{\text{nul}}$ simultaneously.

After simulating the anti-theft game to \mathcal{A} , some tx^* will trigger the winning condition. By weak SE, and since $\pi^{\mathcal{S}'} \in \text{tx}^*$ verifies on $\text{stmt}^{\mathcal{S}'}$ that includes nul (if \mathcal{B} guessed correctly), unless $\text{stmt}^{\mathcal{S}'} = \text{stmt}_i^{\mathcal{S}}$ for some $\text{stmt}_i^{\mathcal{S}} \in \text{Spent}_{\text{nul}}$, we can extract the witness from $\pi^{\mathcal{S}'}$. If extraction is possible, \mathcal{B} obtains (sk, r) such that $\text{NulEval}(\text{sk}, r) = \text{nul}$ for $\text{nul} \in \text{Spent}_{\text{nul}}$. By nullifier collision-resistance it is not possible that the extracted (sk, r) is a different input from (sk^c, r') (where r' is from the critical query). This means that either sk is an actual secret key used in \mathcal{O}_{PRF} if it is instantiated with a nullifier evaluation; or the oracle is random. So \mathcal{B} uses sk to compare \mathcal{O}_{PRF} outputs with $\text{NulEval}(\text{sk}, r)$ and thus break the PRF game.

Therefore, it has to be that $\exists j$ such that $\text{stmt}_j^{\mathcal{S}} = \text{stmt}^{\mathcal{S}'}$ and thus $\text{com}^{\mathcal{S}'} = \text{com}_j^{\mathcal{S}}$. \square

This last reduction can be applied to all the nullifiers in $\text{Nf}^{\mathcal{A}} \cap \text{Nf}$: if \mathcal{A} uses an honest nullifier, we are able to locate the critical query it was taken from, and the related $\text{com}^{\mathcal{S}}$ in tx^* is the same as in that critical query.

Now, in tx^* , the adversary might try to combine several nullifiers or notes from different Spent queries, and for the final reduction we only need to focus on a single such critical query. In other

¹⁵ The fact that each honest query has at least one honest output does not help with query identification: \mathcal{A} does not *have to* include any honest output notes into tx^* , and still must be able to trigger the winning condition of the game.

words, winning condition “ \exists Spent entry | ...” can be triggered by several such entries. Fix any query \hat{E} that has an honest nullifier or a note triggering winning condition¹⁶.

Let tx be the transaction from \hat{E} , which defines M, Nf . Call I_M the indices in tx in which we observe $M^{\mathcal{A}'} \cap M \neq \emptyset$, similarly I_{Nf} for $\text{Nf}^{\mathcal{A}} \cap \text{Nf}$. Similarly, call I_M^*, I_{Nf}^* the corresponding indices in tx^* . (E.g. this notation implies $\forall i : \text{nul}_{I_{\text{Nf}}, i} = \text{nul}_{I_{\text{Nf}}^*, i}^*$)

There exists a set $C_0 = \{\text{com}_i^S\}_{I_{\text{Nf}}} \cup \{\text{com}_i^T\}_{I_M}$ as part of tx . And after the previous two claims on NIZK binding commitments we now know that there is a subset $C_0^* = \{\text{com}_i^{S^*}\}_{I_{\text{Nf}}^*} \cup \{\text{com}_i^{T^*}\}_{I_M^*}$ of commitments C^* in σ^* which is exactly the same as C_0 . But there are also other two disjoint sets of commitments, $C_1 = C \setminus C_0$ and $C_1^* = C^* \setminus C_0^*$. E.g. C_1^* corresponds to the input nullifiers and output notes of tx^* that were not taken from \hat{E} , but from elsewhere (other honest queries, or adversarially generated).

Recall that C_0 is defined w.r.t. $M \cap M^{\mathcal{A}'}$ and $\text{Nf} \cap \text{Nf}^{\mathcal{A}}$, and so nullifiers and notes included with C_1^* in tx^* are, by definition, disjoint with M and Nf in tx . We now present the last claim: if \mathcal{A} triggers the winning condition of the anti-theft game, then it can break security (binding or HID-OR) of the commitment scheme.

Proof (Anti-Theft Claim 3). We first describe the reduction \mathcal{B} to HID-OR. First, it pre-guesses a query \hat{E} in which it will embed. When \mathcal{A} asks query \hat{E} , \mathcal{B} takes any two distinct indices corresponding to inputs and outputs, which define two commitments C_X and C_Y it will embed into. The reduction will create all commitments except these two honestly, and regarding the two it will put there HID-OR output challenge for either: (1) original two original values $(a_1, \text{ty}_1), (a_2, \text{ty}_2)$; or (2) them swapped as $(a_2, \text{ty}_1), (a_1, \text{ty}_2)$ if both indices correspond to inputs or both to outputs, and swapped with negation $(-a_2, \text{ty}_2), (-a_1, \text{ty}_1)$ otherwise. This way to embed guarantees the HID-OR requirement that $\Delta_{\text{ty}, 0} = \Delta_{\text{ty}, 1}$. The joint randomness rc_0 for C_X, C_Y the reduction will sum together with the randomness for other honestly produced inputs or outputs to output the final total randomness rc for tx . The only two NIZKs that need to be simulated are the ones that correspond to C_X, C_Y . Note that we always have at least one input and output in the transaction — this is because there is always by $\mathcal{O}_{\text{Spend}}$ mandates $|O| > 0$, and $|I| > 0$ since offer without inputs cannot trigger the winning condition, since it is guaranteed to be classified as honest by SplitTx because of note uniqueness.

Continue simulating the anti-theft game to \mathcal{A} . This does not require any changes — further queries to $\mathcal{O}_{\text{Spend}}$ do not depend on our embedding, and can be performed as before.

In the end of the game, \mathcal{A} provides a tx^* which triggers the winning condition with some notes or nullifiers corresponding to commitments C_0 . The reduction will find C_x in C_0 (abort if it is not present), it will assert that $C_Y \notin C$ (abort otherwise). Then it will call NIZK extractor to obtain the randomness rc_i^* for all commitments in C^* except for C_X , which gives joint rc^* for the whole C_1^* when summed up, and thus \mathcal{B} computes $\text{rc}_0^* = \text{rc}^* - \text{rc}^{x'}$ (where rc^* is total joint randomness of tx^*). The reduction will return $b = 1$ iff $C_X = \text{Com}((a_1, \text{ty}_1), \text{rc}_0^*)$.

Now we argue that the reduction breaks binding or HID-OR of the commitment scheme if \mathcal{A} wins the anti-theft game. First, the total probability of all guesses to be correct is $1/|Q|m^2$, which is polynomial, where Q is a number of queries to $\mathcal{O}_{\text{Spend}}$, and $m = \text{poly}(\lambda)$ is a maximum number of inputs and outputs in a \hat{E} transaction). The guess inside \hat{E} is correct if $C_X \in C_0$ and $C_Y \in C_1$ — therefore, the probability of the guess being correct is at least $1/m^2 = \text{poly}(\lambda)$.

Now, assume all guesses are correct, and that \mathcal{A} wins the anti-theft game — then \hat{E} was not included completely in tx^* , so $C_X \in C_0^*, C_Y \notin C^*$. We know by the previous two claims that C_X in tx^* is the same as in tx (since the nullifier or the note is the same, because it triggered the

¹⁶ Not all $\mathcal{O}_{\text{Spend}}$ queries *just triggering* winning condition here work: when fixing \hat{E} that shares a *nullifier* with tx^* , recall that some other $\mathcal{O}_{\text{Spend}}$ query can also have nul, but by the second claim we can locate the “true” critical query for that nul by comparing com^S . Fix only such a “truly” critical query.

winning condition). We cannot extract from the corresponding (simulated) NIZK π_X — however, by NIZK SE we can still extract¹⁷ from all the other NIZKs in tx^* (or they were honestly produced in tx , which is equivalent), which is what reduction does. So the extractor in \mathcal{B} will succeed, and \mathcal{B} will obtain rc_0^* . This rc_0^* should give C_X when message is either (a_1, ty_2) or (a_2, ty_2) ; and thus \mathcal{B} wins HID-OR. And if C_X does not match $\text{Com}((a_b, \text{ty}_b), \text{rc}_0^*)$ for both $b \in \{0, 1\}$, it means that binding of C_X is broken, that is \mathcal{A} found a different value and randomness giving rise to the same C_X . Thus \mathcal{B} can either determine whether C_X commits to the original value or the “swapped” one, which is enough to break HID-OR; or \mathcal{B} breaks commitment binding. \square

This concludes the anti-theft proof. \square

E Balance Proof

Proof (Theorem 2). We progress by the following sequence of games:

Game₁: We start by introducing transaction extractor into the balance game. The game **Balance₁**, presented below, is different from the standard **Balance** in two aspects:

1. the setup algorithm now generates the AoK in the dishonest way, not disposing of the trapdoor τ from **AoK.Setup**;
2. when the game processes transactions in the loop, the extractor \mathcal{E}_A is introduced, which uses τ to obtain pre-transaction **ptx** from every **tx**. This extracted **ptx** is then asserted to be an input to **tx**.

Lines marked with a star * indicate new or changed lines.

```

BalanceA, EA1(1λ)
  % Use simulated setup for the AoK.
  (p, τ) ← Setup'(1λ) *
  (st*, I0, I, SK*) ← AoKeyGen · oSpend · oInsert(p)
  ...
  (Ins', ·) ← GetLog(st*)
  for all (st, ·, tx, σ) ∈ Ins' do
    ...
    Parse tx as ( {nuli*}i=1|S| { (noteiT*, Ci) }i=1|T|, {Δa,ty}ty ∈ Ty ) *
    % Extract pre-transaction used to create tx
    ptxE ← EA(p, τ, tx, σ) *
    Parse SE as { (skiS, noteiS, nuli, pathi, (aiS, tyiS), riS) }i=1|S| *
    Parse TE as { (pkiT, noteiT, CiT, (aiT, tyiT), riT) }i=1|T| *
    (·, st̄) ← GetLog(st) *
    assert CheckPTx(ptxE, st) = 1 ∧ CompleteTx(ptxE) = tx *
    ...
  if ∃ ty : vA[ty] > v0[ty] + vH-[ty] - vH+[ty] then return 1
  else return 0

```

Formally, we claim that for every adversary \mathcal{A} there exists \mathcal{E}_A s.t. $\Pr[\text{Balance}_{\mathcal{A}, \mathcal{E}_A}^1(1^\lambda) = 1] \geq \Pr[\text{Balance}_{\mathcal{A}}(1^\lambda) = 1] - \text{negl}(\lambda)$, assuming KS of the NIZKs holds, and commitments are binding:

¹⁷ Again, there is a catch in how we use SE here. In the original SE, all proofs are simulated, and one can extract from those which do not have the same instances **stmt** as simulated ones **stmt_i^{sim}**. In our case, only some (two) proofs are simulated, and other proofs are honest. But from honest proofs we can straightforwardly extract (since we produced them), so still it is true that extraction is possible when $\forall i : \text{stmt} \neq \text{stmt}_i^{\text{sim}}$.

Proof (Theorem 2, Transition 1). The extractor $\mathcal{E}_{\mathcal{A}}$ is essentially a wrapper around two AoK extractors (for spend and output proofs): it calls $\text{AoK}[\mathcal{L}^{\text{spend}}].\mathcal{E}_{\mathcal{A}}$ and $\text{AoK}[\mathcal{L}^{\text{output}}].\mathcal{E}_{\mathcal{A}}$, both of which are guaranteed to exist by KS of the NIZKs, and concatenates their results into \mathcal{S} and \mathcal{T} .

Note that the `ptx` assertion is the only way the games are different in terms of possibility of different outcome. The only other line that can fail is the internal `GetLog(st)`; but since it queries a state which was returned by the previous `GetLog(st*)`, it will not abort. Therefore we must only argue why the assertion containing `CheckPTx` and `CompleteTx` will hold every time – in other words the probability for adversary to win will not be hindered by the assertion failing.

We first argue why $\text{CheckPTx}(\text{ptx}_{\mathcal{E}}, \text{st}) = 1$. The check that $\text{nul}_i \notin \text{st.MT}$ is directly checked in `Verify`. All checks except for the `CheckBalance` reduce to the NIZK knowledge-soundness. Looking at every $\mathcal{L}^{\text{spend}}$ NIZK first, the fact it verifies implies that $\text{AoK}[\mathcal{L}^{\text{spend}}].\mathcal{E}_{\mathcal{A}}$ extracts $w = (\text{path}_i, \text{sk}_i^S, a_i^S, \text{ty}_i^S, r_i^S, \text{rc}_i^S)$, which by NIZK KS guarantees for all $i \in [|\mathcal{S}|]$:

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

Conditions (1) and (3) are satisfied by $\mathcal{L}^{\text{open}}$, and (2) is by \mathcal{L}^{nul} (see $\mathcal{L}^{\text{spend}}$ structure). Similarly, the output NIZK for the $\mathcal{L}^{\text{output}}$ language guarantees that $\text{AoK}[\mathcal{L}^{\text{output}}].\mathcal{E}_{\mathcal{A}}$ will return $(\text{pk}_i^T, a_i^T, \text{ty}_i^T, r_i^T, \text{rc}_i^T)$ such that:

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

Which again holds by the NIZK KS and the structure of $\mathcal{L}^{\text{open}}$, being part of $\mathcal{L}^{\text{output}}$.

The last balance check that $\text{CheckBalance}(\mathcal{S}, \mathcal{T}) = 1$, which is implemented as $\forall \text{ty} \in \{\text{ty}_i^S\}_{i=1}^{|\mathcal{S}|} \cup \{\text{ty}_i^T\}_{i=1}^{|\mathcal{T}|} : \sum_{a \in \{a_i^S | \text{ty}_i^S = \text{ty}\}_{i=1}^{|\mathcal{S}|}} a - \sum_{a \in \{a_i^T | \text{ty}_i^T = \text{ty}\}_{i=1}^{|\mathcal{T}|}} a = \Delta_{\text{ty}}$ succeeds by the commitment homomorphic property in `Verify`. We know that `tx` verifies, therefore the homomorphic sum in `Verify` holds. By the previous steps we also know that the extracted type-value pairs (a_i^S, ty_i^S) and (a_i^T, ty_i^T) are inputs to the commitments com^S and com^T correspondingly. By the $a \in [2^\alpha]$ check in NIZK, and by the β check in `Verify` (and by the choice of α and β), we know that homomorphic sums of $\text{com}^S, \text{com}^T$ will not overflow. Since the commitments in `Verify` (including commitment to Δ_{ty}) sum to the identity, the committed values a_i sum to exactly Δ_{ty} per type.

Therefore $\text{CheckPTx} = 1$ with overwhelming probability by existence of NIZK extractors.

Finally, we claim that $\text{CompleteTx} = 1$. `CompleteTx` does two things: first, it removes secret information from `ptx`, and, second, it computes the imbalances Δ_{ty} . First, the public information in $\text{tx} \cap \text{ptx}$ is input nullifiers, and note-ciphertext output pairs – because these are in the statements of corresponding NIZKs, they are bound to be exactly the same (formally, these values are the input of \mathcal{E} as `stmt`, and not its output). Therefore all parts of `tx` except for the deltas are exactly like in the `ptx` extracted. Second, that the deltas in `tx` are also the same as computed from a_i follows from the commitment binding. And the fact that they sum up to Δ we have just shown when arguing $\text{CheckPTx} = 1$.

Therefore the gap $\text{negl}(\lambda)$ in this game consists of probability of failing, or binding being broken. \square

Now that we have `ptx` values extracted in clear, the main balance property will follow inductively by following the $(\text{st}, \text{st}', \text{tx}, \sigma) \in \text{Ins}'$ loop step by step. But before we present the

main proof reasoning, we must introduce several auxiliary constructions into our game. Our intention within next several games is to 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.

Game₂: We add an additional condition on the extracted data by asserting that it is equal to the type-value pairs we use when modifying $v_{\mathcal{H}+}, v_{\mathcal{H}-}$.

```

Balance $\mathcal{A}, \mathcal{E}_{\mathcal{A}}$ 2( $1^\lambda$ )
...
for all (st, ·, tx,  $\sigma$ ) ∈  $\text{Ins}'$  do
...
ptx $\mathcal{E}$ @( $\mathcal{S}_{\mathcal{E}}, \mathcal{T}_{\mathcal{E}}$ ) ←  $\mathcal{E}_{\mathcal{A}}$ (p,  $\tau$ , tx,  $\sigma$ )
...
for (Nf,  $M$ ) ∈ tx $\mathcal{H}$  do
  Find (·,  $I$ , Nf,  $M$ ) ∈ Spent
  for (note',  $C'$ ) ∈  $I$  do
    (·, (a, ty), ·) ← TryReceive(note',  $C'$ , SK)
     $v_{\mathcal{H}-}[\text{ty}] := v_{\mathcal{H}-}[\text{ty}] + a$ 
    assert (a, ty) = ( $a_i^S, \text{ty}_i^S$ ) *
  for (note',  $C'$ ) ∈  $M$  where (·,  $M$ ) ∈ tx do
    res ← TryReceive(note',  $C'$ , SK)
    if res = (·, (a, ty), ·) ≠ ⊥ then
       $v_{\mathcal{H}+}[\text{ty}] := v_{\mathcal{H}+}[\text{ty}] + a$ 
      assert (a, ty) = ( $a_i^T, \text{ty}_i^T$ ) *
...

```

This game transition is by OTA binding and nullifier collision-resistance.

Proof (Theorem 2, Transition 2). The first assertion is reached because $(\cdot, I, \text{Nf}, M) \in \text{Spent}$ for (Nf, M) located by SplitTx to be honest on the basis of $(\text{Nf}, M) \in \text{tx}$, where tx is a currently processed transaction. Since notes that are added to I pass checks in BuildPTx (inside $\mathcal{O}_{\text{Spend}}$), we know that (a, ty) are valid inputs producing $\text{nul} \in \text{Nf}$ and note. The argument is exactly the same as second claim of proof of Theorem 1: this requires nullifier pseudorandomness, collision-resistance, and weak SE of the NIZK. By this argument we know that the NIZK statements are the same, thus value commitments are the same, and thus the values inside the commitments (by value commitment binding) are the same.

The second assertion will hold by OTA binding: TryReceive guarantees that (a, ty) are valid input to note', and so are (a_i^S, ty_i^S) by CheckPTx and the previous game. Therefore these must be equal.

□

Game₃: In the next game Balance³ we add more meaning to the extracted data by linking input notes to the notes mentioned in previous transactions. We assert that all the extracted input notes are the same as some output notes generated before, as the notes originally present in I_0 ; and that they are not an input to any previous transaction (thus not double spent).

```

BalanceA, EA3(1λ)
...
for all (st, ·, tx, σ) ∈ Ins' do
...
ptxE ⊕ (SE, TE) ← EA(p, τ, tx, σ)
...
for nul ∈ tx do
  assert The corresponding extracted noteS:
    (1) Is present at least once
    (1.1) in some TE of some previous transaction; or
    (1.2) in I0 directly.
    (2) Is not part of SE in any previous txs.
...
if ∃ ty : vA[ty] > v0[ty] + vH-[ty] - vH+[ty] then return 1
else return 0

```

The transition is by MT binding and nullifier uniqueness.

Proof (Theorem 2, Transition 3). By NIZK KS note^S must be in the Merkle tree. Notes are put into the MT at previous steps, or they are present in $\text{st}_0.\text{MT}$ — this is how MT is populated — so we can always find the previous step where note^S was introduced. In other words, we can always find the previous state with $\text{Com}(\text{note}^S)$ present. The note inserted into MT at that step is equal to note^S because MT is a binding commitment scheme. This proves the first part of the condition: there always exists the previous step with note^S extracted, or this note^S was present in $\text{st}_0.\text{MT}$.

Regarding the second condition of the assertion, namely that note^S is not an input to any previous tx. Let nul be the nullifier of note^S ; by \mathcal{L}^{nul} and previous game we know that $\text{note}^S = \text{Gen}(\text{pk}, (a, \text{ty}), r)$ and $\text{nul} = \text{NulEval}(\text{sk}, r)$. Now assume the contrary, that there is a previous spend which extracts note^S too. Backtrace to this spend, and let nul_0 be its nullifier, together with r, sk_0 all jointly satisfying the same \mathcal{L}^{nul} equation. It must be that $\text{nul}_0 \neq \text{nul}$ where nul belongs to note^S — if $\text{nul}_0 = \text{nul}$, tx cannot be verified at the current state (nullifier's set is append-only by construction). This is enough to break nullifier uniqueness, since we just observed two nullifiers for the same note:

$$\text{note}^S = \text{Gen}(\text{OTA}.P(\text{sk}), (a, \text{ty}), r) = \text{Gen}(\text{OTA}.P(\text{sk}_0), (a_0, \text{ty}_0), r_0) \\ \text{NulEval}(\text{sk}, r) \neq \text{NulEval}(\text{sk}_0, r_0)$$

Thus double-spending is not allowed. \square

Note that the first condition of the game is more nuanced. In fact, because (malicious) transaction creator has control over the output note randomness, it is possible to create several output notes with the same value and randomness. However, since all these notes have the same nullifier, only one of them can be spent¹⁸. Hence we only require finding “at least one output”.

Game₄: In the next game Balance_4 we will add a (multi-)set $N_{\overline{H}}$ initially populated with notes in I_0 . This set will track all non-honest notes, where “not honest” here means not just adversarial notes (that \mathcal{A} can claim similarly to how $v_{\mathcal{A}}$ is computed) but also notes that are “burned” — that cannot be accessed by anyone.

Each loop iteration will

1. remove adversarial input notes from $N_{\overline{H}}$, and
2. add to $N_{\overline{H}}$ all the output notes that cannot be received by SK.

¹⁸ Similar to the “Faerie Gold” attack in zcash

For both actions we need the data extracted in Balance^1 , 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”¹⁹ assets of \mathcal{A}).

```

Balance4 $\mathcal{A}, \mathcal{E}_{\mathcal{A}}$ (1 $\lambda$ )
...
N $\overline{\mathcal{H}}$  ← ∅ *
for (note $i$ , C $i$ ) ∈ I0 do *
  N $\overline{\mathcal{H}}$  = N $\overline{\mathcal{H}}$  ∪ {note $i$ } *
  (·, nul, (a, ty), ·) ← TryReceive(note $i$ , C $i$ , SK*)
  assert nul ∉ st0.NF
  v0[ty] := v0[ty] + a
v $\mathcal{H}-$ , v $\mathcal{H}+$  ← (ty ↦ 0)
(Ins', ·) ← GetLog(st*)
for all (st, ·, tx, σ) ∈ Ins' do
  ...
  pt $\mathcal{E}$ @(S $\mathcal{E}$ , T $\mathcal{E}$ ) ← E $\mathcal{A}$ (p, τ, tx, σ)
  ...
  for note' ∈ S $\mathcal{E}$  | corresponding skS ∉ SK do *
    N $\overline{\mathcal{H}}$  := N $\overline{\mathcal{H}}$  \ {note'} *
    v $\overline{\mathcal{H}}$ [tyS] := v $\overline{\mathcal{H}}$ [tyS] - aS % tyS, aS as extracted by E $\mathcal{A}$  *
  for (note', C') ∈ M where (·, M) ∈ tx do
    ...
    res ← TryReceive(note', C', SK)
    if res = (·, (a, ty), ·) ≠ ⊥ then
      v $\mathcal{H}+$ [ty] := v $\mathcal{H}+$ [ty] + a
    else *
      N $\overline{\mathcal{H}}$  := N $\overline{\mathcal{H}}$  ∪ {note'} *
      v $\overline{\mathcal{H}}$ [tyT] := v $\overline{\mathcal{H}}$ [tyT] + aT % tyT, aT as extracted by E $\mathcal{A}$  *
  if ∃ ty : v $\mathcal{A}$ [ty] > v0[ty] + v $\mathcal{H}-$ [ty] - v $\mathcal{H}+$ [ty] then return 1
  else return 0

```

As can be seen in the part where $N_{\overline{\mathcal{H}}}$ is populated, it contains all the notes that cannot be received by honest parties, which includes: (1) adversarial notes, with or without correct ciphertext (this does not matter), (2) notes to keys that are not controlled by both adversary and honest parties (burned), (3) notes to honest parties with malformed ciphertext (also effectively burned).

We now argue that $\Pr[\text{Balance}_{\mathcal{A}, \mathcal{E}_{\mathcal{A}}}^4(1^\lambda) = 1] \geq \Pr[\text{Balance}_{\mathcal{A}, \mathcal{E}_{\mathcal{A}}}^2(1^\lambda) = 1] - \text{neg}(\lambda)$ by OTA binding.

Proof (Theorem 2, Transition 4). This change only *adds* parallel logic into our computation that almost does not interact with any previous logic. It does not abort in all cases — when we append to the set and add or subtract from the corresponding value variable — but one. The only exception is the set subtraction, which we argue does not fail because of Balance^3 . There we prove that each input note is present at least once in some outputs, and is not present in the inputs. The latter guarantees that the note will not be removed twice. The former guarantees that the note will be present once to be removed: when attempting to remove a note, given that we know that it was in some outputs, the only exception would be that this note was not put into $N_{\overline{\mathcal{H}}}$ at that point, which means that it is receivable using SK, but to trigger removal sk^S at the current step must be $\notin \text{SK}$. Finding two different secret keys, one in SK, another not in SK, for the same note, is not possible by OTA binding. □

¹⁹ E.g. \mathcal{A} can “refresh” the note $\text{note} \in I_0$ by moving its funds fully to another adversarial $\text{note}' \notin I$, and without $N_{\overline{\mathcal{H}}}$ this will not be reflected anywhere.

Game₅: Next we add an “intermediate balance assertion”, similarly to the final one, but with $v_{\overline{\mathcal{H}}}[\mathbf{ty}]$ in place of $v_{\mathcal{A}}[\mathbf{ty}]$, to the end of each iteration (and before the loop).

```

Balance5 $\mathcal{A}, \varepsilon_{\mathcal{A}}$ ( $1^\lambda$ )
...
( $\text{Ins}', \cdot$ )  $\leftarrow$  GetLog( $\text{st}^*$ )
assert  $\forall \mathbf{ty} : v_{\overline{\mathcal{H}}}[\mathbf{ty}] \leq v_0[\mathbf{ty}] + v_{\mathcal{H}-}[\mathbf{ty}] - v_{\mathcal{H}+}[\mathbf{ty}]$  *
for all ( $\text{st}, \cdot, \mathbf{tx}, \sigma$ )  $\in$   $\text{Ins}'$  do
...
assert  $\forall \mathbf{ty} : v_{\overline{\mathcal{H}}}[\mathbf{ty}] \leq v_0[\mathbf{ty}] + v_{\mathcal{H}-}[\mathbf{ty}] - v_{\mathcal{H}+}[\mathbf{ty}]$  *
if  $\exists \mathbf{ty} : v_{\mathcal{A}}[\mathbf{ty}] > v_0[\mathbf{ty}] + v_{\mathcal{H}-}[\mathbf{ty}] - v_{\mathcal{H}+}[\mathbf{ty}]$  then return 1
else return 0

```

We argue that it is not harder to win Balance^5 than to win Balance^4 by OTA soundness and binding.

Proof (Theorem 2, Transition 5). It could be that an adversarial strategy that worked with Balance^4 will fail because the assertion fails. So we must argue that the assertion never fails. We proceed inductively.

In the base case, before the first loop iteration (before any transaction is processed), the assertion holds since $v_{\overline{\mathcal{H}}}[\mathbf{ty}] = v_0[\mathbf{ty}]$ and $v_{\mathcal{H}+} = v_{\mathcal{H}-} = 0$ – so trivially $\forall \mathbf{ty}. v_{\overline{\mathcal{H}}}[\mathbf{ty}] \leq v_0[\mathbf{ty}]$.

Assume now that the assertion holds in the beginning of the loop, we will show it persists through the loop execution. This essentially reduces to the observation that the end-of-the-loop equation is updated by differences that satisfy this equation; which reduces to the balancing property of a single transaction.

Fix a type \mathbf{ty} , and apply the following reasoning to each type in the transaction (all of which are available in the extracted data). Compute the local *difference* values $v_{\mathcal{H}-}^\Delta, v_{\mathcal{H}+}^\Delta$ as prescribed by the game, but without updating the old $v_{\mathcal{H}-}, v_{\mathcal{H}+}$ immediately. Similarly compute $v_{\overline{\mathcal{H}}}^\Delta$ from the extracted (input and output) NIZKs by adding all the non-honest output values and subtracting all the non-honest adversarial input values. Note that v_0 is at no point updated after it is initialised before the main loop.

Now we need to prove that $v_{\overline{\mathcal{H}}}^\Delta \leq v_{\mathcal{H}-}^\Delta - v_{\mathcal{H}+}^\Delta$, then the updated end-of-the-loop equation will still hold. This reduces to the commitment balancing condition. First, observe that $v_{\overline{\mathcal{H}}}^\Delta = \sum a_i^{\mathcal{T}\overline{\mathcal{H}}} - \sum a_i^{\mathcal{S}\overline{\mathcal{H}}}$: we compute $v_{\overline{\mathcal{H}}}^\Delta$ using these extracted values a_i directly, summing all outputs and subtracting all inputs. Similarly, $v_{\mathcal{H}-}^\Delta = \sum a_i^{\mathcal{S}\mathcal{H}}$ and $v_{\mathcal{H}+}^\Delta = \sum a_i^{\mathcal{T}\mathcal{H}}$, the only difference being that values a_i here are obtained not from the extracted data, but by directly receiving the corresponding notes via $\text{TryReceive}(\text{note}, C, \text{SK})$. But these “received” values $((a, \mathbf{ty}), r)$ are equal to the extracted ones as we asserted in the second game.

By the last (balancing) check in CheckPTx introduced in Balance^1 , substituting the extracted a_i values we just discussed, we obtain

$$\sum a_i^{\mathcal{S}\overline{\mathcal{H}}} + \sum a_i^{\mathcal{S}\mathcal{H}} - \sum a_i^{\mathcal{T}\overline{\mathcal{H}}} - \sum a_i^{\mathcal{T}\mathcal{H}} = 0$$

Which translates into $-v_{\overline{\mathcal{H}}}^\Delta + v_{\mathcal{H}-}^\Delta - v_{\mathcal{H}+}^\Delta = 0$, equivalent to $v_{\overline{\mathcal{H}}}^\Delta = v_{\mathcal{H}-}^\Delta - v_{\mathcal{H}+}^\Delta$ as we need. So the predicate persists through the loop iteration. \square

Game₆: Our next and final step is Balance^6 where we show that after the loop $v_{\mathcal{A}} \leq v_{\overline{\mathcal{H}}}$. This is because the former counts transactions that can be received with SK^* , and the latter all that cannot be received by SK . Formally this can be shown by comparing all the notes in $N_{\overline{\mathcal{H}}}$ and I : we claim that all notes in I are present in $N_{\overline{\mathcal{H}}}$.

We show it by doing two things. First, 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 `note` $\in I$ in the outputs of `tx` (this `note` is still attempted to be received by `SK*`). To track what we have already counted into $v_{\mathcal{A}}$, we will create a variable $N_{\mathcal{A}}$: by design after the game $N_{\mathcal{A}}$ is exactly all the notes in I . This allows us to track the successive stake accumulation of \mathcal{A} , but only on those coins that it claims in I . And second, every time we locate a coin that goes into $v_{\mathcal{A}}$, we will assert that all notes in I are also in $N_{\overline{\mathcal{H}}}$.

```

Balance6 $\mathcal{A}, \varepsilon_{\mathcal{A}}$ ( $1^\lambda$ )
...
for (·, nul, (a, ty), ·, ·)  $\in I$  do
  assert nul  $\notin$  st*.NF
  % Removed the  $v_{\mathcal{A}}$  population line
  *
...
 $N_{\mathcal{A}} \leftarrow \emptyset$  % This tracks notes in  $v_{\mathcal{A}}$  on the fly
*
for all (st, ·, tx,  $\sigma$ )  $\in$  Ins' do
  ...
  for (note', C')  $\in M$  where (·, M)  $\in$  tx do
    % Compute  $v_{\mathcal{A}}$  on the fly
    *
    if (note', C')  $\in I$  then
      *
      ... % After  $N_{\overline{\mathcal{H}}}$  is updated
      if (note', (a, ty), nul, ...)  $\in I$  then
        *
        assert nul  $\notin$  st*.NF
        *
         $v_{\mathcal{A}}[\text{ty}] := v_{\mathcal{A}}[\text{ty}] + a$ 
        *
         $N_{\mathcal{A}} = N_{\mathcal{A}} \cup \text{note}'$ 
        *
        assert  $\forall \text{note} \in N_{\mathcal{A}}. \text{note} \in N_{\overline{\mathcal{H}}}$ 
        *
      assert  $\forall \text{ty} : v_{\overline{\mathcal{H}}}[\text{ty}] \leq v_0[\text{ty}] + v_{\mathcal{H}-}[\text{ty}] - v_{\mathcal{H}+}[\text{ty}]$ 
      *
    assert  $v_{\mathcal{A}} \leq v_{\overline{\mathcal{H}}}$ 
    *
    if  $\exists \text{ty} : v_{\mathcal{A}}[\text{ty}] > v_0[\text{ty}] + v_{\mathcal{H}-}[\text{ty}] - v_{\mathcal{H}+}[\text{ty}]$  then return 1
    else return 0

```

The transition is by nullifier uniqueness, pseudorandomness, and OTA binding.

Proof (Theorem 2, Transition 6). Moving $v_{\mathcal{A}}$ computation into the loop on its own does not affect the control flow: this is because all the notes in I are present as outputs of transactions in `Ins'`, since we assert (line 4 of the original Balance game) that all the notes from I must be present in the Merkle tree `st.MT` of the final state, and state maintenance oracles guarantee that they were introduced in one of the previous states `st' \in $\vec{\text{st}}$` .

The only thing that is important are the two assertions.

Let us focus on the first assertion $\forall \text{note} \in N_{\mathcal{A}}. \text{note} \in N_{\overline{\mathcal{H}}}$. It holds by induction: assuming on the previous iteration (of the inner loop) this condition holds, it can only fail if (1) some old notes in $N_{\mathcal{A}}$ have been removed from $N_{\overline{\mathcal{H}}}$, or (2) the currently added to $N_{\mathcal{A}}$ note has not been added to $N_{\overline{\mathcal{H}}}$ just a few steps before.

The first condition violates the assumption that notes in I are unspent: if `note` was removed from $N_{\mathcal{A}}$ it means its nullifier has been revealed. But in the final state `st*` this nullifier is not present (this is checked in the beginning of the game). So nullifier uniqueness must be broken.

Second condition can fail if the note was not added to $N_{\overline{\mathcal{H}}}$, which happens only if it can be received honestly. But this means that a note in I , in the beginning of the game, can be received using `SK`, and also \mathcal{A} showed a correct nullifier for it. This is impossible by nullifier pseudorandomness: \mathcal{A} cannot generate nullifiers for notes generated for `SK`.

The last assertion $v_{\mathcal{A}} \leq v_{\overline{\mathcal{H}}}$ holds because we just showed notes inclusion at each step of the iteration; and because of OTA binding the values used to compute $v_{\overline{\mathcal{H}}}$ (extracted in Balance¹) are the same as values provided by \mathcal{A} in I .

Because the predicate $v_{\overline{\mathcal{H}}} \leq \dots$ holds in the end of each iteration for all types, and thus in the end of the last iteration, and because $v_{\mathcal{A}} \leq v_{\overline{\mathcal{H}}}$ as we showed in the last transition, the predicate in the end of the game with $v_{\mathcal{A}} \leq (v_{\overline{\mathcal{H}}} \leq) \dots$ will also hold, and thus \mathcal{A} cannot win the last game unless with negligible probability. Therefore, it cannot win the original balance game. \square

F Privacy Proof

Proof (Deferred reduction from Theorem 3). We start from $\text{Privacy}_{\mathcal{A}}^0(1^\lambda)$.

Game₁: Replace all honest NIZKs (created inside $\mathcal{O}_{\text{Spend}}$ and SignTx) to simulated NIZKs.

Game₂: By nullifier pseudorandomness, replace all the honest nullifiers \mathcal{A} has not seen through $\mathcal{O}_{\text{Spend}}$ by the nullifiers over a different set of secret keys.

To switch an individual $\text{NulEval}(\text{sk}_0, r_0)$ to $\text{NulEval}(\text{sk}_1, r_1)$, we first pick a random function f and switch *all* evaluations of $\text{NulEval}(\text{sk}_0, r)$ to $f(r)$ (for all r). Since the nullifiers in $\text{tx}^{\mathcal{H}}$ have not been queried previously in $\mathcal{O}_{\text{Spend}}$, 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 $\text{tx}^{\mathcal{H}}$) evaluations of $f(\cdot)$ back to $\text{NulEval}(\text{sk}_0, \cdot)$. We now perform exactly the same steps then for the second key, de-idealizing ψ into $\text{NulEval}(\text{sk}_1, r_1)$. First we replace all evaluations of $\text{NulEval}(\text{sk}_1, r)$ to $f(r)$ (for all 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 $\text{NulEval}(\text{sk}_1, r)$, *including* the target nullifier in $\text{tx}^{\mathcal{H}}$. This is how we have just replaced $\text{NulEval}(\text{sk}_0, r_0)$ by $\text{NulEval}(\text{sk}_1, r_1)$.

Repeat the procedure for all pairs of nullifiers in $\text{tx}^{\mathcal{H}}$ in any order (i.e. it does not matter what the source and target of the replacement is since all values are pseudorandom).

Game₃: 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 $\text{OTA.Gen}(\text{pk}_b, (\text{ty}_b, v_b), r)$ and $\text{OTA.Enc}(\text{pk}_b, (\text{ty}_b, v_b, r), \xi)$ from $b = 0$ to $b = 1$ even if there are **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 **Receive** evaluations because of the **Receive** oracle in the pseudorandomness definition), perform the privacy switch, and then return the real NulEval evaluations back.

Game₄: Replace intermediary commitments com with the values from T_1 . Here we apply HID-OR for vectors, as described in Corollary 1: replacing one set of typed commitments with another, given that $\Delta_{\text{ty},0} = \Delta_{\text{ty},1}$ for each type. This transition is perfect.

Game₅: 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 EvalTree , so all NIZKs in $\mathcal{O}_{\text{Spend}}$ 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 EvalTree has the same distribution as $\text{tx}_{\mathcal{H},1}$ except for the (yet simulated) proofs, and in particular it satisfies $\mathcal{L}^{\text{spend}}$ and $\mathcal{L}^{\text{output}}$. So now we enable real NIZKs back as well, and by zero-knowledge we obtain honest proofs for T_1 .

Now **Game₅** is equivalent to $\text{Privacy}_{\mathcal{A}}^1(1^\lambda)$, which concludes the privacy proof. \square