Aggregate Cash Systems: A Cryptographic Investigation of Mimblewimble

Georg Fuchsbauer[1,2], Michele Orrù[2,1], and Yannick Seurin[3]

[1] Inria
[2] École normale supérieure, CNRS, PSL University, Paris, France
[3] ANSSI, Paris, France

{georg.fuchsbauer,michele.orru}@ens.fr
yannick.seurin@m4x.org

Abstract. Mimblewimble is an electronic cash system proposed by an anonymous author in 2016. It combines several privacy-enhancing techniques initially envisioned for Bitcoin, such as Confidential Transactions (Maxwell, 2015), non-interactive merging of transactions (Saxena, Misra, Dhar, 2014), and cut-through of transaction inputs and outputs (Maxwell, 2013). As a remarkable consequence, coins can be deleted once they have been spent while maintaining public verifiability of the ledger, which is not possible in Bitcoin. This results in tremendous space savings for the ledger and efficiency gains for new users, who must verify their view of the system.
In this paper, we provide a provable-security analysis for Mimblewimble. We give a precise syntax and formal security definitions for an abstraction of Mimblewimble that we call an *aggregate cash system*. We then formally prove the security of Mimblewimble in this definitional framework. Our results imply in particular that two natural instantiations (with Pedersen commitments and Schnorr or BLS signatures) are provably secure against inflation and coin theft under standard assumptions.

Keywords: Mimblewimble, Bitcoin, commitments, aggregate signatures.

1 Introduction

Bitcoin and the UTXO model. Proposed in 2008 and launched early 2009, Bitcoin [Nak08] is a decentralized payment system in which transactions are registered in a distributed and publicly verifiable ledger called a blockchain. Bitcoin departs from traditional account-based payment systems where transactions specify an amount moving from one account to another. Instead, each transaction consists of a list of *inputs* and a list of *outputs*.

Each output contains a value (expressed as a multiple of the currency unit, 10^{-8} bitcoin) and a short script specifying how the output can be spent. The most common script is *Pay to Public Key Hash* (P2PKH) and contains the hash of an ECDSA public key, commonly called a Bitcoin address. Each input of a transaction contains a reference to an output of a previous transaction in the blockchain and a script which must match the script of that output. In the case of P2PKH, an input must provide a public key that hashes to the address of the output it spends and a valid signature for this public key.

Each transaction spends one or more previous transaction outputs and creates one or more new outputs, with a total value not larger than the total value of coins being spent. The system is bootstrapped through special transactions called *coinbase* transactions which have outputs but no inputs and therefore create money (and also serve to incentivize the proof-of-work consensus mechanism, which allows users to agree on the valid state of the blockchain).

To avoid double-spending attacks, each output of a transaction can only be referenced once by an input of a subsequent transaction. Note that this implies that an output must necessarily be spent entirely. As transactions can have multiple outputs, change can be realized by having the sender assign part of the outputs to an address she controls. Since all transactions that ever occurred since the inception of the system are publicly available in the blockchain, whether an output has already been spent can be publicly checked. In particular, every transaction output recorded in the blockchain can be classified either as an *unspent transaction output (UTXO)* if it has not been referenced by a subsequent transaction input so far, or a *spent transaction output (STXO)* otherwise. Hence, the UTXO set "encodes" all bitcoins available to be spent, while the STXO set only contains "consumed" bitcoins and could, in theory, be deleted.

The validation mechanics in Bitcoin requires new users to download and validate the entire blockchain in order to check that their view of the system is not compromised.[4] Consequently, the security of the system and its ability to enroll new users relies on (a significant number of) Bitcoin clients to persistently store the entire blockchain. Once a new node has checked the entire blockchain, it is free to "prune" it[5] and retain only the freshly computed UTXO set, but it will not be able to convince another newcomer that this set is valid.

Consider the following toy example. A coinbase transaction creates an output txo_1 for some amount v associated with a public key pk_1. This output is spent by a transaction T_1 creating a new output txo_2 with amount v associated with a public key pk_2. Transaction T_1 contains a valid signature σ_1 under public key pk_1. Once a node has verified σ_1, it is ensured that txo_2 is valid and the node can therefore delete the coinbase transaction and T_1. By doing this, however, he cannot convince anyone else that output txo_2 is indeed valid.

At the time of writing, the size of Bitcoin's blockchain is approximately 175 GB.[6] Downloading and validating the full blockchain can take up to several days with an average connection and standard hardware. In contrast, the size of the UTXO set, containing around 55 millions elements, is only a couple of GB.

Bitcoin privacy. Despite some common misconception, Bitcoin offers a very weak level of privacy. Although users can create multiple pseudonymous addresses at will, the public availability of all transaction data often allows to link them and reveals a surprisingly large amount of identifying information, as shown in many works [AKR+13, MPJ+13, RS13, KKM14].

Several protocols have been proposed with the goal of improving on Bitcoin's privacy properties, such as Cryptonote [vS13] (implemented for example by Monero), Zerocoin [MGGR13] and Zerocash [BCG+14]. On the other hand, there are privacy-enhancing techniques compatible with Bitcoin, for example coin mixing [BNM+14, RMK14, HAB+17], to ensure payer anonymity. Below we describe three specific proposals that have paved the way for Mimblewimble.

Confidential Transactions. Confidential Transactions (CT), described by Maxwell [Max15] based on an initial idea by Back [Bac13] and now implemented by Monero, allow to hide the *values* of transaction outputs. The idea is to replace explicit amounts in transactions by homomorphic commitments: this hides the value contained in each output, but the transaction creator cannot modify this value later on.[7]

[4] So-called *Simplified Verification Payment (SPV)* clients only download much smaller pieces of the blockchain allowing them to verify specific transactions. However, they are less secure than fully validating clients and they do not contribute to the general security of the system [GCKG14, SZ16].

[5] This functionality was introduced in Bitcoin Core v0.11, see `https://github.com/bitcoin/bitcoin/blob/v0.11.0/doc/release-notes.md#block-file-pruning`.

[6] See `https://www.blockchain.com/charts/blocks-size`.

[7] Commitments are actually never publicly opened; however the opening information is used when spending a coin and remains privy to the participants.

More specifically, the amount v in an output is replaced by a Pedersen commitment $C = vH + rG$, where H and G are generators of a discrete-log-hard (additively denoted) group and r is a random value. Using the homomorphic property of the commitment scheme, one can prove that a transaction does not create money out of thin air, i.e., that the sum of the outputs is less than the sum of the inputs. Consider a transaction with input commitments $C_i = v_i H + r_i G$, $1 \leq i \leq n$, and output commitments $\hat{C}_i = \hat{v}_i H + \hat{r}_i G$, $1 \leq i \leq m$. The transaction does not create money *iff* $\sum_{i=1}^{n} v_i \geq \sum_{i=1}^{m} \hat{v}_i$. This can be proved by providing an opening (f, r) with $f \geq 0$ for $\sum_{i=1}^{n} C_i - \sum_{i=1}^{m} \hat{C}_i$, whose validity can be publicly checked. The difference f between inputs and outputs are so-called fees that reward the miner that includes the transaction in a block.

Note that arithmetic on hidden values is done modulo p, the order of the underlying group. Hence, a malicious user could spend an input worth 2 and create two outputs worth 10 and $p - 8$, which would look like a transaction creating two outputs worth 1 each. To ensure that commitments do not contain too large values that cause such mod-p reductions, a non-interactive zero-knowledge (NIZK) proof that the committed value is in $[0, v_{\max}]$ (a so-called *range proof*) is added to each commitment, where $v_{\max}$ is small compared to p.

CoinJoin. When a Bitcoin transaction has multiple inputs and multiple outputs, nothing can be inferred about "which input goes to which output" beyond what is imposed by the values of the coins (e.g., if a transaction has two inputs with values 10 BTC and 1 BTC, and two outputs with values 10 BTC and 1 BTC, all that can be said is that at least 9 BTC flowed from the first input to the first output). CoinJoin [Max13a] builds on this technical principle to let different users create a single transaction that combines all of their inputs and outputs. When all inputs and outputs have the same value, this perfectly mixes the coins. Note that unlike CT, CoinJoin does not require any change to the Bitcoin protocol and is already used in practice. However, this protocol is interactive as participants need all input and output addresses to build the transaction. Saxena *et al.* [SMD14] proposed a modification of the Bitcoin protocol which essentially allows users to perform CoinJoin non-interactively and which relies on so-called *composite* signatures.[8]

Cut-through. A basic property of the UTXO model is that a sequence of two transactions, a first one spending an output txo_1 and creating txo_2, followed by a second one spending txo_2 and creating txo_3, is equivalent to a single *cut-through* transaction spending txo_1 and creating txo_3. While such an optimization is impossible once transactions have been included in the blockchain (as mentioned before, this would violate public verifiability of the blockchain), this has been suggested [Max13b] for *unconfirmed* transactions, i.e., transactions broadcast to the Bitcoin network but not included in a block yet. As we will see, the main added benefit of Mimblewimble is to allow *post-confirmation cut-through*.

Mimblewimble. Mimblewimble was first proposed by an anonymous author in 2016 [Jed16]. The idea was then developed further by Poelstra [Poe16]. At the time of writing, there are at least two independent implementations of Mimblewimble as a cryptocurrency: one is called Grin[9], the other Beam.[10]

Mimblewimble combines in a clever way CT, a non-interactive version of CoinJoin, and cut-through of transaction inputs and outputs. As with CT, a coin is a commitment $C = vH + rG$ to its value v using randomness r, coming with a range proof π. If CT were actually employed in Bitcoin, spending a CT-protected output would require the knowledge of the opening of

[8] An earlier, anonymous version of the paper used the name *one-way aggregate signature* (OWAS), see `https://bitcointalk.org/index.php?topic=290971`. Composite signatures are very similar to aggregate signatures [BGLS03].

[9] See `http://grin-tech.org` and `https://github.com/mimblewimble/grin/blob/master/doc/intro.md`.

[10] See `https://www.beam-mw.com`.

the commitment *and*, as for a standard output, of the secret key associated with the address controlling the coin. Mimblewimble goes one step further and completely abandons the notion of addresses or more generally scripts: spending a coin *only* requires knowledge of the opening of the commitment. As a result, ownership of a coin $C = vH + rG$ is equivalent to the knowledge of its opening, and the randomness r of the commitment now acts as the *secret key* for the coin.

Exactly as in Bitcoin, a Mimblewimble transaction specifies a list $\mathbf{C} = (C_1, \ldots, C_n)$ of input coins (which must be coins existing in the system) and a list $\hat{\mathbf{C}} = (\hat{C}_1, \ldots, \hat{C}_m)$ of output coins, where $C_i = v_i H + r_i G$ for $1 \le i \le n$ and $\hat{C}_i = \hat{v}_i H + \hat{r}_i G$ for $1 \le i \le m$. We will detail later how exactly such a transaction is constructed. Leaving fees aside for simplicity, the transaction is balanced (i.e., does not create money) *iff* $\sum \hat{v}_i - \sum v_i = 0$, which, letting $\sum \mathbf{C}$ denote $\sum_{i=1}^{n} C_i$, is equivalent to

$$\sum \hat{\mathbf{C}} - \sum \mathbf{C} = \sum (\hat{v}_i H + \hat{r}_i G) - \sum (v_i H + r_i G) = \left(\sum \hat{r}_i - \sum r_i\right) G \, .$$

In other words, knowledge of the opening of all coins in the transaction *and* balancedness of the transaction implies knowledge of the discrete logarithm in base G of $E := \sum \hat{\mathbf{C}} - \sum \mathbf{C}$, called the *excess* of the transaction in Mimblewimble jargon. Revealing the opening $(0, r := \sum \hat{r}_i - \sum r_i)$ of the excess E as in CT would leak too much information (e.g., together with the openings of the input coins and of all output coins except one, this would yield the opening of the remaining output coin); however, knowledge of r can be *proved* by providing a valid signature (on the empty message) under public key E using some discrete-log-based signature scheme. Intuitively, as long as the commitment scheme is binding and the signature scheme is unforgeable, it should be infeasible to compute a valid signature for an unbalanced transaction.

Transactions (legitimately) creating money, such as coinbase transactions, can easily be incorporated by letting the *supply* s (i.e., the number of monetary units created by the transaction) be explicitly specified and redefining the excess of the transaction as $E := \sum \hat{\mathbf{C}} - \sum \mathbf{C} - sH$. All in all, a Mimblewimble transaction is a tuple

$$\mathsf{tx} = (s, \mathbf{C}, \hat{\mathbf{C}}, K) \quad \text{with} \quad K := (\boldsymbol{\pi}, \mathbf{E}, \sigma) \, , \tag{1}$$

where s is the supply, $\mathbf{C}$ is the input coin list, $\hat{\mathbf{C}}$ is the output coin list, and K is the so-called *kernel*, which contains the list $\boldsymbol{\pi}$ of range proofs for output coins,[11] the (list of) transaction excesses $\mathbf{E}$ (as there can be several; see below), and a signature σ.[12]

Such transactions can now easily be merged non-interactively *à la* CoinJoin: consider $\mathsf{tx}_0 = (s_0, \mathbf{C}_0, \hat{\mathbf{C}}_0, (\boldsymbol{\pi}_0, E_0, \sigma_0))$ and $\mathsf{tx}_1 = (s_1, \mathbf{C}_1, \hat{\mathbf{C}}_1, (\boldsymbol{\pi}_1, E_1, \sigma_1))$; then the *aggregate transaction* tx resulting from merging tx_0 and tx_1 is simply

$$\mathsf{tx} := \left(s_0 + s_1, \mathbf{C}_0 \parallel \mathbf{C}_1, \hat{\mathbf{C}}_0 \parallel \hat{\mathbf{C}}_1, (\boldsymbol{\pi}_0 \parallel \boldsymbol{\pi}_1, (E_0, E_1), (\sigma_0, \sigma_1))\right) \, . \tag{2}$$

Moreover, if the signature scheme supports aggregation, as for example the BLS scheme [BGLS03, BNN07], the pair (σ_0, σ_1) can be replaced by a compact aggregate signature σ for the public keys $\mathbf{E} := (E_0, E_1)$.

An aggregate transaction $(s, \mathbf{C}, \hat{\mathbf{C}}, (\boldsymbol{\pi}, \mathbf{E}, \sigma))$ is valid if all range proofs verify, σ is a valid aggregate signature for $\mathbf{E}$ and if

$$\sum \hat{\mathbf{C}} - \sum \mathbf{C} - sH = \sum \mathbf{E} \, . \tag{3}$$

As transactions can be recursively aggregated, the resulting kernel will contain a list $\mathbf{E}$ of kernel excesses, one for each transaction that has been aggregated.

[11] Since inputs must be coins that already exist in the system, their range proofs are contained in the kernels of the transactions that created them.

[12] A transaction fee can easily be added to the picture by making its amount f explicit and adding fH to the transaction excess. For simplicity, we omit it in this paper.

The main novelty of Mimblewimble, namely cut-through, naturally emerges from the way transactions are aggregated and validated. Assume that some coin C appears as an output in tx_0 and as an input in tx_1; then, one can erase C from the input and output lists of the aggregate transaction tx, and tx will still be valid since (3) will still hold. Hence, each time an output of a transaction tx_0 is spent by a subsequent transaction tx_1, this output can be "forgotten" without losing the ability to validate the resulting aggregate transaction.

In Mimblewimble the ledger is itself a transaction of the form (1), which starts out empty, and to which transactions are recursively aggregated as they are added to the ledger. We assume that for a transaction to be allowed onto the ledger, its input list must be contained in the output list of the ledger (this corresponds to the natural requirement that only coins that exist in the ledger can be spent). Then, it is easy to see that the following holds:

(i) the supply s of the ledger is equal to the sum of the supplies of all transactions added to the ledger so far;
(ii) the input coin list of the ledger is always empty.

Property (i) follows from the definition of aggregation in (2). Property (ii) follows inductively. At the inception of the system the ledger is empty (thus the first transaction added to the ledger must be a transaction with an empty input coin list and non-zero supply, a *minting* transaction). Any transaction tx added to the ledger must have its input coins contained in the output coin list of the ledger; thus cut-through will remove all of them from the joint input list, hence the updated ledger again has no input coins (and the coins spent by tx are deleted from its outputs). The ledger in Mimblewimble is thus a single aggregate transaction whose supply s is equal to the amount of money that was created in the system and whose output coin list $\hat{\mathbf{C}}$ is the analogue of the UTXO set in Bitcoin. Its kernel K allows to cryptographically verify its validity. The history of all transactions that have occurred is not retained, and only one kernel excess per transaction (a very short piece of information) is recorded.

Our contribution. We believe it is crucial that protocols undergo a formal security assessment and that the cryptographic guarantees they provide must be well understood before deployment. To this end, we provide a provable-security treatment for Mimblewimble. A first attempt at proving its security was partly undertaken by Poelstra [Poe16]. We follow a different approach: we put forward a general syntax and a framework of game-based security definitions for an abstraction of Mimblewimble that we dub an *aggregate cash system*.

Formalizing security for a cash system requires care. For example, Zerocoin [MGGR13] was recently found to be vulnerable to *denial-of-spending* attacks [RTRS18] that were not captured by the security model in which Zerocoin was proved secure. To avoid such pitfalls, we keep the syntax simple, while allowing to express meaningful security definitions. We formulate two natural properties that define the security of a cash system: *inflation-resistance* ensures that the only way money can be created in a system is explicitly via the supply contained in transactions; *resistance to coin theft* guarantees that no one can spend a user's coins as long as she keeps her keys safe. We moreover define a privacy notion, *transaction indistinguishability*, which states that a transaction does not reveal anything about the values it transfers from its inputs to its outputs.

We then give a black-box construction of an aggregate cash system, which naturally generalizes Mimblewimble, from a homomorphic commitment scheme COM, an (aggregate) signature scheme SIG, and a NIZK range-proof system Π. We believe that such a modular treatment will ease the exploration of post-quantum instantiations of Mimblewimble or related systems.

Note that in our description of Mimblewimble, we have note yet explained how to actually create a transaction that transfers some amount ρ of money from a sender to a receiver. It turns out that this is a delicate question. The initial description of the protocol [Jed16] proposed the following one-round procedure:

– the sender selects input coins $\mathbf{C}$ of total value $v \geq \rho$; it creates *change coins* $\mathbf{C}'$ of total value $v - \rho$ and sends $\mathbf{C}$, $\mathbf{C}'$, range proofs for $\mathbf{C}'$ and the opening $(-\rho, k)$ of $\sum \mathbf{C}' - \sum \mathbf{C}$ to the receiver (over a secure channel);
– the receiver creates additional output coins $\mathbf{C}''$ (and range proofs) of total value ρ with keys (k_i''), computes a signature σ with the secret key $k + \sum k_i''$ and defines the transaction $\mathsf{tx} = (0, \mathbf{C}, \mathbf{C}' \parallel \mathbf{C}'', (\boldsymbol{\pi}, E = \sum \mathbf{C}' + \sum \mathbf{C}'' - \sum \mathbf{C}, \sigma))$.

However, a subtle problem arises with this protocol. Once the transaction has been added to the ledger, the change outputs $\mathbf{C}'$ should only be spendable by the sender, who owns them. It turns out that the receiver is also able to spend them by "reverting" the transaction tx. Indeed, he knows the range proofs for coins in $\mathbf{C}$ and the secret key $(-k - \sum k_i'')$ for the transaction with inputs $\mathbf{C}' \parallel \mathbf{C}''$ and outputs $\mathbf{C}$. Arguably, the sender is given back her initial input coins in the process, but (i) she could have deleted the secret keys for these old coins, making them unspendable, and (ii) this violates any meaningful formalization of security against coin theft.

A natural way to prevent such a malicious behavior would be to let the sender and the receiver, each holding a share of the secret key corresponding to public key $E \coloneqq \sum \mathbf{C}' \parallel \mathbf{C}'' - \sum \mathbf{C}$, engage in a two-party interactive protocol to compute σ. Actually, this seems to be the way Grin is taking, although, to the best of our knowledge, the problem described above with the original protocol has never been documented.

We show that the spirit of the original non-interactive protocol can be salvaged, so a sender can make a payment to a receiver without the latter's active involvement. In our solution the sender first constructs a full-fledged transaction tx spending $\mathbf{C}$ and creating change coins $\mathbf{C}'$ as well as a special output coin $C = \rho H + kG$, and sends tx and the opening (ρ, k) of the special coin to the receiver. (Note that, unlike in the previous case, k is now independent from the keys of the coins in $\mathbf{C}$ and $\mathbf{C}'$.) The receiver then creates a second transaction tx' spending the special coin C and creating its own output coins $\mathbf{C}''$ and aggregates tx and tx'. As intended, this results in a transaction with inputs $\mathbf{C}$ and outputs $\mathbf{C}' \parallel \mathbf{C}''$ since C is discarded by cut-through. The only drawback of this procedure is that the final transaction, being the aggregate of two transactions, now has two kernel excesses instead of one for the interactive protocol mentioned above.

After specifying our protocol $\mathsf{MW}[\mathsf{COM}, \mathsf{SIG}, \Pi]$, we turn to proving its security in our definitional framework. To this end, we formulate two security notions, called EUF-NZO and EUF-CRO, tying the commitment scheme and the signature scheme together. Assuming that Π is simulation-extractable [DDO+01, Gro06], we show that EUF-NZO-security for the pair $(\mathsf{COM}, \mathsf{SIG})$ implies that MW is resistant to inflation, while EUF-CRO-security implies that MW is resistant to coin theft. Transaction indistinguishability follows from zero-knowledge of Π and COM being hiding.

Finally, we consider two natural instantiations of $\mathsf{MW}[\mathsf{COM}, \mathsf{SIG}, \Pi]$. For each, we let COM be the Pedersen commitment scheme [Ped92]. When SIG is instantiated with the Schnorr signature scheme [Sch91], we show that the pair $(\mathsf{COM}, \mathsf{SIG})$ is EUF-NZO- and EUF-CRO-secure under the Discrete Logarithm assumption. When SIG is instantiated with the BLS signature scheme [BLS01], we show that the pair $(\mathsf{COM}, \mathsf{SIG})$ is EUF-NZO- and EUF-CRO-secure under the CDH assumption. Both proofs are in the random-oracle model. BLS signatures have the additional benefit of supporting aggregation [BGLS03, BNN07], so that the ledger kernel always contains a short aggregate signature, independently of the number of transactions that have been added to the ledger. We stress that, unlike Zerocash [BCG+14], none of these two instantiations require a trusted setup.

Future work. As already noted by Poelstra [Poe16], given the aggregate of two transactions for which no cut-through occurred, it is possible to distinguish the inputs and outputs of each original transaction by solving a simple subset sum problem based on the two kernel excesses contained in the aggregate transaction. To achieve indistinguishability of aggregate transactions, Grin developers have proposed to add a so-called kernel offset (a random commitment to zero) to each transaction, and to add them when merging transactions (so that any transaction now contains a list of kernel excesses and a single kernel offset). We leave the formal analysis of this proposal for future work.

2 Preliminaries

2.1 General Notation

We denote the (closed) integer interval from a to b by $[a, b]$. We use $[b]$ as shorthand for $[1, b]$. A function $\mu\colon \mathbb{N} \to [0, 1]$ is negligible (denoted $\mu = \mathsf{negl}$) if for all $c \in \mathbb{N}$ there exists $\lambda_c \in \mathbb{N}$ such that $\mu(\lambda) \leq \lambda^{-c}$ for all $\lambda \geq \lambda_c$. A function ν is *overwhelming* if $1 - \nu = \mathsf{negl}$. Given a non-empty finite set S, we let $x \leftarrow\!\!\mathrm{s}\, S$ denote the operation of sampling an element x from S uniformly at random. By $y \coloneqq M(x_1, \ldots; r)$ we denote the operation of running algorithm M on inputs $x_1, \ldots$ and coins r and letting y denote the output. By $y \leftarrow M(x_1, \ldots)$, we denote letting $y \coloneqq M(x_1, \ldots; r)$ for random r, and $[M(x_1, \ldots)]$ is the set of values that have positive probability of being output by M on inputs $x_1, \ldots$ If an algorithm calls a subroutine which returns $\perp$, we assume it stops and returns $\perp$ (note that this does not hold for an adversary calling an *oracle* which returns $\perp$).

A list $\mathbf{L} = (x_1, \ldots, x_n)$, also denoted $(x_i)_{i=1}^{n}$, is a finite sequence. The length of a list $\mathbf{L}$ is denoted $|\mathbf{L}|$. For $i = 1, \ldots, |\mathbf{L}|$, the i-th element of $\mathbf{L}$ is denoted $\mathbf{L}[i]$, or L_i when no confusion is possible. By $\mathbf{L}_0 \,\|\, \mathbf{L}_1$ we denote the list $\mathbf{L}_0$ followed by $\mathbf{L}_1$. The empty list is denoted $(\,)$. Given a list $\mathbf{L}$ of elements of an additive group, we let $\sum \mathbf{L}$ denote the sum of all elements of $\mathbf{L}$. Let $\mathbf{L}_0$ and $\mathbf{L}_1$ be two lists, each without repetition. We write $\mathbf{L}_0 \subseteq \mathbf{L}_1$ *iff* each element of $\mathbf{L}_0$ also appears in $\mathbf{L}_1$. We define $\mathbf{L}_0 \cap \mathbf{L}_1$ to be the list of all elements that simultaneously appear in both $\mathbf{L}_0$ and $\mathbf{L}_1$, ordered as in $\mathbf{L}_0$. The difference between $\mathbf{L}_0$ and $\mathbf{L}_1$, denoted $\mathbf{L}_0 - \mathbf{L}_1$, is the list of all elements of $\mathbf{L}_0$ that do not appear in $\mathbf{L}_1$, ordered as in $\mathbf{L}_0$. So, for example $(1, 2, 3) - (2, 4) = (1, 3)$. We define the *cut-through* of two lists $\mathbf{L}_0$ and $\mathbf{L}_1$, denoted $\mathsf{cut}(\mathbf{L}_0, \mathbf{L}_1)$, as

$$\mathsf{cut}(\mathbf{L}_0, \mathbf{L}_1) \coloneqq (\mathbf{L}_0 - \mathbf{L}_1, \mathbf{L}_1 - \mathbf{L}_0)\ .$$

2.2 Cryptographic Primitives

We introduce the three building blocks we will use to construct an aggregate cash system: a commitment scheme COM, an aggregate signature scheme SIG, and a non-interactive zero-knowledge proof system $\mathsf{\Pi}$. For compatibility reasons, the setup algorithms for each of these schemes are split: a common algorithm $\mathsf{MainSetup}(1^\lambda)$ first returns *main* parameters mp (specifying e.g. an abelian group), and specific algorithms $\mathsf{COM.Setup}$, $\mathsf{SIG.Setup}$, and $\mathsf{\Pi.Setup}$ take as input mp and return the specific parameters cp, sp, and crs for each primitive. We assume that mp is contained in cp, sp, and crs.

Commitment scheme. A commitment scheme COM consists of the following algorithms:

- $\mathsf{cp} \leftarrow \mathsf{COM.Setup}(\mathsf{mp})$: the setup algorithm takes as input main parameters mp and outputs commitment parameters cp, which implicitly define a value space $\mathcal{V}_{\mathsf{cp}}$, a randomness space $\mathcal{R}_{\mathsf{cp}}$, and a commitment space $\mathcal{C}_{\mathsf{cp}}$;
- $C \coloneqq \mathsf{COM.Cmt}(\mathsf{cp}, v, r)$: the (deterministic) commitment algorithm takes as input commitment parameters cp, a value $v \in \mathcal{V}_{\mathsf{cp}}$ and randomness $r \in \mathcal{R}_{\mathsf{cp}}$, and outputs a commitment $C \in \mathcal{C}_{\mathsf{cp}}$.

```
Game $\mathrm{HID}_{\mathsf{COM},\mathcal{A}}(\lambda)$
    $b \leftarrow\!\!\$ \{0,1\}$
    $\mathsf{mp} \leftarrow \mathsf{MainSetup}(1^\lambda)$
    $\mathsf{cp} \leftarrow \mathsf{COM.Setup}(\mathsf{mp})$
    $b' \leftarrow \mathcal{A}^{\textsc{Commit}}(\mathsf{cp})$
    return $b = b'$
```

```
Oracle $\textsc{Commit}(v_0, v_1)$
    $r \leftarrow\!\!\$ \mathcal{R}_{\mathsf{cp}}$
    $C \coloneqq \mathsf{COM.Cmt}(\mathsf{cp}, v_b, r)$
    return $C$
```

```
Game $\mathrm{BND}_{\mathsf{COM},\mathcal{A}}(\lambda)$
    $\mathsf{mp} \leftarrow \mathsf{MainSetup}(1^\lambda)$
    $\mathsf{cp} \leftarrow \mathsf{COM.Setup}(\mathsf{mp})$
    $(v_0, r_0, v_1, r_1) \leftarrow \mathcal{A}(\mathsf{cp})$
    $C_0 \coloneqq \mathsf{COM.Cmt}(\mathsf{cp}, v_0, r_0)$
    $C_1 \coloneqq \mathsf{COM.Cmt}(\mathsf{cp}, v_1, r_1)$
    return $v_0 \neq v_1$ and $C_0 = C_1$
```

Fig. 1. The games for hiding and binding of a commitment scheme COM.

In most instantiations, given a value $v \in \mathcal{V}_{\mathsf{cp}}$, the sender picks $r \leftarrow\!\!\$ \mathcal{R}_{\mathsf{cp}}$ uniformly at random and computes the commitment $C = \mathsf{COM.Cmt}(\mathsf{cp}, v, r)$. To *open* the commitment, the sender reveals (v, r) so anyone can verify that $\mathsf{COM.Cmt}(\mathsf{cp}, v, r) = C$.

We require commitment schemes to be hiding and binding. Loosely speaking, a scheme is *hiding* if the commitment C reveals no information about v. A scheme is *binding* if the sender cannot open the commitment in two different ways. The two corresponding security games are depicted in Figure 1 and the security definitions are given below.

Definition 1 (Hiding). *Let game* HID *be as defined Figure 1. A commitment scheme* COM *is* hiding *if for any p.p.t. adversary* $\mathcal{A}$*:*

$$\mathsf{Adv}^{\mathrm{hid}}_{\mathsf{COM},\mathcal{A}}(\lambda) \coloneqq 2 \cdot \left|\Pr\left[\mathrm{HID}_{\mathsf{COM},\mathcal{A}}(\lambda) = \mathbf{true}\right] - \tfrac{1}{2}\right| = \mathsf{negl}(\lambda) \ .$$

Definition 2 (Binding). *Let game* BND *be as defined in Figure 1. A commitment scheme* COM *is* binding *if for any p.p.t. adversary* $\mathcal{A}$*:*

$$\mathsf{Adv}^{\mathrm{bnd}}_{\mathsf{COM},\mathcal{A}}(\lambda) \coloneqq \Pr\left[\mathrm{BND}_{\mathsf{COM},\mathcal{A}}(\lambda) = \mathbf{true}\right] = \mathsf{negl}(\lambda) \ .$$

Lemma 3 (Collision-resistance). *Let* COM *be a (binding and hiding) commitment scheme. Then for any* $(v_0, v_1) \in \mathcal{V}_{\mathsf{cp}}^2$*, the probability that* $\mathsf{Cmt}(\mathsf{cp}, v_0, r_0) = \mathsf{Cmt}(\mathsf{cp}, v_1, r_1)$ *for* $r_0, r_1 \leftarrow\!\!\$ \mathcal{R}_{\mathsf{cp}}$ *is negligible.*

The proof of the lemma is straightforward: for $v_0 \neq v_1$ this would break binding and for $v_0 = v_1$ it would break hiding.

A commitment scheme is (additively) *homomorphic* if the value, randomness, and commitment spaces are groups (denoted additively) and for any commitment parameters cp, any $v_0, v_1 \in \mathcal{V}_{\mathsf{cp}}$, and any $r_0, r_1 \in \mathcal{R}_{\mathsf{cp}}$, we have:

$$\mathsf{COM.Cmt}(\mathsf{cp}, v_0, r_0) + \mathsf{COM.Cmt}(\mathsf{cp}, v_1, r_1) = \mathsf{COM.Cmt}(\mathsf{cp}, v_0 + v_1, r_0 + r_1) \ .$$

Recursive aggregate signature scheme. An aggregate signature scheme allows to (publicly) combine an arbitrary number n of signatures (from potentially distinct users and on potentially distinct messages) into a single (ideally short) signature [BGLS03, LMRS04, BNN07]. Traditionally, the syntax of an aggregate signature scheme only allows the aggregation algorithm to take as input individual signatures. In this paper, we consider aggregate signature schemes supporting recursive aggregation, meaning that the aggregation algorithm can take as input aggregate signatures (as for example the schemes based on BLS signatures [BGLS03, BNN07]). A recursive aggregate signature scheme SIG consists of the following algorithms:

```
Game EUF-CMA$_{\mathsf{SIG},\mathcal{A}}(\lambda)$
    $\mathsf{Q} := (\,);\ \mathsf{mp} \leftarrow \mathsf{MainSetup}(1^\lambda)$
    $\mathsf{sp} \leftarrow \mathsf{SIG.Setup}(\mathsf{mp})\,;\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{SIG.KeyGen}(\mathsf{sp})$
    $(\mathbf{L}, \sigma) \leftarrow \mathcal{A}^{\textsc{Sign}}(\mathsf{pk})$
    return $(\exists\, m : (\mathsf{pk}, m) \in \mathbf{L} \wedge m \notin \mathsf{Q})$ and $\mathsf{SIG.Ver}(\mathsf{sp}, \mathbf{L}, \sigma)$

Oracle Sign$(m)$
    $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sk}, m)$
    $\mathsf{Q} := \mathsf{Q} \,\|\, (m)$
    return $\sigma$
```

Fig. 2. The EUF-CMA security game for an aggregate signature scheme SIG.

- $\mathsf{sp} \leftarrow \mathsf{SIG.Setup}(\mathsf{mp})$: the setup algorithm takes as input main parameters mp and outputs signature parameters sp, which implicitly define a secret key space $\mathcal{S}_{\mathsf{sp}}$ and a public key space $\mathcal{P}_{\mathsf{sp}}$ (we let the message space be $\{0,1\}^*$);
- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{SIG.KeyGen}(\mathsf{sp})$: the key generation algorithm takes signature parameters sp and outputs a secret key $\mathsf{sk} \in \mathcal{S}_{\mathsf{sp}}$ and a public key $\mathsf{pk} \in \mathcal{P}_{\mathsf{sp}}$;
- $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sp}, \mathsf{sk}, m)$: the signing algorithm takes as input parameters sp, a secret key $\mathsf{sk} \in \mathcal{S}_{\mathsf{sp}}$, and a message $m \in \{0,1\}^*$ and outputs a signature σ;
- $\sigma \leftarrow \mathsf{SIG.Agg}(\mathsf{sp}, (\mathbf{L}_0, \sigma_0), (\mathbf{L}_1, \sigma_1))$: the aggregation algorithm takes parameters sp and two pairs of public-key/message lists $\mathbf{L}_i = ((\mathsf{pk}_{i,j}, m_{i,j}))_{j=1}^{|\mathbf{L}_i|}$ and (aggregate) signatures σ_i, $i = 0, 1$; it returns an aggregate signature σ;
- $\mathit{bool} \leftarrow \mathsf{SIG.Ver}(\mathsf{sp}, \mathbf{L}, \sigma)$: the (deterministic) verification algorithm takes parameters sp, a list $\mathbf{L} = ((\mathsf{pk}_i, m_i))_{i=1}^{|\mathbf{L}|}$ of public-key/message pairs, and an aggregate signature σ; it returns **true** or **false** indicating validity of σ.

Correctness of a recursive aggregate signature scheme is defined recursively. An aggregate signature scheme is *correct* if for every λ, for any message m, for every $\mathsf{mp} \in [\mathsf{MainSetup}(1^\lambda)]$, $\mathsf{sp} \in [\mathsf{SIG.Setup}(\mathsf{mp})]$, $(\mathsf{sk}, \mathsf{pk}) \in [\mathsf{SIG.KeyGen}(\mathsf{sp})]$, $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sp}, \mathsf{sk}, m)$ and every $(\mathbf{L}_0, \sigma_0), (\mathbf{L}_1, \sigma_1)$ such that $\mathsf{SIG.Ver}(\mathsf{sp}, \mathbf{L}_0, \sigma_0) = \mathbf{true} = \mathsf{SIG.Ver}(\mathsf{sp}, \mathbf{L}_1, \sigma_1)$ we have

$$\mathsf{SIG.Ver}(\mathsf{sp}, ((\mathsf{pk}, m)), \sigma) = \mathbf{true} \quad \text{and}$$
$$\mathsf{SIG.Ver}(\mathsf{sp}, \mathbf{L}_0 \,\|\, \mathbf{L}_1, \mathsf{SIG.Agg}(\mathsf{sp}, (\mathbf{L}_0, \sigma_0), (\mathbf{L}_1, \sigma_1))) = \mathbf{true}\ .$$

Note that for any recursive aggregate signature scheme, one can define a different aggregation algorithm $\mathsf{SIG.Agg}'$ which takes as input a list of triples $((\mathsf{pk}_i, m_i, \sigma_i))_{i=1}^n$ and returns an aggregate signature σ for $((\mathsf{pk}_i, m_i))_{i=1}^n$, which is the standard syntax for an aggregate signature scheme. Algorithm $\mathsf{SIG.Agg}'$ can be built from $\mathsf{SIG.Agg}$ by calling it recursively $n-1$ times, aggregating one signature at a time.

The standard security notion for aggregate signature schemes is *existential unforgeability under chosen-message attack* (EUF-CMA) [BGLS03, BNN07].

Definition 4 (EUF-CMA). *Let game* EUF-CMA *be as defined in Figure 2. An aggregate signature scheme* SIG *is* existentially unforgeable under chosen-message attack *if for any p.p.t. adversary* $\mathcal{A}$,

$$\mathsf{Adv}^{\text{euf-cma}}_{\mathsf{SIG},\mathcal{A}}(\lambda) := \Pr\left[\text{EUF-CMA}_{\mathsf{SIG},\mathcal{A}}(\lambda) = \mathbf{true}\right] = \mathsf{negl}(\lambda)\ .$$

Note that any standard signature scheme can be trivially turned into an aggregate signature scheme by letting the aggregation algorithm simply concatenate signatures, i.e., $\mathsf{SIG.Agg}(\mathsf{sp}, (\mathbf{L}_0, \sigma_0), (\mathbf{L}_1, \sigma_1))$ returns (σ_0, σ_1). Although such aggregate signatures are not compact, standard EUF-CMA-security for the original scheme implies EUF-CMA-security in the sense of Definition 4 for this aggregate signature scheme. This allows us to capture standard and (compact) aggregate signature schemes, such as the ones proposed in [BGLS03, BNN07], in a single framework.

```
Game EUF-NZO$_{\mathsf{COM},\mathsf{SIG},\mathcal{A}}(\lambda)$
$\mathsf{mp} \leftarrow \mathsf{MainSetup}(1^\lambda)\,;\ \mathsf{cp} \leftarrow \mathsf{COM.Setup}(\mathsf{mp})\,;\ \mathsf{sp} \leftarrow \mathsf{SIG.Setup}(\mathsf{mp})$
$(\mathbf{L}, \sigma, (v, r)) \leftarrow \mathcal{A}(\mathsf{cp}, \mathsf{sp})$
$((X_i, m_i))_{i=1}^n := \mathbf{L}$
return $v \neq 0$ and $\sum_{i=1}^n X_i = \mathsf{COM.Cmt}(\mathsf{cp}, v, r)$ and $\mathsf{SIG.Ver}(\mathsf{sp}, \mathbf{L}, \sigma)$
```

Fig. 3. The EUF-NZO security game for a pair of compatible additively homomorphic commitment and aggregate signature schemes $(\mathsf{COM}, \mathsf{SIG})$.

Compatibility. For our aggregate cash system, we require the commitment scheme COM and the aggregate signature scheme SIG to satisfy some "combined" security notions. We say that COM and SIG are *compatible* if they use the same $\mathsf{MainSetup}$ and if for any λ, any $\mathsf{mp} \in [\mathsf{MainSetup}(1^\lambda)]$, $\mathsf{cp} \in [\mathsf{COM.Setup}(\mathsf{mp})]$ and $\mathsf{sp} \in [\mathsf{SIG.Setup}(\mathsf{mp})]$, the following holds:

- $\mathcal{S}_{\mathsf{sp}} = \mathcal{R}_{\mathsf{cp}}$, i.e., the secret key space of SIG is the same as the randomness space of COM;
- $\mathcal{P}_{\mathsf{sp}} = \mathcal{C}_{\mathsf{cp}}$, i.e., the public key space of SIG is the same as the commitment space of COM;
- $\mathsf{SIG.KeyGen}$ proceeds by drawing $\mathsf{sk} \leftarrow_{\$} \mathcal{R}_{\mathsf{cp}}$ and setting $\mathsf{pk} := \mathsf{COM.Cmt}(\mathsf{cp}, 0, \mathsf{sk})$.

We give two security definitions for compatible commitment and aggregate signature schemes. The first one roughly states that only commitments to zero can serve as signature-verification keys; more precisely, a p.p.t. adversary cannot simultaneously produce a signature for a (set of) freely chosen public key(s) *and* a non-zero opening of (the sum of) the public key(s).

Definition 5 (EUF-NZO). *Let game* EUF-NZO *be as defined in Figure 3. A pair of compatible homomorphic commitment and aggregate signature schemes* $(\mathsf{COM}, \mathsf{SIG})$ *is* existentially unforgeable with non-zero opening *if for any p.p.t. adversary* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\text{euf-nzo}}_{\mathsf{COM},\mathsf{SIG},\mathcal{A}}(\lambda) := \Pr\left[\text{EUF-NZO}_{\mathsf{COM},\mathsf{SIG},\mathcal{A}}(\lambda) = \mathbf{true}\right] = \mathsf{negl}(\lambda)\ .$$

EUF-NZO-security of the pair $(\mathsf{COM}, \mathsf{SIG})$ implies that COM is binding, as shown in Appendix A.1.

The second security definition is more involved. It roughly states that, given a challenge public key C^*, no adversary can produce a signature under $-C^*$. Moreover, we only require the adversary to make a signature under any keys $X_1, \ldots, X_n$, as long as it knows an opening to the difference between their sum and $-C^*$. This must even hold if the adversary is given a signing oracle for keys related to C^*. Informally, the adversary is faced with the following dilemma: either it picks public keys $X_1, \ldots, X_n$ honestly, so it can produce a signature but it cannot open $\sum X_i + C^*$; or it includes $-C^*$ within the public keys, allowing it to open $\sum X_i + C^*$, but then it cannot produce a signature.

Definition 6 (EUF-CRO). *Let game* EUF-CRO *be as defined in Figure 4. A pair of compatible homomorphic commitment and aggregate signature schemes* $(\mathsf{COM}, \mathsf{SIG})$ *is* existentially unforgeable with challenge-related opening *if for any p.p.t. adversary* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\text{euf-cro}}_{\mathsf{COM},\mathsf{SIG},\mathcal{A}}(\lambda) := \Pr\left[\text{EUF-CRO}_{\mathsf{COM},\mathsf{SIG},\mathcal{A}}(\lambda) = \mathbf{true}\right] = \mathsf{negl}(\lambda)\ .$$

NIZK. Let R be an efficiently computable ternary relation. For triplets $(\mathsf{mp}, u, w) \in \mathsf{R}$ we call u the statement and w the witness. A non-interactive proof system Π for R consists of the following three algorithms:

- $\mathsf{crs} \leftarrow \Pi.\mathsf{Setup}(\mathsf{mp})$: the setup algorithm takes as input main parameters mp and outputs a common reference string (CRS) crs;

```
Game EUF-CRO$_{\mathsf{COM},\mathsf{SIG},\mathcal{A}}(\lambda)$
$\mathsf{mp} \leftarrow \mathsf{MainSetup}(1^\lambda)\,;\ \mathsf{cp} \leftarrow \mathsf{COM.Setup}(\mathsf{mp})$
$\mathsf{sp} \leftarrow \mathsf{SIG.Setup}(\mathsf{mp})\,;\ (r^*, C^*) \leftarrow \mathsf{SIG.KeyGen}(\mathsf{sp})$
$(\mathbf{L}, \sigma, (v, r)) \leftarrow \mathcal{A}^{\textsc{Sign}'}(\mathsf{cp}, \mathsf{sp}, C^*)\,;\ ((X_i, m_i))_{i=1}^n := \mathbf{L}$
return $\sum_{i=1}^n X_i = -C^* + \mathsf{COM.Cmt}(\mathsf{cp}, v, r)$ and $\mathsf{SIG.Ver}(\mathsf{sp}, \mathbf{L}, \sigma)$

Oracle $\textsc{Sign}'(a, m)$
$\mathsf{sk}' := a + r^*$
return $\mathsf{SIG.Sign}(\mathsf{sp}, \mathsf{sk}', m)$
```

Fig. 4. The EUF-CRO security game for a pair of compatible additively homomorphic commitment and aggregate signature schemes $(\mathsf{COM}, \mathsf{SIG})$.

– $\pi \leftarrow \Pi.\mathsf{Prv}(\mathsf{crs}, u, w)$: the prover algorithm takes as input a CRS crs and a pair (u, w) and outputs a proof π;
– *bool* $\leftarrow \Pi.\mathsf{Ver}(\mathsf{crs}, u, \pi)$: the verifier algorithm takes as input a CRS crs, a statement u, and a proof π and outputs **true** or **false**, indicating acceptance of the proof.

A proof system Π is *complete* if for every λ and every (even unbounded) adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{l}\mathsf{mp} \leftarrow \mathsf{MainSetup}(1^\lambda)\\ \mathsf{crs} \leftarrow \Pi.\mathsf{Setup}(\mathsf{mp})\\ (u, w) \leftarrow \mathcal{A}(\mathsf{crs})\\ \pi \leftarrow \Pi.\mathsf{Prv}(\mathsf{crs}, u, w)\end{array} : (\mathsf{mp}, u, w) \in \mathsf{R} \Rightarrow \Pi.\mathsf{Ver}(\mathsf{crs}, u, \pi) = \mathbf{true}\right] = 1\ .$$

A proof system Π is *zero-knowledge* if no information about the witness is leaked by the proof. We define a *simulator* $\Pi.\mathsf{Sim}$ for a proof system Π as a pair of algorithms:

– $(\mathsf{crs}, \tau) \leftarrow \Pi.\mathsf{SimSetup}(\mathsf{mp})$: the simulated setup algorithm takes main parameters mp and outputs a CRS together with a trapdoor τ;
– $\pi^* \leftarrow \Pi.\mathsf{SimPrv}(\mathsf{crs}, \tau, u)$: the simulated prover algorithm takes as input a CRS, a trapdoor τ, and a statement u and outputs a simulated proof π^*.

Definition 7 (Zero-knowledge). *Let game* ZK *be as defined in Figure 5. A proof system* Π *for relation* R *is* zero-knowledge *if there exists a simulator* $\Pi.\mathsf{Sim}$ *such that for any p.p.t. adversary* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathrm{zk}}_{\Pi,\mathsf{R},\mathcal{A}}(\lambda) := 2 \cdot \left|\Pr\left[\mathrm{ZK}_{\Pi,\mathsf{R},\mathcal{A}}(\lambda) = \mathbf{true}\right] - \tfrac{1}{2}\right| = \mathsf{negl}(\lambda)\ .$$

Note that the zero-knowledge advantage can equivalently be defined as

$$\mathsf{Adv}^{\mathrm{zk}}_{\Pi,\mathsf{R},\mathcal{A}}(\lambda) = \left|\Pr\left[\mathrm{ZK}^0_{\Pi,\mathsf{R},\mathcal{A}}(\lambda) = \mathbf{true}\right] - \Pr\left[\mathrm{ZK}^1_{\Pi,\mathsf{R},\mathcal{A}}(\lambda) = \mathbf{true}\right]\right|\ ,$$

```
Game ZK$_{\Pi,\mathsf{R},\mathcal{A}}(\lambda)$
$b \leftarrow_{\$} \{0,1\}\,;\ \mathsf{mp} \leftarrow \mathsf{MainSetup}(1^\lambda)$
$\mathsf{crs}_0 \leftarrow \Pi.\mathsf{Setup}(\mathsf{mp})$
$(\mathsf{crs}_1, \tau) \leftarrow \Pi.\mathsf{SimSetup}(\mathsf{mp})$
$b' \leftarrow \mathcal{A}^{\textsc{SimProve}}(\mathsf{crs}_b)$
return $b = b'$

Oracle $\textsc{SimProve}(u, w)$
if $\neg\,\mathsf{R}(\mathsf{mp}, u, w)$ then return $\bot$
if $b = 0$ then $\pi \leftarrow \Pi.\mathsf{Prv}(\mathsf{crs}_0, u, w)$
else $\pi \leftarrow \Pi.\mathsf{SimPrv}(\mathsf{crs}_1, \tau, u)$
return $\pi$
```

Fig. 5. The non-interactive zero-knowledge game for a proof system Π.

```
Game S-EXT$_{\Pi,\mathsf{R},\mathcal{A}}(\lambda)$
Q := ( ) ; mp ← MainSetup($1^\lambda$)
$(\mathsf{crs}, \tau) \leftarrow \Pi.\mathsf{SimSetup}(\mathsf{mp})$
$(\mathbf{u}, \boldsymbol{\pi}) \leftarrow \mathcal{A}^{\textsc{SimProve}}(\mathsf{crs})$
for $i = 1 \ldots |\mathbf{u}|$ do
  $w_i := \Pi.\mathsf{Ext}(\mathsf{crs}, \tau, u_i, \pi_i)$
return $\bigvee_{i=1}^{|\mathbf{u}|} \big(\Pi.\mathsf{Ver}(\mathsf{crs}, u_i, \pi_i) \wedge (u_i, \pi_i) \notin \mathsf{Q} \wedge \neg\, \mathsf{R}(\mathsf{mp}, u_i, w_i)\big)$

Oracle SimProve($u$)
$\pi \leftarrow \Pi.\mathsf{SimPrv}(\mathsf{crs}, \tau, u)$
$\mathsf{Q} := \mathsf{Q} \,\|\, ((u, \pi))$
return $\pi$
```

Fig. 6. The (multi-statement) simulation-extractability game for a proof system Π.

where the game $\mathrm{ZK}^i_{\Pi,\mathsf{R},\mathcal{A}}(\lambda)$ is defined as $\mathrm{ZK}_{\Pi,\mathsf{R},\mathcal{A}}(\lambda)$ except $b \leftarrow_\$ \{0,1\}$ is replaced by $b := i$ and the game returns b'.

The central security property of a proof system is *soundness*, meaning no adversary can produce a proof for a false statement. A stronger notion is *knowledge-soundness*, meaning that an adversary must know a witness in order to make a proof. This is formalized via an extraction algorithm defined as follows:

– $w := \Pi.\mathsf{Ext}(\mathsf{crs}, \tau, u, \pi)$: the (deterministic) extraction algorithm takes a CRS, a trapdoor τ, a statement u, and a proof π and returns a witness w.

Knowledge-soundness states that from a valid proof for a statement u output by an adversary, $\Pi.\mathsf{Ext}$ can extract a witness for u. In security proofs where the reduction simulates certain proofs knowledge-soundness is not sufficient. The stronger notion *simulation-extractability* guarantees that even then, from every proof output by the adversary, $\Pi.\mathsf{Ext}$ can extract a witness. Note that we define a multi-statement variant of simulation extractability: the adversary returns a list of statements and proofs and wins if there is a least one statement such that the corresponding proof is valid and the extractor fails to extract a witness.

Definition 8 (Simulation-Extractability). *Let game* S-EXT *be as defined in Figure 6. A non-interactive proof system* Π *for* R *with simulator* $\Pi.\mathsf{Sim}$ *is* (multi-statement) simulation-extractable *if there exists an extractor* $\Pi.\mathsf{Ext}$ *such that for any p.p.t. adversary* $\mathcal{A}$,

$$\mathsf{Adv}^{\text{s-ext}}_{\Pi,\mathsf{R},\mathcal{A}}(\lambda) := \Pr\left[\text{S-EXT}_{\Pi,\mathsf{R},\mathcal{A}}(\lambda) = \mathbf{true}\right] = \mathsf{negl}(\lambda) \ .$$

In the instantiation of our cash system, we will deal with *families* of relations, i.e. relations R_δ parametrized by some $\delta \in \mathbb{N}$. For those, we assume that the proof system Π is defined over the family of relations $\mathsf{R} = \{\mathsf{R}_\delta\}_\delta$ and that the setup algorithm $\Pi.\mathsf{Setup}$ takes an additional parameter δ which specifies the particular relation used during the protocol (and which is included in the returned CRS). For instance, in the case of proofs in a certain range $[0, v_{\max}]$, the proof system will be defined over a relation $\mathsf{R}_{v_{\max}}$, where $v_{\max}$ is the maximum integer allowed.

3 Aggregate Cash System

3.1 Syntax

Coins. The public parameters pp set up by the cash system specify a coin space $\mathcal{C}_{\mathsf{pp}}$ and a key space $\mathcal{K}_{\mathsf{pp}}$. A *coin* is an element $C \in \mathcal{C}_{\mathsf{pp}}$; to each coin is associated a *coin key* $k \in \mathcal{K}_{\mathsf{pp}}$, which allows spending the coin. The *value* v of a coin is an integer in $[0, v_{\max}]$, where $v_{\max}$ is a system parameter. We assume that there exists a function mapping pairs $(v, k) \in [0, v_{\max}] \times \mathcal{K}_{\mathsf{pp}}$ to coins in $\mathcal{C}_{\mathsf{pp}}$; however, we do not assume this mapping to be invertible or even injective.

Ledger. Similarly to any ledger-based currency such as Bitcoin, an aggregate cash system keeps track of available coins in the system through a ledger. We assume the ledger to be unique and available at any time to all users. How users are kept in consensus on the ledger is outside the scope of this paper. In our abstraction, a ledger Λ simply provides two attributes: a list of all coins available in the system $\Lambda.\mathsf{out}$, and the total value $\Lambda.\mathsf{sply}$ those coins add up to. We say that a coin C *exists in the ledger* Λ if $C \in \Lambda.\mathsf{out}$.

Transactions. Transactions allow to modify the state of the ledger. Formally, a *transaction* tx provides three attributes: a *coin input list* $\mathsf{tx.in}$, a *coin output list* $\mathsf{tx.out}$, and a *supply* $\mathsf{tx.sply} \in \mathbb{N}$ specifying the amount of money created by tx. We classify transactions into three types. A transaction tx is said to be:

– a *minting transaction* if $\mathsf{tx.sply} > 0$ and $\mathsf{tx.in} = (\,)$; such a transaction creates new coins of total value $\mathsf{tx.sply}$ in the ledger;
– a *transfer transaction* if $\mathsf{tx.sply} = 0$ and $\mathsf{tx.in} \neq (\,)$; such a transaction transfers coins (by spending previous transaction outputs and creating new ones) but does not increase the overall value of coins in the ledger;
– a *mixed transaction* if $\mathsf{tx.sply} > 0$ and $\mathsf{tx.in} \neq (\,)$.

Pre-transactions. Pre-transactions allow users to transfer money to each other. Formally, a *pre-transaction* provides three attributes: a *coin input list* $\mathsf{ptx.in}$, a *list of change coins* $\mathsf{ptx.chg}$, and a *remainder* $\mathsf{ptx.rmdr}$. When Alice wants to send money worth value ρ to Bob, she selects coins of hers of total value $v \geq \rho$ and specifies the desired values for her change coins when $v > \rho$. The resulting pre-transaction ptx has therefore some input coin list $\mathsf{ptx.in}$ with total amount v, a change coin list $\mathsf{ptx.chg}$, and some remainder $\rho = \mathsf{ptx.rmdr}$. Alice sends this pre-transaction (via a secure channel) to Bob, who, in turn, finalizes it into a valid transaction and adds it to the ledger.

Aggregate cash system. An aggregate cash system CASH consists of the following algorithms:

– $(\mathsf{pp}, \Lambda) \leftarrow \mathsf{Setup}(1^\lambda, v_{\max})$: the setup algorithm takes as input the security parameter λ in unary and a maximal coin value $v_{\max}$ and returns public parameters pp and an initial (empty) ledger Λ.
– $(\mathsf{tx}, \mathbf{k}) \leftarrow \mathsf{Mint}(\mathsf{pp}, \mathbf{v})$: the mint algorithm takes as input a list of values $\mathbf{v}$ and returns a minting transaction tx and a list of coin keys $\mathbf{k}$ for the coins in $\mathsf{tx.out}$, such that the supply of tx is the sum of the values $\mathbf{v}$.
– $(\mathsf{ptx}, \mathbf{k}') \leftarrow \mathsf{Send}(\mathsf{pp}, (\mathbf{C}, \mathbf{v}, \mathbf{k}), \mathbf{v}')$: the sending algorithm takes as input a list of coins $\mathbf{C}$ together with the associated lists of values $\mathbf{v}$ and secret keys $\mathbf{k}$ and a list of *change values* $\mathbf{v}'$ whose sum is at most the sum of the input values $\mathbf{v}$; it returns a pre-transaction ptx and a list of keys $\mathbf{k}'$ for the change coins of ptx, such that the remainder of ptx is the sum of the values $\mathbf{v}$ minus the sum of the values $\mathbf{v}'$.
– $(\mathsf{tx}, \mathbf{k}'') \leftarrow \mathsf{Rcv}(\mathsf{pp}, \mathsf{ptx}, \mathbf{v}'')$: the receiving algorithm takes as input a pre-transaction ptx and a list of values $\mathbf{v}''$ whose sum equals the remainder of ptx; it returns a transfer transaction tx and a list of secret keys $\mathbf{k}''$ for the fresh coins in the output of tx, one for each value in $\mathbf{v}''$.
– $\Lambda' \leftarrow \mathsf{Ldgr}(\mathsf{pp}, \Lambda, \mathsf{tx})$: the ledger algorithm takes as input the ledger Λ and a transaction tx to be included in Λ; it returns an updated ledger Λ' or $\bot$.
– $\mathsf{tx} \leftarrow \mathsf{Agg}(\mathsf{pp}, \mathsf{tx}_0, \mathsf{tx}_1)$: the transaction aggregation algorithm takes as input two transactions tx_0 and tx_1 whose input coin lists are disjoint and whose output coin lists are disjoint; it returns a transaction tx whose supply is the sum of the supplies of tx_0 and tx_1 and whose input and output coin list is the cut-through of $\mathsf{tx}_0.\mathsf{in} \parallel \mathsf{tx}_1.\mathsf{in}$ and $\mathsf{tx}_0.\mathsf{out} \parallel \mathsf{tx}_1.\mathsf{out}$.

We say that an aggregate cash system CASH is correct if its procedures Setup, Mint, Send, Rcv, Ldgr, and Agg behave as expected with overwhelming probability (that is, we allow that with negligible probability things can go wrong, typically, because an algorithm could generate the same coin twice). We give a formal definition that uses procedures Ver, which defines validity of a ledger or a (pre-)transaction, and Cons, which checks consistency of a triple (coin, value, key).

Definition 9 (Correctness). *An aggregate cash system* CASH *is* correct *if there exist procedures* $\mathsf{Ver}(\cdot,\cdot)$ *and* $\mathsf{Cons}(\cdot,\cdot,\cdot,\cdot)$ *such that for any* $v_{\max} \in \mathbb{N}$ *and (not necessarily p.p.t.)* $\mathcal{A}_{\mathsf{Mint}}, \mathcal{A}_{\mathsf{Send}}, \mathcal{A}_{\mathsf{Rcv}}, \mathcal{A}_{\mathsf{Agg}}$ *and* $\mathcal{A}_{\mathsf{Ldgr}}$ *the following functions are overwhelming in* λ*:*

$$\Pr\Big[(\mathsf{pp}, \Lambda) \leftarrow \mathsf{Setup}(1^\lambda, v_{\max}) \;:\; \mathsf{Ver}(\mathsf{pp}, \Lambda)\Big]$$

$$\Pr\left[\begin{array}{l}(\mathsf{pp}, \Lambda) \leftarrow \mathsf{Setup}(1^\lambda, v_{\max})\\ \mathbf{v} \leftarrow \mathcal{A}_{\mathsf{Mint}}(\mathsf{pp}, \Lambda)\\ (\mathsf{tx}, \mathbf{k}) \leftarrow \mathsf{Mint}(\mathsf{pp}, \mathbf{v})\end{array} \;:\; \mathbf{v} \in [0, v_{\max}]^* \Rightarrow \begin{pmatrix}\mathsf{Ver}(\mathsf{pp}, \mathsf{tx}) \wedge \mathsf{tx.in} = (\,) \wedge\\ \mathsf{tx.sply} = \sum \mathbf{v} \wedge\\ \mathsf{Cons}(\mathsf{pp}, \mathsf{tx.out}, \mathbf{v}, \mathbf{k})\end{pmatrix}\right]$$

$$\Pr\left[\begin{array}{l}(\mathsf{pp}, \Lambda) \leftarrow \mathsf{Setup}(1^\lambda, v_{\max})\\ (\mathbf{C}, \mathbf{v}, \mathbf{k}, \mathbf{v}') \leftarrow \mathcal{A}_{\mathsf{Send}}(\mathsf{pp}, \Lambda)\\ (\mathsf{ptx}, \mathbf{k}') \leftarrow \mathsf{Send}(\mathsf{pp}, (\mathbf{C}, \mathbf{v}, \mathbf{k}), \mathbf{v}')\\ \\ \\ \end{array} \;:\; \begin{array}{r}\begin{pmatrix}\mathsf{Cons}(\mathsf{pp}, \mathbf{C}, \mathbf{v}, \mathbf{k}) \wedge \mathbf{v} \,\|\, \mathbf{v}' \in [0, v_{\max}]^* \wedge\\ \sum \mathbf{v} - \sum \mathbf{v}' \in [0, v_{\max}]\end{pmatrix}\\ \Rightarrow \begin{pmatrix}\mathsf{Ver}(\mathsf{pp}, \mathsf{ptx}) \wedge \mathsf{ptx.in} = \mathbf{C} \wedge\\ \mathsf{ptx.rmdr} = \sum \mathbf{v} - \sum \mathbf{v}' \wedge\\ \mathsf{Cons}(\mathsf{pp}, \mathsf{ptx.chg}, \mathbf{v}', \mathbf{k}')\end{pmatrix}\end{array}\right]$$

$$\Pr\left[\begin{array}{l}(\mathsf{pp}, \Lambda) \leftarrow \mathsf{Setup}(1^\lambda, v_{\max})\\ (\mathsf{ptx}, \mathbf{v}'') \leftarrow \mathcal{A}_{\mathsf{Rcv}}(\mathsf{pp}, \Lambda)\\ (\mathsf{tx}, \mathbf{k}'') \leftarrow \mathsf{Rcv}(\mathsf{pp}, \mathsf{ptx}, \mathbf{v}'')\\ \\ \end{array} \;:\; \begin{array}{r}\Big(\mathsf{Ver}(\mathsf{pp}, \mathsf{ptx}) \wedge \mathbf{v}'' \in [0, v_{\max}]^* \wedge \mathsf{ptx.rmdr} = \sum \mathbf{v}''\Big)\\ \Rightarrow \begin{pmatrix}\mathsf{Ver}(\mathsf{pp}, \mathsf{tx}) \wedge \mathsf{tx.sply} = 0 \wedge\\ \mathsf{tx.in} = \mathsf{ptx.in} \wedge \mathsf{ptx.chg} \subseteq \mathsf{tx.out} \wedge\\ \mathsf{Cons}(\mathsf{pp}, \mathsf{tx.out} - \mathsf{ptx.chg}, \mathbf{v}'', \mathbf{k}'')\end{pmatrix}\end{array}\right]$$

$$\Pr\left[\begin{array}{l}(\mathsf{pp}, \Lambda) \leftarrow \mathsf{Setup}(1^\lambda, v_{\max})\\ (\mathsf{tx}_0, \mathsf{tx}_1) \leftarrow \mathcal{A}_{\mathsf{Agg}}(\mathsf{pp}, \Lambda)\\ \mathsf{tx} \leftarrow \mathsf{Agg}(\mathsf{pp}, \mathsf{tx}_0, \mathsf{tx}_1)\\ \\ \\ \end{array} \;:\; \begin{array}{r}\begin{pmatrix}\mathsf{Ver}(\mathsf{pp}, \mathsf{tx}_0) \wedge \mathsf{tx}_0.\mathsf{in} \cap \mathsf{tx}_1.\mathsf{in} = (\,) \wedge\\ \mathsf{Ver}(\mathsf{pp}, \mathsf{tx}_1) \wedge \mathsf{tx}_0.\mathsf{out} \cap \mathsf{tx}_1.\mathsf{out} = (\,)\end{pmatrix}\\ \Rightarrow \begin{pmatrix}\mathsf{Ver}(\mathsf{pp}, \mathsf{tx}) \wedge \mathsf{tx.sply} = \mathsf{tx}_0.\mathsf{sply} + \mathsf{tx}_1.\mathsf{sply} \wedge\\ \mathsf{tx.in} = (\mathsf{tx}_0.\mathsf{in} \,\|\, \mathsf{tx}_1.\mathsf{in}) - (\mathsf{tx}_0.\mathsf{out} \,\|\, \mathsf{tx}_1.\mathsf{out}) \wedge\\ \mathsf{tx.out} = (\mathsf{tx}_0.\mathsf{out} \,\|\, \mathsf{tx}_1.\mathsf{out}) - (\mathsf{tx}_0.\mathsf{in} \,\|\, \mathsf{tx}_1.\mathsf{in})\end{pmatrix}\end{array}\right]$$

$$\Pr\left[\begin{array}{l}(\mathsf{pp}, \Lambda) \leftarrow \mathsf{Setup}(1^\lambda, v_{\max})\\ (\Lambda, \mathsf{tx}) \leftarrow \mathcal{A}_{\mathsf{Ldgr}}(\mathsf{pp}, \Lambda)\\ \Lambda' \leftarrow \mathsf{Ldgr}(\mathsf{pp}, \Lambda, \mathsf{tx})\\ \\ \\ \end{array} \;:\; \begin{array}{r}\begin{pmatrix}\mathsf{Ver}(\mathsf{pp}, \Lambda) \wedge \mathsf{Ver}(\mathsf{pp}, \mathsf{tx}) \wedge\\ \mathsf{tx.in} \subseteq \Lambda.\mathsf{out} \wedge \mathsf{tx.out} \cap \Lambda.\mathsf{out} = (\,)\end{pmatrix}\\ \Rightarrow \begin{pmatrix}\Lambda' \neq \bot \wedge \mathsf{Ver}(\mathsf{pp}, \Lambda') \wedge\\ \Lambda'.\mathsf{out} = (\Lambda.\mathsf{out} - \mathsf{tx.in}) \,\|\, \mathsf{tx.out} \wedge\\ \Lambda'.\mathsf{sply} = \Lambda.\mathsf{sply} + \mathsf{tx.sply}\end{pmatrix}\end{array}\right]$$

3.2 Security Definitions

Security against inflation. A sound payment system must ensure that the only way money can be created is via the supply of transactions; typically minting transactions. This means that for any tx, the total value of the output coins should be equal to the sum of the total value of the input coins plus the supply tx.sply of the transaction. Since coin values are not deducible from a transaction (this is one of the privacy features of such a system), we define the property at the level of the ledger Λ.

We say that a cash system is resistant to inflation if no adversary can spend coins from Λ.out worth more than Λ.sply. The adversary's task is thus to create a pre-transaction whose remainder is strictly greater than Λ.sply; validity of the pre-transaction is checked by completing it to a transaction via Rcv and adding it to the ledger via Ldgr. This is captured by the definition below.

```
Game $\mathrm{INFL}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max})$
$(\mathsf{pp}, \Lambda) \leftarrow \mathsf{CASH.Setup}(1^\lambda, v_{\max})$
$(\Lambda, \mathsf{ptx}, \mathbf{v}) \leftarrow \mathcal{A}(\mathsf{pp}, \Lambda)$
$(\mathsf{tx}, \mathbf{k}) \leftarrow \mathsf{CASH.Rcv}(\mathsf{pp}, \mathsf{ptx}, \mathbf{v})$
return $\perp \not\leftarrow \mathsf{CASH.Ldgr}(\mathsf{pp}, \Lambda, \mathsf{tx})$ and $\Lambda.\mathsf{sply} < \sum \mathbf{v}$
```

Fig. 7. Game formalizing resistance to inflation of a cash system CASH.

Definition 10 (Inflation-resistance). *We say that an aggregate cash system* CASH *is secure against inflation if for any $v_{\max}$ and any p.p.t. adversary $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathrm{infl}}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max}) := \Pr\left[\mathrm{INFL}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max}) = \mathbf{true}\right] = \mathsf{negl}(\lambda) \ ,$$

where $\mathrm{INFL}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max})$ is defined in Figure 7.

Security against coin theft. Besides inflation, which protects the soundness of the system as a whole, the second security notion protects individual users. It requires that only a user can spend coins belonging to him, where ownership of a coin amounts to knowledge of the coin secret key. This is formalized by the experiment in Figure 8, which proceeds as follows. The challenger sets up the system and maintains the ledger Λ throughout the game (we assume that the consensus protocol provides this). The adversary can add any valid transaction to the ledger through an oracle LEDGER.

The challenger also simulates an honest user and manages her coins; in particular, it maintains a list Hon, which represents the coins that the honest user expects to own in the ledger. The game also maintains two hash tables Val and Key mapping coins produced by the game to their values and keys. We write e.g. $\mathsf{Val}(C) := v$ to mean that the pair (C, v) is added to Val and let $\mathsf{Val}(C)$ denote the value v for which (C, v) is in Val. This naturally generalizes to lists letting $\mathsf{Val}(\mathbf{C})$ be the list $\mathbf{v}$ such that (C_i, v_i) is in Val for all i.

The adversary can interact with the honest user and the ledger using the following oracles:

- MINT is an oracle that mints coins for the honest user. It takes as input a vector of values $\mathbf{v}$, creates a minting transaction tx together with the secret keys of the output coins, adds tx to the ledger and appends the newly created coins to Hon.
- RECEIVE lets the adversary send coins to the honest user. The oracle takes as input a pre-transaction ptx and output values $\mathbf{v}$; it completes ptx to a transaction tx creating output coins with values $\mathbf{v}$, adds tx to the ledger, and appends the newly created coins to Hon.
- SEND lets the adversary make an honest user send coins to it. It takes as input a list $\mathbf{C}$ of coins contained in Hon and a list of change values $\mathbf{v}'$; it also checks that none of the coins in $\mathbf{C}$ has been queried to SEND before. It returns a pre-transaction ptx spending the coins from $\mathbf{C}$ and creating change output coins with values $\mathbf{v}'$. The oracle only produces a pre-transaction and returns it to the adversary, but it does not alter the ledger. This is why the list Hon of honest coins is not altered either; in particular, the sent coins $\mathbf{C}$ still remain in Hon.
- LEDGER lets the adversary commit a transaction tx to the ledger. If the transaction output contains the (complete) set of change coins of a pre-transaction ptx previously sent to the adversary, then these change coins are added to Hon, while the input coins of ptx are removed from Hon.

Note that the list Hon represents the coins that the honest user should consider hers, given the system changes induced by the oracle calls: coins received directly from the adversary via

```
Game $\mathrm{STEAL}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max})$
$(\mathsf{pp}, \Lambda) \leftarrow \mathsf{CASH.Setup}(1^\lambda, v_{\max})$
$\mathsf{Hon}, \mathsf{Val}, \mathsf{Key}, \mathsf{Ptx} := ()$
$\mathcal{A}^{\textsc{Mint},\textsc{Send},\textsc{Receive},\textsc{Ledger}}(\mathsf{pp}, \Lambda)$
return $(\mathsf{Hon} \not\subseteq \Lambda.\mathsf{out})$

Aux function $\mathsf{Store}(\mathbf{C}, \mathbf{v}, \mathbf{k})$
$\mathsf{Val}(\mathbf{C}) := \mathbf{v}\,;\ \mathsf{Key}(\mathbf{C}) := \mathbf{k}$

Oracle $\textsc{Mint}(\mathbf{v})$
$(\mathsf{tx}, \mathbf{k}) \leftarrow \mathsf{CASH.Mint}(\mathsf{pp}, \mathbf{v})$
$\Lambda \leftarrow \mathsf{CASH.Ldgr}(\mathsf{pp}, \Lambda, \mathsf{tx})$
$\mathsf{Hon} := \mathsf{Hon} \,\|\, \mathsf{tx.out}$
$\mathsf{Store}(\mathsf{tx.out}, \mathbf{v}, \mathbf{k})$
return $\mathsf{tx}$

Oracle $\textsc{Send}(\mathbf{C}, \mathbf{v}')$
if $\mathbf{C} \not\subseteq \mathsf{Hon}$ or $\bigcup_{\mathsf{ptx}\in\mathsf{Ptx}} \mathsf{ptx.in} \cap \mathbf{C} \neq ()$
    return $\bot$ // only honest coins never sent can be queried
$(\mathsf{ptx}, \mathbf{k}') \leftarrow \mathsf{CASH.Send}\big(\mathsf{pp}, \mathbf{C}, \mathsf{Val}(\mathbf{C}), \mathsf{Key}(\mathbf{C}), \mathbf{v}'\big)$
$\mathsf{Store}(\mathsf{ptx.chg}, \mathbf{v}', \mathbf{k}')\,;\ \mathsf{Ptx} := \mathsf{Ptx} \,\|\, (\mathsf{ptx})$
return $\mathsf{ptx}$
```

```
Oracle $\textsc{Receive}(\mathsf{ptx}, \mathbf{v})$
$(\mathsf{tx}, \mathbf{k}) \leftarrow \mathsf{CASH.Rcv}(\mathsf{pp}, \mathsf{ptx}, \mathbf{v})$
$\Lambda' \leftarrow \textsc{Ledger}(\mathsf{tx})$ // updates $\mathsf{Hon}$
if $\Lambda' = \bot$ then return $\bot$
$\mathsf{Hon} := \mathsf{Hon} \,\|\, (\mathsf{tx.out} - \mathsf{ptx.chg})$
$\mathsf{Store}(\mathsf{tx.out} - \mathsf{ptx.chg}, \mathbf{v}, \mathbf{k})$
return $\mathsf{tx}$

Oracle $\textsc{Ledger}(\mathsf{tx})$
$\Lambda' \leftarrow \mathsf{CASH.Ldgr}(\mathsf{pp}, \Lambda, \mathsf{tx})$
if $\Lambda' = \bot$ then return $\bot$ else $\Lambda := \Lambda'$
for all $\mathsf{ptx} \in \mathsf{Ptx}$ do
    if $\mathsf{ptx.chg} \subseteq \mathsf{tx.out}$
    // if all change of $\mathsf{ptx}$ now in ledger
        $\mathsf{Ptx} := \mathsf{Ptx} - (\mathsf{ptx})$
        $\mathsf{Hon} := (\mathsf{Hon} - \mathsf{ptx.in}) \,\|\, \mathsf{ptx.chg}$
        // consider input of $\mathsf{ptx}$ consumed
return $\Lambda$
```

Fig. 8. Game formalizing resistance to coin theft of a cash system CASH.

Receive or as fresh coins via Mint are added to Hon. Coins sent to the adversary in a pre-transaction ptx via Send are only removed *once all change coins of* ptx *have been added* to the ledger via Ledger. Note also that, given these oracles, the adversary can also simulate transfers between honest users. It can simply call Send to receive an honest pre-transaction ptx and then call Receive to have the honest user receive ptx.

The winning condition of the game is now simply that Hon does not reflect what the honest user would expect, namely Hon is not fully contained in the ledger (because the adversary managed to spend a coin that is still in Hon, which amounts to stealing a coin from the honest user).

Definition 11 (Theft-resistance). *We say that an aggregate cash system* CASH *is* secure against coin theft *if for any $v_{\max}$ and any p.p.t. adversary $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathrm{steal}}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max}) := \Pr\left[\mathrm{STEAL}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max}) = \mathbf{true}\right] = \mathsf{negl}(\lambda) ,$$

where $\mathrm{STEAL}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max})$ is defined in Figure 8.

Transaction indistinguishability. An important security feature that Mimblewimble inherits from Confidential Transactions [Max15] is that the amounts involved in a transaction are hidden so that only the sender and the receiver know how much money is involved. In addition, a transaction completely hides which inputs paid which outputs and which coins were change and which were added by the receiver.

```
Game IND-TX$_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max})$
$b \leftarrow_{\$} \{0,1\}$
$(\mathsf{pp}, \Lambda) \leftarrow \mathsf{Setup}(1^\lambda, v_{\max})$
$b' \leftarrow \mathcal{A}^{\mathrm{TX}}(\mathsf{pp})$
return $b = b'$
```

```
Oracle TX$((\mathbf{v}_0, \mathbf{v}'_0, \mathbf{v}''_0), (\mathbf{v}_1, \mathbf{v}'_1, \mathbf{v}''_1))$
if not $(\mathbf{v}_0, \mathbf{v}'_0, \mathbf{v}''_0, \mathbf{v}_1, \mathbf{v}'_1, \mathbf{v}''_1 \in [0, v_{\max}]^*)$
    return $\perp$
if $|\mathbf{v}_0| \neq |\mathbf{v}_1|$ or $|\mathbf{v}'_0| + |\mathbf{v}''_0| \neq |\mathbf{v}'_1| + |\mathbf{v}''_1|$
    return $\perp$ // as number of coins is not hidden
if $\sum \mathbf{v}_0 \neq \sum(\mathbf{v}'_0 \parallel \mathbf{v}''_0)$ or $\sum \mathbf{v}_1 \neq \sum(\mathbf{v}'_1 \parallel \mathbf{v}''_1)$
    return $\perp$ // as transactions must be balanced
$(\mathsf{tx}, \mathbf{k}) \leftarrow \mathsf{Mint}(\mathsf{pp}, \mathbf{v}_b)$
$(\mathsf{ptx}, \mathbf{k}') \leftarrow \mathsf{Send}(\mathsf{pp}, (\mathsf{tx.out}, \mathbf{v}_b, \mathbf{k}), \mathbf{v}'_b)$
$(\mathsf{tx}^*, \mathbf{k}'') \leftarrow \mathsf{Rcv}(\mathsf{pp}, \mathsf{ptx}, \mathbf{v}''_b)$
return $\mathsf{tx}^*$
```

Fig. 9. Game formalizing transaction indistinguishability of a cash system CASH.

We formalize this via the following game, specified in Figure 9. The adversary submits two sets of values $(\mathbf{v}_0, \mathbf{v}'_0, \mathbf{v}''_0)$ and $(\mathbf{v}_1, \mathbf{v}'_1, \mathbf{v}''_1)$ representing possibles values for input coins, change coins and receiver's coins of a transaction. The game creates a transaction with values either from the first or the second set and the adversary must guess which. For the transaction to be valid, we must have $\sum \mathbf{v}_b = \sum \mathbf{v}'_b + \sum \mathbf{v}''_b$ for both $b = 0, 1$. Moreover, transactions do not hide the *number* of input and output coins. We therefore also require that $|\mathbf{v}_0| = |\mathbf{v}_1|$ and $|\mathbf{v}'_0| + |\mathbf{v}''_0| = |\mathbf{v}'_1| + |\mathbf{v}''_1|$.

Definition 12 (Transaction indistinguishability). *We say that an aggregate cash system* CASH *is* transaction-indistinguishable *if for any* $v_{\max}$ *and any p.p.t. adversary* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\text{tx-ind}}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max}) := 2 \cdot \left| \Pr\left[\text{TX-IND}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max}) = \mathbf{true}\right] - \tfrac{1}{2} \right| = \mathsf{negl}(\lambda) \ ,$$

where $\text{TX-IND}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max})$ *is defined in Figure 9.*

4 Construction of an Aggregate Cash System

4.1 Description

Let COM be a homomorphic commitment scheme such that for $\mathsf{cp} \leftarrow \mathsf{COM.Setup}(\mathsf{MainSetup}(1^\lambda))$ we have value space $\mathcal{V}_{\mathsf{cp}} = \mathbb{Z}_p$ with p of length λ (such as the Pedersen scheme). Let SIG be an aggregate signature scheme that is compatible with COM. For $v_{\max} \in \mathbb{N}$, let $\mathsf{R}_{v_{\max}}$ be the (efficiently computable) relation on commitments with values at most $v_{\max}$, i.e.,

$$\mathsf{R}_{v_{\max}} := \left\{ (\mathsf{mp}, (\mathsf{cp}, C), (v, r)) \;\middle|\; \mathsf{mp} = \mathsf{mp}_{\mathsf{cp}} \wedge C = \mathsf{COM.Cmt}(\mathsf{cp}, v, r) \wedge v \in [0, v_{\max}] \right\} ,$$

where $\mathsf{mp}_{\mathsf{cp}}$ are the main parameters contained in cp (recall that we assume that for $\mathsf{cp} \in [\mathsf{COM.Setup}(\mathsf{mp})]$, mp is contained in cp). Let Π be a simulation-extractable NIZK proof system for the family of relations $\mathsf{R} = \{\mathsf{R}_{v_{\max}}\}_{v_{\max}}$.

For notational simplicity, we will use the following vectorial notation for COM, R, and Π: given $\mathbf{C}$, $\mathbf{v}$, and $\mathbf{r}$ with $|\mathbf{C}| = |\mathbf{v}| = |\mathbf{r}|$, we let

$$\mathsf{COM.Cmt}(\mathsf{cp}, \mathbf{v}, \mathbf{r}) := \left(\mathsf{COM.Cmt}(\mathsf{cp}, v_i, r_i)\right)_{i=1}^{|\mathbf{v}|} ,$$

$$\mathsf{R}_{v_{\max}}((\mathsf{cp}, \mathbf{C}), (\mathbf{v}, \mathbf{r})) := \textstyle\bigwedge_{i=1}^{|\mathbf{C}|} \mathsf{R}_{v_{\max}}(\mathsf{mp}_{\mathsf{cp}}, (\mathsf{cp}, C_i), (v_i, r_i)) ,$$

$$\Pi.\mathsf{Prv}(\mathsf{crs}, (\mathsf{cp}, \mathbf{C}), (\mathbf{v}, \mathbf{r})) := \left(\Pi.\mathsf{Prv}(\mathsf{crs}, (\mathsf{cp}, C_i), (v_i, r_i))\right)_{i=1}^{|\mathbf{C}|} ,$$

$$\mathsf{\Pi.Ver}(\mathsf{crs}, (\mathsf{cp}, \mathbf{C}), \boldsymbol{\pi}) := \bigwedge_{i=1}^{|\mathbf{C}|} \mathsf{\Pi.Ver}(\mathsf{crs}, (\mathsf{cp}, C_i), \pi_i) ,$$

and likewise for $\mathsf{\Pi.SimPrv}$. We also assume that messages are the empty string ε if they are omitted from $\mathsf{SIG.Ver}$ and $\mathsf{SIG.Agg}$; that is, we overload notation and let

$$\mathsf{SIG.Ver}(\mathsf{sp}, (X_i)_{i=1}^n, \sigma) := \mathsf{SIG.Ver}(\mathsf{sp}, ((X_i, \varepsilon))_{i=1}^n, \sigma)$$

and likewise for $\mathsf{SIG.Agg}(\mathsf{sp}, ((X_{0,i})_{i=1}^{n_0}, \sigma_0), ((X_{1,i})_{i=1}^{n_1}, \sigma_1))$.

From COM, SIG and Π, we construct an aggregate cash system $\mathsf{MW}[\mathsf{COM}, \mathsf{SIG}, \Pi]$ as follows. The public parameters pp consist of commitment and signature parameters cp, sp, and a CRS for Π. A *coin key* $k \in \mathcal{K}_{\mathsf{pp}}$ is an element of the randomness space $\mathcal{R}_{\mathsf{cp}}$ of the commitment scheme, i.e., $\mathcal{K}_{\mathsf{pp}} = \mathcal{R}_{\mathsf{cp}}$. A *coin* $C = \mathsf{COM.Cmt}(\mathsf{cp}, v, k)$ is a commitment to the value v of the coin using the coin key k as randomness. Hence, $\mathcal{C}_{\mathsf{pp}} = \mathcal{C}_{\mathsf{cp}}$.

A transaction $\mathsf{tx} = (s, \mathbf{C}, \hat{\mathbf{C}}, K)$ consists of a supply $\mathsf{tx.sply} = s$, an input coin list $\mathsf{tx.in} = \mathbf{C}$, an output coin list $\mathsf{tx.out} = \hat{\mathbf{C}}$, and a kernel K. The kernel K is a triple $(\boldsymbol{\pi}, \mathbf{E}, \sigma)$ where $\boldsymbol{\pi}$ is a list of range proofs for the output coins, $\mathbf{E}$ is a non-empty list of signature-verification keys (which are of the same form as commitments) called *kernel excesses*, and σ is an (aggregate) signature. We define the *excess of the transaction* tx, denoted $\mathsf{Exc}(\mathsf{tx})$, as the sum of outputs minus the sum of inputs, with the supply s converted to a coin with $k = 0$:

$$\mathsf{Exc}(\mathsf{tx}) := \sum \hat{\mathbf{C}} - \sum \mathbf{C} - \mathsf{COM.Cmt}(\mathsf{cp}, s, 0) . \tag{4}$$

Intuitively, $\mathsf{Exc}(\mathsf{tx})$ should be a commitment to 0, as the committed input and output values of the transaction should cancel out; this is evidenced by giving a signature under key $\mathsf{Exc}(\mathsf{tx})$ (which could be represented as the sum of elements (E_i) due to aggregation; see below).

A transaction $\mathsf{tx} = (s, \mathbf{C}, \hat{\mathbf{C}}, K)$ with $K = (\boldsymbol{\pi}, \mathbf{E}, \sigma)$ is said to be valid if all range proofs are valid, $\mathsf{Exc}(\mathsf{tx}) = \sum \mathbf{E}$, and σ is a valid signature for $\mathbf{E}$ (with all messages ε).[13]

When a user wants to make a payment of an amount ρ, she creates a transaction tx with input coins $\mathbf{C}$ of values $\mathbf{v}$ with $\sum \mathbf{v} \geq \rho$ and with output coins a list of fresh change coins of values $\mathbf{v}'$ so that $\sum \mathbf{v}' = \sum \mathbf{v} - \rho$. She also appends one more special coin of value ρ to the output. The pre-transaction ptx is then defined as this transaction tx, the remainder $\mathsf{ptx.rmdr} := \rho$ and the key for the special coin.

When receiving a pre-transaction $\mathsf{ptx} = (\mathsf{tx}, \rho, k)$, the receiver first checks that tx is valid and that k is a key for the special coin $C' := \mathsf{tx.out}[|\mathsf{tx.out}|]$ of value ρ. He then creates a transaction tx' that spends C' (using its key k) and creates coins of combined value ρ. Aggregating tx and tx' yields a transaction tx'' with $\mathsf{tx}''.\mathsf{sply} = 0$, $\mathsf{tx}''.\mathsf{in} = \mathsf{ptx.in}$ and $\mathsf{tx}''.\mathsf{out}$ containing $\mathsf{ptx.chg}$ and the freshly created coins. The receiver then submits tx'' to the ledger.

The ledger accepts a transaction if it is valid (as defined above) and if its input coins are contained in the output coin list of the ledger (which corresponds to the UTXO set in other systems). We do not consider any other conditions related to the consensus mechanism, such as fees being included in a transaction to incentivize its inclusion in the ledger or a proof-of-work being included in a minting transaction.

We first define some auxiliary algorithms in Figure 10, which create coins and transactions and verify their validity by instantiating the procedures Ver and Cons from Definition 9. Using these we then formally define $\mathsf{MW}[\mathsf{COM}, \mathsf{SIG}, \Pi]$ in Figure 11.

[13] If $\mathbf{E}$ in a transaction tx consists of a single element, it must be $E = \mathsf{Exc}(\mathsf{tx})$, so E could be omitted from the transaction; we keep it for consistency.

```
$\mathsf{MW.Coin}((\mathsf{cp}, \mathsf{sp}, \mathsf{crs}), \mathbf{v})$
$\mathbf{k} \leftarrow_{\$} \mathcal{R}_{\mathsf{cp}}^{|\mathbf{v}|}$
$\mathbf{C} := \mathsf{COM.Cmt}(\mathsf{cp}, \mathbf{v}, \mathbf{k})$
$\boldsymbol{\pi} \leftarrow \Pi.\mathsf{Prv}(\mathsf{crs}, (\mathsf{cp}, \mathbf{C}), (\mathbf{v}, \mathbf{k}))$
return $(\mathbf{C}, \mathbf{v}, \boldsymbol{\pi})$
```

```
$\mathsf{MW.MkTx}((\mathsf{cp}, \mathsf{sp}, \mathsf{crs}), (\mathbf{C}, \mathbf{v}, \mathbf{k}), \hat{\mathbf{v}})$
if $\neg\mathsf{Cons}(\mathsf{pp}, \mathbf{C}, \mathbf{v}, \mathbf{k})$ then return $\perp$
$s := \sum \hat{\mathbf{v}} - \sum \mathbf{v}$
if $\mathbf{v} \,\|\, \hat{\mathbf{v}} \not\subseteq [0, v_{\max}]^*$ or $s < 0$
    then return $\perp$
$(\hat{\mathbf{C}}, \hat{\mathbf{k}}, \hat{\boldsymbol{\pi}}) \leftarrow \mathsf{MW.Coin}(\mathsf{pp}, \hat{\mathbf{v}})$
$E := \sum \hat{\mathbf{C}} - \sum \mathbf{C} - \mathsf{COM.Cmt}(\mathsf{cp}, s, 0)$
$\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sp}, \sum \hat{\mathbf{k}} - \sum \mathbf{k}, \varepsilon)$
$K := (\hat{\boldsymbol{\pi}}, E, \sigma)$
$\mathsf{tx} := (s, \mathbf{C}, \hat{\mathbf{C}}, K)$
return $(\mathsf{tx}, \hat{\mathbf{k}})$
```

```
$\mathsf{MW.Cons}((\mathsf{cp}, \mathsf{sp}, \mathsf{crs}), \mathbf{C}, \mathbf{v}, \mathbf{k})$
return $|\mathbf{C}| = |\mathbf{v}| = |\mathbf{k}|$ and $\mathbf{v} \in [0, v_{\max}]^*$
        and $(\forall\, i \neq j : C_i \neq C_j)$
        and $\mathbf{C} = \mathsf{COM.Cmt}(\mathsf{cp}, \mathbf{v}, \mathbf{k})$
// $\mathsf{Cons}(\mathsf{pp}, (\,), (\,), (\,))$ returns true
```

```
$\mathsf{MW.Ver}((\mathsf{cp}, \mathsf{sp}, \mathsf{crs}), \mathsf{tx})$
if $\mathsf{tx} = (0, (\,), (\,), ((\,), (\,), \varepsilon))$ then return true
$(s, \mathbf{C}, \hat{\mathbf{C}}, K) := \mathsf{tx}$;  $(\boldsymbol{\pi}, \mathbf{E}, \sigma) := K$
$\mathsf{Exc} := \sum \hat{\mathbf{C}} - \sum \mathbf{C} - \mathsf{COM.Cmt}(\mathsf{cp}, s, 0)$
return $(\forall\, i \neq j : C_i \neq C_j \wedge \hat{C}_i \neq \hat{C}_j)$ and $\mathbf{C} \cap \hat{\mathbf{C}} = (\,)$
    and $s \geq 0$ and $\Pi.\mathsf{Ver}(\mathsf{crs}, \hat{\mathbf{C}}, \boldsymbol{\pi})$ and
    $\sum \mathbf{E} = \mathsf{Exc}$ and $\mathsf{SIG.Ver}(\mathsf{sp}, \mathbf{E}, \sigma)$
```

```
$\mathsf{MW.Ver}(\mathsf{pp}, \mathsf{ptx})$
$(\mathsf{tx}, \rho, k') := \mathsf{ptx}$
return $\mathsf{MW.Ver}(\mathsf{pp}, \mathsf{tx})$ and $\mathsf{tx.sply} = 0$ and
    $\mathsf{MW.Cons}(\mathsf{pp}, \mathsf{tx.out}[|\mathsf{tx.out}|], \rho, k')$
```

```
$\mathsf{MW.Ver}(\mathsf{pp}, \Lambda)$
$\mathsf{tx} := \Lambda$  // interpret $\Lambda$ as transaction
return $\mathsf{MW.Ver}(\mathsf{pp}, \mathsf{tx})$ and $\mathsf{tx.in} = (\,)$
```

Fig. 10. Auxiliary algorithms for the MW aggregate cash system.

Correctness. We start with showing some properties of the auxiliary algorithms in Figure 10. For any $\mathbf{v} \in [0, v_{\max}]^*$ and $(\mathbf{C}, \mathbf{k}, \boldsymbol{\pi}) \leftarrow \mathsf{Coin}(\mathsf{pp}, \mathbf{v})$, we have $\mathsf{Cons}(\mathsf{pp}, \mathbf{C}, \mathbf{v}, \mathbf{k})$ with overwhelming probability due to Lemma 3. Moreover, correctness of SIG and Π implies that MkTx run on consistent $(\mathbf{C}, \mathbf{v}, \mathbf{k})$ and values $\hat{\mathbf{v}} \in [0, v_{\max}]^*$ with $\sum \hat{\mathbf{v}} \geq \sum \mathbf{v}$ produces a tx which is accepted by Ver with overwhelming probability and whose supply is the difference $\sum \mathbf{v} - \sum \hat{\mathbf{v}}$.

We now show that the protocol MW[COM, SIG, Π] described in Figure 11 satisfies Definition 9. It is immediate that an empty ledger output by $\mathsf{Setup}(1^\lambda, v_{\max})$ verifies. As Mint invokes MkTx on empty inputs and output values $\mathbf{v}$, correctness of Mint follows from correctness of MkTx. Correctness of Send also follows from correctness of MkTx when the preconditions on the values, consistency of the coins and the supply, and $\sum \mathbf{v} - \sum \mathbf{v}' = \rho \in [0, v_{\max}]$ hold (note that $\mathsf{ptx.rmdr} = \rho$). Therefore, with overwhelming probability the pre-transaction is valid, and the change coins are consistent. Correctness of Agg is straightforward: it returns a transaction with the desired supply, input, and output coin list whose validity follows from correctness of SIG.Agg and Π.Ver for the coins resulting from cut-through. For any adversary $\mathcal{A}_{\mathsf{Ldgr}}$ returning (Λ, tx), if $\mathsf{Ver}(\mathsf{pp}, \Lambda) = \mathbf{true}$, then $\Lambda.\mathsf{in} = (\,)$ and Λ is valid when interpreted as a transaction. Since the input list of Λ is empty, $\mathsf{Ldgr}(\mathsf{pp}, \Lambda, \mathsf{tx}) = \mathsf{Agg}(\mathsf{pp}, \Lambda, \mathsf{tx})$ and so Ldgr is correct because Agg is.

Finally, we consider Rcv, which is slightly more involved. Consider an adversary $\mathcal{A}_{\mathsf{Rcv}}$ returning $(\mathsf{ptx}, \mathbf{v}'')$ with $\mathsf{ptx} = (\mathsf{tx}, \rho, k')$ and let $(\mathsf{tx}'', \mathbf{k}'') \leftarrow \mathsf{MW.Rcv}(\mathsf{pp}, \mathsf{ptx}, \mathbf{v}'')$. First, the preconditions trivially guarantee that the output is not $\perp$. Consider the call $(\mathsf{tx}', \mathbf{k}'') \leftarrow \mathsf{MW.MkTx}(\mathsf{pp}, (C', \rho, k'), \mathbf{v}'')$ inside MW.Rcv. We claim that with overwhelming probability, $(\mathsf{tx.in} \,\|\, \mathsf{tx}'.\mathsf{in}) \cap (\mathsf{tx.out} \,\|\, \mathsf{tx}'.\mathsf{out}) = (C')$. First, $\mathsf{tx.in} \cap \mathsf{tx.out} = (\,)$ as otherwise $\mathsf{Ver}(\mathsf{pp}, \mathsf{tx}) = \mathbf{false}$

```
$\mathsf{MW.Setup}(1^\lambda, v_{\max})$
  $\mathsf{mp} \leftarrow \mathsf{MainSetup}(1^\lambda)$
  $\mathsf{cp} \leftarrow \mathsf{COM.Setup}(\mathsf{mp})$
  $\mathsf{sp} \leftarrow \mathsf{SIG.Setup}(\mathsf{mp})$
  $\mathsf{crs} \leftarrow \Pi\mathsf{.Setup}(\mathsf{mp}, v_{\max})$
  $\Lambda := \big(0, (\,), (\,), ((\,), (\,), \varepsilon)\big)$
  return $\big(\mathsf{pp} := (\mathsf{cp}, \mathsf{sp}, \mathsf{crs}), \Lambda\big)$

$\mathsf{MW.Agg}(\mathsf{pp}, \mathsf{tx}_0, \mathsf{tx}_1)$
  if $\neg\,\mathsf{MW.Ver}(\mathsf{pp}, \mathsf{tx}_0)$ or
    $\neg\,\mathsf{MW.Ver}(\mathsf{pp}, \mathsf{tx}_1)$ or
    $\mathsf{tx}_0\mathsf{.in} \cap \mathsf{tx}_1\mathsf{.in} \neq (\,)$ or
    $\mathsf{tx}_0\mathsf{.out} \cap \mathsf{tx}_1\mathsf{.out} \neq (\,)$
      return $\perp$
  $\big(s_0, \mathbf{C}_0, \hat{\mathbf{C}}_0, (\boldsymbol{\pi}_0, \mathbf{E}_0, \sigma_0)\big) := \mathsf{tx}_0$
  $\big(s_1, \mathbf{C}_1, \hat{\mathbf{C}}_1, (\boldsymbol{\pi}_1, \mathbf{E}_1, \sigma_1)\big) := \mathsf{tx}_1$
  $\mathbf{C} := \mathbf{C}_0 \,\|\, \mathbf{C}_1 - \hat{\mathbf{C}}_0 \,\|\, \hat{\mathbf{C}}_1$
  $\hat{\mathbf{C}} := \hat{\mathbf{C}}_0 \,\|\, \hat{\mathbf{C}}_1 - \mathbf{C}_0 \,\|\, \mathbf{C}_1$
  $\boldsymbol{\pi} := (\pi_{0,i})_{i \in I_0} \,\|\, (\pi_{1,i})_{i \in I_1}$
      where $I_j := \{i : \hat{C}_{j,i} \notin \mathbf{C}_{1-j}\}$
  // $\boldsymbol{\pi}$ contains the proofs for coins in $\hat{\mathbf{C}}$
  $\sigma \leftarrow \mathsf{SIG.Agg}\big(\mathsf{sp}, (\mathbf{E}_0, \sigma_0), (\mathbf{E}_1, \sigma_1)\big)$
  $K := (\boldsymbol{\pi}, \mathbf{E}_0 \,\|\, \mathbf{E}_1, \sigma)$
  return $(s_0 + s_1, \mathbf{C}, \hat{\mathbf{C}}, K)$

$\mathsf{MW.Mint}(\mathsf{pp}, \hat{\mathbf{v}})$
  $(\mathsf{tx}, \hat{\mathbf{k}}) \leftarrow \mathsf{MW.MkTx}(\mathsf{pp}, ((\,), (\,), (\,)), \hat{\mathbf{v}})$
  return $(\mathsf{tx}, \hat{\mathbf{k}})$   // If $\perp \leftarrow \mathsf{MkTx}$, $\mathsf{Mint}$ returns $\perp$

$\mathsf{MW.Send}(\mathsf{pp}, (\mathbf{C}, \mathbf{v}, \mathbf{k}), \mathbf{v}')$
  $\rho := \sum \mathbf{v} - \sum \mathbf{v}'$
  $(\mathsf{tx}, \hat{\mathbf{k}}) \leftarrow \mathsf{MW.MkTx}(\mathsf{pp}, (\mathbf{C}, \mathbf{v}, \mathbf{k}), \mathbf{v}' \,\|\, \rho)$
  $\mathsf{ptx} := (\mathsf{tx}, \rho, \hat{k}_{|\mathbf{v}'|+1})$
  return $(\mathsf{ptx}, (\hat{k}_i)_{i=1}^{|\mathbf{v}'|})$

$\mathsf{MW.Rcv}(\mathsf{pp}, \mathsf{ptx}, \mathbf{v}'')$
  $(\mathsf{tx}, \rho, k') := \mathsf{ptx}$
  if $\neg\,\mathsf{MW.Ver}(\mathsf{pp}, \mathsf{ptx})$ or $\rho \neq \sum \mathbf{v}''$
    return $\perp$
  $C' := \mathsf{tx.out}[|\mathsf{tx.out}|]$
  $(\mathsf{tx}', \mathbf{k}'') \leftarrow \mathsf{MW.MkTx}(\mathsf{pp}, (C', \rho, k'), \mathbf{v}'')$
  $\mathsf{tx}'' \leftarrow \mathsf{MW.Agg}(\mathsf{pp}, \mathsf{tx}, \mathsf{tx}')$
  return $(\mathsf{tx}'', \mathbf{k}'')$

$\mathsf{MW.Ldgr}(\mathsf{pp}, \Lambda, \mathsf{tx})$
  if $\Lambda\mathsf{.in} \neq (\,)$ or $\mathsf{tx.in} \not\subseteq \Lambda\mathsf{.out}$
    return $\perp$
  return $\mathsf{MW.Agg}(\mathsf{pp}, \Lambda, \mathsf{tx})$   // returns $\perp$ if $\Lambda$ or $\mathsf{tx}$ invalid
```

Fig. 11. The MW aggregate cash system. (Recall that algorithms return $\perp$ when one of their subroutines returns $\perp$.)

and $\mathsf{Ver}(\mathsf{pp}, \mathsf{ptx}) = \textbf{false}$. By definition of MkTx, $\mathsf{tx}'.\mathsf{in} = (C')$ and by Lemma 3, $\mathsf{tx}'.\mathsf{out} \cap (\mathsf{tx.in} \,\|\, (C')) = (\,)$ with overwhelming probability. Hence,

$$(\mathsf{tx.in} \,\|\, \mathsf{tx}'.\mathsf{in}) \cap (\mathsf{tx.out} \,\|\, \mathsf{tx}'.\mathsf{out}) = (C') \cap \mathsf{tx.out} = (C')$$

and by correctness of Agg, C' is the only coin removed by cut-through during the call $\mathsf{tx}'' \leftarrow \mathsf{MW.Agg}(\mathsf{pp}, \mathsf{tx}, \mathsf{tx}')$. Thus, the input coin list of tx'' is the same as that of ptx and the change is contained in the output coin list of tx''. The pre-conditions $\mathsf{Ver}(\mathsf{pp}, \mathsf{ptx})$ and $\sum \mathbf{v}'' = \rho$ imply that $\mathsf{tx.sply} = 0$ and $\mathsf{tx}'.\mathsf{sply} = 0$, respectively. Hence, $\mathsf{tx}''.\mathsf{sply} = 0$ by correctness of Agg. Validity of tx'' and consistency of the new coins follow from correctness of Agg (and validity of the output of MkTx).

4.2 Security

We show that $\mathsf{MW}[\mathsf{COM}, \mathsf{SIG}, \Pi]$ is inflation-resistant, resistant to coin theft and that it satisfies transaction indistinguishability.

Theorem 13 (Inflation-resistance (Def. 10)). *Assume that the pair* $(\mathsf{COM}, \mathsf{SIG})$ *is EUF-NZO-secure and that* Π *is zero-knowledge and simulation-extractable. Then the aggregate cash* *system* $\mathsf{MW}[\mathsf{COM}, \mathsf{SIG}, \Pi]$ *is secure against inflation. More precisely, for any* $v_{\max}$ *and any p.p.t. adversary* $\mathcal{A}$*, there exists a negligible function* $\nu_{\mathcal{A}}$ *and p.p.t. adversaries* $\mathcal{B}$*,* $\mathcal{B}_{\mathrm{zk}}$ *and* $\mathcal{B}_{\mathrm{se}}$ *such that*

$$\mathsf{Adv}^{\mathrm{infl}}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \le \mathsf{Adv}^{\mathrm{euf\text{-}nzo}}_{\mathsf{COM},\mathsf{SIG},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathrm{zk}}_{\Pi,\mathsf{R}_{v_{\max}},\mathcal{B}_{\mathrm{zk}}}(\lambda) + \mathsf{Adv}^{\mathrm{s\text{-}ext}}_{\Pi,\mathsf{R}_{v_{\max}},\mathcal{B}_{\mathrm{se}}}(\lambda) + \nu_{\mathcal{A}}(\lambda) \ .$$

The proof can be found in Appendix A.2; it follows directly from EUF-NZO security and *extractability* of Π (we do not actually require *simulation*-extractability, but instead of formally defining extractability we simply relied on Definitions 7 and 8 implying it).

Consider an adversary $\mathcal{A}$ in game $\mathrm{INFL}_{\mathsf{MW}}$ in Figure 7. To win the game, $\mathcal{A}$ must return a valid ledger Λ, a valid ptx and $\mathbf{v}$ with

(i) $\mathsf{ptx.in} \subseteq \Lambda.\mathsf{out}$ and (ii) $\sum \mathbf{v} = \mathsf{ptx.rmdr}$

(otherwise Rcv and/or Ldgr return $\perp$). All coins in $\Lambda.\mathsf{out}$, $\mathsf{ptx.in}$ and $\mathsf{ptx.chg}$ have valid range proofs: the former two in the ledger's kernel $K_\Lambda = (\boldsymbol{\pi}_\Lambda, \mathbf{E}_\Lambda, \sigma_\Lambda)$ (by (i)), and $\mathsf{ptx.chg}$ in the kernel of $\mathsf{tx}_{\mathsf{ptx}}$ contained in ptx. From these proofs the reduction extracts the values $\mathbf{v}_{\Lambda.\mathsf{out}}, \mathbf{v}_{\mathsf{ptx.in}}, \mathbf{v}_{\mathsf{ptx.chg}} \in [0, v_{\max}]^*$ and keys $\mathbf{k}_{\Lambda.\mathsf{out}}, \mathbf{k}_{\mathsf{ptx.in}}, \mathbf{k}_{\mathsf{ptx.chg}} \in \mathcal{K}^*_{\mathsf{pp}}$ of every coin. We first argue that

(iii) $\sum \mathbf{v}_{\Lambda.\mathsf{out}} - \Lambda.\mathsf{sply} = 0$ and (iv) $\sum \mathbf{v}_{\mathsf{ptx.chg}} + \mathsf{ptx.rmdr} - \sum \mathbf{v}_{\mathsf{ptx.in}} = 0$.

If (iii) was not the case then ($v^* := \sum \mathbf{v}_{\Lambda.\mathsf{out}} - \Lambda.\mathsf{sply}$, $k^* := \sum \mathbf{k}_{\Lambda.\mathsf{out}}$) would be a non-zero opening of the excess Exc of Λ. Since furthermore $\mathsf{Exc} = \sum \mathbf{E}_\Lambda$ and σ_Λ is valid for $\mathbf{E}_\Lambda$, the tuple $(\mathbf{E}_\Lambda, \sigma_\Lambda, (v^*, k^*))$ would be an EUF-NZO solution.

Likewise, a non-zero left-hand side of (iv) can be used together with the kernel of $\mathsf{tx}_{\mathsf{ptx}}$ to break EUF-NZO. From (i)–(iv) we now get

$$\sum \mathbf{v} \overset{\text{(ii)}}{=} \mathsf{ptx.rmdr} \overset{\text{(iv)}}{=} \sum \mathbf{v}_{\mathsf{ptx.in}} - \sum \mathbf{v}_{\mathsf{ptx.chg}} \le \sum \mathbf{v}_{\mathsf{ptx.in}} \overset{\text{(i)}}{\le} \sum \mathbf{v}_{\Lambda.\mathsf{out}} \overset{\text{(iii)}}{=} \Lambda.\mathsf{sply} \ ,$$

which contradicts the fact that $\mathcal{A}$ won $\mathrm{INFL}_{\mathsf{MW}}$, as this requires $\sum \mathbf{v} > \Lambda.\mathsf{sply}$. (The function $\nu_{\mathcal{A}}$ accounts for (iii) (or (iv)) only holding over $\mathbb{Z}_p$ but not over $\mathbb{Z}$; this would imply $|\Lambda.\mathsf{out}| \ge p/v_{\max}$, which can only happen with negligible probability $\nu_{\mathcal{A}}$ for a p.p.t. $\mathcal{A}$.)

Theorem 14 (Theft-resistance (Def. 11)). *Assume that the pair* $(\mathsf{COM}, \mathsf{SIG})$ *is EUF-CRO-secure and that* Π *is zero-knowledge and simulation-extractable. Then the aggregate cash system* $\mathsf{MW}[\mathsf{COM}, \mathsf{SIG}, \Pi]$ *is secure against coin theft. More precisely, for any* $v_{\max}$ *and any p.p.t. adversary* $\mathcal{A}$*, which, via its oracle calls, makes the challenger create at most* $h_{\mathcal{A}}$ *coins and whose queries* $(\mathbf{C}, \mathbf{v}')$ *to* $\mathrm{S{\scriptsize END}}$ *satisfy* $|\mathbf{v}'| \le n_{\mathcal{A}}$*, there exists a negligible function* ν*, a p.p.t. adversary* $\mathcal{B}$ *making a single signing query, and p.p.t. adversaries* $\mathcal{B}_{\mathrm{zk}}$ *and* $\mathcal{B}_{\mathrm{se}}$ *such that*

$$\mathsf{Adv}^{\mathrm{steal}}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \le h_{\mathcal{A}}(\lambda) \cdot n_{\mathcal{A}}(\lambda) \cdot \big(\mathsf{Adv}^{\mathrm{euf\text{-}cro}}_{\mathsf{COM},\mathsf{SIG},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathrm{zk}}_{\Pi,\mathsf{R}_{v_{\max}},\mathcal{B}_{\mathrm{zk}}}(\lambda) + \mathsf{Adv}^{\mathrm{s\text{-}ext}}_{\Pi,\mathsf{R}_{v_{\max}},\mathcal{B}_{\mathrm{se}}}(\lambda)\big) + \nu(\lambda) \ .$$

The proof can be found in Appendix A.3. Here we give some proof intuition. We first assume that all coins created by the challenger are different. By Lemma 3 the probability $\nu(\lambda)$ that two coins collide is negligible.

Since in game STEAL the ledger is maintained by the challenger we have:

(i) the kernel of Λ contains a valid range proof for each coin in $\Lambda.\mathsf{out}$.

In order to win the game, the adversary must at some point *steal* some coin $\widetilde{C}$ from the challenger, by creating a transaction tx^* with $\widetilde{C}$ among its inputs, that is, $\mathsf{tx}^* = (s, \mathbf{C}, \hat{\mathbf{C}}, (\boldsymbol{\pi}, \mathbf{E}, \sigma))$ with $\widetilde{C} \in \mathbf{C}$. For tx^* to be accepted to the ledger, we must have:

(ii) $\mathbf{C} \subseteq \Lambda.\mathsf{out}$;
(iii) tx^* is valid, meaning
  (a) the signature σ verifies under key list $\mathbf{E}$;
  (b) $\sum \mathbf{E} = \sum \hat{\mathbf{C}} - \sum \mathbf{C} - \mathsf{Cmt}(\mathsf{cp}, s, 0)$;
  (c) all proofs $\boldsymbol{\pi}$ for coins $\hat{\mathbf{C}}$ are valid.

From (i), (ii) and (iii)(c) we have that all coins in $\mathbf{C}$ and $\hat{\mathbf{C}}$ have valid proofs, which means we can extract (except for $\widetilde{C}$, as we will see later) their values $\mathbf{v}$ and $\hat{\mathbf{v}}$ and keys $\mathbf{k}$ and $\hat{\mathbf{k}}$. This means, we can write (iii)(b) as:

$$\sum \mathbf{E} = -\widetilde{C} + \mathsf{Cmt}(\mathsf{cp}, \underbrace{\sum \hat{\mathbf{v}} - \sum \mathbf{v} - s}_{=:v^*}, \underbrace{\sum \hat{\mathbf{k}} - \sum \mathbf{k}}_{=:k^*}) \ . \tag{5}$$

Now, if we had set $\widetilde{C} = C^*$ with C^* a challenge for EUF-CRO then (iii)(a) and Equation (5) together would imply that $(\mathbf{E}, \sigma, (v^*, k^*))$ is a solution for C^* in EUF-CRO. So the basic proof idea is to embed a challenge C^* as one of the honest coins $\widetilde{C}$ created in the system and hope that the adversary will steal $\widetilde{C}$. When $\widetilde{C}$ is first created, it can be during a call to MINT, SEND or RECEIVE, each of which will create a transaction tx using $\mathsf{MW.MkTx}$; we thus set $\mathsf{tx.out}[j] = \widetilde{C}$ for some j. Now tx must contain a range proof for $\widetilde{C}$, which we produce using the zero-knowledge simulator, and a signature under verification key

$$\sum \mathsf{tx.out} - \sum \mathsf{tx.in} = \big(\sum_{i \neq j} \mathsf{tx.out}[i] - \sum \mathsf{tx.in}\big) + \widetilde{C} \ . \tag{6}$$

The coin keys of $\mathsf{tx.in}$ are input to $\mathsf{MW.MkTx}$ and those of $(\mathsf{tx.out}[i])_{i \neq j}$ are created by it. So we know the secret key a for the expression in parentheses in (6) and can therefore make a query $\mathrm{SIGN}'(a)$ to the related-key signing oracle to obtain the signature.

While this shows that simulating the creation of coin $\widetilde{C}$ is easily dealt with, what complicates the proof is when the adversary queries $\mathrm{SEND}(\mathbf{C}, \mathbf{v}')$ with $\widetilde{C} \in \mathbf{C}$, which should produce a pre-transaction $\widetilde{\mathsf{ptx}}$. Since $\widetilde{C}$ is a (say the j-th) input of $\widetilde{\mathsf{ptx}}$, this would require a signature related to $-C^*$ for which we cannot use the SIGN' oracle. Instead, we pick one random, say the $\tilde{\imath}$-th, change coin $\overline{C}$ and embed the challenge C^* in $\overline{C}$ as well. (If there are no change coins, we abort; we justify this below) To complete $\widetilde{\mathsf{ptx}}$ we now need a signature for key

$$\sum_{i \neq \tilde{\imath}} \mathsf{tx.out}[i] + \overline{C} - \sum_{i \neq j} \mathsf{tx.in}[i] - \widetilde{C} \ ,$$

and since the two occurrences of C^* cancel out, the simulation knows the signing key of the above expression. (The way the reduction actually embeds C^* in a coin $\widetilde{C}$ which in the game is supposed to have value v is by setting $\widetilde{C} := C^* + \mathsf{Cmt}(\mathsf{cp}, v, k)$.)

Let's look again at the transaction tx^* with which the adversary steals $\widetilde{C}$: for tx^* to actually steal $\widetilde{C}$, we must have $\widetilde{\mathsf{ptx}}.\mathsf{chg} \not\subseteq \mathsf{tx}^*.\mathsf{out}$ (where $\widetilde{\mathsf{ptx}}$ was the pre-transaction sending $\widetilde{C}$) as otherwise tx^* could simply be a transaction that completes $\widetilde{\mathsf{ptx}}$. If we were lucky when choosing $\overline{C}$ and $\overline{C}$ is one of the coins that the adversary did not include in $\mathsf{tx}^*.\mathsf{out}$, then tx^* satisfies all the properties in (iii) above, in particular (5), meaning we have a solution to EUF-CRO.

Unfortunately, there is one more complication: the adversary could have included $\overline{C}$ as one of the *inputs* of tx^*, in which case we cannot solve EUF-CRO, since (5) would be of the form

$$\sum \mathbf{E} = -2 \cdot \widetilde{C} + \mathsf{Cmt}(\mathsf{cp}, v^*, k^*) \ . \tag{7}$$

But intuitively, in this case the adversary has also "stolen" $\overline{C}$ and if we had randomly picked $\overline{C}$ when first embedding C^* then we could also solve EUF-CRO.

Unfortunately, "stealing" a change output that has not been added to Hon yet does *not* constitute a win according to game STEAL. To illustrate the issue, consider an adversary making the following queries (where all coins C_1 through C_5 have value 1), which the sketched reduction cannot use to break EUF-CRO:

– LEDGER(tx) with $\mathsf{tx} = (2, (\,), (C_1, C_2), K)$ $\quad\rightarrow\ \Lambda.\mathsf{out} = (C_1, C_2)$, $\mathsf{Hon} = (\,)$
– MINT((1)), creating coin C_3 $\quad\rightarrow\ \Lambda.\mathsf{out} = (C_1, C_2, C_3)$, $\mathsf{Hon} = (C_3)$
– SEND($(C_3), (1, 1)$), creating change C_4, C_5 $\quad\rightarrow\ \Lambda.\mathsf{out} = (C_1, C_2, C_3)$, $\mathsf{Hon} = (C_3)$

– $\textsc{Ledger}((0,(C_1),(C_4),K'))$ $\rightarrow$ $\Lambda.\mathsf{out} = (C_2, C_3, C_4)$, $\mathsf{Hon} = (C_3)$
– $\textsc{Ledger}((0,(C_2),(C_5),K''))$ $\rightarrow$ $\Lambda.\mathsf{out} = (C_3, C_4, C_5)$, $\mathsf{Hon} = (C_3)$
– $\textsc{Ledger}((0,(C_3,C_4,C_5),(C_6),K^*) =: \mathsf{tx}^*)$ $\rightarrow$ $\Lambda.\mathsf{out} = (C_6)$, $\mathsf{Hon} = (C_3)$

Note that all calls $\textsc{Ledger}(\mathsf{tx})$ leave Hon unchanged, since for ptx created during the $\textsc{Send}$ call we have $(C_4, C_5) = \mathsf{ptx.chg} \not\subseteq \mathsf{tx.out}$. The adversary wins the game since it stole C_3, so the reduction must have set $\widetilde{C} = C_3$; moreover, in order to simulate the $\textsc{Send}$ query, it must set $\overline{C}$ to C_4 or C_5. But now tx^* is of the form as in (7), which the reduction cannot use to break EUF-CRO.

The solution to making the reduction always work is to actually prove a *stronger* security notion, where the adversary not only wins when it spends a coin from Hon (in a way that is not simply a completion of a pre-transaction obtained from $\textsc{Send}$), but also if the adversary spends a change output which has not been included in Hon yet. Let us denote the set of all such coins by Chg and stress that if the adversary steals a coin from Chg, which the reduction guessed correctly, then there exists only one coin with the challenge embedded in it and so the situation as in (7) cannot arise.

In the proof of this strengthened notion the reduction now guesses the first coin that was stolen from Chg or Hon and if both happen in the same transaction it only accepts a coin from Chg as the right guess. (In the example above, the guesses $\widetilde{C} = C_4$ or $\widetilde{C} = C_5$ would be correct.)

It remains to argue that the reduction can abort when the adversary makes a query $\textsc{Send}(\mathbf{C}, (\,))$ with $\widetilde{C} \in \mathbf{C}$: in this case its guess $\widetilde{C}$ must have been wrong: for ptx returned by this oracle call we have $\mathsf{ptx.chg} \subseteq \mathsf{tx.out}$ for any tx, so $\mathsf{ptx.in}$ and thus $\widetilde{C}$ is removed from Hon whenever $\mathcal{A}$ makes a $\textsc{Ledger}$ call (which it must make in order to steal a coin), assuming w.l.o.g. that the adversary stops as soon as it has made its stealing transaction.

Finally, what happens if the adversary makes a query $\textsc{Send}(\mathbf{C}, \mathbf{v}')$ with $\overline{C} \in \mathbf{C}$? We could embed the challenge a *third* time, in one of the change coins of the pre-transaction we need to simulate. Instead of complicating the analysis, the reduction can actually safely abort if such a query is made, since its guess must have been wrong: $\textsc{Send}$ must be queried on honest coins, so we must have $\overline{C} \in \mathsf{Hon}$. As only the $\textsc{Ledger}$ oracle can add existing coins to Hon, it must have been queried with some tx such that $\widetilde{\mathsf{ptx}}.\mathsf{chg} \subseteq \mathsf{tx.out}$, as then $\widetilde{\mathsf{ptx}}.\mathsf{chg} \ni \overline{C}$ would be added to Hon; however at the same time this removes $\widetilde{\mathsf{ptx}}.\mathsf{in} \ni \widetilde{C}$ from Hon, which means that $\widetilde{C}$ cannot be the coin the adversary steals, because $\widetilde{C}$ cannot be included in Hon a second time. (As just analyzed for $\overline{C}$ above, the only way to add an existing coin $\widetilde{C}$ to Hon is if $\widetilde{C}$ was created as change during a query $\mathsf{ptx} \leftarrow \textsc{Send}(\mathbf{C}, \mathbf{v})$. But since $\widetilde{C}$ had already been in Hon, there must have been a call $\textsc{Ledger}(\mathsf{tx})$ with tx completing ptx, after which ptx is discarded from the list Ptx of pre-transactions awaiting inclusion in the ledger; see Figure 8).

Theorem 15 (Transaction indistinguishability (Def. 12)). *Assume that* COM *is a homomorphic hiding commitment scheme,* SIG *a compatible signature scheme, and* Π *is a zero-knowledge proof system. Then the aggregate cash system* $\mathsf{MW}[\mathsf{COM}, \mathsf{SIG}, \Pi]$ *is transaction-indistinguishable. More precisely, for any* $v_{\max}$ *and any p.p.t. adversary* $\mathcal{A}$ *which makes at most* $q_{\mathcal{A}}$ *queries to its oracle* $\textsc{Tx}$*, there exist p.p.t. adversaries* $\mathcal{B}_{\text{zk}}$ *and* $\mathcal{B}_{\text{hid}}$ *such that*

$$\mathsf{Adv}^{\text{tx-ind}}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \leq \mathsf{Adv}^{\text{zk}}_{\Pi,\mathsf{R}_{v_{\max}},\mathcal{B}_{\text{zk}}}(\lambda) + q_{\mathcal{A}} \cdot \mathsf{Adv}^{\text{hid}}_{\mathsf{COM},\mathcal{B}_{\text{hid}}}(\lambda)\ .$$

The proof can be found in Appendix A.4 and intuitively follows from commitments being hiding and proofs zero-knowledge, and that the coin $C^* = \mathsf{Cmt}(\mathsf{cp}, \rho, k^*)$ that is contained in a pre-transaction together with its key k^* (C^* is then spent by Rcv and eliminated from the final transaction by cut-through) acts as a randomizer between E' and E''. We moreover use the fact that because COM is homomorphic, for any values with $\sum \mathbf{v}'_0 + \sum \mathbf{v}''_0 - \sum \mathbf{v}_0 = \sum \mathbf{v}'_1 + \sum \mathbf{v}''_1 - \sum \mathbf{v}_1$,

```
Game $\mathrm{DL}_{\mathsf{GrGen},\mathcal{A}}(\lambda)$
    $\Gamma \coloneqq (p, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda)$
    $x \leftarrow\!\!\$\, \mathbb{Z}_p\,;\ X \coloneqq xG$
    $x' \leftarrow \mathcal{A}(\Gamma, X)$
    $\mathbf{return}\ x' = x$
```

```
Game $\mathrm{CDH}_{\mathsf{GrGen}',\mathcal{A}}(\lambda)$
    $\Gamma' \coloneqq (p, \mathbb{G}, \mathbb{G}_T, e, G) \leftarrow \mathsf{GrGen}'(1^\lambda)$
    $x \leftarrow\!\!\$\, \mathbb{Z}_p\,; y \leftarrow\!\!\$\, \mathbb{Z}_p$
    $Z \leftarrow \mathcal{A}(\Gamma, xG, yG)$
    $\mathbf{return}\ Z = xyG$
```

Fig. 12. The Discrete Logarithm and Computational Diffie-Hellman games.

the tuple

$$\big(\mathbf{C} \coloneqq \mathsf{Cmt}(\mathsf{cp}, \mathbf{v}_b, \mathbf{k}), \mathbf{C}' \coloneqq \mathsf{Cmt}(\mathsf{cp}, \mathbf{v}'_b, \mathbf{k}') \,\|\, \mathbf{C}'' \coloneqq \mathsf{Cmt}(\mathsf{cp}, \mathbf{v}''_b, \mathbf{k}''), \overline{k}\big) \tag{8}$$

hides the bit b even when $\overline{k} \coloneqq \sum \mathbf{k}' + \sum \mathbf{k}'' - \sum \mathbf{k}$ is revealed.

We prove Theorem 15 by showing that transactions returned by oracle Tx when $b = 0$ are indistinguishable from transactions returned when $b = 1$. These are of the form

$$\mathsf{tx}^* = (0, \mathbf{C}, \mathbf{C}' \,\|\, \mathbf{C}'', (\boldsymbol{\pi}' \,\|\, \boldsymbol{\pi}'', (E', E''), \sigma^*))\ , \tag{9}$$

where $E' = \sum \mathbf{C}' + C^* - \sum \mathbf{C}$ and $E'' = \sum \mathbf{C}'' - C^*$, and σ^* is an aggregation of signatures σ' and σ'' under keys $r' \coloneqq \sum \mathbf{k}' + k^* - \sum \mathbf{k}$ and $r'' \coloneqq \sum \mathbf{k}'' - k^*$, respectively. We thus have $E' = \mathsf{Cmt}(\mathsf{cp}, 0, r')$ and $E'' = \mathsf{Cmt}(\mathsf{cp}, 0, r'')$.

Together with the fact that k^* is uniform and never revealed, indistinguishability of (8) implies indistinguishability of tx^*, as we can create a tuple as in (9) from a tuple as in (8): simulate the proofs $\boldsymbol{\pi}' \,\|\, \boldsymbol{\pi}''$, choose a random r^* and set $E' = \mathsf{Cmt}(\mathsf{cp}, 0, r^*)$, $E'' = \mathsf{Cmt}(\mathsf{cp}, 0, \overline{k} - r^*)$, $\sigma' \leftarrow \mathsf{Sign}(\mathsf{sp}, r^*, \varepsilon)$ and $\sigma'' \leftarrow \mathsf{Sign}(\mathsf{sp}, \overline{k} - r^*, \varepsilon)$ and aggregate σ' and σ''.

5 Instantiations

5.1 Schemes

Throughout this section, we make use of prime-order abelian groups, potentially equipped with a (symmetric) bilinear map. A group description is a tuple $\Gamma = (p, \mathbb{G}, G)$ where p is an odd prime of length λ, $\mathbb{G}$ is an additive group of order p, and G is a generator of $\mathbb{G}$. A bilinear group description is a tuple $\Gamma' = (p, \mathbb{G}, \mathbb{G}_T, e, G)$ where p is an odd prime of length λ, $\mathbb{G}$ and $\mathbb{G}_T$ are groups of order p (we denote $\mathbb{G}_T$ multiplicatively), G is a generator of $\mathbb{G}$ and e is an efficiently computable non-degenerate bilinear map $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ (i.e., the map e is such that for all $U, V \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, $e(aU, bV) = e(U, V)^{ab}$, and $e(G, G)$ is a generator of $\mathbb{G}_T$). We assume the existence of a p.p.t. algorithm GrGen (GrGen') that, given as input the security parameter in unary 1^λ, outputs a group description Γ (a bilinear group description Γ').

For groups generated by GrGen we will make the assumption that discrete logarithms (DL) are hard to compute, while for bilinear groups generated by GrGen' we will make the computational Diffie-Hellman (CDH) assumption. They state that $\mathsf{Adv}^{\mathrm{dl}}_{\mathsf{GrGen},\mathcal{A}_{\mathrm{dl}}}(\lambda) \coloneqq \Pr\left[\mathrm{DL}_{\mathsf{GrGen},\mathcal{A}_{\mathrm{dl}}}(\lambda) = \mathbf{true}\right]$ and $\mathsf{Adv}^{\mathrm{cdh}}_{\mathsf{GrGen}',\mathcal{A}_{\mathrm{cdh}}}(\lambda) \coloneqq \Pr\left[\mathrm{CDH}_{\mathsf{GrGen}',\mathcal{A}_{\mathrm{cdh}}}(\lambda) = \mathbf{true}\right]$ are negligible in λ for all p.p.t. adversaries $\mathcal{A}_{\mathrm{dl}}$ and $\mathcal{A}_{\mathrm{cdh}}$, where games DL and CDH are specified in Figure 12.

For the Pedersen-Schnorr (Pedersen-BLS) instantiation, the main setup algorithm $\mathsf{MainSetup}$ consists of a (bilinear) group generation algorithm GrGen (GrGen').

Pedersen Commitments. The homomorphic commitment scheme proposed by Pedersen [Ped92], denoted PDS, is defined in Figure 13 (we only consider the case where the main parameters are a group description Γ; this translates immediately to the case of a bilinear group description Γ'). In order to commit to a value $v \in \mathbb{Z}_p$, one samples $r \leftarrow\!\!\$\ \mathbb{Z}_p$ and returns

$$C \coloneqq \mathsf{PDS.Cmt}(v, r) = vH + rG \ ,$$

where H is a generator contained in the parameters. A commitment C is opened by providing the value v and the randomness r. Pedersen commitments are computationally binding under the DL assumption and perfectly hiding. Since $\mathsf{PDS.Cmt}(v_0, r_0) + \mathsf{PDS.Cmt}(v_1, r_1) = \mathsf{PDS.Cmt}(v_0 + v_1, r_0 + r_1)$, Pedersen commitments are additively homomorphic.

Schnorr Signatures. We recall the Schnorr signature scheme [Sch91] in Figure 14. Note that we use the key-prefixed variant of the scheme, where the public key is hashed together with the commitment and the message. This corresponds to the *strong* Fiat-Shamir transform as defined in [BPW12], which ensures extractability in situations where the adversary can select public keys adaptively, which is the case in the EUF-NZO and EUF-CRO security games. Note that no non-interactive aggregation procedure is known for Schnorr signatures other than trivially concatenating individual signatures.

BLS Signatures. The Boneh–Lynn–Shacham (BLS) signature scheme [BLS01] is a simple deterministic signature scheme based on pairings. It is defined in Figure 15. We consider the key-prefixed variant of the scheme (i.e., the public key is hashed together with the message) which allows to securely aggregate signatures on the same message [BGLS03, BNN07]. EUF-CMA-security can be proved in the random oracle model under the CDH assumption.

The proof system. In contrast to COM and SIG, there are no compatibility or joint security requirements for the proof system. In practice, the Bulletproofs scheme [BBB$^+$18] could be used, although under which assumptions it satisfies Definition 8 remains to be studied.

5.2 Security of Pedersen-Schnorr

Our security proofs for the Pedersen-Schnorr pair are in the random oracle model and make use of the standard rewinding technique of Pointcheval and Stern [PS96] for extracting discrete logarithms from a successful adversary. This requires some particular care since in both the EUF-NZO and the EUF-CRO games, the adversary can output multiple signatures for distinct public keys for which the reduction must extract discrete logarithms. Fortunately, a generalized forking lemma by Bagherzandi, Cheon, and Jarecki [BCJ08] shows that for Schnorr signatures, one can perform multiple extractions efficiently. We recall this result in Appendix A.5. Equipped with it, we can prove the following two lemmas, whose proofs can be found in Appendix A.6.

Lemma 16. *The pair* $(\mathsf{PDS}, \mathsf{SCH})$ *is EUF-NZO-secure in the random oracle model under the DL assumption. More precisely, for any p.p.t. adversary* $\mathcal{A}$ *making at most* q_h *random oracle queries and returning a forgery for a list of size at most* N*, there exists a p.p.t. adversary* $\mathcal{B}$

```
PDS.Setup($\Gamma$)
    $(p, \mathbb{G}, G) \coloneqq \Gamma$;  $H \leftarrow\!\!\$\ \mathbb{G}$
    return $\mathsf{cp} \coloneqq (\Gamma, H)$

PDS.Cmt(cp, $v$, $r$)
    $((p, \mathbb{G}, G), H) \coloneqq \mathsf{cp}$
    return $C \coloneqq vH + rG$
```

Fig. 13. The Pedersen commitment scheme.

```
SCH.Setup($\mathit{\Gamma}$)
    $(p, \mathbb{G}, G) := \mathit{\Gamma}$
    Select $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$
    return $\mathsf{sp} := (\mathit{\Gamma}, \mathcal{H})$

SCH.KeyGen(sp)
    $((p, \mathbb{G}, G), \mathcal{H}) := \mathsf{sp}$
    $x \leftarrow\$ \mathbb{Z}_p;\ X := xG$
    $\mathsf{sk} := x;\ \mathsf{pk} := X$
    return $(\mathsf{sk}, \mathsf{pk})$

SCH.Agg(sp, $(\mathbf{L}_0, \boldsymbol{\sigma}_0), (\mathbf{L}_1, \boldsymbol{\sigma}_1)$)
    return $\boldsymbol{\sigma}_0 \parallel \boldsymbol{\sigma}_1$

SCH.Sign(sp, sk, $m$)
    $((p, \mathbb{G}, G), \mathcal{H}) := \mathsf{sp}$
    $x := \mathsf{sk};\ X := xG$
    $r \leftarrow\$ \mathbb{Z}_p;\ R := rG$
    $c := \mathcal{H}(X, R, m);\ s := r + cx$
    return $\sigma := (R, s)$

SCH.Ver(sp, $\mathbf{L}, \sigma$)
    $((p, \mathbb{G}, G), \mathcal{H}) := \mathsf{sp}$
    $((X_i, m_i))_{i=1}^n := \mathbf{L}$
    $((R_i, s_i))_{i=1}^n := \boldsymbol{\sigma}$
    for $i$ in $[1, n]$ do $c_i := \mathcal{H}(X_i, R_i, m_i)$
    return $\bigwedge_{i=1}^n (s_i G = R_i + c_i X_i)$
```

Fig. 14. The Schnorr signature scheme.

running in time at most $8N^2 q_h/\delta_\mathcal{A} \cdot \ln(8N/\delta_\mathcal{A}) \cdot t_\mathcal{A}$, where $\delta_\mathcal{A} = \mathsf{Adv}^{\text{euf-nzo}}_{\mathsf{PDS,SCH},\mathcal{A}}(\lambda)$ and $t_\mathcal{A}$ is the running time of $\mathcal{A}$, such that

$$\mathsf{Adv}^{\text{euf-nzo}}_{\mathsf{PDS,SCH},\mathcal{A}}(\lambda) \le 8 \cdot \mathsf{Adv}^{\text{dl}}_{\mathsf{GrGen},\mathcal{B}}(\lambda) .$$

Lemma 17. *The pair* $(\mathsf{PDS}, \mathsf{SCH})$ *is EUF-CRO-secure in the random oracle model under the DL assumption. More precisely, for any p.p.t. adversary $\mathcal{A}$ making at most q_h random oracle queries and q_s signature queries, returning a forgery for a list of size at most N, and such that $\delta_\mathcal{A} = \mathsf{Adv}^{\text{euf-cro}}_{\mathsf{PDS,SCH},\mathcal{A}}(\lambda) \ge 2q_s/p$, there exists a p.p.t. adversary $\mathcal{B}$ running in time at most $16N^2(q_h + q_s)/\delta_\mathcal{A} \cdot \ln(16N/\delta_\mathcal{A}) \cdot t_\mathcal{A}$, where $t_\mathcal{A}$ is the running time of $\mathcal{A}$, such that*

$$\mathsf{Adv}^{\text{euf-cro}}_{\mathsf{PDS,SCH},\mathcal{A}}(\lambda) \le 8 \cdot \mathsf{Adv}^{\text{dl}}_{\mathsf{GrGen},\mathcal{B}}(\lambda) + \frac{q_s + 8}{p} .$$

Corollary 18. $\mathsf{MW}[\mathsf{PDS}, \mathsf{SCH}, \mathsf{\Pi}]$ *with* $\mathsf{\Pi}$ *zero-knowledge and simulation-extractable is inflation-resistant and theft-resistant in the random oracle model under the DL assumption.*

5.3 Security of Pedersen-BLS

The security proofs for the Pedersen-BLS pair are also in the random oracle model but do not use rewinding. They are reminiscent of the proof of [BGLS03, Theorem 3.2] and can be found in Appendix A.7. Note that EUF-CRO-security is only proved for adversaries making a constant number of signing queries. Fortunately, adversary $\mathcal{B}$ constructed in Theorem 14 makes a single signing query.

Lemma 19. *The pair* $(\mathsf{PDS}, \mathsf{BLS})$ *is EUF-NZO-secure in the random oracle model under the CDH assumption. More precisely, for any p.p.t. adversary $\mathcal{A}$ making at most q_h random oracle queries and returning a forgery for a list of size at most N, there exists a p.p.t. adversary $\mathcal{B}$ running in time at most $t_\mathcal{A} + (q_h + N + 2)t_M$, where $t_\mathcal{A}$ is the running time of $\mathcal{A}$ and t_M is the time of a scalar multiplication in $\mathbb{G}$, such that*

$$\mathsf{Adv}^{\text{cdh}}_{\mathsf{GrGen},\mathcal{B}}(\lambda) = \mathsf{Adv}^{\text{euf-nzo}}_{\mathsf{PDS,BLS},\mathcal{A}}(\lambda) .$$

```
$\mathsf{BLS.Setup}(\Gamma')$
    $(p, \mathbb{G}, \mathbb{G}_T, e, G) := \Gamma'$
    Select $\mathcal{H} : \{0,1\}^* \to \mathbb{G}$
    return $\mathsf{sp} := (\Gamma', \mathcal{H})$

$\mathsf{BLS.KeyGen}(\mathsf{sp})$
    $((p, \mathbb{G}, \mathbb{G}_T, e, G), \mathcal{H}) := \mathsf{sp}$
    $x \leftarrow\!\$\ \mathbb{Z}_p;\ X := xG$
    $\mathsf{sk} := x;\ \mathsf{pk} := X$
    return $(\mathsf{sk}, \mathsf{pk})$

$\mathsf{BLS.Sign}(\mathsf{sp}, \mathsf{sk}, m)$
    $((p, \mathbb{G}, \mathbb{G}_T, e, G), \mathcal{H}) := \mathsf{sp}$
    $x := \mathsf{sk};\ X := xG$
    $Q := \mathcal{H}(X, m)$
    return $\sigma := xQ$

$\mathsf{BLS.Agg}(\mathsf{sp}, (\mathbf{L}_0, \sigma_0), (\mathbf{L}_1, \sigma_1))$
    return $\sigma_0 + \sigma_1$

$\mathsf{BLS.Ver}(\mathsf{sp}, \mathbf{L}, \sigma)$
    $((p, \mathbb{G}, \mathbb{G}_T, e, G), \mathcal{H}) := \mathsf{sp}$
    $((X_i, m_i))_{i=1}^n := \mathbf{L}$
    return $e(\sigma, G) = \prod_{i=1}^n e(X_i, \mathcal{H}(X_i, m_i))$
```

Fig. 15. The BLS aggregate signature scheme.

Lemma 20. *The pair* $(\mathsf{PDS}, \mathsf{BLS})$ *is EUF-CRO-secure in the random oracle model under the CDH assumption. More precisely, for any p.p.t. adversary* $\mathcal{A}$ *making at most* q_h *random oracle queries and* $q_s = O(1)$ *signature queries and returning a forgery for a list of size at most* N*, there exists a p.p.t. adversary* $\mathcal{B}$ *running in time at most* $t_{\mathcal{A}} + (2q_h + 3q_s + N + 2)t_M$*, where* $t_{\mathcal{A}}$ *is the running time of* $\mathcal{A}$ *and* t_M *is the time of a scalar multiplication in* $\mathbb{G}$*, such that*

$$\mathsf{Adv}^{\mathrm{cdh}}_{\mathsf{GrGen},\mathcal{B}}(\lambda) \geq \frac{1}{4 \cdot (2N)^{q_s}} \cdot \mathsf{Adv}^{\mathrm{euf\text{-}cro}}_{\mathsf{PDS},\mathsf{BLS},\mathcal{A}}(\lambda)\ .$$

Corollary 21. $\mathsf{MW}[\mathsf{PDS}, \mathsf{BLS}, \Pi]$ *with* Π *zero-knowledge and simulation-extractable is inflation-resistant and theft-resistant in the random oracle model under the CDH assumption.*

Acknowledgements. The first author is supported by the French ANR EfTrEC project (ANR-16-CE39-0002). The second author is supported by ERC grant 639554 (project aSCEND).

References

[AKR+13] Elli Androulaki, Ghassan Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in Bitcoin. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography 2013*, volume 7859 of *LNCS*, pages 34–51. Springer, Heidelberg, April 2013.

[Bac13] Adam Back. Bitcoins with homomorphic value (validatable but encrypted), October 2013. BitcoinTalk post, https://bitcointalk.org/index.php?topic=305791.0.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

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

[BCJ08] Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 08*, pages 449–458. ACM Press, October 2008.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, Heidelberg, May 2003.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001.

[BNM+14] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A. Kroll, and Edward W. Felten. Mixcoin: Anonymity for bitcoin with accountable mixes. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography 2014*, volume 8437 of *LNCS*, pages 486–504. Springer, Heidelberg, March 2014.

[BNN07] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *ICALP 2007*, volume 4596 of *LNCS*, pages 411–422. Springer, Heidelberg, July 2007.

[BPW12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 626–643. Springer, Heidelberg, December 2012.

[DDO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Heidelberg, August 2001.

[GCKG14] Arthur Gervais, Srdjan Capkun, Ghassan O. Karame, and Damian Gruber. On the privacy provisions of bloom filters in lightweight bitcoin clients. In Charles N. Payne Jr., Adam Hahn, Kevin R. B. Butler, and Micah Sherr, editors, *Annual Computer Security Applications Conference - ACSAC 2014*, pages 326–335. ACM, 2014.

[Gro06] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 444–459. Springer, Heidelberg, December 2006.

[HAB+17] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. TumbleBit: An untrusted bitcoin-compatible anonymous payment hub. In *NDSS 2017*. The Internet Society, February/March 2017.

[Jed16] Tom Elvis Jedusor. Mimblewimble, 2016. Available at `https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.txt`.

[KKM14] Philip Koshy, Diana Koshy, and Patrick McDaniel. An analysis of anonymity in bitcoin using P2P network traffic. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography 2014*, volume 8437 of *LNCS*, pages 469–485. Springer, Heidelberg, March 2014.

[LMRS04] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 74–90. Springer, Heidelberg, May 2004.

[Max13a] Gregory Maxwell. CoinJoin: Bitcoin privacy for the real world, August 2013. BitcoinTalk post, `https://bitcointalk.org/index.php?topic=279249.0`.

[Max13b] Gregory Maxwell. Transaction cut-through, August 2013. BitcoinTalk post, `https://bitcointalk.org/index.php?topic=281848.0`.

[Max15] Gregory Maxwell. Confidential Transactions, 2015. Available at `https://people.xiph.org/~greg/confidential_values.txt`.

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed E-cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411. IEEE Computer Society Press, May 2013.

[MPJ+13] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In Konstantina Papagiannaki, P. Krishna Gummadi, and Craig Partridge, editors, *Internet Measurement Conference, IMC 2013*, pages 127–140. ACM, 2013.

[Nak08] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. Available at `http://bitcoin.org/bitcoin.pdf`.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

[Poe16] Andrew Poelstra. Mimblewimble, 2016. Available at `https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf`.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Heidelberg, May 1996.

[RMK14] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. CoinShuffle: Practical decentralized coin mixing for bitcoin. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014, Part II*, volume 8713 of *LNCS*, pages 345–364. Springer, Heidelberg, September 2014.

[RS13] Dorit Ron and Adi Shamir. Quantitative analysis of the full Bitcoin transaction graph. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography 2013*, volume 7859 of *LNCS*, pages 6–24. Springer, Heidelberg, April 2013.

[RTRS18] Tim Ruffing, Sri Aravinda Thyagarajan, Viktoria Ronge, and Dominique Schröder. Burning Zerocoins for Fun and for Profit: A Cryptographic Denial-of-Spending Attack on the Zerocoin Protocol. IACR Cryptology ePrint Archive, Report 2018/612, 2018. Available at https://eprint.iacr.org/2018/612.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[SMD14] Amitabh Saxena, Janardan Misra, and Aritra Dhar. Increasing Anonymity in Bitcoin. In Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith, editors, *1st Workshop on Bitcoin Research - Bitcoin 2014*, volume 8438, pages 122–139. Springer, 2014.

[SZ16] Yonatan Sompolinsky and Aviv Zohar. Bitcoin's Security Model Revisited, 2016. Manuscript available at http://arxiv.org/abs/1605.09193.

[vS13] Nicolas van Saberhagen. CryptoNote v 2.0, 2013. Manuscript available at https://cryptonote.org/whitepaper.pdf.

A Omitted Proofs

A.1 EUF-NZO implies BND

Lemma 22. *Let* $(\mathsf{COM}, \mathsf{SIG})$ *be a pair of compatible additively homomorphic commitment and aggregate signature schemes. Then, for any p.p.t. adversary* $\mathcal{A}$*, there exists a p.p.t. adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\text{bnd}}_{\mathsf{COM},\mathcal{A}}(\lambda) = \mathsf{Adv}^{\text{euf-nzo}}_{\mathsf{COM},\mathsf{SIG},\mathcal{B}}(\lambda) \ .$$

Proof. Let $\mathcal{A}$ be an adversary against the binding security of COM. We construct an adversary $\mathcal{B}$ against the EUF-NZO security of $(\mathsf{COM}, \mathsf{SIG})$. On input $(\mathsf{cp}, \mathsf{sp})$, $\mathcal{B}$ simply runs $\mathcal{A}(\mathsf{cp})$ which returns (v_0, r_0) and (v_1, r_1) such that $v_0 \neq v_1$ and (using the homomorphic property) $\mathsf{COM.Cmt}(\mathsf{cp}, v_0 - v_1, r_0 - r_1) = \mathsf{COM.Cmt}(\mathsf{cp}, 0, 0)$. Then, $\mathcal{B}$ draws $r \leftarrow_\$ \mathcal{R}_{\mathsf{cp}} = \mathcal{S}_{\mathsf{sp}}$ and computes $C = \mathsf{COM.Cmt}(\mathsf{cp}, 0, r)$. Clearly, $\mathcal{B}$ can produce a signature for public key C for any message since it knows the corresponding secret key r. On the other hand, $C = \mathsf{COM.Cmt}(\mathsf{cp}, v_0 - v_1, r + r_0 - r_1)$, so that $\mathcal{B}$ also has an opening to a non-zero value for C, and hence can win the EUF-NZO game. Since $\mathcal{B}$ is successful exactly when $\mathcal{A}$ is, the result follows. □

A.2 Proof of Inflation-resistance of MW

Theorem 13 (Inflation-resistance (Def. 10)). *Assume that the pair* $(\mathsf{COM}, \mathsf{SIG})$ *is EUF-NZO-secure and that* Π *is zero-knowledge and simulation-extractable. Then the aggregate cash system* $\mathsf{MW}[\mathsf{COM}, \mathsf{SIG}, \Pi]$ *is secure against inflation. More precisely, for any* $v_{\max}$ *and any p.p.t. adversary* $\mathcal{A}$*, there exists a negligible function* $\nu_{\mathcal{A}}$ *and p.p.t. adversaries* $\mathcal{B}$*,* $\mathcal{B}_{\text{zk}}$ *and* $\mathcal{B}_{\text{se}}$ *such that*

$$\mathsf{Adv}^{\text{infl}}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \leq \mathsf{Adv}^{\text{euf-nzo}}_{\mathsf{COM},\mathsf{SIG},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\text{zk}}_{\Pi,\mathsf{R}_{v_{\max}},\mathcal{B}_{\text{zk}}}(\lambda) + \mathsf{Adv}^{\text{s-ext}}_{\Pi,\mathsf{R}_{v_{\max}},\mathcal{B}_{\text{se}}}(\lambda) + \nu_{\mathcal{A}}(\lambda) \ .$$

Proof. Consider an adversary $\mathcal{A}$ that runs on input public parameters $\mathsf{pp} = (\mathsf{cp}, \mathsf{sp}, \mathsf{crs})$ and an empty ledger, and wins the game $\text{INFL}_{\mathcal{A},\mathsf{MW}}(\lambda, v_{\max})$ with non-negligible probability. The proof proceeds via the following sequence of games, defined in Figure 16 (and whose indistinguishability we argue below):

Game$_0$. This is the original inflation game as presented in Definition 10 for the specific instantiation $\mathsf{CASH} := \mathsf{MW}$ where $\mathsf{MW.Setup}$ has been written out and the first winning condition $\perp \not\leftarrow \mathsf{MW.Ldgr}(\mathsf{pp}, \Lambda, \mathsf{tx})$ has been expanded with all those that are necessary for the specific case of MW. Hence,

$$\mathsf{Adv}^{\text{Game}_0}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) = \mathsf{Adv}^{\text{infl}}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \ . \tag{10}$$

```
Game $\mathsf{Game}_0$
$\mathsf{mp} \leftarrow \mathsf{MainSetup}(1^\lambda)\,;\ \mathsf{cp} \leftarrow \mathsf{COM.Setup}(\mathsf{mp})\,;\ \mathsf{sp} \leftarrow \mathsf{SIG.Setup}(\mathsf{mp})$
$\mathsf{crs} \leftarrow \Pi.\mathsf{Setup}(\mathsf{mp}, v_{\max})\,;\ \mathsf{pp} := (\mathsf{cp}, \mathsf{sp}, \mathsf{crs})\,; \Lambda_0 := \big(0, (\,), (\,), ((\,), (\,), \varepsilon)\big)$
$(\Lambda, \mathsf{ptx}, \mathbf{v}) \leftarrow \mathcal{A}(\mathsf{pp}, \Lambda_0)$
$(\mathsf{tx}, \mathbf{k}) \leftarrow \mathsf{MW.Rcv}(\mathsf{pp}, \mathsf{ptx}, \mathbf{v})$
return $\mathsf{Ver}(\mathsf{pp}, \Lambda)$ and $\mathsf{Ver}(\mathsf{pp}, \mathsf{tx})$ and $\Lambda.\mathsf{in} = (\,)$
        and $\mathsf{tx.in} \subseteq \Lambda.\mathsf{out}$ and $\Lambda.\mathsf{sply} < \sum \mathbf{v}$

Games $\mathsf{Game}_1$ and $\boxed{\mathsf{Game}_2}$
$\mathsf{mp} \leftarrow \mathsf{MainSetup}(1^\lambda)\,;\ \mathsf{cp} \leftarrow \mathsf{COM.Setup}(\mathsf{mp})\,;\ \mathsf{sp} \leftarrow \mathsf{SIG.Setup}(\mathsf{mp})$
$\mathsf{crs} \leftarrow \Pi.\mathsf{Setup}(\mathsf{mp}, v_{\max})\,;\ \mathsf{pp} := (\mathsf{cp}, \mathsf{sp}, \mathsf{crs})\,; \Lambda_0 := \big(0, (\,), (\,), ((\,), (\,), \varepsilon)\big)$
$(\Lambda, \mathsf{ptx}, \mathbf{v}) \leftarrow \mathcal{A}(\mathsf{pp}, \Lambda_0)$
$(\mathsf{tx}, \mathbf{k}) \leftarrow \mathsf{MW.Rcv}(\mathsf{pp}, \mathsf{ptx}, \mathbf{v})$
$\boxed{\text{if } p/v_{\max} \leq |\mathbf{v}| + |\Lambda.\mathsf{out}| + |\mathsf{ptx.chg}| \text{ then return false}}$ (I)
return $\mathsf{Ver}(\mathsf{pp}, \Lambda)$ and $\mathsf{Ver}(\mathsf{pp}, \mathsf{tx})$ and $\Lambda.\mathsf{in} = (\,)$
        and $\mathsf{ptx.in} \subseteq \Lambda.\mathsf{out}$ and $\Lambda.\mathsf{sply} < \mathsf{ptx.rmdr}$ and $\mathsf{Ver}(\mathsf{pp}, \mathsf{ptx})$

Games $\mathsf{Game}_3$ and $\boxed{\mathsf{Game}_4}$
$\mathsf{mp} \leftarrow \mathsf{MainSetup}(1^\lambda)\,;\ \mathsf{cp} \leftarrow \mathsf{COM.Setup}(\mathsf{mp})\,;\ \mathsf{sp} \leftarrow \mathsf{SIG.Setup}(\mathsf{mp})$
$(\mathsf{crs}, \tau) \leftarrow \Pi.\mathsf{SimSetup}(\mathsf{mp}, v_{\max})\,;\ \mathsf{pp} := (\mathsf{cp}, \mathsf{sp}, \mathsf{crs})\,; \Lambda_0 := \big(0, (\,), (\,), ((\,), (\,), \varepsilon)\big)$
$(\Lambda, \mathsf{ptx}, \mathbf{v}) \leftarrow \mathcal{A}(\mathsf{pp}, \Lambda_0)$
$(s, \mathbf{C}_\Lambda, \hat{\mathbf{C}}, K) := \Lambda\,; (\boldsymbol{\pi}, \mathbf{E}, \sigma) := K$        // just parsing
$(\mathsf{tx}', \rho, k^*) := \mathsf{ptx}\,;\ (s', \mathbf{C}, \mathbf{C}' \| \,(C^*)\,, K') := \mathsf{tx}'\,;\ (\boldsymbol{\pi}' \| \,(\pi^*)\,, \mathbf{E}', \sigma') := K'$        // just parsing
$(\mathsf{tx}, \mathbf{k}) \leftarrow \mathsf{MW.Rcv}(\mathsf{pp}, \mathsf{ptx}, \mathbf{v})$
if $p/v_{\max} \leq |\mathbf{v}| + |\hat{\mathbf{C}}| + |\mathbf{C}'|$ then return false
$\boxed{(\hat{\mathbf{v}} \| \mathbf{v}', \hat{\mathbf{k}} \| \mathbf{k}') := \Pi.\mathsf{Ext}(\mathsf{crs}, \tau, (\mathsf{cp}, \hat{\mathbf{C}} \| \mathbf{C}'), \boldsymbol{\pi} \| \boldsymbol{\pi}')}$
$\boxed{\text{if } \neg \mathsf{R}_{v_{\max}}\big((\mathsf{cp}, \hat{\mathbf{C}} \| \mathbf{C}'), (\hat{\mathbf{v}} \| \mathbf{v}', \hat{\mathbf{k}} \| \mathbf{k}')\big) \text{ then return false}}$
return $\mathsf{Ver}(\mathsf{pp}, \Lambda)$ and $\mathsf{Ver}(\mathsf{pp}, \mathsf{tx})$ and $\mathbf{C}_\Lambda = (\,)$
        and $\mathbf{C} \subseteq \hat{\mathbf{C}}$ and $s < \rho$ and $\mathsf{Ver}(\mathsf{pp}, \mathsf{ptx})$
```

Fig. 16. Games modifying $\mathrm{INFL}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max})$. Changes w.r.t. previous games are highlighted or $\boxed{\text{boxed}}$.

Game$_1$. We slightly change the last two winning conditions of Game$_0$ and add an extra one. We claim that this is perfectly indistinguishable from Game$_0$. Indeed, since $\mathsf{Ver}(\mathsf{pp}, \mathsf{tx}) = \textbf{true}$, this implies in particular that MW.Rcv did not return $\perp$. Then, $\mathsf{tx.in} = \mathsf{ptx.in}$ by correctness of MW.Rcv and, by inspection of MW.Rcv, $\mathsf{ptx.rmdr} = \sum \mathbf{v}$ and $\mathsf{Ver}(\mathsf{pp}, \mathsf{ptx})$ hold whenever MW.Rcv does not return $\perp$. Hence,

$$\mathsf{Adv}^{\mathrm{Game}_1}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) = \mathsf{Adv}^{\mathrm{Game}_0}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max})\ . \tag{11}$$

Game$_2$. We modify Game$_1$ so that the experiment returns **false** whenever the adversary creates too many coins (which may cause the sum of their values to be larger than p). We claim that the two games are computationally indistinguishable. Since the adversary $\mathcal{A}$ runs in polynomial time, there is a polynomial $t_{\mathcal{A}}(\lambda)$ that upper-bounds the output length of $\mathcal{A}$, and in particular

$|\mathbf{v}| + |\mathit{\Lambda}.\mathsf{out}| + |\mathsf{ptx.chg}|$. On the other hand, the value space $\mathcal{V}_{\mathsf{cp}} = \mathbb{Z}_p$ is such that $\lfloor \log p \rfloor + 1 = \lambda$; in other words, $p \geq 2^{\lambda-1}$. Therefore, there exists $\lambda_0 \in \mathbb{N}$ such that

$$v_{\max} \cdot t_{\mathcal{A}}(\lambda) < 2^{\lambda-1} \leq p \qquad \text{for any } \lambda \geq \lambda_0 .$$

Let $\nu_{\mathcal{A}}(\lambda)$ denote the probability that Game$_2$ returns **false** in line (I). We thus have

$$\mathsf{Adv}^{\mathrm{Game}_2}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \geq \mathsf{Adv}^{\mathrm{Game}_1}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) - \nu_{\mathcal{A}}(\lambda) \ . \tag{12}$$

Since there exists $\lambda_0 \in \mathbb{N}$ such that $|\mathbf{v}| + |\mathit{\Lambda}.\mathsf{out}| + |\mathsf{ptx.chg}| \leq t_{\mathcal{A}}(\lambda) < p/v_{\max}$ for all $\lambda > \lambda_0$, we have $\nu_{\mathcal{A}}(\lambda) = 0$ for all such λ. Therefore, $\nu_{\mathcal{A}}(\lambda)$ is negligible.

Game$_3$. The only difference with Game$_2$ is that crs is now generated together with an extraction trapdoor τ via $\mathsf{\Pi.SimSetup}$. We also parse $\mathit{\Lambda}$ and ptx and make some basic notational substitutions. Consider the following adversary $\mathcal{B}_{\mathrm{zk}}$ for game $\mathrm{ZK}_{\mathsf{\Pi},\mathsf{R}_{v_{\max}}}$. It receives crs (which is either produced by $\mathsf{\Pi.Setup}$ or by $\mathsf{\Pi..SimSetup}$) and simulates Game$_2$: it retrieves mp from crs, generates $\mathsf{cp} \leftarrow \mathsf{COM.Setup}(\mathsf{mp})$ and $\mathsf{sp} \leftarrow \mathsf{SIG.Setup}(\mathsf{mp})$, and runs $\mathcal{A}$ on input $(\mathsf{cp}, \mathsf{sp}, \mathsf{crs})$ and an empty ledger (note that the simulation does not require to query oracle SimProve). $\mathcal{B}_{\mathrm{zk}}$ returns 1 if $\mathcal{A}$ wins Game$_2$ and 0 otherwise. Since $\mathcal{B}_{\mathrm{zk}}$ perfectly simulates Game$_2$ or Game$_3$ depending on the bit of its ZK challenger, we have

$$\mathsf{Adv}^{\mathrm{Game}_3}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \geq \mathsf{Adv}^{\mathrm{Game}_2}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) - \mathsf{Adv}^{\mathrm{zk}}_{\mathsf{\Pi},\mathsf{R}_{v_{\max}},\mathcal{B}_{\mathrm{zk}}}(\lambda) \ . \tag{13}$$

Game$_4$. This final game attempts to extract valid openings for $\hat{\mathbf{C}}$ (the output coins of the ledger) and $\mathbf{C}'$ (the change coins of the pre-transaction) from the proofs contained in the kernels of $\mathit{\Lambda}$ and tx'; it returns **false** if extraction fails for any of these coins. The rest of the game is left unchanged. Consider an adversary $\mathcal{B}_{\mathrm{se}}$ for game S-EXT$_{\mathsf{\Pi}}$ which on input a simulated CRS crs simulates Game$_3$ for $\mathcal{A}$ (again this does not require to query oracle SimProve) and returns $(\hat{\mathbf{C}} \| \mathbf{C}', \boldsymbol{\pi} \| \boldsymbol{\pi}')$ if Game$_3$ returns **true** and aborts otherwise. Note that for Game$_3$ to return **true**, all range proofs in the kernel of $\mathit{\Lambda}$ and tx' must be valid. Hence, $\mathcal{B}_{\mathrm{se}}$ wins game S-EXT$_{\mathsf{\Pi}}$ exactly when Game$_3$ returns **true** and Game$_4$ returns **false**, so that

$$\mathsf{Adv}^{\mathrm{Game}_4}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) = \mathsf{Adv}^{\mathrm{Game}_3}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) - \mathsf{Adv}^{\mathrm{s\text{-}ext}}_{\mathsf{\Pi},\mathsf{R}_{v_{\max}},\mathcal{B}_{\mathrm{se}}}(\lambda) \ . \tag{14}$$

The reduction to EUF-NZO. We now construct an adversary $\mathcal{B}$ against EUF-NZO-security of $(\mathsf{COM}, \mathsf{SIG})$. $\mathcal{B}$ takes as input $(\mathsf{cp}, \mathsf{sp})$ and simulates Game$_4$: it retrieves mp from $(\mathsf{cp}, \mathsf{sp})$, generates $\mathsf{crs} \leftarrow \mathsf{\Pi.Sim.Setup}(\mathsf{mp})$, and runs $\mathcal{A}$ on input $(\mathsf{cp}, \mathsf{sp}, \mathsf{crs})$ and an empty ledger. If the game returns **false**, then $\mathcal{B}$ aborts. Otherwise, we show that $\mathcal{B}$ can break EUF-NZO each time Game$_4$ returns **true**.

Game$_4$ returning **true** implies in particular (in all the following we use the notation of Figure 16):

(i) $\mathsf{Ver}(\mathsf{pp}, \mathit{\Lambda}) = \mathbf{true}$; (ii) $\mathbf{C}_{\mathit{\Lambda}} = (\,)$; (iii) $\mathbf{C} \subseteq \hat{\mathbf{C}}$; (iv) $s < \rho$;

(v) $(|\mathbf{v}| + |\hat{\mathbf{C}}| + |\mathbf{C}'|) \cdot v_{\max} < p$;

(vi) $\mathsf{Ver}(\mathsf{pp}, \mathsf{ptx}) = \mathbf{true}$, which in turn implies:
  - (a) $\mathsf{Ver}(\mathsf{pp}, \mathsf{tx}') = \mathbf{true}$;
  - (b) $s' := \mathsf{tx}'.\mathsf{sply} = 0$;
  - (c) $C^* = \mathsf{COM.Cmt}(\mathsf{cp}, \rho, k^*)$.

Let $\hat{v} := \sum \hat{\mathbf{v}}$ and $\hat{k} := \sum \hat{\mathbf{k}}$. Two cases may occur.

– $\hat{v} \not\equiv s \bmod p$: In this case, $\mathcal{B}$ returns $(\mathbf{E}, \sigma, (\hat{v} - s, \hat{k}))$. We claim that this is a valid EUF-NZO solution. By (i), σ is a valid signature for $\mathbf{E}$ and $\sum \mathbf{E} = \mathsf{Exc}(\Lambda)$ where by definition:

$$\begin{aligned}\mathsf{Exc}(\Lambda) &= \textstyle\sum \hat{\mathbf{C}} - \sum \mathbf{C}_\Lambda - \mathsf{COM.Cmt}(\mathsf{cp}, s, 0)\\ &= \textstyle\sum \hat{\mathbf{C}} - \mathsf{COM.Cmt}(\mathsf{cp}, s, 0) && \text{(by (ii))}\\ &= \textstyle\sum_{i=1}^{|\hat{\mathbf{C}}|} \mathsf{COM.Cmt}(\mathsf{cp}, \hat{v}_i, \hat{k}_i) - \mathsf{COM.Cmt}(\mathsf{cp}, s, 0)\\ &= \mathsf{COM.Cmt}(\mathsf{cp}, \hat{v} - s, \hat{k})\ .\end{aligned}$$

Since $\hat{v} - s \not\equiv 0 \bmod p$, $(\hat{v} - s, \hat{k})$ is a non-zero opening for $\sum \mathbf{E}$.

– $\hat{v} \equiv s \bmod p$: In this case, $\mathcal{B}$ must exploit the pre-transaction (unlike in the previous case where the ledger was sufficient). Let $v' := \sum \mathbf{v}'$ and $k' := \sum \mathbf{k}'$. Let us denote with I the set of indexes such that $\mathbf{C} = (\hat{\mathbf{C}}[i])_{i \in I}$, which exists by (iii). By (vi)(a), σ' is a valid signature for $\mathbf{E}'$ and $\sum \mathbf{E}' = \mathsf{Exc}(\mathsf{tx}')$ where by definition:

$$\begin{aligned}\mathsf{Exc}(\mathsf{tx}') &= \textstyle\sum \mathbf{C}' + C^* - \sum \mathbf{C} - \mathsf{COM.Cmt}(\mathsf{cp}, s', 0)\\ &= \textstyle\sum \mathbf{C}' + C^* - \sum \mathbf{C} && \text{(by (vi)(b))}\\ &= \textstyle\sum \mathbf{C}' + \mathsf{COM.Cmt}(\mathsf{cp}, \rho, k^*) - \sum \mathbf{C} && \text{(by (vi)(c))}\\ &= \mathsf{COM.Cmt}(\mathsf{cp}, v', k') + \mathsf{COM.Cmt}(\mathsf{cp}, \rho, k^*) - \textstyle\sum_{i \in I} \mathsf{COM.Cmt}(\mathsf{cp}, \hat{v}_i, \hat{k}_i)\\ &= \mathsf{COM.Cmt}(\mathsf{cp}, v' + \rho - \textstyle\sum_{i \in I} \hat{v}_i, k' + k^* - \sum_{i \in I} \hat{k}_i)\ ,\end{aligned}$$

Let $v'' := v' + \rho - \sum_{i \in I} \hat{v}_i$ and $k'' := k' + k^* - \sum_{i \in I} \hat{k}_i$, so that (v'', k'') is an opening of $\mathsf{Exc}(\mathsf{tx}')$. At this stage, we only need to prove that $v'' \not\equiv 0 \bmod p$. To see it, note that:

$$\rho \overset{\text{(iv)}}{>} s \geq \hat{v} = \sum_{i \in [1, |\hat{\mathbf{C}}|]} \hat{v}_i \geq \sum_{i \in I} \hat{v}_i,$$

where the second inequality follows from $s \equiv \hat{v} \pmod p$, $s \geq 0$ (by (i)), and $0 \leq \hat{v} < |\hat{\mathbf{C}}| \cdot v_{\max} < p$ (by (v)). Since $v'' \geq \rho - \sum_{i \in I} \hat{v}_i$, this implies that $v'' > 0$. On the other hand,

$$v'' \leq v' + \rho \leq (|\mathbf{C}'| + |\mathbf{v}|) \cdot v_{\max} < p$$

again by (v). Hence, $0 < v'' < p$, and this proves our claim. Wrapping up, $(\mathbf{E}', \sigma', (v'', k''))$ is a valid EUF-NZO solution.

In both cases, $\mathcal{B}$ wins the EUF-NZO game every time Game_4 returns **true**. We thus have:

$$\mathsf{Adv}^{\text{euf-nzo}}_{\mathcal{B}}(\lambda) \geq \mathsf{Adv}^{\text{Game}_4}_{\mathcal{A}}(\lambda, v_{\max})\ . \tag{15}$$

The theorem follows from Eqs. (10) to (15). □

A.3 Proof of Theft-resistance of MW

Theorem 14 (Theft-resistance (Def. 11)). *Assume that the pair* (COM, SIG) *is EUF-CRO-secure and that* Π *is zero-knowledge and simulation-extractable. Then the aggregate cash system* MW[COM, SIG, Π] *is secure against coin theft. More precisely, for any* $v_{\max}$ *and any p.p.t. adversary* $\mathcal{A}$*, which, via its oracle calls, makes the challenger create at most* $h_{\mathcal{A}}$ *coins and whose queries* $(\mathbf{C}, \mathbf{v}')$ *to* SEND *satisfy* $|\mathbf{v}'| \leq n_{\mathcal{A}}$*, there exists a negligible function* ν*, a p.p.t. adversary* $\mathcal{B}$ *making a single signing query, and p.p.t. adversaries* $\mathcal{B}_{\text{zk}}$ *and* $\mathcal{B}_{\text{se}}$ *such that*

$$\mathsf{Adv}^{\text{steal}}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \leq h_{\mathcal{A}}(\lambda) \cdot n_{\mathcal{A}}(\lambda) \cdot (\mathsf{Adv}^{\text{euf-cro}}_{\mathsf{COM},\mathsf{SIG},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\text{zk}}_{\Pi,\mathsf{R}_{v_{\max}},\mathcal{B}_{\text{zk}}}(\lambda) + \mathsf{Adv}^{\text{s-ext}}_{\Pi,\mathsf{R}_{v_{\max}},\mathcal{B}_{\text{se}}}(\lambda)) + \nu(\lambda)\ .$$

Proof. To simplify the analysis, we first modify game STEAL in that it aborts if the experiment generates the same coin twice by chance. By Lemma 3, the probability $\nu(\lambda)$ of this happening is negligible. Now consider an adversary $\mathcal{A}$ that wins the (modified) game STEAL in Figure 8, thus $\mathsf{Hon} \not\subseteq \Lambda.\mathsf{out}$ holds when the adversary terminates. We proceed via a sequence of games.

```
Games $\mathsf{Game}_0$ and $\boxed{\mathsf{Game}_1}$
$(\mathsf{pp}, \Lambda) \leftarrow \mathsf{CASH.Setup}(1^\lambda, v_{\max})$
$\boxed{\mathsf{Chg},}$ $\mathsf{Hon}, \mathsf{Val}, \mathsf{Key}, \mathsf{Ptx} := (\,)$
$\mathcal{A}^{\textsc{Mint},\textsc{Send},\textsc{Receive},\textsc{Ledger}}(\mathsf{pp}, \Lambda)$
return false

Aux function $\mathsf{Store}(\mathbf{C}, \mathbf{v}, \mathbf{k})$
$\mathsf{Val}(\mathbf{C}) := \mathbf{v}$; $\mathsf{Key}(\mathbf{C}) := \mathbf{k}$

Oracle $\textsc{Mint}(\mathbf{v})$
$(\mathsf{tx}, \mathbf{k}) \leftarrow \mathsf{CASH.Mint}(\mathsf{pp}, \mathbf{v})$
$\Lambda \leftarrow \mathsf{CASH.Ldgr}(\mathsf{pp}, \Lambda, \mathsf{tx})$
$\mathsf{Hon} := \mathsf{Hon} \,\|\, \mathsf{tx.out}$
$\mathsf{Store}(\mathsf{tx.out}, \mathbf{v}, \mathbf{k})$
return tx

Oracle $\textsc{Send}(\mathbf{C}, \mathbf{v}')$
if $\mathbf{C} \not\subseteq \mathsf{Hon}$ or $\bigcup_{\mathsf{ptx} \in \mathsf{Ptx}} \mathsf{ptx.in} \cap \mathbf{C} \neq (\,)$
  return $\perp$
$(\mathsf{ptx}, \mathbf{k}') \leftarrow \mathsf{CASH.Send}\big(\mathsf{pp}, \mathbf{C}, \mathsf{Val}(\mathbf{C}), \mathsf{Key}(\mathbf{C}), \mathbf{v}'\big)$
$\mathsf{Store}(\mathsf{ptx.chg}, \mathbf{v}', \mathbf{k}')$; $\mathsf{Ptx} := \mathsf{Ptx} \,\|\, (\mathsf{ptx})$
$\boxed{\mathsf{Chg} := \mathsf{Chg} \,\|\, (\mathsf{ptx.chg})}$
return ptx

Oracle $\textsc{Receive}(\mathsf{ptx}, \mathbf{v})$
$(\mathsf{tx}, \mathbf{k}) \leftarrow \mathsf{CASH.Rcv}(\mathsf{pp}, \mathsf{ptx}, \mathbf{v})$
$\Lambda' \leftarrow \textsc{Ledger}(\mathsf{tx})$
if $\Lambda' = \perp$ then return $\perp$
$\mathsf{Hon} := \mathsf{Hon} \,\|\, (\mathsf{tx.out} - \mathsf{ptx.chg})$
$\mathsf{Store}(\mathsf{tx.out} - \mathsf{ptx.chg}, \mathbf{v}, \mathbf{k})$
return tx

Oracle $\textsc{Ledger}(\mathsf{tx})$
$\Lambda' \leftarrow \mathsf{CASH.Ldgr}(\mathsf{pp}, \Lambda, \mathsf{tx})$
if $\Lambda' = \perp$ then return $\perp$ else $\Lambda := \Lambda'$
for all $\mathsf{ptx} \in \mathsf{Ptx}$ do
  if $\mathsf{ptx.chg} \subseteq \mathsf{tx.out}$
    $\mathsf{Ptx} := \mathsf{Ptx} - (\mathsf{ptx})$
    $\mathsf{Hon} := (\mathsf{Hon} - \mathsf{ptx.in}) \,\|\, \mathsf{ptx.chg}$
    $\boxed{\mathsf{Chg} := \mathsf{Chg} - \mathsf{ptx.chg}}$
$\boxed{\text{if } \mathsf{tx.in} \cap \mathsf{Chg} \neq (\,)}$   // change stolen!
  $\boxed{\text{then stop game returning true}}$   (I)
if $\mathsf{tx.in} \cap \mathsf{Hon} \neq (\,)$   // coin stolen!
  then stop game returning true (II)
return $\Lambda$
```

Fig. 17. Reformulated and $\boxed{\text{strengthened}}$ coin stealing game.

Game$_0$. Inspection of the STEAL game shows the following:

- a coin in Hon must have been added to Hon during an oracle call to MINT, RECEIVE or LEDGER; during this, it is also added to $\Lambda.\mathsf{out}$;
- in order for a coin to be *removed* from $\Lambda.\mathsf{out}$, it has to be in $\mathsf{tx.in}$ for some tx queried to LEDGER;
- if after such a call the coin is still in Hon and the adversary stops, then it has won.

Following this analysis, we further modify the game STEAL as Game$_0$ in Figure 17 (ignore the boxes for now), so that it declares $\mathcal{A}$ won whenever the condition $\mathsf{Hon} \not\subseteq \Lambda.\mathsf{out}$ is first satisfied. We have highlighted the changes w.r.t. the original game in gray. By the above analysis we have

$$\mathsf{Adv}^{\mathrm{Game}_0}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \geq \mathsf{Adv}^{\mathrm{steal}}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) - \nu(\lambda) \ . \tag{16}$$

(Note that $\mathcal{A}$ could win Game$_0$ but not STEAL by putting a stolen coin back into the ledger.)

Game$_1$. To simplify the proof, we *strengthen* the security notion by defining a game that is easier to win than Game$_0$ and then show that even this is infeasible. Consider an adversary that queries SEND, which creates a pre-transaction ptx with change coins $\mathsf{ptx.chg}$, and then queries LEDGER on a transaction tx that spends coins of $\mathsf{ptx.chg}$ that are not in Hon yet. In the original game, this does not constitute a win, since only coins in Hon can be stolen.

Our strengthened game Game_1 does consider such behavior as winning the game. In particular, the game stores the change coins generated during $\textsc{Send}$ calls in a list Chg and removes them from Chg once they are added to Hon. It also stops and declares the game won if the adversary manages to spend a coin from Chg. Game_1 is also defined in Figure 17 by including the boxes. Since every adversary that wins Game_0 also wins Game_1, we have:

$$\mathsf{Adv}^{\mathrm{Game}_1}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \geq \mathsf{Adv}^{\mathrm{Game}_0}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \ . \tag{17}$$

Inspection of Game_1 yields that at any point during the execution the following holds: $\mathsf{Chg} \cup \mathsf{Hon}$ contains exactly all coins ever produced by the game and $\mathsf{Chg} \cap \mathsf{Hon} = (\,)$: coins are produced by $\textsc{Mint}$, $\textsc{Receive}$ or $\textsc{Send}$, where the former two add the coins to Hon and the latter adds them to Chg; further, all coins removed from Chg by $\textsc{Ledger}$ are added to Hon.

Game_2. Consider a winning execution of Game_1 and let tx^* denote the transaction such that $\mathcal{A}$ wins during call $\textsc{Ledger}(\mathsf{tx}^*)$. Define a coin $\widetilde{C}$ as follows: if $\mathcal{A}$ stole a coin from Chg, i.e., it won in line (I), then $\widetilde{C}$ is the first coin in $\mathsf{tx}^*.\mathsf{in}$ that is also in Chg; if the adversary won by stealing a coin from Hon, i.e., in line (II), then $\widetilde{C}$ is the first coin in $\mathsf{tx}^*.\mathsf{in}$ that is also in Hon.

Now consider the case the adversary wins in line (II) *and* previously made a query $\textsc{Send}(\mathbf{C}, \mathbf{v}')$ with $\widetilde{C} \in \mathbf{C}$. Let $\widetilde{\mathsf{ptx}}$, with $\widetilde{C} \in \widetilde{\mathsf{ptx}}.\mathsf{in}$, be the pre-transaction returned by $\textsc{Send}$. Then we must have $(*)$ $\widetilde{\mathsf{ptx}}.\mathsf{chg} \not\subseteq \mathsf{tx}^*.\mathsf{out}$, as otherwise, during the final call to $\textsc{Ledger}$, Hon would have been updated to $\mathsf{Hon} := (\mathsf{Hon} - \widetilde{\mathsf{ptx}}.\mathsf{in}) \,\|\, \widetilde{\mathsf{ptx}}.\mathsf{chg}$, thereby removing $\widetilde{C}$ from Hon, which contradicts the definition of $\widetilde{C}$, as no coin can ever be re-added to Hon. We define $\tilde{\imath}$ as the index of the first coin in $\widetilde{\mathsf{ptx}}.\mathsf{chg}$ that is not in $\mathsf{tx}^*.\mathsf{out}$, which by $(*)$ exists.

Having now defined a coin $\widetilde{C}$, which uniquely exists for every winning execution of Game_1 and $\tilde{\imath}$, which uniquely exists if the adversary won in line (II) and queried $\mathbf{C} \ni \widetilde{C}$ to $\textsc{Send}$, we define Game_2. Let $h_{\mathcal{A}}$ and $n_{\mathcal{A}}$ be upper bounds on the number of coins created during the execution and on the number of change coins in a pre-transaction. Game_2 is defined like Game_1, except that at the beginning, Game_2 samples $\tilde{\imath} \leftarrow_{\$} [n_{\mathcal{A}}]$ and guesses $\widetilde{C}$ among all (at most $h_{\mathcal{A}}$) produced coins and returns **false** in case the adversary lost *or* the guess was not correct. Game_2 for the cash system MW is depicted in Figure 18. Since the guess is uniform and perfectly hidden from the adversary, we have:

$$\mathsf{Adv}^{\mathrm{Game}_2}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \geq \frac{1}{h_{\mathcal{A}} \cdot n_{\mathcal{A}}} \cdot \mathsf{Adv}^{\mathrm{Game}_1}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \ . \tag{18}$$

Game_3. In Game_3 we introduce two modifications in how queries to $\textsc{Send}$ are handled (Figure 18, including the boxes). First, the game returns **false** if the adversary makes a call $\textsc{Send}(\mathbf{C}, \mathbf{v}')$ with $\widetilde{C} \in \mathbf{C}$ and $|\mathbf{v}'| < \tilde{\imath}$. If there is a call $\mathsf{ptx} \leftarrow \textsc{Send}(\mathbf{C}, \mathbf{v}')$ with $\widetilde{C} \in \mathbf{C}$ and $|\mathbf{v}'| \geq \tilde{\imath}$, then the game defines $\overline{C} := \mathsf{ptx}.\mathsf{chg}[\tilde{\imath}]$. Once $\overline{C}$ has been defined, the game returns **false** if the adversary makes a call $\textsc{Send}(\mathbf{C}, \mathbf{v}')$ with $\overline{C} \in \mathbf{C}$.

We claim that Game_3 returns **true** with exactly the same probability as Game_2, so that

$$\mathsf{Adv}^{\mathrm{Game}_3}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) = \mathsf{Adv}^{\mathrm{Game}_2}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \ . \tag{19}$$

This is because the modifications only make the game return **false** *earlier*. To see this, we consider an execution of Game_2 which returns **true** in line (I) or (II) and show that the corresponding execution of Game_3 also returns **true**. Consider first an execution which returns **true** in line (I). Since $\widetilde{C} \in \mathsf{Chg}$ and $\mathsf{Chg} \cap \mathsf{Hon} = (\,)$ (see argument after (17)), $\widetilde{C}$ is never queried to $\textsc{Send}$, so Game_3 cannot return **false** in line (IV); moreover, $\overline{C}$ is never defined, so Game_3 cannot return **false** in line (III) either. Hence, Game_3 also returns **true** in line (I). Consider now an execution which returns **true** in line (II) during a query $\textsc{Ledger}(\mathsf{tx}^*)$. We consider several cases depending on the value of $\widetilde{\mathsf{ptx}}$ at the end of the execution:

```
Games $\mathrm{Game}_2$ and $\boxed{\mathrm{Game}_3}$

$\tilde{c} \leftarrow\!\!\$\ [h_{\mathcal{A}}]$ ; $\tilde{\imath} \leftarrow\!\!\$\ [n_{\mathcal{A}}]$ ; $c := 0$ ;
$\widetilde{C} := \bot$ ; $\widetilde{\mathsf{ptx}} := \bot$ ; $\boxed{\overline{C} := \bot}$
$(\mathsf{pp}, \Lambda) \leftarrow \mathsf{MW.Setup}(1^\lambda, v_{\max})$
$\mathsf{Chg}, \mathsf{Hon}, \mathsf{Val}, \mathsf{Key}, \mathsf{Ptx} := (\,)$
$\mathcal{A}^{\textsc{Mint},\textsc{Send},\textsc{Receive},\textsc{Ledger}}(\mathsf{pp}, \Lambda)$
return false

Aux function $\mathsf{Store}(\mathbf{C}, \mathbf{v}, \mathbf{k})$

$\mathsf{Val}(\mathbf{C}) := \mathbf{v}$ ; $\mathsf{Key}(\mathbf{C}) := \mathbf{k}$

$\mathsf{MW.Coin}(\mathsf{pp}, \mathbf{v})$

for $i = 1 \ldots |\mathbf{v}|$ do
  $c := c + 1$ // increase coin ctr
  $k_i \leftarrow\!\!\$\ \mathcal{R}_{\mathsf{cp}}$ ; $C_i := \mathsf{COM.Cmt}(\mathsf{cp}, v_i, k_i)$
  $\pi_i \leftarrow \mathsf{\Pi.Prv}(\mathsf{crs}, (\mathsf{cp}, C_i), (v_i, k_i))$
  if $c = \tilde{c}$ then $\widetilde{C} := C_i$
return $\big((C_i)_{i=1}^{|\mathbf{v}|}, (k_i)_{i=1}^{|\mathbf{v}|}, (\pi_i)_{i=1}^{|\mathbf{v}|}\big)$

Oracle $\textsc{Send}(\mathbf{C}, \mathbf{v}')$

if $\mathbf{C} \not\subseteq \mathsf{Hon}$ or $\bigcup_{\mathsf{ptx} \in \mathsf{Ptx}} \mathsf{ptx.in} \cap \mathbf{C} \neq (\,)$
  return $\bot$
$(\mathsf{ptx}, \mathbf{k}') \leftarrow \mathsf{MW.Send}\big(\mathsf{pp}, \mathbf{C}, \mathsf{Val}(\mathbf{C}), \mathsf{Key}(\mathbf{C}), \mathbf{v}'\big)$
if $\widetilde{C} \in \mathbf{C}$ then $\widetilde{\mathsf{ptx}} := \mathsf{ptx}$
                    // this can only happen once
$\mathsf{Store}(\mathsf{ptx.chg}, \mathbf{v}', \mathbf{k}')$ ; $\mathsf{Ptx} := \mathsf{Ptx} \,\|\, (\mathsf{ptx})$
$\mathsf{Chg} := \mathsf{Chg} \,\|\, (\mathsf{ptx.chg})$ ; return ptx
```

```
$\mathsf{MW.Send}(\mathsf{pp}, (\mathbf{C}, \mathbf{v}, \mathbf{k}), \mathbf{v}')$

$\rho := \sum \mathbf{v} - \sum \mathbf{v}'$
if $\neg\,\mathsf{Cons}(\mathsf{pp}, \mathbf{C}, \mathbf{v}, \mathbf{k})$ then return $\bot$
if $\mathbf{v} \,\|\, \mathbf{v}' \,\|\, \rho \not\subseteq [0, v_{\max}]^*$ then return $\bot$
┌
│ if $\overline{C} \in \mathbf{C}$ then stop game returning false        (III)
│ if $\widetilde{C} \in \mathbf{C}$ // ptx being created will be $\widetilde{\mathsf{ptx}}$
│   if $|\mathbf{v}'| < \tilde{\imath}$ then stop game returning false (IV)
│   $(\hat{\mathbf{C}}, \hat{\mathbf{k}}, \hat{\boldsymbol{\pi}}) \leftarrow \mathsf{MW.Coin}(\mathsf{pp}, \mathbf{v}' \,\|\, \rho)$
│   $\overline{C} := \hat{\mathbf{C}}[\tilde{\imath}]$
└
$\boxed{\text{else}}$ $(\hat{\mathbf{C}}, \hat{\mathbf{k}}, \hat{\boldsymbol{\pi}}) \leftarrow \mathsf{MW.Coin}(\mathsf{pp}, \mathbf{v}' \,\|\, \rho)$
$\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sp}, \sum \hat{\mathbf{k}} - \sum \mathbf{k}, \varepsilon)$
$\mathsf{tx} := \Big(0, \mathbf{C}, \hat{\mathbf{C}}, \big(\hat{\boldsymbol{\pi}}, \sum \hat{\mathbf{C}} - \sum \mathbf{C}, \sigma\big)\Big)$
return $\Big(\mathsf{ptx} := (\mathsf{tx}, \rho, \hat{k}_{|\mathbf{v}'|+1}), (\hat{k}_i)_{i=1}^{|\mathbf{v}'|}\Big)$

Oracle $\textsc{Ledger}(\mathsf{tx})$

$\Lambda' \leftarrow \mathsf{MW.Ldgr}(\mathsf{pp}, \Lambda, \mathsf{tx})$
if $\Lambda' = \bot$ then return $\bot$ else $\Lambda := \Lambda'$
for all $\mathsf{ptx} \in \mathsf{Ptx}$ do
  if $\mathsf{ptx.chg} \subseteq \mathsf{tx.out}$
    $\mathsf{Ptx} := \mathsf{Ptx} - (\mathsf{ptx})$
    $\mathsf{Hon} := (\mathsf{Hon} - \mathsf{ptx.in}) \,\|\, \mathsf{ptx.chg}$
    $\mathsf{Chg} := \mathsf{Chg} - \mathsf{ptx.chg}$
if $\mathsf{tx.in} \cap \mathsf{Chg} \neq (\,)$
  if $\widetilde{C} = \mathsf{tx.in}[j]$ with $j = \min\{i \mid \mathsf{tx.in}[i] \in \mathsf{Chg}\}$
    stop game returning true                                        (I)
  else stop game returning false
if $\mathsf{tx.in} \cap \mathsf{Hon} \neq (\,)$
  if $\widetilde{C} = \mathsf{tx.in}[j]$ with $j = \min\{i \mid \mathsf{tx.in}[i] \in \mathsf{Hon}\}$
  $\wedge\, (\widetilde{\mathsf{ptx}} = \bot \vee \tilde{\imath} = \min\{i \mid \widetilde{\mathsf{ptx}}.\mathsf{chg}[i] \notin \mathsf{tx.out}\})$
    stop game returning true                                        (II)
  else stop game returning false
return $\Lambda$
```

Fig. 18. Game_2 and Game_3 for CASH := MW, where oracles Mint and Receive are defined as in Figure 17 and MW.Setup, MW.Mint, MW.Ldgr, and MW.Rcv are defined as in Figure 11. Changes from Game_1 to Game_2 are highlighted and changes from Game_2 to Game_3 are $\boxed{\text{boxed}}$.

1. $\widetilde{\mathsf{ptx}} = \bot$: the analysis is like in case (I): $\widetilde{C}$ is never queried to Send (as otherwise $\widetilde{\mathsf{ptx}}$ would get defined) and $\overline{C}$ is never defined, hence Game_3 cannot return **false** in line (III) or (IV).
2. $\widetilde{\mathsf{ptx}} \neq \bot$, that is, $\widetilde{C}$ has been queried to Send: First, since Game_2 arriving in line (II) implies that the $\tilde{\imath}$-th change output of $\widetilde{\mathsf{ptx}}$ exists, Game_3 cannot have returned **false** in line (IV). Hence, $\overline{C}$ was created and added to Chg. We first argue that $\overline{C}$ must still be in Chg at the end of the game: $\overline{C}$ can only be removed from Chg during an oracle call Ledger(tx) with $\widetilde{\mathsf{ptx}}.\mathsf{chg} \subseteq \mathsf{tx.out}$. However, at the same time such a call removes $\widetilde{\mathsf{ptx}}.\mathsf{in}$ from Hon. This

however contradicts that Game_2 returns **true** in line (II), which implies $\widetilde{C} \in \mathsf{Hon}$ when by construction we have $\widetilde{C} \in \widetilde{\mathsf{ptx}}.\mathsf{in}$. We conclude that $\overline{C} \in \mathsf{Chg}$ when the game returns. Hence $\overline{C}$ cannot have been queried to $\textsc{Send}$ (for which it must be in Hon) and therefore Game_3 does not return **false** in line (III).

Game_4. We define Game_4, a slight variation of Game_3 where at the beginning in $\mathsf{MW.Setup}$, instead of setting $\mathsf{crs} \leftarrow \Pi.\mathsf{Setup}(\mathsf{mp}, v_{\max})$, it sets $(\mathsf{crs}, \tau) \leftarrow \Pi.\mathsf{SimSetup}(\mathsf{mp}, v_{\max})$. When $\widetilde{C}$ is created, it sets $\pi_i \leftarrow \Pi.\mathsf{SimPrv}(\mathsf{crs}, \tau, (\mathsf{cp}, C_i))$ (instead of running $\Pi.\mathsf{Prv}$ on the witness (v, k)); if $\mathcal{A}$ queries $\textsc{Send}(\mathbf{C}, \mathbf{v}')$ with $\widetilde{C} \in \mathbf{C}$ and there is an $\tilde{\imath}$-th change coin $\overline{C}$, it also simulates the proof for $\overline{C}$.

Game_4 is easily shown to be indistinguishable from Game_3 by constructing the following adversary $\mathcal{B}_{\text{zk}}$ for game $\text{ZK}_{\Pi,\mathsf{R}_{v_{\max}}}$: it receives crs (which is either produced by $\Pi.\mathsf{Setup}$ or by $\Pi.\mathsf{SimSetup}$) and simulates Game_3, querying its oracle $\textsc{SimProve}((\mathsf{cp}, C), (v, k))$ when producing the proof for $C = \widetilde{C}$ or $C = \overline{C}$; $\mathcal{B}_{\text{zk}}$ returns 1 if $\mathcal{A}$ wins Game_3 and 0 otherwise. Since $\mathcal{B}_{\text{zk}}$ perfectly simulates Game_3 or Game_4 depending on the bit of its ZK challenger, we have:

$$\mathsf{Adv}^{\text{Game}_4}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \geq \mathsf{Adv}^{\text{Game}_3}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) - \mathsf{Adv}^{\text{zk}}_{\Pi,\mathsf{R}_{v_{\max}},\mathcal{B}_{\text{zk}}}(\lambda) \; . \tag{20}$$

Game_5. Our final game will be Game_5, which is defined as Game_4, except that if a call $\textsc{Ledger}(\mathsf{tx}^*)$ ends up in line (I) or (II), then instead of immediately returning **true**, Game_5 does the following: let $\mathsf{tx}^* = (s, \mathbf{C}, \hat{\mathbf{C}}, (\hat{\boldsymbol{\pi}}, \mathbf{E}, \sigma))$; let $\mathbf{C}' := \mathbf{C} - (\widetilde{C})$ and for all $\mathbf{C}'$ (for which we have $\mathbf{C}' \subseteq \Lambda.\mathsf{out}$; otherwise $\mathsf{MW.Ldgr}$, and thus $\textsc{Ledger}$, would return $\perp$), it collects the corresponding proofs $\boldsymbol{\pi}'$ in the kernel of the ledger and it runs:

– $(\mathbf{v}, \mathbf{k}) := \Pi.\mathsf{Ext}(\mathsf{crs}, \tau, (\mathsf{cp}, \mathbf{C}'), \boldsymbol{\pi}')$
– $(\hat{\mathbf{v}}, \hat{\mathbf{k}}) := \Pi.\mathsf{Ext}(\mathsf{crs}, \tau, (\mathsf{cp}, \hat{\mathbf{C}}), \hat{\boldsymbol{\pi}})$

If we have $\neg\, \mathsf{R}_{v_{\max}}((\mathsf{cp}, \mathbf{C}' \,\|\, \hat{\mathbf{C}}), (\mathbf{v} \,\|\, \hat{\mathbf{v}}, \mathbf{k} \,\|\, \hat{\mathbf{k}}))$ then Game_5 returns **false**; otherwise, it returns **true**.

We show that Game_5 is indistinguishable from Game_4. To start with, note that since $\mathsf{MW.Ldgr}(\mathsf{pp}, \Lambda, \mathsf{tx}^*)$ did not return $\perp$, both Λ and tx^* are valid and thus the following hold:

– $\Pi.\mathsf{Ver}(\mathsf{crs}, \mathbf{C}', \boldsymbol{\pi}')$ and
– $\Pi.\mathsf{Ver}(\mathsf{crs}, \hat{\mathbf{C}}, \hat{\boldsymbol{\pi}})$.

We construct an adversary $\mathcal{B}_{\text{se}}$ against simulation-extractability of Π, which receives a simulated CRS crs and has access to an oracle $\textsc{SimProve}$, as follows. Adversary $\mathcal{B}_{\text{se}}$ simulates Game_4, using its oracle $\textsc{SimProve}$ (since it does not have the simulation trapdoor) for the simulated proofs for $\widetilde{C}$ and $\overline{C}$; if Game_4 ends up in lines (I) or (II), $\mathcal{B}_{\text{se}}$ returns $(\mathbf{C}' \,\|\, \hat{\mathbf{C}}, \boldsymbol{\pi}' \,\|\, \hat{\boldsymbol{\pi}})$; otherwise $\mathcal{B}_{\text{se}}$ aborts. Since $\mathcal{B}_{\text{se}}$ wins game S-EXT_Π whenever Game_4 returns **true** while Game_5 would not, we have

$$\mathsf{Adv}^{\text{Game}_5}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \geq \mathsf{Adv}^{\text{Game}_4}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) - \mathsf{Adv}^{\text{s-ext}}_{\Pi,\mathsf{R}_{v_{\max}},\mathcal{B}_{\text{se}}}(\lambda) \; . \tag{21}$$

The reduction to EUF-CRO. We now construct an adversary $\mathcal{B}$ against EUF-CRO of $(\mathsf{COM}, \mathsf{SIG})$, which makes a single call to its $\textsc{Sign}'$ oracle, and show that $\mathcal{B}$ breaks EUF-CRO with the same probability as $\mathcal{A}$ wins Game_5. The modified procedures $\mathsf{MW.Coin}$, $\mathsf{MW.MkTx}$ (which uses $\mathsf{MW.Coin}$ and is used by $\textsc{Mint}$ and $\textsc{Receive}$) and $\mathsf{MW.Send}$ are formally defined in Figure 19 ($\mathsf{MW.Coin}$ now has an additional parameter j, which is set to $\perp$ by default).

$\mathcal{B}$ receives input $(\mathsf{cp}, \mathsf{sp}, C^*)$. It computes $(\mathsf{crs}, \tau) \leftarrow \Pi.\mathsf{SimSetup}(\mathsf{mp}, v_{\max})$ and picks random $\tilde{c}$ and $\tilde{\imath}$. It runs $\mathcal{A}$ on $\mathsf{pp} := (\mathsf{cp}, \mathsf{sp}, \mathsf{crs})$ and simulates all of $\mathcal{A}$'s oracles as defined by Game_5 (cf. also Figure 18), except that it *embeds* the challenge C^* into $\widetilde{C}$ and, in case $\widetilde{C}$ is queried to $\textsc{Send}$, it also embeds C^* into the $\tilde{\imath}$-th change coin $\overline{C}$. If there is no $\tilde{\imath}$-th change coin, in particular, if there are *no* change coins, then $\mathcal{B}$ aborts (as Game_5 would return **false** anyway).

By "embedding" C^* into $\widetilde{C}$, we mean that instead of computing $\widetilde{C}$ as $\mathsf{Cmt}(\mathsf{cp}, v, k)$, $\mathcal{B}$ sets $\widetilde{C} := C^* + \mathsf{Cmt}(\mathsf{cp}, v, k)$ and does thus not know $\widetilde{C}$'s actual key $\tilde{k} = k + r^*$, where $C^* = r^*G$.

$\mathcal{B}$ uses the zero-knowledge simulator to produce the range proofs for the two coins $\widetilde{C}$ and $\overline{C}$. Moreover, when $\widetilde{C}$ is created by MINT, SEND or RECEIVE, the transaction containing it as its j-th output must be signed. $\mathcal{B}$ uses its related-key signing oracle SIGN$'$ to do this: letting $(k_i)_i$ and $(\hat{k}_i)_{i\neq j}$ be the keys of the inputs and other outputs of the transaction, $\mathcal{B}$ requires a signature under $\sum_{i\neq j} \hat{k}_i + (k_j + r^*) - \sum_i k_i$; it thus makes a query $\textsc{Sign}'(\sum \hat{\mathbf{k}} - \sum \mathbf{k}, \varepsilon)$ (this is the only time $\mathcal{B}$ makes a query).

Finally, consider the query $\textsc{Send}(\mathbf{C}, \mathbf{v}')$ with $\mathbf{C}[j] = \widetilde{C}$ for some j, which would also require the signing key for $\widetilde{C}$ in order to compute the resulting pre-transaction

$$\widetilde{\mathsf{ptx}} = \big(\mathsf{tx} = (0, \mathbf{C}, \hat{\mathbf{C}}, (\hat{\boldsymbol{\pi}}, \textstyle\sum \hat{\mathbf{C}} - \sum \mathbf{C}, \sigma)), \rho, \hat{k}_{|\mathbf{v}|+1}\big) \ ,$$

where σ is a signature for the verification key $\sum \hat{\mathbf{C}} - \sum \mathbf{C}$. Since $\mathcal{B}$ embedded C^* in $\mathbf{C}$ via $\widetilde{C}$ and in $\hat{\mathbf{C}}$ via $\overline{C}$, the two occurrences cancel out and $\mathcal{B}$ knows the corresponding signing key. More precisely, $\sum \hat{\mathbf{k}} - \sum \mathbf{k}$ is the signing key for $\sum \hat{\mathbf{C}} - \sum \mathbf{C}$, since:

$$\begin{aligned}\textstyle\sum \hat{\mathbf{C}} - \sum \mathbf{C} = {} & \textstyle\sum_{i\neq j} \mathsf{Cmt}(\mathsf{cp}, \hat{v}_i, \hat{k}_i) + (C^* + \mathsf{Cmt}(\mathsf{cp}, v_j, k_j)) \\ & \qquad - \textstyle\sum_{i\neq \tilde{\imath}} \mathsf{Cmt}(\mathsf{cp}, v_i, k_i) - (C^* + \mathsf{Cmt}(\mathsf{cp}, v_{\tilde{\imath}}, k_{\tilde{\imath}})) \\ = {} & \mathsf{Cmt}\big(\mathsf{cp}, 0, \textstyle\sum_i \hat{k}_i - \sum k_i\big) \ .\end{aligned}$$

Note that there can only be one SEND query containing $\widetilde{C}$. The *only* other query which $\mathcal{B}$ cannot answer (as it lacks a necessary coin key) is $\textsc{Send}(\mathbf{C}, \mathbf{v}')$ with $\overline{C} \in \mathbf{C}$. If this happens then $\mathcal{B}$ aborts. Hence, $\mathcal{B}$ perfectly simulates Game_5.

We now show how $\mathcal{B}$ computes a solution for the EUF-CRO challenge whenever $\mathcal{A}$ wins Game_5. Figure 19 specifies $\mathcal{B}$'s simulation of the oracle LEDGER and its behavior in case $\mathcal{A}$ wins Game_5 (via the procedure $\mathsf{Finalize}$). We claim that whenever $\mathsf{Finalize}(\mathsf{tx}^*)$ is called, the following holds:

(i) $\widetilde{C} \in \mathsf{tx}^*.\mathsf{in}$; (ii) $\widetilde{C} \notin \mathsf{tx}^*.\mathsf{out}$; (iii) $\overline{C} \notin \mathsf{tx}^*.\mathsf{in}$; (iv) $\overline{C} \notin \mathsf{tx}^*.\mathsf{out}$.

Property (i) is clearly necessary for $\mathsf{Finalize}(\mathsf{tx}^*)$ to be called. Property (ii) must hold as otherwise $\mathsf{tx}^*.\mathsf{in} \cap \mathsf{tx}^*.\mathsf{out} \neq (\,)$, which implies $\mathsf{Ver}(\mathsf{pp}, \mathsf{tx}^*) = \mathbf{false}$ and hence $\mathsf{MW.Agg}(\mathsf{pp}, \Lambda, \mathsf{tx}^*)$ (and hence $\mathsf{MW.Ldgr}(\mathsf{pp}, \Lambda, \mathsf{tx}^*)$) would return $\perp$. To prove (iii) and (iv), we distinguish two cases. Assume first that $\mathsf{Finalize}(\mathsf{tx}^*)$ is called in line (I). Since $\widetilde{C} \in \mathsf{Chg}$ and $\mathsf{Chg} \cap \mathsf{Hon} = (\,)$ (see argument after (17)), $\widetilde{C}$ has never been queried to SEND, thus $\overline{C}$ has never been defined and (iii) and (iv) trivially hold. Assume now that $\mathsf{Finalize}(\mathsf{tx}^*)$ is called in line (II). If $\widetilde{\mathsf{ptx}} = \perp$ then as before $\overline{C} = \perp$ and (iii) and (iv) trivially hold. If $\widetilde{\mathsf{ptx}} \neq \perp$, then necessarily (by inspection of the code) $\widetilde{\mathsf{ptx}}.\mathsf{chg}[\tilde{\imath}] = \overline{C} \notin \mathsf{tx}^*.\mathsf{out}$ and (iv) holds. It remains to prove (iii). As in the reasoning for Game_3, we have $\overline{C} \in \mathsf{Chg}$. Moreover, we have $\mathsf{tx}^*.\mathsf{in} \cap \mathsf{Chg} = (\,)$, since otherwise $\mathcal{B}$ would have returned in line (I) of Figure 19. Together this implies $\overline{C} \notin \mathsf{tx}^*.\mathsf{in}$.

It is easily seen that all coins in $\Lambda.\mathsf{out}$ have a valid proof in the ledger's kernel (otherwise $\mathsf{MW.Ldgr}$ (and thus LEDGER) would not have included them in $\Lambda.\mathsf{out}$). The reduction can thus use the extractor to obtain the values and keys of the coins in $\mathbf{C} - (\widetilde{C})$: $(v_i)_{i\in[|\mathbf{C}|]\setminus(j)}$ and $(k_i)_{i\in[|\mathbf{C}|]\setminus(j)}$. From the proofs $\boldsymbol{\pi}$ contained in tx^*, it can moreover extract the values and keys of the output coins $\hat{\mathbf{C}}$: $(\hat{v}_i)_{i\in[|\hat{\mathbf{C}}|]}$ and $(\hat{k}_i)_{i\in[|\hat{\mathbf{C}}|]}$. Since $\mathsf{MW.Ver}(\mathsf{pp}, \mathsf{tx}^*)$, we have:

(a) $\mathsf{SIG.Ver}(\mathsf{sp}, \mathbf{E}, \sigma)$

```
Adversary $\mathcal{B}^{\textsc{Sign}'}(\mathsf{cp}, \mathsf{sp}, C^*)$

$\tilde{c} \leftarrow\!\!\$\, [h_{\mathcal{A}}]$ ; $\tilde{\imath} \leftarrow\!\!\$\, [n_{\mathcal{A}}]$ ; $c := 0$
$\widetilde{C} := \bot$ ; $\widetilde{\mathsf{ptx}} := \bot$ ; $\overline{C} := \bot$
$(\mathsf{crs}, \tau) \leftarrow \Pi.\mathsf{SimSetup}(\mathsf{mp}, v_{\max})$
$\Lambda := \big(0, (\,), (\,), ((\,), (\,), \varepsilon)\big)$
$\mathsf{Chg}, \mathsf{Hon}, \mathsf{Val}, \mathsf{Key}, \mathsf{Ptx} := (\,)$
$\mathcal{A}^{\textsc{Mint},\textsc{Send},\textsc{Receive},\textsc{Ledger}}((\mathsf{cp}, \mathsf{sp}, \mathsf{crs}), \Lambda)$
return $\bot$   // if not stopped earlier
```

```
$\mathsf{MW.Coin}(\mathsf{pp}, \mathbf{v}, j)$

for $i = 1 \ldots |\mathbf{v}|$ do
  $c := c + 1$ ; $k_i \leftarrow\!\!\$\, \mathcal{R}_{\mathsf{cp}}$
  if $c = \tilde{c}$ or $i = j$ then
    $C_i := C^* + \mathsf{COM.Cmt}(\mathsf{cp}, v_i, k_i)$   // embed
    $\pi_i \leftarrow \Pi.\mathsf{SimPrv}(\mathsf{crs}, \tau, (\mathsf{cp}, C_i))$   // challenge
  else $C_i := \mathsf{COM.Cmt}(\mathsf{cp}, v_i, k_i)$
    $\pi_i \leftarrow \Pi.\mathsf{Prv}(\mathsf{crs}, (\mathsf{cp}, C_i), (v_i, k_i))$
  if $c = \tilde{c}$ then $\widetilde{C} := C_i$
return $\big( (C_i)_{i=1}^{|\mathbf{v}|}, (k_i)_{i=1}^{|\mathbf{v}|}, (\pi_i)_{i=1}^{|\mathbf{v}|} \big)$
```

```
$\mathsf{MW.MkTx}(\mathsf{pp}, (\mathbf{C}, \mathbf{v}, \mathbf{k}), \hat{\mathbf{v}})$

if $\neg\,\mathsf{Cons}(\mathsf{pp}, \mathbf{C}, \mathbf{v}, \mathbf{k})$ then return $\bot$
$s := \sum \hat{\mathbf{v}} - \sum \mathbf{v}$
if $\mathbf{v} \,\|\, \hat{\mathbf{v}} \not\subseteq [0, v_{\max}]^*$ or $s < 0$
  return $\bot$
$(\hat{\mathbf{C}}, \hat{\mathbf{k}}, \hat{\boldsymbol{\pi}}) \leftarrow \mathsf{MW.Coin}(\mathsf{pp}, \hat{\mathbf{v}})$
if $\widetilde{C} \in \hat{\mathbf{C}}$   // $\widetilde{C}$ created in this tx
  $\textsc{Sign}'(\sum \hat{\mathbf{k}} - \sum \mathbf{k}, \varepsilon)$
else $\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sp}, \sum \hat{\mathbf{k}} - \sum \mathbf{k}, \varepsilon)$
$\mathsf{tx} := \big(s, \mathbf{C}, \hat{\mathbf{C}}, \big(\hat{\boldsymbol{\pi}}, \sum \hat{\mathbf{C}} - \sum \mathbf{C}, \sigma\big)\big)$
return $(\mathsf{tx}, \hat{\mathbf{k}})$
```

```
Procedure $\mathsf{Finalize}(\mathsf{tx})$

$\mathsf{tx} = (s, \mathbf{C}, \hat{\mathbf{C}}, (\hat{\boldsymbol{\pi}}, \mathbf{E}, \sigma))$ ; $\mathbf{C}' := \mathbf{C} - (\widetilde{C})$
let $\boldsymbol{\pi}'$ be the proofs for $\mathbf{C}'$ in $\Lambda$
$(\mathbf{v}, \mathbf{k}) := \Pi.\mathsf{Ext}(\mathsf{crs}, \tau, (\mathsf{cp}, \mathbf{C}'), \boldsymbol{\pi}')$
$(\hat{\mathbf{v}}, \hat{\mathbf{k}}) := \Pi.\mathsf{Ext}(\mathsf{crs}, \tau, (\mathsf{cp}, \hat{\mathbf{C}}), \hat{\boldsymbol{\pi}})$
if $\neg\,\mathsf{R}_{v_{\max}}\big((\mathsf{cp}, \mathbf{C}' \,\|\, \hat{\mathbf{C}}), (\mathbf{v} \,\|\, \hat{\mathbf{v}}, \mathbf{k} \,\|\, \hat{\mathbf{k}})\big)$
  abort
return $(\mathbf{E}, \sigma, \sum \hat{\mathbf{v}} - \sum \mathbf{v} - s, \sum \hat{\mathbf{k}} - \sum \mathbf{k})$
```

```
$\mathsf{MW.Send}(\mathsf{pp}, (\mathbf{C}, \mathbf{v}, \mathbf{k}), \mathbf{v}')$

$\rho := \sum \mathbf{v} - \sum \mathbf{v}'$
if $\neg\,\mathsf{Cons}(\mathsf{pp}, \mathbf{C}, \mathbf{v}, \mathbf{k})$ then return $\bot$
if $\mathbf{v} \,\|\, \mathbf{v}' \,\|\, \rho \not\subseteq [0, v_{\max}]^*$ then return $\bot$
if $\overline{C} \in \mathbf{C}$ then abort
if $\widetilde{C} \in \mathbf{C}$   // $\mathsf{ptx}$ being created will be $\widetilde{\mathsf{ptx}}$
  if $|\mathbf{v}'| < \tilde{\imath}$ then abort
  // embed challenge in $\tilde{\imath}$-th change coin:
  $(\hat{\mathbf{C}}, \hat{\mathbf{k}}, \hat{\boldsymbol{\pi}}) \leftarrow \mathsf{MW.Coin}(\mathsf{pp}, \mathbf{v}' \,\|\, \rho, \tilde{\imath})$
  $\overline{C} := \hat{\mathbf{C}}[\tilde{\imath}]$
else $(\hat{\mathbf{C}}, \hat{\mathbf{k}}, \hat{\boldsymbol{\pi}}) \leftarrow \mathsf{MW.Coin}(\mathsf{pp}, \mathbf{v}' \,\|\, \rho)$
$\sigma \leftarrow \mathsf{SIG.Sign}(\mathsf{sp}, \sum \hat{\mathbf{k}} - \sum \mathbf{k}, \varepsilon)$
$\mathsf{tx} := \big(0, \mathbf{C}, \hat{\mathbf{C}}, \big(\hat{\boldsymbol{\pi}}, \sum \hat{\mathbf{C}} - \sum \mathbf{C}, \sigma\big)\big)$
return $\big(\mathsf{ptx} := (\mathsf{tx}, \rho, \hat{k}_{|\mathbf{v}'|+1}), (\hat{k}_i)_{i=1}^{|\mathbf{v}'|}\big)$
```

```
Oracle $\textsc{Ledger}(\mathsf{tx})$

$\Lambda' \leftarrow \mathsf{MW.Ldgr}(\mathsf{pp}, \Lambda, \mathsf{tx})$
if $\Lambda' = \bot$ then return $\bot$ else $\Lambda := \Lambda'$
for all $\mathsf{ptx} \in \mathsf{Ptx}$ do
  if $\mathsf{ptx.chg} \subseteq \mathsf{tx.out}$
    $\mathsf{Ptx} := \mathsf{Ptx} - (\mathsf{ptx})$
    $\mathsf{Hon} := (\mathsf{Hon} - \mathsf{ptx.in}) \,\|\, \mathsf{ptx.chg}$
    $\mathsf{Chg} := \mathsf{Chg} - \mathsf{ptx.chg}$
if $\mathsf{tx.in} \cap \mathsf{Chg} \neq (\,)$
  if $\widetilde{C} = \mathsf{tx.in}[j]$ with $j = \min\{i \,|\, \mathsf{tx.in}[i] \in \mathsf{Chg}\}$
    then $\mathsf{Finalize}(\mathsf{tx})$ else abort                         (I)
if $\mathsf{tx.in} \cap \mathsf{Hon} \neq (\,)$
  if $\widetilde{C} = \mathsf{tx.in}[j]$ with $j = \min\{i \,|\, \mathsf{tx.in}[i] \in \mathsf{Hon}\}$
      $\wedge\ (\widetilde{\mathsf{ptx}} = \bot \vee \tilde{\imath} = \min\{i \,|\, \widetilde{\mathsf{ptx}}.\mathsf{chg}[i] \notin \mathsf{tx.out}\})$
                                                  // $\overline{C} = \widetilde{\mathsf{ptx}}.\mathsf{chg}[\tilde{\imath}]$
    then $\mathsf{Finalize}(\mathsf{tx})$ else abort                         (II)
return $\Lambda$
```

Fig. 19. Reduction $\mathcal{B}$ simulating Game$_5$ (only showing oracles that differ from Figure 18).

(b) $$\begin{aligned}\sum \mathbf{E} &= \sum \hat{\mathbf{C}} - \sum \mathbf{C} - \mathsf{Cmt}(\mathsf{cp}, s, 0)\\ &= \sum_i \hat{C}_i - \big(\sum_{i \neq j} C_i + C^* + \mathsf{Cmt}(\mathsf{cp}, v_j, k_j)\big) - \mathsf{Cmt}(\mathsf{cp}, s, 0)\\ &= -C^* + \mathsf{Cmt}(\mathsf{cp}, \underbrace{\textstyle\sum_i \hat{v}_i - \sum_i v_i - s}_{=:v}, \underbrace{\textstyle\sum_i \hat{k}_i - \sum_i k_i}_{=:r})\end{aligned}$$

$\mathcal{B}$ thus returns $(\mathbf{E}, \sigma, (v, r))$, which makes it win the game EUF-CRO.

Together this shows that whenever $\mathcal{A}$ wins Game$_5$ then $\mathcal{B}$ wins EUF-CRO, that is

$$\mathsf{Adv}^{\text{euf-cro}}_{\mathsf{COM},\mathsf{SIG},\mathcal{B}}(\lambda) = \mathsf{Adv}^{\text{Game}_5}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \ . \tag{22}$$

The theorem now follows from Eqs. (16) to (22). □

A.4 Proof of Transaction-indistinguishability of MW

Theorem 15 (Transaction indistinguishability (Def. 12)). *Assume that* COM *is a homomorphic hiding commitment scheme,* SIG *a compatible signature scheme, and* Π *is a zero-knowledge proof system. Then the aggregate cash system* MW[COM, SIG, Π] *is transaction-indistinguishable. More precisely, for any* $v_{\max}$ *and any p.p.t. adversary* $\mathcal{A}$ *which makes at most* $q_{\mathcal{A}}$ *queries to its oracle* Tx, *there exist p.p.t. adversaries* $\mathcal{B}_{\text{zk}}$ *and* $\mathcal{B}_{\text{hid}}$ *such that*

$$\mathsf{Adv}^{\text{tx-ind}}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \leq \mathsf{Adv}^{\text{zk}}_{\Pi,\mathsf{R}_{v_{\max}},\mathcal{B}_{\text{zk}}}(\lambda) + q_{\mathcal{A}} \cdot \mathsf{Adv}^{\text{hid}}_{\mathsf{COM},\mathcal{B}_{\text{hid}}}(\lambda) \ .$$

Before proving the theorem, we show a fact that will be useful for the proof. Consider commitment parameters cp for COM and let $v_b, v_b' \in \mathcal{V}_{\mathsf{cp}}$; then the following distributions are all equivalent:

$$\big[r, r' \leftarrow_\$ \mathcal{R}_{\mathsf{cp}} : C := \mathsf{Cmt}(\mathsf{cp}, v_b, r),\ C' := \mathsf{Cmt}(\mathsf{cp}, v_b', r'),\ k := r + r'\big] \ , \tag{23}$$
$$\big[r, r' \leftarrow_\$ \mathcal{R}_{\mathsf{cp}} : C := \mathsf{Cmt}(\mathsf{cp}, v_b, r),\ C' := \mathsf{Cmt}(\mathsf{cp}, v_b + v_b', r + r') - C,\ k := r + r'\big] \ , \tag{24}$$
$$\big[r, k \leftarrow_\$ \mathcal{R}_{\mathsf{cp}} : C := \mathsf{Cmt}(\mathsf{cp}, v_b, r),\ C' := \mathsf{Cmt}(\mathsf{cp}, v_b + v_b', k) - C,\ k\big] \ . \tag{25}$$

(The distribution in) (24) is equivalent to (the one in) (23) since COM is additively homomorphic; (25) is equivalent to (24) since $\mathcal{R}_{\mathsf{cp}}$ is a group and therefore $r + r'$ and k are equally distributed.

Now consider an adversary $\mathcal{A}$ that chooses $v_0, v_0', v_1, v_1' \in \mathcal{V}_{\mathsf{cp}}$ such that $v_0 + v_0' = v_1 + v_1'$ and receives a tuple (C, C', k) as defined in (23) for a random $b \leftarrow_\$ \{0, 1\}$ and $\mathcal{A}$ has to guess b. Then if COM is hiding, $\mathcal{A}$'s advantage will be negligible; intuitively, this is because (23) is distributed as (25) and in the latter the only thing depending on b is C (since $v_0 + v_0' = v_1 + v_1'$). More formally, one can construct an adversary $\mathcal{B}$ for the HID$_{\mathsf{COM}}$ game which queries its challenge oracle on (v_0, v_1) to get C and simulates distribution (23) that $\mathcal{A}$ expects using (25).

We generalize this indistinguishability notion to vectors of values of length more than two as follows:

Definition 23 (HID-PRR). *Let game* HID-PRR *be as defined in Figure 20. A commitment scheme* COM *is* hiding under partially revealed randomness *if for any p.p.t. adversary* $\mathcal{A}$:

$$\mathsf{Adv}^{\text{hid-prr}}_{\mathsf{COM},\mathcal{A}}(\lambda) := \left|2 \cdot \Pr\left[\text{HID-PRR}_{\mathsf{COM},\mathcal{A}}(\lambda) = \mathbf{true}\right] - 1\right| = \mathsf{negl}(\lambda) \ .$$

The following is proved by generalizing reduction $\mathcal{B}$ sketched above.

Lemma 24. *Any hiding homomorphic commitment scheme* COM *is also HID-PRR. More precisely, for any p.p.t. adversary* $\mathcal{A}$*, there exists a p.p.t. adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\text{hid-prr}}_{\mathsf{COM},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\text{hid}}_{\mathsf{COM},\mathcal{B}}(\lambda) \ .$$

```
Game HID-PRR_{COM,A}(λ)
    $b \leftarrow_\$ \{0,1\}$ ; $\mathsf{mp} \leftarrow \mathsf{MainSetup}(1^\lambda)$
    $\mathsf{cp} \leftarrow \mathsf{COM.Setup}(\mathsf{mp})$
    $(\mathbf{v}_0, \mathbf{v}_1, \mathsf{state}) \leftarrow \mathcal{A}(\mathsf{cp})$
    if $|\mathbf{v}_0| \neq |\mathbf{v}_1|$ or $\sum \mathbf{v}_0 \neq \sum \mathbf{v}_1$
        return $bool \leftarrow_\$ \{\mathbf{true}, \mathbf{false}\}$
    $\mathbf{r} \leftarrow_\$ (\mathcal{R}_{\mathsf{cp}})^{|\mathbf{v}_0|}$ ; $\mathbf{C} := \mathsf{COM.Cmt}(\mathsf{cp}, \mathbf{v}_b, \mathbf{r})$
    $b' \leftarrow \mathcal{A}(\mathbf{C}, \sum \mathbf{r}, \mathsf{state})$
    return $b = b'$

Adversary $\mathcal{B}(\mathsf{cp})$
    $(\mathbf{v}_0, \mathbf{v}_1, \mathsf{state}) \leftarrow \mathcal{A}(\mathsf{cp})$
    if $|\mathbf{v}_0| \neq |\mathbf{v}_1|$ or $\sum \mathbf{v}_0 \neq \sum \mathbf{v}_1$
        return $b' \leftarrow_\$ \{0,1\}$
    for $i = 1 \ldots |\mathbf{v}_0| - 1$ do
    // oracle query in game HID_COM:
        $C_i \leftarrow \textsc{Commit}(v_0[i], v_1[i])$
    $\mathbf{C}' := (C_i)_{i=1}^{|\mathbf{v}_0|-1}$ ; $k \leftarrow_\$ \mathcal{R}_{\mathsf{cp}}$
    $b' \leftarrow \mathcal{A}\big(\mathbf{C}' \,\|\, \big(\mathsf{Cmt}(\mathsf{cp}, \sum \mathbf{v}_0, k) - \sum \mathbf{C}'\big), k), \mathsf{state}\big)$
    return $b'$
```

Fig. 20. Game HID-PRR and adversary $\mathcal{B}$ for Lemma 24.

Proof. Fix values $\mathbf{v}_b$ and let $\mathbf{v}'_b$ denote the first $|\mathbf{v}_b| - 1$ components of $\mathbf{v}_b$. Then, as above, the following distributions can be shown to be the same:

$$\big[\mathbf{r} \leftarrow_\$ (\mathcal{R}_{\mathsf{cp}})^{|\mathbf{v}_b|} : \mathbf{C} := \mathsf{Cmt}(\mathsf{cp}, \mathbf{v}_b, \mathbf{r}),\ k := \textstyle\sum \mathbf{r}\big] ,$$
$$\big[\mathbf{r}' \leftarrow_\$ (\mathcal{R}_{\mathsf{cp}})^{|\mathbf{v}_b|-1}, k \leftarrow_\$ \mathcal{R}_{\mathsf{cp}} : (\mathbf{C}' := \mathsf{Cmt}(\mathsf{cp}, \mathbf{v}'_b, \mathbf{r}')) \,\|\, \big(\mathsf{Cmt}(\mathsf{cp}, \textstyle\sum \mathbf{v}_b, k) - \sum \mathbf{C}'\big),\ k\big] .$$

Let $\mathcal{A}$ be an adversary for game HID-PRR$_{\mathsf{COM}}$. We construct an adversary $\mathcal{B}$ for game HID$_{\mathsf{COM}}$ as shown in Figure 20. By the observation above, game HID-PRR$_{\mathsf{COM}}$ is perfectly simulated by $\mathcal{B}$. Hence, $\mathcal{B}$ wins HID$_{\mathsf{COM}}$ with the same advantage as $\mathcal{A}$ distinguishes $b = 0$ from $b = 1$ in game HID-PRR$_{\mathsf{COM}}$. □

Before proving Theorem 15, we state another simple fact.

Lemma 25. *Let* COM *be an (additively) homomorphic commitment scheme, parameters* cp *be output by* COM.Setup, $v \in \mathcal{V}_{\mathsf{cp}}$ *and* $r \in \mathcal{R}_{\mathsf{cp}}$. *Then* $\mathsf{Cmt}(\mathsf{cp}, -v, -r) = -\mathsf{Cmt}(\mathsf{cp}, v, r)$.

Proof. We have $\mathsf{Cmt}(\mathsf{cp}, 0, 0) = 0$, since $\mathsf{Cmt}(\mathsf{cp}, v, r) = \mathsf{Cmt}(\mathsf{cp}, 0 + v, 0 + r) = \mathsf{Cmt}(\mathsf{cp}, 0, 0) + \mathsf{Cmt}(\mathsf{cp}, v, r)$. This implies $0 = \mathsf{Cmt}(\mathsf{cp}, v - v, r - r) = \mathsf{Cmt}(\mathsf{cp}, v, r) + \mathsf{Cmt}(\mathsf{cp}, -v, -r)$, which shows the statement. □

Proof (of Theorem 15). We start with instantiating CASH in Figure 9 with MW and write out game TX-IND$_{\mathsf{MW}}$ in Figure 21, where the boxes should be ignored. (We have simplified the description by omitting checks for inputs that are created correctly by the experiment.) We next define a game Game$_1$ (Figure 21, including the boxes) where all range proofs are simulated. It is straightforward to construct an adversary $\mathcal{B}_{\mathrm{zk}}$ so that

$$\mathsf{Adv}^{\mathrm{Game}_1}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) \geq \mathsf{Adv}^{\text{tx-ind}}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) - \mathsf{Adv}^{\mathrm{zk}}_{\Pi,\mathsf{R}_{v_{\max}},\mathcal{B}_{\mathrm{zk}}}(\lambda) .$$

By Lemma 24, in order to prove the theorem, it suffices to construct $\mathcal{B}$ such that

$$\mathsf{Adv}^{\text{hid-prr}}_{\mathsf{COM},\mathcal{B}}(\lambda) \geq \tfrac{1}{q_{\mathcal{A}}} \cdot \mathsf{Adv}^{\mathrm{Game}_1}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max}) . \tag{26}$$

Consider a Tx-oracle query $(\mathbf{v}_0, \mathbf{v}'_0, \mathbf{v}''_0), (\mathbf{v}_1, \mathbf{v}'_1, \mathbf{v}''_1)$ with $\mathbf{v}_0, \mathbf{v}'_0, \mathbf{v}''_0, \mathbf{v}_1, \mathbf{v}'_1, \mathbf{v}''_1 \in [0, v_{\max}]^*$ and

$$|\mathbf{v}_0| = |\mathbf{v}_1|$$
$$|\mathbf{v}'_0| + |\mathbf{v}''_0| = |\mathbf{v}'_1| + |\mathbf{v}''_1| \qquad \textstyle\sum \mathbf{v}'_0 + \sum \mathbf{v}''_0 - \sum \mathbf{v}_0 = 0 = \sum \mathbf{v}'_1 + \sum \mathbf{v}''_1 - \sum \mathbf{v}_1 . \tag{27}$$

```
$\text{IND-TX}_{\mathsf{MW},\mathcal{A}}(\lambda, v_{\max})$ $\boxed{\text{Game}_1}$
mp $\leftarrow$ MainSetup$(1^\lambda)$
cp $\leftarrow$ COM.Setup(mp)
sp $\leftarrow$ SIG.Setup(mp)
crs $\leftarrow$ Π.Setup(mp, $v_{\max}$)
$\boxed{(\mathsf{crs}, \tau) \leftarrow \Pi.\mathsf{SimSetup}(\mathsf{mp}, v_{\max})}$
$b \leftarrow_\$ \{0, 1\}$
$b' \leftarrow \mathcal{A}^{\mathrm{TX}}(\mathsf{pp})$
return $b = b'$
```

```
MW.Coin((cp, sp, crs), $\mathbf{v}$)
for $i = 1 \ldots |\mathbf{v}|$ do
    $k_i \leftarrow_\$ \mathcal{R}_{\mathsf{cp}}$
    $C_i :=$ COM.Cmt(cp, $v_i, k_i$)
    $\pi_i \leftarrow$ Π.Prv(crs, (cp, $C_i$), ($v_i, k_i$))
    $\boxed{\pi_i \leftarrow \Pi.\mathsf{SimPrv}(\mathsf{crs}, \tau, (\mathsf{cp}, C_i))}$
return $\big((C_i)_{i=1}^{|\mathbf{v}|}, (k_i)_{i=1}^{|\mathbf{v}|}, (\pi_i)_{i=1}^{|\mathbf{v}|}\big)$
```

```
MW.MkTx(pp, ($\mathbf{C}, \mathbf{v}, \mathbf{k}$), $\hat{\mathbf{v}}$)
$s := \sum \hat{\mathbf{v}} - \sum \mathbf{v}$
$(\hat{\mathbf{C}}, \hat{\mathbf{k}}, \hat{\boldsymbol{\pi}}) \leftarrow$ MW.Coin(pp, $\hat{\mathbf{v}}$)
$E := \sum \hat{\mathbf{C}} - \sum \mathbf{C}$ − COM.Cmt(cp, $s$, 0)
$\sigma \leftarrow$ SIG.Sign(sp, $\sum \hat{\mathbf{k}} - \sum \mathbf{k}, \varepsilon$)
return (tx := ($s, \mathbf{C}, \hat{\mathbf{C}}, (\hat{\boldsymbol{\pi}}, E, \sigma)$), $\hat{\mathbf{k}}$)
```

```
Oracle Tx(($\mathbf{v}_0, \mathbf{v}'_0, \mathbf{v}''_0$), ($\mathbf{v}_1, \mathbf{v}'_1, \mathbf{v}''_1$))
if not ($\mathbf{v}_0, \mathbf{v}'_0, \mathbf{v}''_0, \mathbf{v}_1, \mathbf{v}'_1, \mathbf{v}''_1 \in [0, v_{\max}]^*$)
    return ⊥
if $|\mathbf{v}_0| \neq |\mathbf{v}_1|$ or $|\mathbf{v}'_0| + |\mathbf{v}''_0| \neq |\mathbf{v}'_1| + |\mathbf{v}''_1|$
    return ⊥
if $\sum \mathbf{v}_0 \neq \sum(\mathbf{v}'_0 \,\|\, \mathbf{v}''_0)$ or $\sum \mathbf{v}_1 \neq \sum(\mathbf{v}'_1 \,\|\, \mathbf{v}''_1)$
    return ⊥
// (tx, $\mathbf{k}$) $\leftarrow$ Mint(pp, $\mathbf{v}_b$)
(tx, $\mathbf{k}$) $\leftarrow$ MW.MkTx(pp, ((), (), ()), $\mathbf{v}_b$)
($\sum \mathbf{v}_b$, (), $\mathbf{C}, K$) := tx
// (ptx, $\mathbf{k}'$) $\leftarrow$ Send(pp, (tx.out, $\mathbf{v}_b, \mathbf{k}$), $\mathbf{v}'_b$)
$\rho := \sum \mathbf{v}_b - \sum \mathbf{v}'_b$
(tx′, $\mathbf{k}' \,\|\, k^*$) $\leftarrow$ MW.MkTx(pp, ($\mathbf{C}, \mathbf{v}_b, \mathbf{k}$), $\mathbf{v}'_b \,\|\, \rho$)
$\big(0, \mathbf{C}, \mathbf{C}' \,\|\, C^*, (\boldsymbol{\pi}' \,\|\, \pi^*, E', \sigma')\big)$ := tx′
        // where $E' = \sum \mathbf{C}' + C^* - \sum \mathbf{C} - 0$
        // and $\sigma' \leftarrow$ Sign$\big(\mathsf{sp}, \sum \mathbf{k}' + k^* - \sum \mathbf{k}, \varepsilon\big)$
// (tx*, $\mathbf{k}''$) $\leftarrow$ Rcv(pp, ptx, $\mathbf{v}''_b$) with ptx := (tx′, $\rho, k^*$)
        // note that $\rho = \sum \mathbf{v}''_b$
(tx″, $\mathbf{k}''$) $\leftarrow$ MW.MkTx(pp, ($C^*, \rho, k^*$), $\mathbf{v}''_b$)
$\big(0, (C^*), \mathbf{C}'', (\boldsymbol{\pi}'', E'', \sigma'')\big)$ := tx″
        // where $E'' = \sum \mathbf{C}'' - C^* - 0$
        // and $\sigma'' \leftarrow$ Sign$\big(\mathsf{sp}, \sum \mathbf{k}'' - k^*, \varepsilon\big)$
tx* $\leftarrow$ MW.Agg(pp, tx′, tx″)
        // if tx* $\neq$ ⊥:
$\big(0, \mathbf{C}, \mathbf{C}' \,\|\, \mathbf{C}'', (\boldsymbol{\pi}' \,\|\, \boldsymbol{\pi}'', (E', E''), \sigma^*)\big)$ := tx*
    // where $\sigma^* \leftarrow$ SIG.Agg$\big(\mathsf{sp}, ((E'), \sigma'), ((E''), \sigma'')\big)$
return tx*
```

Fig. 21. Transaction-indistinguishability game for MW and $\boxed{\text{hybrid game}}$.

We will show that the response of Tx is independent of b. (If one of the conditions in (27) does not hold, then Tx returns ⊥, independently of b). By Figure 21, the oracle reply is of the form

$$\mathsf{tx}^* = \Big(0, \mathbf{C}, \mathbf{C}' \,\|\, \mathbf{C}'', \Big(\boldsymbol{\pi}' \,\|\, \boldsymbol{\pi}'', (E', E''), \mathsf{SIG.Agg}\big(\mathsf{sp}, ((E'), \sigma'), ((E''), \sigma'')\big)\Big)\Big) \quad \text{with} \tag{28}$$

$$\sigma' \leftarrow \mathsf{Sign}\big(\mathsf{sp}, \textstyle\sum \mathbf{k}' + k^* - \sum \mathbf{k}, \varepsilon\big)$$

$$\sigma'' \leftarrow \mathsf{Sign}\big(\mathsf{sp}, \textstyle\sum \mathbf{k}'' - k^*, \varepsilon\big)$$

$$\begin{aligned} E' &= \textstyle\sum \mathbf{C}' + C^* - \sum \mathbf{C} = \mathsf{Cmt}(\mathsf{cp}, \sum \mathbf{v}'_b + \rho - \sum \mathbf{v}_b, \sum \mathbf{k}' + k^* - \sum \mathbf{k}) \\ &= \mathsf{Cmt}(\mathsf{cp}, 0, \textstyle\sum \mathbf{k}' + k^* - \sum \mathbf{k}) \end{aligned} \tag{29}$$

$$E'' = \textstyle\sum \mathbf{C}'' - C^* = \mathsf{Cmt}(\mathsf{cp}, \sum \mathbf{v}''_b - \rho, \sum \mathbf{k}'' - k^*) = \mathsf{Cmt}(\mathsf{cp}, 0, \sum \mathbf{k}'' - k^*)\ , \tag{30}$$

where the last equations in (29) and (30) follow since COM is homomorphic and $\sum \mathbf{v}_b - \sum \mathbf{v}'_b = \rho = \sum \mathbf{v}''_b$, by the definition of ρ and (27).

On the other hand, for vectors $\mathbf{v}_b, \mathbf{v}'_b, \mathbf{v}''_b$, for which (27) holds, we have that the distribution

$$\Big[\mathbf{k} \,\|\, \mathbf{k}' \,\|\, \mathbf{k}'' \leftarrow_\$ \mathcal{R}_{\mathsf{cp}}^{|\mathbf{v}_0|+|\mathbf{v}'_0|+|\mathbf{v}''_0|} : \mathbf{C} \,\|\, \mathbf{C}' \,\|\, \mathbf{C}'',\ \bar{k} := \textstyle\sum \mathbf{k}' + \sum \mathbf{k}'' - \sum \mathbf{k}\Big] \tag{31}$$

with $\mathbf{C} \coloneqq \mathsf{Cmt}(\mathsf{cp}, \mathbf{v}_b, \mathbf{k})$, $\mathbf{C}' \coloneqq \mathsf{Cmt}(\mathsf{cp}, \mathbf{v}'_b, \mathbf{k}')$ and $\mathbf{C}'' \coloneqq \mathsf{Cmt}(\mathsf{cp}, \mathbf{v}''_b, \mathbf{k}'')$ are indistinguishable for $b = 0$ or $b = 1$. This follows by applying Lemma 24 to vectors $-\mathbf{v}_b \,\|\, \mathbf{v}'_b \,\|\, \mathbf{v}''_b$, for $b = 0, 1$, and then using Lemma 25. From (31) we get that

$$\Big[\mathbf{k} \,\|\, \mathbf{k}' \,\|\, \mathbf{k}'' \,\|\, (k^*) \leftarrow\!\!\$\, \mathcal{R}_{\mathsf{cp}}^{|\mathbf{v}_0|+|\mathbf{v}'_0|+|\mathbf{v}''_0|+1} : \mathbf{C} \,\|\, \mathbf{C}' \,\|\, \mathbf{C}'',\, r' \coloneqq \textstyle\sum \mathbf{k}' + k^* - \sum \mathbf{k},\, r'' \coloneqq \sum \mathbf{k}'' - k^*\Big] \quad (32)$$

is also indistinguishable for $b = 0$ and $b = 1$: We could construct a reduction $\mathcal{B}_{(32)}$ that, given $(\mathbf{C} \,\|\, \mathbf{C}' \,\|\, \mathbf{C}'', \overline{k})$ distributed as in (31), samples $r \leftarrow \mathcal{R}_{\mathsf{cp}}$ and runs a distinguisher for (32) on $(\mathbf{C} \,\|\, \mathbf{C}' \,\|\, \mathbf{C}'', r, \overline{k} - r)$; the latter is distributed correctly, since r' and r'' are uniform conditioned on $r' + r'' = \overline{k}$.

Since oracle Tx does not reveal C^* and thus k^* is perfectly hidden, (32) implies that tx^* in (28) is also indistinguishable for $b = 0$ or $b = 1$: we can construct an adversary $\mathcal{B}_{(28)}$ which, given an output of the form (32), computes a tuple of the form (28) by setting:

$$\begin{array}{lll} \boldsymbol{\pi}' \leftarrow \Pi.\mathsf{SimPrv}(\mathsf{crs}, \tau, (\mathsf{cp}, \mathbf{C}')) & \sigma' \leftarrow \mathsf{Sign}(\mathsf{sp}, r', \varepsilon) & E' = \mathsf{Cmt}(\mathsf{cp}, 0, r') \\ \boldsymbol{\pi}'' \leftarrow \Pi.\mathsf{SimPrv}(\mathsf{crs}, \tau, (\mathsf{cp}, \mathbf{C}'')) & \sigma'' \leftarrow \mathsf{Sign}(\mathsf{sp}, r'', \varepsilon) & E'' = \mathsf{Cmt}(\mathsf{cp}, 0, r'')\ , \end{array}$$

where we additionally used that $|\mathbf{C}_0| = |\mathbf{C}_1|$ and $|(\mathbf{C}'_0 \,\|\, \mathbf{C}''_0)| = |(\mathbf{C}'_1 \,\|\, \mathbf{C}''_1)|$, as implied by (27). All oracle replies are thus distinguishable with advantage at most $\mathsf{Adv}^{\text{hid-prr}}_{\mathsf{COM},\mathcal{B}}(\lambda)$, where $\mathcal{B}$ combines adversaries $\mathcal{B}_{(32)}$ and $\mathcal{B}_{(28)}$, defined above. This shows (26) and thus the theorem. □

A.5 A Generalized Forking Lemma

Let $\mathcal{A}$ be a randomized algorithm which on input inp and $\mathbf{h} = (h_1, \ldots, h_{q_h}) \in (\mathbb{Z}_p)^{q_h}$ returns a tuple $(\mathsf{outp}, \mathbf{j}, \varPhi)$ where outp is some main output,[14] $\mathbf{j}$ is an ordered list of integers in $[1, q_h]$ of size at most N, and $\varPhi = (\phi_j)_{j\in\mathbf{j}}$ is a list of so-called side outputs. We say that $\mathcal{A}$ "accepts" if $\mathbf{j} \neq (\,)$ and we let $\mathsf{acc}(\mathcal{A})$ denote the accepting probability of $\mathcal{A}$, defined as

$$\Pr[\mathsf{inp} \leftarrow \mathsf{IG}, \mathbf{h} \leftarrow\!\!\$\, (\mathbb{Z}_p)^{q_h}, \omega \leftarrow\!\!\$\, \varOmega_{\mathcal{A}}, (\mathsf{outp}, \mathbf{j}, \varPhi) \leftarrow \mathcal{A}(\mathsf{inp}, \mathbf{h}; \omega) : \mathbf{j} \neq (\,)]\ ,$$

where IG is a randomized algorithm called input generator and $\varOmega_{\mathcal{A}}$ is the space of random coins for $\mathcal{A}$. The generalized forking algorithm associated with $\mathcal{A}$, denoted $\mathsf{GenFork}_{\mathcal{A}}$, is defined in Figure 22. It takes as input inp and returns either a tuple $(\mathsf{outp}, \mathbf{j}, \varPhi, \varPhi')$, where $\varPhi$ and $\varPhi'$ are two lists of side outputs indexed by $\mathbf{j}$, or a distinguished symbol $\perp$ indicating failure. Let $\mathsf{frk}(\mathcal{A})$ be the success probability of $\mathsf{GenFork}_{\mathcal{A}}$ defined as

$$\mathsf{frk}(\mathcal{A}) \coloneqq \Pr[\mathsf{inp} \leftarrow \mathsf{IG} : \perp \not\leftarrow \mathsf{GenFork}_{\mathcal{A}}(\mathsf{inp})]\ .$$

Lemma 26 (Generalized Forking Lemma [BCJ08]). *Let p, q_h, and N be integers and let $\mathcal{A}$ be as above. Assume that $\mathsf{acc}(\mathcal{A}) \geq 8Nq_h/p$. Then $\mathsf{frk}(\mathcal{A}) \geq \mathsf{acc}(\mathcal{A})/8$. Moreover, $\mathsf{GenFork}_{\mathcal{A}}$ runs in time at most*

$$\frac{8N^2 q_h}{\mathsf{acc}(\mathcal{A})} \cdot \ln\left(\frac{8N}{\mathsf{acc}(\mathcal{A})}\right) \cdot t_{\mathcal{A}}\ ,$$

where $t_{\mathcal{A}}$ is an upper bound on the running time of $\mathcal{A}$.

[14] Note that this main output is empty in [BCJ08]. This does not modify the lemma.

```
Algorithm $\mathsf{GenFork}_{\mathcal{A}}(\mathsf{inp})$

$\omega \leftarrow\!\!\$\ \Omega_{\mathcal{A}}$
$h_1, \dots, h_{q_h} \leftarrow\!\!\$\ \mathbb{Z}_p;\ \mathbf{h} := (h_1, \dots, h_{q_h})$
$(\mathsf{outp}, \mathbf{j}, \Phi) \leftarrow \mathcal{A}(\mathsf{inp}, \mathbf{h}; \omega)$      (I)
if $\mathbf{j} = (\,)$ then return $\perp$
$(j_1, \dots, j_n) := \mathbf{j}$      // $n \le N$ and $j_1 \le \dots \le j_n$
$\Phi' := (\,);\ k_{\max} := 8nq_h/\mathsf{acc}(\mathcal{A}) \cdot \ln(8n/\mathsf{acc}(\mathcal{A}))$
for $i = 1, \dots, n$ do      (II)
    $\mathsf{succ}_i := 0;\ k_i := 0$
    while $\mathsf{succ}_i = 0$ and $k_i \le k_{\max}$ do
        $k_i := k_i + 1$
        $h'_{j_i}, \dots, h'_{q_h} \leftarrow\!\!\$\ \mathbb{Z}_p;\ \mathbf{h}' := (h_1, \dots, h_{j_i - 1}, h'_{j_i}, \dots, h'_{q_h})$
        $(\mathsf{outp}', \mathbf{j}', (\phi'_j)_{j \in \mathbf{j}'}) \leftarrow \mathcal{A}(\mathsf{inp}, \mathbf{h}'; \omega)$
        if $\mathbf{j}' \ne (\,)$ and $j_i \in \mathbf{j}'$ and $h'_{j_i} \ne h_{j_i}$
            $\Phi' := \Phi' \,\|\, (\phi'_{j_i});\ \mathsf{succ}_i := 1$
if $\bigwedge_{i=1}^{n} (\mathsf{succ}_i = 1)$ then return $(\mathsf{outp}, \mathbf{j}, \Phi, \Phi')$
else return $\perp$
```

Fig. 22. The generalized forking algorithm $\mathsf{GenFork}_{\mathcal{A}}$ associated with $\mathcal{A}$.

A.6 Proofs of Security for Pedersen-Schnorr

Lemma 16. *The pair* $(\mathsf{PDS}, \mathsf{SCH})$ *is EUF-NZO-secure in the random oracle model under the DL assumption. More precisely, for any p.p.t. adversary* $\mathcal{A}$ *making at most* q_h *random oracle queries and returning a forgery for a list of size at most* N*, there exists a p.p.t. adversary* $\mathcal{B}$ *running in time at most* $8N^2 q_h/\delta_{\mathcal{A}} \cdot \ln(8N/\delta_{\mathcal{A}}) \cdot t_{\mathcal{A}}$*, where* $\delta_{\mathcal{A}} = \mathsf{Adv}^{\text{euf-nzo}}_{\mathsf{PDS},\mathsf{SCH},\mathcal{A}}(\lambda)$ *and* $t_{\mathcal{A}}$ *is the running time of* $\mathcal{A}$*, such that*

$$\mathsf{Adv}^{\text{euf-nzo}}_{\mathsf{PDS},\mathsf{SCH},\mathcal{A}}(\lambda) \le 8 \cdot \mathsf{Adv}^{\text{dl}}_{\mathsf{GrGen},\mathcal{B}}(\lambda) \ .$$

Proof. Let $\mathcal{A}$ be an adversary against the EUF-NZO-security of $(\mathsf{PDS}, \mathsf{SCH})$ and let $\delta_{\mathcal{A}} := \mathsf{Adv}^{\text{euf-nzo}}_{\mathsf{PDS},\mathsf{SCH},\mathcal{A}}(\lambda)$. We first define a wrapper $\mathcal{A}'$ that can be used in the generalized forking lemma. Algorithm $\mathcal{A}'$ takes as input $\mathsf{inp} = (\mathsf{cp}, \mathsf{sp})$ and a tuple $(h_1, \dots, h_{q_h}) \in (\mathbb{Z}_p)^{q_h}$. It picks random coins ω for $\mathcal{A}$ and runs $\mathcal{A}(\mathsf{cp}, \mathsf{sp}; \omega)$. It answers the i-th random oracle query made by $\mathcal{A}$ with h_i. If $\mathcal{A}$ fails, then $\mathcal{A}'$ returns $(\varepsilon, (\,), (\,))$. Otherwise, $\mathcal{A}$ returns $(\mathbf{L}, \boldsymbol{\sigma}, (v, r))$, where $\mathbf{L} = ((X_i, m_i))_{i=1}^n$ and $\boldsymbol{\sigma} = ((R_i, s_i))_{i=1}^n$, such that the following holds:

$$\sum_{i=1}^{n} X_i = vH + rG \ \text{ with } \ v \ne 0 \tag{33}$$

$$s_i G = R_i + \mathcal{H}(X_i, R_i, m_i) X_i \ \text{ for } \ 1 \le i \le n. \tag{34}$$

We assume *wlog* that $\mathcal{A}$ never repeats a query and that it made all queries $\mathcal{H}(X_i, R_i, m_i)$, $1 \le i \le n$, before returning its output (as otherwise it is valid with only negligible probability). For $i \in [1, n]$, let j_i be the (necessarily unique) index such that $\mathcal{H}(X_i, R_i, m_i) = h_{j_i}$ and assume that $j_1 \le \dots \le j_n$ (this is *wlog* since this can be achieved by simple reordering $\mathbf{L}$ and $\boldsymbol{\sigma}$). For $i \in [1, n]$, $\mathcal{A}'$ defines $\phi_{j_i} := (h_{j_i}, X_i, R_i, s_i)$ and returns $(\mathsf{outp}, \mathbf{j}, \Phi)$ where $\mathsf{outp} = (v, r)$, $\mathbf{j} = (j_i)_{i=1}^n$, and $\Phi = (\phi_j)_{j \in \mathbf{j}}$. Note that $\mathsf{acc}(\mathcal{A}') = \delta_{\mathcal{A}}$ and that $\mathcal{A}'$ runs in time similar to $\mathcal{A}$.

From $\mathcal{A}'$, we construct an adversary $\mathcal{B}$ against the DL problem as follows. On input the group description $\Gamma = (p, \mathbb{G}, G)$ and a group element H for which it must solve the DL problem, $\mathcal{B}$

lets $\mathsf{cp} \coloneqq (\varGamma, H)$ and $\mathsf{sp} \coloneqq (\varGamma, \mathcal{H})$, where $\mathcal{H}$ is a dummy interface for the random oracle, and runs $\mathsf{GenFork}_{\mathcal{A}'}(\mathsf{cp}, \mathsf{sp})$. By Lemma 26, with probability at least $\mathsf{acc}(\mathcal{A}')/8$, $\mathsf{GenFork}_{\mathcal{A}'}$ returns a tuple $(\mathsf{outp}, \mathbf{j}, \varPhi, \varPhi')$, where $\mathsf{outp} = (v, r)$, $\mathbf{j} = (j_i)_{i=1}^n$, $\varPhi = ((h_{j_i}, X_i, R_i, s_i))_{i=1}^n$, and $\varPhi' = ((h'_{j_i}, X'_i, R'_i, s'_i))_{i=1}^n$. Since outp and $\varPhi$ result from the main execution of $\mathcal{A}'$ at line (I) of Figure 22, Equation (33) holds, while Equation (34) implies that

$$s_i G = R_i + h_{j_i} X_i \ \text{ for } \ 1 \le i \le n. \tag{35}$$

Moreover, since each tuple $(h'_{j_i}, X'_i, R'_i, s'_i)$ was obtained from a valid forgery, one has

$$s'_i G = R'_i + h'_{j_i} X'_i \ \text{ for } \ 1 \le i \le n. \tag{36}$$

We now argue that for $1 \le i \le n$, $X_i = X'_i$ and $R_i = R'_i$. This follows by inspection of $\mathsf{GenFork}_{\mathcal{A}'}$: in the i-th iteration of the **for** loop at line (II) of Figure 22, the repeated executions of $\mathcal{A}'$ (and hence $\mathcal{A}$) share the same random tape, the same input, and the same random oracle answers $(h_1, \ldots, h_{j_i - 1})$, so that the i-th random oracle query is the same in all executions. This implies in particular that $X_i = X'_i$ and $R_i = R'_i$.[15]

Hence, for $1 \le i \le n$, $\mathcal{B}$ can compute the discrete logarithm of X_i from Equation (35) and Equation (36) as $x_i = (s_i - s'_i)/(h_{j_i} - h'_{j_i}) \bmod p$, where $h_{j_i} \neq h'_{j_i}$ by construction of $\mathsf{GenFork}_{\mathcal{A}'}$. From Equation (33), the discrete logarithm of H can be computed as $(\sum_{i=1}^n x_i - r)/v \bmod p$.

$\mathcal{B}$ is succeeds exactly when $\mathsf{GenFork}_{\mathcal{A}'}$ does, hence $\mathcal{B}$'s advantage is at least $\delta_{\mathcal{A}}/8$. Moreover, $\mathcal{B}$ runs in time similar to $\mathsf{GenFork}_{\mathcal{A}'}$, which by Lemma 26 is at most $8N^2 q_h/\delta_{\mathcal{A}} \cdot \ln(8N/\delta_{\mathcal{A}}) \cdot t_{\mathcal{A}}$. This concludes the proof. □

Lemma 17. *The pair* $(\mathsf{PDS}, \mathsf{SCH})$ *is EUF-CRO-secure in the random oracle model under the DL assumption. More precisely, for any p.p.t. adversary* $\mathcal{A}$ *making at most* q_h *random oracle queries and* q_s *signature queries, returning a forgery for a list of size at most* N*, and such that* $\delta_{\mathcal{A}} = \mathsf{Adv}^{\text{euf-cro}}_{\mathsf{PDS},\mathsf{SCH},\mathcal{A}}(\lambda) \ge 2q_s/p$*, there exists a p.p.t. adversary* $\mathcal{B}$ *running in time at most* $16N^2(q_h + q_s)/\delta_{\mathcal{A}} \cdot \ln(16N/\delta_{\mathcal{A}}) \cdot t_{\mathcal{A}}$*, where* $t_{\mathcal{A}}$ *is the running time of* $\mathcal{A}$*, such that*

$$\mathsf{Adv}^{\text{euf-cro}}_{\mathsf{PDS},\mathsf{SCH},\mathcal{A}}(\lambda) \le 8 \cdot \mathsf{Adv}^{\text{dl}}_{\mathsf{GrGen},\mathcal{B}}(\lambda) + \frac{q_s + 8}{p} \, .$$

Proof. Let $\mathcal{A}$ be an adversary against the EUF-CRO-security of $(\mathsf{PDS}, \mathsf{SCH})$ and let $\delta_{\mathcal{A}} \coloneqq \mathsf{Adv}^{\text{euf-cro}}_{\mathsf{PDS},\mathsf{SCH},\mathcal{A}}(\lambda)$. We first define a wrapper $\mathcal{A}'$ that can be used in the generalized forking lemma. Algorithm $\mathcal{A}'$ takes as input $\mathsf{inp} = (\mathsf{cp}, \mathsf{sp}, C^*)$ and a tuple $(h_1, \ldots, h_{q_h + q_s}) \in (\mathbb{Z}_p)^{q_h + q_s}$. It picks random coins ω for $\mathcal{A}$ and runs $\mathcal{A}(\mathsf{cp}, \mathsf{sp}, C^*; \omega)$. In order to simulate $\mathcal{H}$ and the signature oracle, $\mathcal{A}'$ initializes a counter $\mathsf{ctr} \coloneqq 0$ and an empty table T and proceeds as follows:

- Simulation of $\mathcal{H}$: on input (X, R, m), if $\mathsf{T}(X, R, m)$ is undefined, then $\mathcal{A}'$ increments ctr and sets $\mathsf{T}(X, R, m) \coloneqq h_{\mathsf{ctr}}$; then, it returns $\mathsf{T}(X, R, m)$.
- Simulation of $\textsc{Sign}'$: on input (a, m), $\mathcal{A}'$ proceeds as follows. Let $X \coloneqq C^* + aG$, for which $\mathcal{A}'$ must return a valid forgery on message m. $\mathcal{A}'$ increments ctr, lets $c \coloneqq h_{\mathsf{ctr}}$, draws $s \leftarrow\!\!\$\, \mathbb{Z}_p$, and lets $R \coloneqq sG - cX$. If $\mathsf{T}(X, R, m)$ is defined, then $\mathcal{A}'$ aborts and returns $(\varepsilon, (\,), (\,))$. Otherwise, $\mathcal{A}'$ defines $\mathsf{T}(X, R, m) \coloneqq c$ and returns $\sigma \coloneqq (R, s)$.

If $\mathcal{A}$ fails, then $\mathcal{A}'$ returns $(\varepsilon, (\,), (\,))$. Otherwise, $\mathcal{A}$ returns $(\mathbf{L}, \boldsymbol{\sigma}, (v, r))$, where $\mathbf{L} = ((X_i, m_i))_{i=1}^n$ and $\boldsymbol{\sigma} = ((R_i, s_i))_{i=1}^n$, such that the following holds:

$$\sum_{i=1}^n X_i = vH + rG - C^* \tag{37}$$

$$s_i G = R_i + \mathcal{H}(X_i, R_i, m_i) X_i \ \text{ for } \ 1 \le i \le n. \tag{38}$$

[15] Note that this reasoning requires the random oracle calls to include the public key.

We assume *wlog* that $\mathcal{A}$ never repeats a query and that all values $\mathsf{T}(X_i, R_i, m_i)$, $1 \le i \le n$, are defined when $\mathcal{A}$ returns its output (as otherwise it is valid with only negligible probability). Note that some values $\mathsf{T}(X_i, R_i, m_i)$ might have been defined during a query to the $\textsc{Sign}'$ oracle, i.e., there was some query $(R_i, s_i) \leftarrow \textsc{Sign}'(a_i, m_i)$ such that $X_i = C^* + a_i G$. To simplify the notation, assume *wlog* (this can be achieved by reordering $\mathbf{L}$ and $\boldsymbol{\sigma}$) that there exists $t \in [0, n]$ such that

- for $1 \le i \le t$, there have been queries $(R_i, s_i) \leftarrow \textsc{Sign}'(a_i, m_i)$ with
$$X_i = C^* + a_i G \; ; \tag{39}$$
- for $t+1 \le i \le n$, for all queries $(R, s) \leftarrow \textsc{Sign}'(a, m_i)$, either $X_i \neq C^* + aG$ or $R_i \neq R$.

(Note that it could be the case that $t = n$.) For $i \in [t+1, n]$, let j_i be the (necessarily unique) index such that $\mathsf{T}(X_i, R_i, m_i) = h_{j_i}$ and assume that $j_{t+1} \le \cdots \le j_n$ (again this is *wlog* since this can be achieved by simple reordering $\mathbf{L}$ and $\boldsymbol{\sigma}$). For $i \in [t+1, n]$, $\mathcal{A}'$ defines $\phi_{j_i} := (h_{j_i}, X_i, R_i, s_i)$ and returns $(\mathsf{outp}, \mathbf{j}, \varPhi)$ where $\mathsf{outp} = ((v, r), (a_1, \ldots, a_t))$, $\mathbf{j} = (j_i)_{i=t+1}^n$, and $\varPhi = (\phi_j)_{j \in \mathbf{j}}$.

Unless $\mathcal{A}'$ aborts, the random oracle and the signing oracle are perfectly simulated by $\mathcal{A}'$, so that
$$\mathsf{acc}(\mathcal{A}') \ge \delta_{\mathcal{A}} - q_s/p \; , \tag{40}$$
where q_s/p accounts for the probability that $\mathcal{A}'$ aborts during a signature query. Moreover, $\mathcal{A}'$ runs in time similar to $\mathcal{A}$.

From $\mathcal{A}'$, we construct an adversary $\mathcal{B}$ against the DL problem as follows. On input the group description $\varGamma = (p, \mathbb{G}, G)$ and a group elements H for which it must solve the DL problem, $\mathcal{B}$ lets $\mathsf{cp} := (\varGamma, H)$ and $\mathsf{sp} := (\varGamma, \mathcal{H})$, where $\mathcal{H}$ is a dummy interface for the random oracle, draws $v', r' \leftarrow_\$ \mathbb{Z}_p$, lets $C^* := v'H + r'G$ and runs $\mathsf{GenFork}_{\mathcal{A}'}(\mathsf{cp}, \mathsf{sp}, C^*)$. By Lemma 26, with probability at least $\mathsf{acc}(\mathcal{A}')/8$, $\mathsf{GenFork}_{\mathcal{A}'}$ returns a tuple $(\mathsf{outp}, \mathbf{j}, \varPhi, \varPhi')$, where $\mathsf{outp} = ((v, r), (a_1, \ldots, a_t))$, $\mathbf{j} = (j_i)_{i=t+1}^n$, $\varPhi = ((h_{j_i}, X_i, R_i, s_i))_{i=t+1}^n$, and $\varPhi' = ((h'_{j_i}, X'_i, R'_i, s'_i))_{i=t+1}^n$. Since outp and $\varPhi$ result from the main execution of $\mathcal{A}'$ at line (I) of Figure 22, Equation (37) holds, while Equation (38) implies that
$$s_i G = R_i + h_{j_i} X_i \;\text{ for }\; t+1 \le i \le n. \tag{41}$$
Moreover, since each tuple $(h'_{j_i}, X'_i, R'_i, s'_i)$ was obtained from a valid forgery, one has
$$s'_i G = R'_i + h'_{j_i} X'_i \;\text{ for }\; t+1 \le i \le n. \tag{42}$$

We now argue that for $t+1 \le i \le n$, $X_i = X'_i$ and $R_i = R'_i$. This follows by inspection of $\mathsf{GenFork}_{\mathcal{A}'}$: in the i-th iteration of the **for** loop at line (II) of Figure 22, the repeated executions of $\mathcal{A}'$ (and hence $\mathcal{A}$) share the same random tape, the same input, and the same random oracle answers $(h_1, \ldots, h_{j_i - 1})$, so that the i-th random oracle query is the same in all executions. This implies in particular that $X_i = X'_i$ and $R_i = R'_i$.

Hence, for $t+1 \le i \le n$, $\mathcal{B}$ can compute the discrete logarithm of X_i from Equation (41) and Equation (42) as $x_i = (s_i - s'_i)/(h_{j_i} - h'_{j_i}) \bmod p$, where $h_{j_i} \neq h'_{j_i}$ by construction of $\mathsf{GenFork}_{\mathcal{A}'}$. Combining this with Equation (37) and Equation (39), one has
$$tC^* + (\textstyle\sum_{i=1}^t a_i)G + (\sum_{i=t+1}^n x_i)G = vH + rG - C^* \; .$$
Since $C^* = v'H + r'G$, it follows that
$$\underbrace{(v - (t+1)v')}_{v''} H = \underbrace{(\textstyle\sum_{i=1}^t a_i + (\sum_{i=t+1}^n x_i - r + (t+1)r')}_{r''} G \; .$$
If $v - (t+1)v' = 0$, then $\mathcal{B}$ aborts. Otherwise, it can compute the discrete log of H as $r''/v'' \bmod p$.

$\mathcal{B}$ succeeds if $\mathsf{GenFork}_{\mathcal{A}'}$ does and $v' \neq -v/(t+1) \bmod p$. Since $\mathcal{A}'$'s view is independent from v', the latter happens with probability at most $1/p$. Hence,

$$\mathsf{Adv}^{\mathrm{dl}}_{\mathsf{GrGen},\mathcal{B}}(\lambda) \geq \frac{\mathsf{acc}(\mathcal{A}')}{8} - \frac{1}{p} \overset{(40)}{\geq} \frac{\delta_{\mathcal{A}} - q_s/p}{8} - \frac{1}{p} \, .$$

Moreover, $\mathcal{B}$ runs in time similar to $\mathsf{GenFork}_{\mathcal{A}'}$, which by Lemma 26 is at most

$$\frac{8N^2(q_h+q_s)}{\delta_{\mathcal{A}} - q_s/p} \cdot \ln\left(\frac{8N}{\delta_{\mathcal{A}} - q_s/p}\right) \cdot t_{\mathcal{A}} \leq \frac{16N^2(q_h+q_s)}{\delta_{\mathcal{A}}} \cdot \ln\left(\frac{16N}{\delta_{\mathcal{A}}}\right) \cdot t_{\mathcal{A}} \, ,$$

where we used that $\delta_{\mathcal{A}} \geq 2q_s/p$. This concludes the proof. □

A.7 Proofs of Security for Pedersen-BLS

Lemma 19. *The pair* $(\mathsf{PDS}, \mathsf{BLS})$ *is EUF-NZO-secure in the random oracle model under the CDH assumption. More precisely, for any p.p.t. adversary* $\mathcal{A}$ *making at most* q_h *random oracle queries and returning a forgery for a list of size at most* N*, there exists a p.p.t. adversary* $\mathcal{B}$ *running in time at most* $t_{\mathcal{A}} + (q_h + N + 2)t_M$*, where* $t_{\mathcal{A}}$ *is the running time of* $\mathcal{A}$ *and* t_M *is the time of a scalar multiplication in* $\mathbb{G}$*, such that*

$$\mathsf{Adv}^{\mathrm{cdh}}_{\mathsf{GrGen},\mathcal{B}}(\lambda) = \mathsf{Adv}^{\mathrm{euf\text{-}nzo}}_{\mathsf{PDS},\mathsf{BLS},\mathcal{A}}(\lambda) \, .$$

Proof. Let $\mathcal{A}$ be an adversary against the EUF-NZO-security of $(\mathsf{PDS}, \mathsf{BLS})$. We construct an adversary $\mathcal{B}$ against the CDH problem as follows. On input the bilinear group description $\Gamma' = (p, \mathbb{G}, \mathbb{G}_T, e, G)$ and two group elements H and Y for which it must solve the CDH problem, $\mathcal{B}$ runs $\mathcal{A}$ on input $\mathsf{cp} \coloneqq (\Gamma', H)$ and $\mathsf{sp} \coloneqq (\Gamma', \mathcal{H})$ where the random oracle $\mathcal{H}$ is simulated by $\mathcal{B}$ as follows: $\mathcal{B}$ maintains a table T mapping pairs (X, m) to values α in $\mathbb{Z}_p$; on input (X, m), $\mathcal{B}$ draws $\alpha \leftarrow\!\!\$\, \mathbb{Z}_p$, sets $\mathsf{T}(X, m) \coloneqq \alpha$, and returns $Y + \alpha G$. Eventually, $\mathcal{A}$ returns $(\mathbf{L}, \sigma, (v, r))$, where $\mathbf{L} = ((X_i, m_i))_{i=1}^n$, such that the following holds:

$$\sum_{i=1}^n X_i = vH + rG \text{ with } v \neq 0 \tag{43}$$

$$e(\sigma, G) = \prod_{i=1}^n e(X_i, \mathcal{H}(X_i, m_i)) \, . \tag{44}$$

We assume *wlog* that $\mathcal{A}$ made all queries $\mathcal{H}(X_i, m_i)$, $i \in [1, n]$, before returning its output (as otherwise it is valid with only negligible probability). For $i \in [1, n]$, let $\alpha_i \coloneqq \mathsf{T}(X_i, m_i)$. Then $\mathcal{B}$ returns $Z \coloneqq v^{-1}(\sigma - \sum_{i=1}^n \alpha_i X_i - rY)$ as its guess for the solution of the CDH problem for (H, Y) (note that $v \neq 0$). To see that this solution is valid, note that

$$\begin{aligned}
e(Z, G)^v &= e\left(v^{-1}(\sigma - \textstyle\sum_{i=1}^n \alpha_i X_i - rY), G\right)^v \\
&= e(\sigma, G) \cdot e(-\textstyle\sum_{i=1}^n \alpha_i X_i, G) \cdot e(-rY, G) \\
&\overset{(44)}{=} \textstyle\prod_{i=1}^n e(X_i, \underbrace{\mathcal{H}(X_i, m_i)}_{=Y+\alpha_i G}) \cdot \prod_{i=1}^n e(X_i, \alpha_i G)^{-1} \cdot e(-rG, Y) \\
&= \textstyle\prod_{i=1}^n e(X_i, Y) \cdot e(-rG, Y) \\
&= e(\underbrace{\textstyle\sum_{i=1}^n X_i}_{\overset{(43)}{=} vH + rG} - rG, Y) = e(vH, Y) = e(H, Y)^v \, .
\end{aligned}$$

Since $v \neq 0$, we have $e(Z, G) = e(H, Y)$, meaning Z is a CDH solution to (H, Y).

$\mathcal{B}$ succeeds exactly when $\mathcal{A}$ does, hence $\mathcal{B}$'s advantage is equal to $\mathcal{A}$'s advantage. The running time of $\mathcal{B}$ is upper bounded by the sum of the running time of $\mathcal{A}$, the time to simulate $\mathcal{H}$, which requires at most q_h multiplications, and the time to compute Z, which requires at most $N+2$ multiplications. This concludes the proof. $\square$

Lemma 20. *The pair* $(\mathsf{PDS}, \mathsf{BLS})$ *is EUF-CRO-secure in the random oracle model under the CDH assumption. More precisely, for any p.p.t. adversary* $\mathcal{A}$ *making at most* q_h *random oracle queries and* $q_s = O(1)$ *signature queries and returning a forgery for a list of size at most* N*, there exists a p.p.t. adversary* $\mathcal{B}$ *running in time at most* $t_{\mathcal{A}} + (2q_h + 3q_s + N + 2)t_M$*, where* $t_{\mathcal{A}}$ *is the running time of* $\mathcal{A}$ *and* t_M *is the time of a scalar multiplication in* $\mathbb{G}$*, such that*

$$\mathsf{Adv}^{\mathrm{cdh}}_{\mathsf{GrGen},\mathcal{B}}(\lambda) \geq \frac{1}{4 \cdot (2N)^{q_s}} \cdot \mathsf{Adv}^{\mathrm{euf\text{-}cro}}_{\mathsf{PDS},\mathsf{BLS},\mathcal{A}}(\lambda) \ .$$

Proof. Let $\mathcal{A}$ be an adversary against the EUF-CRO-security of $(\mathsf{PDS}, \mathsf{BLS})$. We construct an adversary $\mathcal{B}$ against the CDH problem as follows. On input the bilinear group description $\Gamma' = (p, \mathbb{G}, \mathbb{G}_T, e, G)$ and two group elements H and Y for which it must solve the CDH problem, $\mathcal{B}$ proceeds as follows. First, $\mathcal{B}$ draws $v', r' \leftarrow\!\!\$\, \mathbb{Z}_p$ and lets

$$C^* := v'H + r'G. \tag{45}$$

Then, it runs $\mathcal{A}$ on input C^*, $\mathsf{cp} := (\Gamma', H)$, and $\mathsf{sp} := (\Gamma', \mathcal{H})$ where the random oracle $\mathcal{H}$ and the related-key signature oracle $\textsc{Sign}'$ are simulated by $\mathcal{B}$ as detailed below.

- Simulation of $\mathcal{H}$: $\mathcal{B}$ maintains a table T mapping pairs (X, m) to pairs (α, β) in $\mathbb{Z}_p \times \{0,1\}$; on input (X, m), if $\mathsf{T}(X, m)$ is undefined, then $\mathcal{B}$ draws $\alpha \leftarrow\!\!\$\, \mathbb{Z}_p$ and $\beta \in \{0,1\}$ such that $\Pr[\beta = 1] = \mu$, where μ will be determined later, and sets $\mathsf{T}(X, m) := (\alpha, \beta)$; then, it returns $\beta Y + \alpha G$.
- Simulation of $\textsc{Sign}'$: on input (a, m), $\mathcal{B}$ runs the simulation for $\mathcal{H}(C^* + aG, m)$, obtaining a corresponding pair (α, β); if $\beta = 1$, then $\mathcal{B}$ aborts; otherwise, $\mathcal{H}(C^* + aG, m) = \alpha G$; then $\mathcal{B}$ computes $\sigma = \alpha(C^* + aG)$ and returns σ; observe that σ is a valid signature for m under public key $C^* + aG$.

Eventually, $\mathcal{A}$ returns $(\mathbf{L}, \sigma, (v, r))$, where $\mathbf{L} = ((X_i, m_i))_{i=1}^n$, such that the following holds:

$$\sum_{i=1}^{n} X_i = vH + rG - C^* \tag{46}$$

$$e(\sigma, G) = \prod_{i=1}^{n} e(X_i, \mathcal{H}(X_i, m_i)) \ . \tag{47}$$

We assume *wlog* that all values $\mathsf{T}(X_i, m_i)$, $i \in [1, n]$, are defined when $\mathcal{A}$ returns its output (as otherwise it is valid with only negligible probability). For $i \in [1, n]$, let $(\alpha_i, \beta_i) := \mathsf{T}(X_i, m_i)$. Note that some values $\mathsf{T}(X_i, m_i)$ might have been defined during a query to the $\textsc{Sign}'$ oracle, i.e., there was some query $\textsc{Sign}'(a_i, m_i)$ such that $X_i = C^* + a_iG$. To simplify the notation, assume *wlog* that there exists $t \in [0, n]$ such that

- for $1 \leq i \leq t$, there have been queries $\textsc{Sign}'(a_i, m_i)$ with

$$X_i = C^* + a_iG \ ; \tag{48}$$

- for $t + 1 \leq i \leq n$, for all queries $\textsc{Sign}'(a, m_i)$, we have $X_i \neq C^* + aG$.

(Note that it could be the case that $t = n$.) Define

$$\begin{aligned} v'' &:= v - (t+1)v' \\ r'' &:= r - (t+1)r' - \textstyle\sum_{i=1}^{t} a_i \; . \end{aligned} \tag{49}$$

If $v'' = 0$ or if $\beta_i = 0$ for some $i \in [t+1, n]$, then $\mathcal{B}$ aborts.

Otherwise, $\mathcal{B}$ returns

$$Z := (v'')^{-1}(\sigma - \textstyle\sum_{i=1}^{n} \alpha_i X_i - r''Y) \tag{50}$$

as its guess for the solution of the CDH problem for (H, Y). Let us prove that this solution is valid. Note that, assuming $\mathcal{B}$ did not abort during Sign$'$ queries, for $1 \le i \le t$ one necessarily has $\beta_i = 0$ and hence $\mathcal{H}(X_i, m_i) = \alpha_i G$. On the other hand, for $t+1 \le i \le n$ we have $\mathcal{H}(X_i, m_i) = Y + \alpha_i G$ and therefore:

$$\begin{aligned} e(Z,G)^{v''} &\overset{(50)}{=} e\left((v'')^{-1}(\sigma - \textstyle\sum_{i=1}^{n} \alpha_i X_i - r''Y), G\right)^{v''} \\ &= e(\sigma, G) \cdot e(-\textstyle\sum_{i=1}^{n} \alpha_i X_i, G) \cdot e(-r''Y, G) \\ &\overset{(47)}{=} \textstyle\prod_{i=1}^{n} e(X_i, \mathcal{H}(X_i, m_i)) \cdot \prod_{i=1}^{n} e(X_i, \alpha_i G)^{-1} \cdot e(-r''G, Y) \\ &= \textstyle\prod_{i=1}^{t} e(X_i, \underbrace{\mathcal{H}(X_i, m_i)}_{=\alpha_i G}) \cdot \prod_{i=t+1}^{n} e(X_i, \underbrace{\mathcal{H}(X_i, m_i)}_{=Y+\alpha_i G}) \\ &\qquad \cdot \textstyle\prod_{i=1}^{n} e(X_i, \alpha_i G)^{-1} \cdot e(-r''G, Y) \\ &= e(\textstyle\sum_{i=t+1}^{n} X_i, Y) e(-r''G, Y) \\ &= e(\textstyle\sum_{i=1}^{n} X_i - \sum_{i=1}^{t} X_i - r''G, Y) \\ &\overset{(46),(48)}{=} e(vH + rG - C^* - \textstyle\sum_{i=1}^{t}(C^* + a_i G) - r''G, Y) \\ &\overset{(45)}{=} e(vH + rG - v'H - r'G - \textstyle\sum_{i=1}^{t}(v'H + r'G + a_i G) - r''G, Y) \\ &= e((v - (t+1)v')H + (r - (t+1)r' - \textstyle\sum_{i=1}^{t} a_i)G - r''G, Y) \\ &\overset{(49)}{=} e(v''H, Y) = E(H, Y)^{v''} . \end{aligned}$$

Since $v'' \neq 0$, we have $e(Z, G) = e(H, Y)$, meaning Z is a CDH solution to (H, Y).

It remains to lower-bound the success probability of $\mathcal{B}$. Consider the following four events:

- A_1: $\mathcal{B}$ does not abort during a Sign$'$ query;
- A_2: $\mathcal{A}$ successfully returns a valid output;
- A_3: $v'' \neq 0$;
- A_4: $\beta_i = 1$ for all $i \in [t+1, n]$.

$\mathcal{B}$ succeeds if those four events happen. We will lower-bound $\Pr\left[\bigwedge_{i=1}^{4} A_i\right]$ using

$$\Pr\left[\textstyle\bigwedge_{i=1}^{4} A_i\right] = \Pr[A_1] \cdot \Pr[A_2 | A_1] \cdot \Pr[A_3 | A_1 \wedge A_2] \cdot \Pr[A_4 | A_1 \wedge A_2 \wedge A_3] \; .$$

We now consider the four probabilities appearing in the product. Let $\delta_{\mathcal{A}} := \mathsf{Adv}^{\text{euf-cro}}_{\mathsf{PDS},\mathsf{BLS},\mathcal{A}}(\lambda)$.

- Recall that when answering a random-oracle query we choose $\beta = 1$ with probability μ. We first claim that $\Pr[A_1] \ge (1 - \mu)^{q_s}$. Assume that $\mathcal{B}$ did not abort up to the $(i-1)$-th signature query, and consider the i-th signature query (a, m) made by $\mathcal{A}$. Let $X := C^* + aG$ and $(\alpha, \beta) := \mathsf{T}(X, m)$. We claim that β is independent from $\mathcal{A}$'s view. This is obvious if $\mathcal{A}$ did not query $\mathcal{H}$ on input (X, m) before the signature query. Otherwise, note that the distribution of $\mathcal{H}(X, m)$ is the same for $\beta = 0$ and $\beta = 1$. Hence, the probability that this query causes $\mathcal{A}$ to abort is exactly μ. By induction, the probability that $\mathcal{A}$ does not abort after ℓ signature queries is at most $(1 - \mu)^{\ell}$. Since $\mathcal{A}$ makes at most q_s queries, the claim follows.

- Next, we claim that $\Pr[A_2|A_1] \geq \delta_{\mathcal{A}}$. Note that, conditioned on $\mathcal{B}$ not aborting during signature queries, the view of $\mathcal{A}$ is identical to the real security experiment: the group elements H and C^* given as input are uniformly random in $\mathbb{G}$, the answers to RO queries are uniformly random in $\mathbb{G}$, and the answers to signature queries are correct. Hence, $\mathcal{A}$ returns a valid output with probability at least $\delta_{\mathcal{A}}$.
- Then, we claim that $\Pr[A_3|A_1 \wedge A_2] \geq (p-1)/p$. Indeed, v' is independent from $\mathcal{A}$'s view, hence the probability that $v' = v(t+1)^{-1} \bmod p$ is exactly $1/p$.
- Finally, we claim that $\Pr[A_4|A_1 \wedge A_2 \wedge A_3] \geq \mu^N$, where N is an upper-bound on the size of $\mathbf{L}$ returned by $\mathcal{A}$. Indeed, for $i \in [t+1, n]$, the fact that pairs (X_i, m_i) did not appear in $\textsc{Sign}'$ queries implies that the view of $\mathcal{A}$ is independent from β_i (even if $\mathcal{A}$ queried $\mathcal{H}(X_i, m_i)$ since the answer is distributed the same way whether $\beta_i = 0$ or $\beta_i = 1$). Hence, the probability that $\beta_i = 1$ for $i \in [t+1, n]$ is at least $\mu^{n-t} \geq \mu^N$.

Combining all the claims above and setting $\mu = 1 - 1/(2N)$, we obtain

$$\begin{aligned}\Pr\left[\textstyle\bigwedge_{i=1}^{4} A_i\right] &\geq (1-\mu)^{q_s} \cdot \delta_{\mathcal{A}} \cdot \frac{p-1}{p} \cdot \mu^N \\ &\geq \frac{1}{(2N)^{q_s}} \cdot \delta_{\mathcal{A}} \cdot \frac{1}{2} \cdot \left(1 - \frac{1}{2N}\right)^N \\ &\geq \frac{\delta_{\mathcal{A}}}{4 \cdot (2N)^{q_s}} \ ,\end{aligned}$$

where we used that $(1 - 1/(2x))^x \geq 1/2$ for all $x \geq 1$.

Finally, the running time of $\mathcal{B}$ is upper-bounded by the sum of the running time of $\mathcal{A}$, the time to simulate $\mathcal{H}$, which requires at most q_h multiplications, the time to simulate $\textsc{Sign}'$, which requires at most $3q_s$ multiplications, and the time to compute Z, which requires at most $N+2$ multiplications. This concludes the proof. □