Non-interactive Mimblewimble transactions, revisited

Georg Fuchsbauer[1] and Michele Orrù[2]

[1] TU Wien, Austria
[2] UC Berkeley, USA
first.last@{tuwien.ac.at,berkeley.edu}

Abstract. Mimblewimble is a cryptocurrency protocol that promises to overcome notorious blockchain scalability issues and provides user privacy. For a long time its wider adoption has been hindered by the lack of non-interactive transactions, that is, payments for which only the sender needs to be online. Yu proposed a way of adding non-interactive transactions to stealth addresses to Mimblewimble, but this turned out to be flawed. Building on Yu and integrating ideas from Burkett, we give a fixed scheme and provide a rigorous security analysis strenghtening the previous security model from Eurocrypt'19. Our protocol is considered for implementation by MimbleWimbleCoin and a variant is now deployed as MimbleWimble Extension Blocks (MWEB) in Litecoin.

1 Introduction

Mimblewimble (MW) is a cryptocurrency protocol that addresses the problem of ever-growing blockchain data that needs to be stored by full nodes in the system. While in all other cryptocurrencies, such as Bitcoin, the full transaction history must be kept for ever,[1] in MW, coins can be deleted after having been spent while *maintaining public verifiability of the ledger*. Instead of growing linearly (like Bitcoin [Nak08], whose blockchain is now > 400 GB)[2], MW-based currencies only need to store the currently existing coins (the *UTXO set*) plus some small data per transaction.

Mimblewimble achieves this by cleverly combining three ideas that were initially envisioned for Bitcoin: (1) *Confidential transactions* [Max15] hide the transacted amount by only including *commitments* to the amounts of the inputs and outputs and giving proofs that the sum of the input values equals that of the output values, showing the transaction is "balanced". Thus, no transaction creates fresh money (apart from *coinbase transactions*, which create money explicitly). Confidential transactions are now implemented e.g. in Monero.[3]

(2) *CoinJoin* [Max13a] is the idea of merging (or *aggregating*) several transactions into one big transaction, in a way that makes it impossible to associate the inputs and outputs of the original transactions. In Bitcoin, this can only be done by having the creators of the transactions interact in order to merge them before being included in the blockchain. In contrast, in MW merging can be done a posteriori without involving the original creators. The result is that in a MW blockchain all transactions are merged into one huge transaction, and there is no information about which inputs led to which outputs.

(3) *Transaction cut-through* [Max13b] is the idea that if a transaction spends an output (which corresponds to a "coin" in the system) txo_1 and creates txo_2, which is then spent by another transaction creating txo_3, then this should be equivalent to a "cut-through" transaction spending txo_1 and creating txo_3. While in Bitcoin this could only be done for "unconfirmed transactions", i.e., ones not yet included in any block, MW allows cut-through to be done after confirmation, which is what enables MW's space-efficiency improvements. As every spent coin is removed, the

[1] An exception are recent proposals building on more speculative technology such as recursive zk-SNARKs; cf. https://minaprotocol.com/lightweight-blockchain
[2] https://www.blockchain.com/charts/blocks-size
[3] https://www.getmonero.org/resources/moneropedia/stealthaddress.html

result is that the huge transaction representing a MW ledger only has inputs that are the coinbase transactions and outputs that are the unspent coins. In addition, this greatly improves user privacy, as the blockchain reveals neither the transacted amounts nor the transaction graph defining how coins are being transferred (in Bitcoin all this can be inferred from the blockchain).

The main shortcoming of Mimblewimble is that the sender and the receiver(s) of a transaction need to compute the transaction in an interactive protocol. It is thus not possible for a sender to simply transfer money to a destination address without any involvement of the owner of that address, which is the standard setting in all major cryptocurrencies.

Mimblewimble (MW) was first proposed by an anonymous author in 2016 [Jed16]. After being initially investigated by Poelstra [Poe16], a formal model and an analysis of MW were provided by Fuchsbauer, Orrù and Seurin (FOS) [FOS19] in 2019. In 2020, Burkett [Bur20] proposed an extension of Mimblewimble supporting non-interactive transactions, later refined by Yu [Yu20]. We will refer to this extension as MW-Yu. In this work, we first asses the security of MW-Yu [Yu20, §2.1] and describe discovered vulnerabilities. We then fix the scheme, also integrating an idea from a more recent proposal from Burkett [Bur21] and give security proofs that our scheme satisfies (an appropriate adaptation of) the rigorous FOS [FOS19] security model for *aggregate cash systems*.

MimbleWimbleCoin plans to implement the protocol by year-end 2022.[4] A variant of our protocol is used in the *MimbleWimble Extension Blocks* (MWEB), which are now supported by Litecoin (one of the top cryptocurrencies with a market capitalization of > 4 billion USD).[5]

The Mimblewimble protocol. MW uses a group $\mathbb{G}$ (which we denote additively) of prime order p with two generators G and H. As with confidential transactions [Max15], a *coin* is a Pedersen commitment $C = \mathsf{Cmt}(v, q) := vH + qG$ to its value v using some randomness $q \in \mathbb{Z}_p$, together with a so-called *range proof* π guaranteeing that v is contained in some interval of admissible values. In MW, knowledge of the opening q of the commitment enables spending the coin. Similarly to Bitcoin, a transaction in MW contains a list of input coins $\boldsymbol{C} \in \mathbb{G}^n$ and output coins $\hat{\boldsymbol{C}} \in \mathbb{G}^{\hat{n}}$, where

$$C_i = v_i H + q_i G \quad \text{for } i \in [n] \quad \text{and} \quad \hat{C}_i = \hat{v}_i H + \hat{q}_i G \quad \text{for } i \in [\hat{n}] \ .$$

Leaving fees and coinbase (a.k.a. *minting*) transactions aside, a transaction is *balanced* if and only if $\sum \hat{\boldsymbol{v}} - \sum \boldsymbol{v} = 0$ (where for a vector $\boldsymbol{v} = (v_1, \ldots, v_n)$, we let $\sum \boldsymbol{v} := \sum_{i=1}^{n} v_i$). For coins as defined above, this is equivalent to

$$\sum \hat{\boldsymbol{C}} - \sum \boldsymbol{C} = \left(\sum \hat{\boldsymbol{q}} - \sum \boldsymbol{q}\right) G \ ,$$

a quantity called the *kernel excess* $E \in \mathbb{G}$ in MW. If the transaction is balanced, then knowledge of the openings $\hat{\boldsymbol{q}}$ and $\boldsymbol{q}$ of all involved coins implies knowledge of the discrete logarithm $\log E$ (to base G) of the excess E. Intuitively, if the producer of the transaction proves knowledge of $\log E$ then, together with the binding property of Pedersen commitments, this should guarantee that the transaction is balanced. In MW this is done by generating a signature σ under public key E, using its discrete logarithm $\sum \hat{\boldsymbol{q}} - \sum \boldsymbol{q}$ as the signing key.

FOS [FOS19] prove that when using Schnorr signatures (and assuming the range proofs are *simulation-extractable*; cf. Section 5), balancedness follows from the hardness of computing discrete logarithms in $\mathbb{G}$ in the random-oracle model. They also show that as long as a user owning a coin C in the ledger keeps the opening private, no one can steal C (i.e., create a transaction that spends C).

Transactions in Mimblewimble can easily be merged non-interactively, in a similar way to CoinJoin [Max13a]. Consider two transactions $\mathsf{tx}_1 = (\hat{\boldsymbol{C}}_1, \boldsymbol{C}_1, \boldsymbol{\pi}_1, E_1, \sigma_1)$ and $\mathsf{tx}_2 = (\hat{\boldsymbol{C}}_2, \boldsymbol{C}_2, \boldsymbol{\pi}_2, E_2, \sigma_2)$. The *aggregate transaction* tx is defined as the concatenation of inputs and outputs, that is, (letting "$\|$" concatenation)

$$\mathsf{tx} = \left(\hat{\boldsymbol{C}}_1 \| \hat{\boldsymbol{C}}_2, \boldsymbol{C}_1 \| \boldsymbol{C}_2, \boldsymbol{\pi}_1 \| \boldsymbol{\pi}_2, E_1 \| E_2, \sigma_1 \| \sigma_2\right) \ .$$

[4] https://www.mwc.mw/mimble-wimble-coin-articles/mimblewimble-non-interactive-transactions-review

[5] https://blog.litecoin.org/litecoin-core-v0-212-release-282f5405aa11 and https://twitter.com/DavidBurkett38/status/1555100039822954496

A transaction $\mathsf{tx} = (\boldsymbol{C}, \hat{\boldsymbol{C}}, \boldsymbol{\pi}, \boldsymbol{E}, \boldsymbol{\sigma})$ is valid if all π's and σ's verify and if

$$\sum \hat{\boldsymbol{C}} - \sum \boldsymbol{C} = \sum \boldsymbol{E} \; . \tag{0}$$

As outputs in one transaction that also appear as inputs in the other cancel out in Equation (0) for tx, they can simply be removed from the input and output list (together with their range proofs), while validity of tx will be maintained. This has been called *transaction cut-through* in the literature [Max13b]. In MW, the ledger is defined as the cut-through of the aggregation of all transactions. Since every spent coin (a.k.a. "transaction output", TXO) is removed by cut-through, the outputs in the ledger are precisely the *unspent TXOs* (UTXO), representing the current state of the ledger. FOS [FOS19] further remarked that if the used signature scheme supports aggregation, then $\sigma_1 \| \sigma_2$ can be replaced by their aggregation to save further space. Then the only trace of a transaction whose outputs have been spent in the ledger is the value E.

Even if the lists of inputs and outputs in an aggregate transaction tx are shuffled, one can still link inputs and outputs come from the same component transaction, as tx will contain an excess value E that equals the difference between the sum of the outputs and inputs of the original transaction. This can be prevented by using *kernel offsets* [Dev20b], where E is replaced by $E + tG$ for a random $t \leftarrow\!\$\; \mathbb{Z}_p$ and t is included in the transaction; the aggregate of two transactions with (E_1, t_1) and (E_2, t_2) will then contain $(E_1 \| E_2, t_1 + t_2)$. A consequence of kernel offsets is that, given an aggregated transaction, nothing can be deduced about which inputs and outputs belonged to the same original transaction. This is implied by our notion of *transaction privacy* (Section 6.6), which we prove our scheme to satisfy.

Our analysis concerns the application layer, and we do not provide network-level privacy guarantees. Network adversaries that observe transactions being broadcast, or traffic analysis in general, constitute an entirely different problem that we consider outside the security of the Mimblewimble protocol per se. In practice, protocols like Dandelion [VFV17] and Tor[6] can help mitigating attacks at the network level. See for instance their adoption in Grin[7].

Non-interactive transactions. Most implementations of MW create new transactions via an *interactive* protocol between sender and receiver in order to produce the Schnorr signature σ under a secret key that depends on the openings of the sender's and the receiver's coins.[8] FOS [FOS19] proposed a transaction protocol, where the sender creates all output coins, so that she can compute σ on her own. She then sends the receiver (through a separate private channel) the transaction along with the secret key associated to one of the output coins. The latter creates a transaction spending this coin, merges it with the received transaction and broadcasts the aggregate transaction to the miners. The downside of this approach is that there is a window of time in which both sender and receiver can spend a coin, which can lead to deniability issues for payments.

In 2020, Yu [Yu20] posted on ePrint an extension of MW for achieving non-interactive transactions by adding *stealth addresses* [vS13,Tod14]. Each user has a *wallet* (or stealth address) $(A, B) \in \mathbb{G}^2$. Given a destination wallet, a sender can derive a one-time address, unique for every transaction, to which she sends the money. These one-time addresses are unlinkable to the wallet they correspond to, yet the owner of the wallet is (the only one) able to derive the secret key for it. In detail, the sender chooses a uniform element $r \leftarrow\!\$\; \mathbb{Z}_p$ and defines the one-time key for stealth address $(A = aG, B = bG)$ as $P = \mathrm{H}(rA) \cdot G + B$, where H is a cryptographic hash function. Being provided $R := rG$, the owner of (A, B) can derive the secret key $\log P$ as $\mathrm{H}(aR) + b$.

[6] https://www.torproject.org/

[7] https://docs.grin.mw/wiki/miscellaneous/dandelion/

[8] For instance, in Grin this is documented in the grin-wallet documentation: https://raw.githubusercontent.com/mimblewimble/grin-wallet/master/doc/transaction/basic-transaction-wf.png.
In Beam, this is documented in the developer documentation: https://github.com/BeamMW/beam/wiki/Cryptographic-primitives.

Integrating stealth addresses into MW is not straightforward, as the currency itself does not provide addresses for sending money. Yu's proposal [Yu20], built on top of Burkett's [Bur20], received multiple feedbacks from the community, and as a consequence it was further updated with notes describing possible attacks and countermeasures. In essence, the idea is to extend an output (C, π) in MW by a one-time key P chosen by the sender for a destination address, as well as an ephemeral key R that allows the receiver to compute the secret key for P. To prevent the value P from being modified (which would mean stealing it from the receiver), a signature ρ on P under signature-verification key R (of which the sender knows the logarithm) is added. An output is thus of the form $(C, \pi, R, \rho, P, \chi)$.

Moreover, the mechanism for letting the receiver derive the secret key for P can also be used to let the receiver obtain the opening of the commitment C (Yu [Yu20, §2.1.1] does this by setting $q = \mathrm{H}(\mathrm{H}(rA)G + B)$). Note that knowing the so-called "view key" (a, B) of stealth address (aG, B), one can derive from R both P (and thus check if the payment is for that address) and q (and thus check if C commits to a given amount).

Usually in cryptocurrencies, when spending an output linked to a key P, the transaction is signed with the secret key of P. In Mimblewimble however, aggregation of transactions should hide which inputs and outputs come from the same component transaction. Yu therefore proposes to use logarithms of the values P in the inputs and the values $\hat{R}$ in the outputs to "authenticate" the spending, similarly to how the openings of the input coins authenticate the output coins via Equation (0) in MW. Namely by proving knowledge of the logarithm of $\sum \hat{\boldsymbol{R}} - \sum \boldsymbol{P}$.

Yu proposes to simply arrange the $\hat{\boldsymbol{R}}$ values so that the above results in the excess E (defined as $\sum \hat{\boldsymbol{C}} - \sum \boldsymbol{C}$). However, this is only possible if one of the outputs $\hat{C}_i$ goes back to the sender (who can choose the corresponding value $\hat{q}_i$ arbitrarily); for all other coins, $\hat{R}_j$ (together with the stealth address) defines $\hat{q}_j$, which defines E, for which $\hat{R}_j$ has to be chosen, which is infeasible. We therefore modify the scheme and introduce a *stealth excess* $X := \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{P}$, under which we add (as for E) a signature to the transaction. Our scheme then supports transactions for which all outputs are sent to destination addresses. At the time of writing, the core proposal in [Yu20, §2.1] is still affected by further issues. The ones known before our analysis are the following:

- As illustrated in [Yu20, §2.9.1], MW-Yu is susceptible to a rogue-key attack [Yu20, §2.9.3]. A fix was also proposed, which requires the addition of one signature per transaction input, namely a signature proving knowledge of the logarithm of P. The security and correctness analysis of this proposed change are not detailed further.
- Mixing NIT with non-NIT transactions, as envisaged in [Yu20], leads to correctness issues within the balance equations.[9] No argument for why the security is preserved is given, especially w.r.t. [Yu20, Eq. ②]. (We do not considered "mixed transactions" in our scheme.)

A major drawback of Yu's and our scheme (and MWEB) is the lack of support for transaction cut-through. Since an output is associated with a value $\hat{R}_i$ and an input is associated with a value P_j, if an output is spent via an input in an aggregated transaction, these values do not cancel out, and removing them would thus change the stealth excess $\sum \hat{\boldsymbol{R}} - \sum \boldsymbol{P}$ (we discuss this in detail in Section 4.3). We note that in practice, nodes would only store and check validity of the most recent excesses and perform cut-trough for coins that have been spent in the past beyond a so-called *horizon* (cf. [Bur21, §4]). Cut-through enables attacks by miners who could change outputs of transactions (and violate *transaction-binding*, see below), which is only relevant for *recent* transactions; once they are in a block beyond the horizon, they are "protected" by the consensus mechanism (a security layer that is outside of our model).

Differences to MWEB. The variant used by Litecoin [Bur21] differs in how exactly the secrets for an output are derived from a stealth address (A, B): In our scheme, from a Diffie-Hellman (DH)

[9] https://forum.mwc.mw/t/non-interactive-transaction-and-stealth-address/32

share $R = rG$, we derive $(k, q) := \mathrm{H}(rA)$, which defines the one-time address as $P := kG + B$ and the coin as $C := vH + qG$. Outputs in [Bur21] have an additional element K_e (in addition to K_s, which corresponds to our R) used as the Diffie-Hellman share (deriving its randomness from $\log K_s$). A symmetric key, derived from the DH-shared key, is then used to encrypt v and derive q. Our variant is arguably simpler, which facilitates our formal analysis.

Our contributions

Scheme. We propose a new protocol for non-interactive transactions, greatly inspired by Yu [Yu20] and using an idea by Burkett [Bur21] to overcome one of the found issues. Our protocol is a variant of what is now already being used by Litecoin in its MimbleWimble Extension Blocks. In Section 4 we discuss further issues that emerged after the publication of [Yu20].

Model. We then analyze our protocol in a strengthening of the model proposed in [FOS19], which did not protect against a malleability attack by miners (discussed below). We only consider non-interactive transactions, which greatly simplifies the security notions. We define security experiments that capture the following attacks:

(i) creating money other than via coinbase transactions *(inflation resistance)*
(ii) spending someone else's output in the ledger *(theft prevention)*
(iii) stealing money from a transaction not yet merged with the ledger *(transaction-binding)*
(iv) breaking privacy by learning anything about the transacted amounts, the destination addresses or the relations of the inputs and outputs in an aggregated transaction *(transaction privacy)*

Inflation resistance and *theft* prevention are straightforward adaptations of the notions from [FOS19, Def. 10 and 11]. *Transaction privacy* is stronger than FOS's privacy notion [FOS19, Def. 12], which only guarantees that amounts are hidden (FOS's scheme does not use kernel offsets, which means one can "disaggregate" transactions). To concisely capture all anonymity, privacy and confidentiality guarantees, we define a simulation-based notion requiring that a transaction can be simulated without knowledge of any information that transactions are supposed to hide.

We introduce *transaction-binding*, a notion that protects users against malicious miners by guaranteeing that no outputs can be removed from a transaction. In particular, after a transaction tx that spends some output *txo* was broadcast, no one can create a transaction tx' that spends *txo* but does *not* include *all* outputs of tx. We note that *theft resistance* [FOS19, Def. 11] (which deals with interactive transactions) only guarantees the following: a user that engages in a protocol with the adversary spending value in $\boldsymbol{C}$ and creating change $\boldsymbol{C}'$ for herself is guaranteed that $\boldsymbol{C}'$ are included in the ledger as soon as any of $\boldsymbol{C}$ is spent.

Proof. We prove the security of our protocol by following the provable-security methodology and giving security reductions of the different security notions to standard computational hardness assumptions in idealized models.

In our security proofs, we assume that the *discrete logarithm problem* is hard in the underlying group $\mathbb{G}$ and for transaction privacy we additionally make the *decisional Diffie-Hellman (DDH) assumption*. Our main building block is a *zero-knowledge* proof system for proving knowledge of discrete logarithms that satisfies *strong simulation-extractability* (defined in Section 5).

We show that the Schnorr signature scheme, and a variant thereof, which we use to improve efficiency of our scheme, satisfy these notions in an idealized model, namely the combination of the *algebraic group model* [FKL18] and the *random oracle model* without making any computational assumptions. Finally, we assume that the used range proofs are merely proofs of knowledge[10] and do not require that they are simulation-extractable as in previous analyses [FOS19].

[10] This is in some sense minimal, since for Pedersen commitments the language (see Section 2) is trivial; cf. Section 6.1.

2 Preliminaries

Let ε denote the empty string, and $[a]$ the set $\{1, \ldots, a\}$ (for some $a \in \mathbb{N}$). We assume the existence of a group $\mathbb{G}$ of prime order p and two "nothing-up-my-sleeve" generators $G, H \in \mathbb{G}$ (that is, the discrete logarithm of H to base G is not known to anyone). The length of the prime p is the security parameter λ. (A typical choice could be the group `Secp256k1` and hence $\lambda = 256$.) For $X \in \mathbb{G}$, we let $\log X$ denote the discrete logarithm of X to base G, that is $\log X = x$ with $X = xG$.

Proofs of possession. We consider a cryptographic hash function which we model as a random oracle and denote by $\mathrm{H}(\cdot)$. We use (key-prefixed) Schnorr signatures (see also Figure 2, page 15), which are unforgeable under the DL assumption in the random oracle model (ROM), and whose security has been extensively studied in the past literature [PS00,Seu12]. Here we interpret Schnorr signatures as (zero-knowledge) proofs of knowledge of the secret key, that is, if $X = xG \in \mathbb{G}$ is the public key then a Schnorr signature is a proof of knowledge of $x = \log X$.

We generalize this to a proof of knowledge of two logarithms that has the same size as a Schnorr signature. Interpreting knowledge of $\log X$ as "possessing" X, we call the proof system PoP for "proof of possession". More formally, PoP is a proof system for the following NP-relation (whose statements contains a part m, sometimes called a "tag"), defined w.r.t. a group description $(p, \mathbb{G}, G)$:

$$\left\{ \big((X, Y, m), (x, y)\big) \,:\, X = xG \wedge Y = yG \wedge m \in \{0,1\}^* \right\} .$$

A proof for a statement $(X, Y, m) \in \mathbb{G}^2 \times \{0,1\}^*$ is computed via $\mathsf{PoP.P}$ using the witness (x, y) by picking a uniform $r \leftarrow\!\!\$\ \mathbb{Z}_p$, defining $R := rG$, computing $(c, d) := \mathrm{H}(X, Y, m, R)$ and returning a proof $(R, s) \in \mathbb{G} \times \mathbb{Z}_p$ with $s := r + c \cdot x + d \cdot y \bmod p$. The verification algorithm $\mathsf{PoP.V}((X, Y, m), (R, s))$ computes $(c, d) := \mathrm{H}(X, Y, m, R)$ and checks whether $sG = R + c \cdot X + d \cdot Y$ (see also Figure 3, page 15).

The system PoP can also be used to prove knowledge of a witness $x \in \mathbb{Z}_p$ for statements $(X = xG, m)$ by using Schnorr signing and verification (defined like above but setting $y := 0$). A proof of possession of X with tag m is thus a Schnorr signature on m under public key X. We use PoP proofs for different *types* of values (proofs ψ for (P, D), proofs σ for excesses (E, X) and proofs ρ for R). We assume that these are "domain-separated", which can easily be achieved by including the type in the tag of the statement. We also assume that random oracles H used elsewhere in the scheme (e.g. to derive the value q) are domain-separated from H used for PoP.

In Section 5 we show that in the *algebraic group model* [FKL18] combined with the ROM, the proof system PoP is a *strongly simulation-sound zero-knowledge proof of knowledge* of logarithms, a property that will be central in the security analysis of our protocol. *Zero-knowledge* means that there exists a simulator (which here controls the random oracle) that can simulate proofs for any statements without being given a witness that are indistinguishable from honestly generated proofs. *Proof of knowledge* (PoK) means that there exists an extractor that from any prover (which here is assumed to be algebraic; see Section 5) that produces a valid proof ψ for a statement (X, m) (or (X, Y, m)) can extract the witness $\log X$ (or $(\log X, \log Y)$). Proofs are *simulation-sound* (also called *simulation-extractable* (SE) for PoKs) if a witness can be extracted from a prover even if the prover can obtain simulated proofs ψ_i for statements (X_i, m_i) of its choice (except the one it is proving). For *strong* SE the only restriction is that the pair $((X, m), \psi)$ must be different from all query/response pairs $((X_i, m_i), \psi_i)$.

Pedersen commitments. We employ Pedersen commitments, which are homomorphic w.r.t. the committed values and the used randomness. A value $v \in \mathbb{Z}_p$ is committed by sampling $q \leftarrow\!\!\$\ \mathbb{Z}_p$ and setting

$$C = \mathsf{Cmt}(v, r) := vH + qG .$$

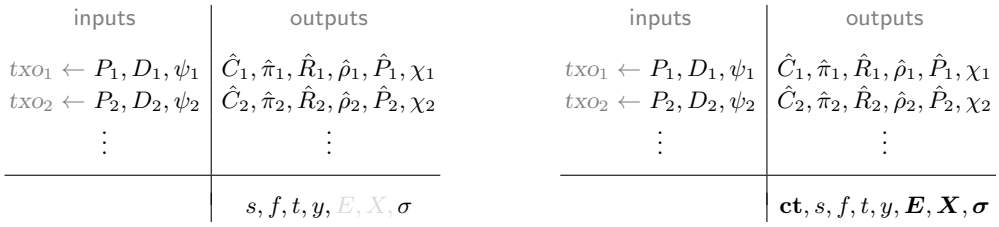

Fig. 1. Left: Visualization of a (simple) MW-NIT transaction. Inputs consist of a value P_i contained in a previous transaction output, a "doubling key" D_i and a proof of possession (PoP) ψ_i of P_i and D_i. Outputs consist of a commitment $\hat{C}_i$ to their value, an associated range proof $\hat{\pi}_i$, an ephemeral key $\hat{R}_i$, a signature (or PoP) $\hat{\rho}_i$ under $\hat{R}_i$, the one-time address $\hat{P}_i$ and an epoch χ_i. The kernel consists of the supply s, the fee f, the offsets t and y and the PoP σ of E and X. The excess E and the stealth excess X can be computed as in (1) and (2). **Right:** Visualization of an aggregated MW-NIT transaction. It additionally contains a cut-through list **ct**, and lists of excesses, stealth excesses and corresponding PoPs.

A commitment is opened by sending (v and) q and testing $C \stackrel{?}{=} vH + qG$. Pedersen commitments are perfectly hiding (i.e., no information about the value is leaked) and computationally binding (i.e., under the discrete logarithm (DL) assumption, no adversary can change their mind about a committed value, that is, find a commitment and two openings $v \neq v'$ to it).

Range proofs. We require a proof system for statements on commitments, in particular for the NP language defined by the following relation asserting that a committed value is contained in an admissible interval:

$$\left\{\left(C, (v, r)\right) \ : \ C = \mathsf{Cmt}(v, r) \ \wedge \ v \in [0, v_{\max}]\right\}$$

We assume a zero-knowledge proof system RaP for proofs of knowledge of a witness (v, r) for a statement C such that $0 \leq v \leq v_{\max}$. (Note that we do not assume RaP to be simulation-sound [FOS19, Def. 8], whereas FOS required this in their proof of theft prevention. Our scheme could thus be potentially instantiated with a more efficient range proof system than theirs.) We denote the prover algorithm by $\pi \leftarrow \mathsf{RaP.P}(C, (v, r))$ and the verifier by $b \leftarrow \mathsf{RaP.V}(C, \pi)$. A typical choice of proof system for RaP are Bulletproofs [BBB+18], which do not introduce any new trust assumption (as its parameters are random group elements, as for Pedersen commitments).

3 Proposal for MW with non-interactive transactions

We start with presenting the scheme and then discuss the reason for our design choices, such as adding *stealth excesses* (Section 4.1) and *doubling keys* (which prevent a sub-exponential-time attack; Section 4.2).

3.1 Data structures

The extension of Mimblewimble to non-interactive transactions introduces the notion of addresses.

A **stealth address** is a pair $(A = aG, B = bG) \in \mathbb{G}^2$, for which we call $(a, B) \in \mathbb{Z}_p \times \mathbb{G}$ the **view key** and $(a, b) \in \mathbb{Z}_p^2$ the **spend key**.

A **transaction** in MW-NIT is composed of (see also Figure 1):

– A list of **outputs**: tuples of the form $txo = (\hat{C}, \hat{\pi}, \hat{R}, \hat{\rho}, \hat{P}, \hat{\chi})$, each implicitly associated to an **output address** (A, B), composed of:
  - an **ephemeral key** $\hat{R} = \hat{r}G \in \mathbb{G}$, chosen by the sender, which defines two keys as:

$$(\hat{k}, \hat{q}) \coloneqq \mathrm{H}(\hat{r}A)$$

  (note that $(\hat{k}, \hat{q})$ can be computed from the view key and $\hat{R}$, as $\hat{r}A = a\hat{R}$)

- a **commitment** $\hat{C} \coloneqq \hat{v}H + \hat{q}G$ to the coin value $\hat{v}$, using randomness $\hat{q}$
- a **range proof** $\hat{\pi}$ proving knowledge of an opening (v, q) of $\hat{C}$, with $v \in [0, v_{\max}]$
- a **one-time output public key** $\hat{P} \in \mathbb{G}$, computed from $\hat{k}$ as $\hat{P} \coloneqq \hat{B} + kG$ (note that the *spend* key is required to compute $\log \hat{P}$)
- a **proof of possession** $\hat{\rho}$ of $\hat{R}$ with tag $\hat{C}\|\hat{\pi}\|\hat{P}\|\hat{\chi}$
- an **epoch** $\hat{\chi}$ in which the output was created

– A list of **inputs** of the form (P, D, ψ) where
- $P \in \mathbb{G}$ is the one-time **public key** of the transaction output being spent (each value P is only allowed once in the ledger)
- $D \in \mathbb{G}$ is the one-time **doubling key**, chosen by the sender, that "doubles" P
- ψ is a **proof of possession** of P and D with tag the transaction output being spent

– The **kernel**, which is composed of:
- the **supply** $s \in [0, v_{\max}]$, indicating the amount of money created in the transaction
- the **fee** $f \in [0, v_{\max}]$, indicating the fee paid for the current transaction
- the **offset** $t \in \mathbb{Z}_p$
- the **excess** $E \in \mathbb{G}$, defined as the difference between the commitments in the outputs (including the fee) and the inputs (including the supply), shifted by the offset. If C_i is the i-th input commitment, that is, the value contained in the output in which P_i appears, then

$$E \coloneqq \sum \hat{\boldsymbol{C}} + fH - \sum \boldsymbol{C} - sH - tG \,, \tag{1}$$

  which can be seen as $E \coloneqq E' - tG$ in terms of the *true excess* $E' \coloneqq \sum \hat{\boldsymbol{C}} + fH - \sum \boldsymbol{C} - sH$
- the **stealth offset** $y \in \mathbb{Z}_p$
- the **stealth excess** $X \in \mathbb{G}$, defined as the difference between the ephemeral keys $\hat{R}_i$ from the outputs and the doubling one-time keys D_i from the inputs, shifted by the stealth offset y

$$X \coloneqq \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{D} - yG \tag{2}$$

- a **proof of possession** σ of E and X (with empty tag ε)

A (simple, i.e., non-aggregated; see below) transaction is thus of the form:

$$\mathsf{tx} = \big((\boldsymbol{P}, \boldsymbol{D}, \boldsymbol{\psi}), (\hat{\boldsymbol{C}}, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{R}}, \hat{\boldsymbol{\rho}}, \hat{\boldsymbol{P}}, \hat{\boldsymbol{\chi}}), (s, f, t, y, \sigma)\big)$$

3.2 Transaction creation

Consider a transaction output $txo = (C, \pi, R, \rho, P, \chi)$ spent to an address (A', B').

– Given the corresponding view key (a', B'), one can compute the shared keys k and q (the opening for the commitment C) as:

$$(k, q) \coloneqq \mathrm{H}(a'R) \,.$$

– Given the corresponding spend key (a', b'), one can compute the secret key for P as $\log P = b' + k$.

To create a transaction that, in an epoch $\hat{\chi}$, spends transaction outputs txo_i of values v_i from one-time keys P_i, for $i \in [n]$, and creates outputs of values $\{\hat{v}_i\}_{i \in [\hat{n}]}$ for destination addresses $\{(A_i, B_i)\}_{i \in [\hat{n}]}$, creating an amount s of new money and paying f in fees so that $\hat{\mathbf{v}} \in [0, v_{\max}]^{\hat{n}}$ and $\sum \hat{\mathbf{v}} + f = \sum \boldsymbol{v} + s$, do the following:

– for each input index $i \in [n]$:
- compute all values q_i and $p_i \coloneqq \log P_i$, for $i \in [n]$, as described above
- select a random $d_i \leftarrow\!\!\$\, \mathbb{Z}_p$ and set $D_i \coloneqq d_i G$
- compute a proof of possession

$$\psi_i \leftarrow \mathsf{PoP.P}((P_i, D_i, txo_i), (p_i, d_i))$$

– for each output index $i \in [\hat{n}]$:
  - select a random ephemeral key $\hat{r}_i \leftarrow\!\!\$\, \mathbb{Z}_p$ and set $\hat{R}_i \coloneqq \hat{r}_i G$
  - compute the shared secrets for the destination address (A_i, B_i)

$$(\hat{k}_i, \hat{q}_i) \coloneqq \mathrm{H}(\hat{r}_i A_i) \tag{3}$$

    and from them compute the output commitment and the one-time key

$$\hat{C}_i \coloneqq \hat{v}_i H + \hat{q}_i G \tag{4}$$

$$\hat{P}_i \coloneqq \hat{B}_i + \hat{k}_i G \tag{5}$$

  - compute a range proof $\hat{\pi}_i \leftarrow \mathsf{RaP.P}(\hat{C}_i, (\hat{v}_i, \hat{q}_i))$
  - compute a proof of possession of the output ephemeral key

$$\hat{\rho}_i \leftarrow \mathsf{PoP.P}((\hat{R}_i, \hat{C}_i \| \hat{\pi}_i \| \hat{P}_i \| \hat{\chi}), \hat{r}_i) \tag{6}$$

– sample uniformly at random $t \leftarrow\!\!\$\, \mathbb{Z}_p$ and compute

$$e \coloneqq \sum \hat{\boldsymbol{q}} - \sum \boldsymbol{q} - t = \log E$$

  with E as in (1)
– sample uniformly at random $y \leftarrow\!\!\$\, \mathbb{Z}_p$ and compute

$$x \coloneqq \sum \hat{\boldsymbol{r}} - \sum \boldsymbol{d} - y = \log X$$

  with X as in (2).
– compute a proof of possession of E and X with empty tag: $\sigma \leftarrow \mathsf{PoP.P}((E, X, \varepsilon), (e, x))$

The final transaction is

$$\mathsf{tx} \coloneqq \big((P_i, D_i, \psi_i)_{i\in[n]},\ (\hat{C}_i, \hat{\pi}_i, \hat{R}_i, \hat{\rho}_i, \hat{P}_i, \hat{\chi})_{i\in[\hat{n}]},\ (s, f, t, y, \sigma)\big)\ . \tag{7}$$

3.3 Transaction aggregation

Aggregate transactions are essentially concatenations of the composing transactions. In contrast to Jedusor's [Jed16] and FOS' [FOS19] protocols, *MW-NIT does not perform any cut-through*, as this is insecure, as we show in Section 4.3. Outputs of one transaction that are spent as inputs of another one in the aggregation are therefore kept in a *cut-through list* **ct**, which stores the concatenation of the output and the input spending the latter.

While simple transactions do not (need to) contain the (stealth) excesses (displayed in light gray in Figure 1), aggregate transactions (also displayed in Figure 1) contain lists of excesses $\boldsymbol{E}$, stealth excesses $\boldsymbol{X}$ and associated proofs $\boldsymbol{\sigma}$. An aggregated transaction is thus of the form

$$\mathsf{tx} = \big((P_i, D_i, \psi_i)_{i\in[n]},\ (\hat{C}_i, \hat{\pi}_i, \hat{R}_i, \hat{\rho}_i, \hat{P}_i, \hat{\chi}_i)_{i\in[\hat{n}]},\ (\mathbf{ct}, s, f, t, y, (E_i, X_i, \sigma_i)_{i\in[\bar{n}]})\big) \tag{8}$$

where $\mathbf{ct} \coloneqq \big(C'_i, \pi'_i, R'_i, \rho'_i, P'_i, \chi'_i, P'_i, D'_i, \psi'_i\big)_{i\in[n']}$. A simple transaction (7) can be cast as (8) by setting $\mathbf{ct} \coloneqq ()$, $\bar{n} = 1$ and computing E and X as in Equations (1) and (2). Given transactions tx_1 and tx_2, assuming w.l.o.g. that they are of the form (8), compute their aggregation tx as follows:

– define **txi** as the concatenation of the inputs of tx_1 and tx_2, and **txo** as the concatenation of their outputs, **ct** as the concatenation of their cut-through lists and and **ker** as the concatenation of lists of excesses, stealth excesses and associated signatures σ.
– if the same value P appears in two entries of **txi**, or if the same value P appears in two entries of **txo**, then abort
– if a value P appears in an entry of **txi** and in an entry of **txo**, remove the two entries from their resp. lists, concatenate the entries and add them to **ct**

– sort each list **txi**, **txo** and **ct** (lexicographically) and sort **ker** by the E values (required to hide which inputs and outputs comes from which transaction)
– compute the aggregated supply, fee, offset, and stealth offset (from the supplies s_i, etc., of tx_i):

$$s \coloneqq s_1 + s_2 \qquad f \coloneqq f_1 + f_2 \qquad t \coloneqq t_1 + t_2 \qquad y \coloneqq y_1 + y_2$$

– return $\mathsf{tx} := (\mathbf{txi}, \mathbf{txo}, (\mathbf{ct}, s, f, t, y, \mathbf{ker}))$.

3.4 Output verification

We define when a view key (a, B) accepts a transaction output. Given an amount v and an output $txo = (\hat{C}, \hat{\pi}, \hat{R}, \hat{\rho}, \hat{P}, \hat{\chi})$, compute $(k, q) \coloneqq \mathrm{H}(a\hat{R})$, and accept the output if txo has never been previously received in epoch $\hat{\chi}$ and

$$\hat{C} = vH + qG \quad \text{and} \quad \hat{P} = B + kG \ .$$

3.5 Transaction verification

Simple transactions. A transaction

$$\mathsf{tx} = \big((P_i, D_i, \psi_i)_{i\in[n]},\ (\hat{C}_i, \hat{\pi}_i, \hat{R}_i, \hat{\rho}_i, \hat{P}_i, \hat{\chi}_i)_{i\in[\hat{n}]},\ (s, f, t, y, \sigma)\big) \ ,$$

is valid w.r.t. a list of (previous) outputs **txo** with $txo_i = (C_i, \pi_i, R_i, \rho_i, P_i, \chi_i)$ if the P_i values in the inputs of tx and **txo** coincide and if the following hold:

(i) all input proofs are valid: for all $i \in [n]$: $\mathsf{PoP.V}((P_i, D_i, txo_i), \psi_i) = \mathbf{true}$
(ii) all range proofs are valid: for all $i \in [\hat{n}]$: $\mathsf{RaP.V}(\hat{C}_i, \hat{\pi}) = \mathbf{true}$
(iii) all proofs of possession of $\hat{R}$ are valid: for all $i \in [\hat{n}]$: $\mathsf{PoP.V}(\hat{R}_i, \hat{C}_i \| \hat{\pi}_i \| \hat{P}_i \| \hat{\chi}_i, \hat{\rho}_i) = \mathbf{true}$
(iv) the excess proof of possession is valid: $\mathsf{PoP.V}((E, X, \varepsilon), \sigma) = \mathbf{true}$, for

$$E \coloneqq \sum \hat{\boldsymbol{C}} - \sum \boldsymbol{C} + (f - s)H - tG \tag{9}$$

$$X \coloneqq \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{D} - yG \tag{10}$$

Aggregate transactions. An aggregate transaction as in (8)

$$\mathsf{tx} = \big(\mathbf{txi}, \widehat{\mathbf{txo}}, (\mathbf{ct}, s, f, t, y, \boldsymbol{E}, \boldsymbol{X}, \boldsymbol{\sigma})\big)$$

with $\mathbf{txi} = (P_i, D_i, \psi_i)_{i\in[n]}$, $\widehat{\mathbf{txo}} = (\hat{C}_i, \hat{\pi}_i, \hat{R}_i, \hat{\rho}_i, \hat{P}_i, \hat{\chi}_i)_{i\in[\hat{n}]}$, $\mathbf{ct} \coloneqq (\overline{\mathbf{txo}} \| \overline{\mathbf{txi}}) = (\overline{txo_i} \| \overline{txi_i})_{i\in[\bar{n}]}$ is verified w.r.t. previous outputs **txo** as follows:

Check that **in**, $\widehat{\mathbf{txo}}$ and **ct** are sorted lexicographically, then re-arrange the cut-through terms: set the outputs being spent and the inputs spending them, as well as the freshly created outputs, as

$$\mathbf{txo}^* := \mathbf{txo} \,\|\, \overline{\mathbf{txo}} = (C_i, \pi_i, R_i, \rho_i, P_i, \chi_i)_i$$
$$\mathbf{txi}^* := \mathbf{txi} \,\|\, \overline{\mathbf{txi}} = (P_i, D_i, \psi_i)_i$$
$$\widehat{\mathbf{txo}}^* := \widehat{\mathbf{txo}} \,\|\, \overline{\mathbf{txo}} = (\hat{C}_i, \hat{\pi}_i, \hat{R}_i, \hat{\rho}_i, \hat{P}_i, \hat{\chi}_i)_i$$

(Note that the last n' elements of $\mathbf{txo}^*$ and $\widehat{\mathbf{txo}}^*$ are equal but denoted differently). Verify the transaction $(\mathbf{in}^*, \widehat{\mathbf{txo}}^*, (s, f, t, y, \boldsymbol{E}, \boldsymbol{X}, \boldsymbol{\sigma}))$ w.r.t. $\mathbf{txo}^*$: check (i)–(iii) as for simple transactions, and the following instead of (iv):

(iv′) for all $i \in [\bar{n}]$: $\mathsf{PoP.V}((E_i, X_i, \varepsilon), \sigma_i) = \mathbf{true}$
(v′) additionally, the following "balance equations" are checked:

$$\sum \boldsymbol{E} = \sum \hat{\boldsymbol{C}} - \sum \boldsymbol{C} + (f - s)H - tG \tag{11}$$

$$\sum \boldsymbol{X} = \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{D} - yG \tag{12}$$

(Note that in (11) it suffices to sum the $\hat{\boldsymbol{C}}$ up to $\hat{n}$ and the $\boldsymbol{C}$ up to n, as the remaining terms from **ct** cancel out.)

3.6 Inclusion of transactions in the ledger

A ledger Λ is simply an aggregated transaction of the form (8) without any inputs (as the inputs of any transaction added Λ must spend existing outputs, these are thus moved to the cut-through list.

A transaction tx of the form (7) or (8) is included in Λ by aggregating Λ and tx to Λ', checking that Λ' has no inputs (thus all inputs of tx were spent/cut-through) and checking validity of Λ' w.r.t. an empty list $\mathbf{txo}$, as defined in Section 3.5. If any of the checks fail, return $\bot$, otherwise Λ'. (If Λ is known to be valid, it suffices to identify, for every transaction input (P_j, D_j, ψ_j), the ledger output txo_{i_j} containing P_j, and check validity of tx is w.r.t. $txo_{i_1}, \ldots, txo^*_{i_n}$.)

4 Fallacies in the initial proposal

We list below the main attacks found in Yu's scheme, and which motivated the design choices for our scheme in Section 3. Originally [Yu20, §2.2.2], transactions did not include the values D_i, nor the stealth excess X with the respective offset y. Instead, a valid transaction had to satisfy

$$E + tG = \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{P} \ , \tag{13}$$

(in our notation) [Yu20, Eq. ②] instead of Equation (10), and ψ and σ only proved knowledge of the discrete logarithms of P and E, respectively.

4.1 Correctness

Equation (13) can be achieved if one of the outputs, say the i-th, goes back to the creator of the transaction (e.g., because it is a "change output"). She can just set $\hat{R}_i \coloneqq E + tG + \sum \boldsymbol{P} - \sum_{j \neq i} \hat{R}_j$ (for which she knows $\log \hat{R}_i$) and then sample q_i uniformly. However, it is infeasible to create a transaction whose outputs are all linked to destination addresses, e.g., a transaction with a single output: $\hat{R}$ (together with the address) determines the coin opening q, which defines the value E; but (re-)defining $\hat{R}$ so that (13) holds would lead to a new value E. (In Yu's notation [Yu20, Eq. ②], the value r_o depends on q, which in turn is computed from r_o.)

In order not to restrict the format of transactions (and because it allows us to prove the scheme secure), we introduced stealth excesses X, which along with the proof of possession σ accounts for the "excess" in stealth addresses. We also introduce a *stealth offset* to preserve privacy of aggregated transactions.

4.2 The Feed-Me attack

It turns out that merely adding a stealth excess leads to an attack (against *transaction-binding*, see Section 6.5), which was first found by `@south_lagoon77`, alias `kurt`.[11] Consider the scheme in Section 3 but without any D values, which are replaced by the corresponding P values in the equations; in particular, the balance equation for one-time keys is

$$X = \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{P} - y^* G \tag{10*}$$

instead of Equation (10).

To explain the attack, it suffices to focus on non-aggregated transactions. Consider Alice, an honest user with address (aG, B), that creates two transactions tx_1 and tx_2, both with one input

[11] See: https://twitter.com/davidburkett38/status/1466460568525713413

and one output, transferring respectively v_1 to Bob and $v_2 > v_1$ to Charlie:

$$(C_1, \dots) = txo_1 \quad \leftarrow \quad \boxed{P_1, \psi_1 \mid \dots, \hat{R}_1, \hat{\rho}_1, \dots}_{t_1, y_1, E_1, X_1, \dots} \tag{tx_1}$$

$$(C_2, \dots) = txo_2 \quad \leftarrow \quad \boxed{P_2, \psi_2 \mid \dots, \hat{R}_2, \hat{\rho}_2, \dots}_{t_2, y_2, E_2, X_2, \dots} \tag{tx_2}$$

Both transaction are broadcast to the miners. A malicious miner can now forge a new transaction, transferring the amount $v_2 - v_1$ to himself, as follows:

$$txo_2 \quad \leftarrow \quad \boxed{P_2, \psi_2 \mid \dots, \hat{R}_1, \hat{\rho}_1, \dots \| \dots R^*, \rho^*, \dots}_{t_1, y^*, E_1, X_1, \dots} \tag{tx^*}$$

It combines the input of tx_2 and the output and the excesses from tx_1. A second output is computed "honestly", choosing $r^* \leftarrow\!\$\ \mathbb{Z}_p$, setting $R^* = r^*G$ and signing any P value (knowing $\log P$) via ρ^*. The miner also creates the corresponding coin C^*, so tx^* satisfies Equation (9), by setting $C^* := \mathsf{Cmt}(v_2 - v_1, q_2 - q_1)$, where q_1 and q_2 are the openings of C_1 and C_2 (which the miner can either obtain by knowing Alice's view key, or *by having sent the outputs that are now being spent by* tx_1 *and* tx_2 *to Alice in the first place.*) Finally, the miner needs to compute y^* so that tx^* satisfies Equation (10*), that is,

$$y^*G = R^* + \hat{R}_1 - P_2 - X_1 \ .$$

By validity of tx_1, again from (10*), we get $0 = -\hat{R}_1 + P_1 + y_1 G + X_1$ and by adding the two equations:

$$y^*G = R^* + P_1 - P_2 + y_1 G \ . \tag{14}$$

The crucial observation now is that if the miner knows the values k_1 and k_2 that define P_1 and P_2, resp. (which it can either obtain by knowing Alice's view key or by being the creator of the outputs spent by tx_1 and tx_2), then it knows the discrete logarithm of $P_1 - P_2 = (k_1 - k_2)G$, as the value B from Alice's address *cancels out*. The miner can thus compute y^* satisfying (14), thus completing the forged transaction tx^* that transfers the value $v_2 - v_1$ to the miner's address.

Fixes. To prevent this attack, our first fix was to derive the one-time key multiplicatively, as $P_i = k_i B$ in Equation (5). The term $P_1 - P_2$ then becomes $(k_1 - k_2)B$, which is non-zero with overwhelming probability and therefore it becomes hard to compute y^*. More generally, as long as the adversary cannot find distinct hash function outputs $k_{1,1}, \dots, k_{1,n_0}, k_{2,1}, \dots, k_{2,n_1}$ in Equation (3) so that

$$\sum \boldsymbol{k}_1 - \sum \boldsymbol{k}_2 = 0 \ , \tag{15}$$

this variant of the scheme (without the D values) satisfies transaction binding.

However, Wagner's *k-list tree algorithm* [Wag02] can be used to find such values in sub-exponential time. In order to be protected against active adversaries ready to invest substantial computing power, a user therefore would need to limit the number of its pending transactions at any point in time (which could degrade scalability of the system). To overcome this downside, we follow Burkett's approach [Bur21] and introduce an additional group element D in every transaction input which replaces P in the balance equation (10*), yielding (12). Since the D_i's are chosen by the creator of the transaction (whereas the P_i's are chosen by the previous spender, who in *Feed-Me* types of attacks is malicious), the values corresponding to the $k_{i,j}$ above are random, and thus the probability that (15) holds is negligible.

Since the D values are chosen by the honest user, the adversary in the Feed-Me attack does not know $\log(D_1 - D_2)$ (after replacing P values by D values in (14)), so we just reverted to the original format $P := B + kG$ for one-time keys and prove this variant secure.

4.3 On transaction cut-through

Suppose that, in an aggregate transaction, an output $(C, \pi, R, \rho, P, \chi)$ of one transaction is spent as input (P, D, ψ) of another transaction. One may wonder if, as with original MW, cut-through can be applied, that is, remove the spent output and the input referring to it from the aggregate transaction.

While validity of the coin-balance equation (11) would be maintained, this is not the case for the "address equation" (12). One may thus consider (as Yu does [Yu20, §2.1.1] for (sufficiently old parts of) the ledger) adding a value Z defined as the difference of the sum of all removed $\hat{R}$ values and all removed $\hat{D}$ values to the aggregated transaction. The check in Equation (12) would be replaced by $\sum \hat{\boldsymbol{R}} - \sum \boldsymbol{D} + Z = \sum \boldsymbol{X} + yG$, where the sums are only over the indices that have not been removed in the outputs ($\sum \hat{\boldsymbol{R}}$) and the inputs ($\sum \boldsymbol{D}$).

However, since Z is not bound to anything, this scheme would be insecure. Consider a miner that collects transactions and aggregates them. Then she simply replaces one of the remaining $\hat{R}$ values by a value of which she knows the discrete logarithm, puts a new $\hat{P}$ value, produces a corresponding proof $\hat{\rho}$ and defines Z as $\sum \boldsymbol{X} + yG + \sum \boldsymbol{D}$ minus the sum of the $\hat{\boldsymbol{R}}$ values including her own. This results in a valid transaction of which the miner now owns one of the outputs (assuming the miner knows the view key of the stealth address it stole the coin from; otherwise it made the coin unspendable by its owner).

We suspect that Yu assumed all R and P values remain for each (possibly aggregated) transaction when included in the blockchain, since in [Yu20, Eq. ③], it says :

$$SUM(R - P')_{spent\ at\ height}$$

which suggests that all these values need to be present. In addition, we note that simply removing cut-through inputs or outputs would make Equations ③ and ④ incorrect.

Example. Consider two 1-input/1-output transactions tx_1 and tx_2 (assume that all supplies and fees are 0).

$$txo_1 \quad \leftarrow \quad \boxed{P_1, D_1, \psi_1 \mid C_2, \pi_2, R_2, \rho_2, P_2, \chi_2}_{t_2, y_2, E_2, X_2, \sigma_2} \qquad (\mathsf{tx}_2)$$

$$\leftarrow \quad \boxed{P_2, D_2, \psi_2 \mid C_3, \pi_3, R_3, \rho_3, P_3, \chi_3}_{t_3, y_3, E_3, X_3, \sigma_3} \qquad (\mathsf{tx}_3)$$

where tx_2 spends some output txo_1, creating one output, which is then spent by tx_3 (in particular, we have $\mathsf{PoP.V}((P_2, D_2, \mathsf{tx}_2.\mathsf{out}), \psi_2) = \mathbf{true}$). Suppose tx_2 and tx_3 could be merged as

$$txo_1 \quad \leftarrow \quad \boxed{P_1, D_1, \psi_1 \mid C_3, \pi_3, R_3, \rho_3, P_3, \chi_3}_{t_2+t_3, y_2+y_3, (E_2, E_3), (X_2, X_3), (\sigma_2, \sigma_3), Z}$$

which is valid if ψ_1 and ρ_3 (and π_3), as well as σ_2, σ_3, are valid on their respective messages, and the following holds:

$$C_3 - C_1 = E_2 + E_3 + (t_2 + t_3)G$$
$$R_3 - D_1 + Z = X_2 + X_1 + (y_2 + y_3)G$$

Then a miner could simply choose r^*, p^*, χ_3^* set $R_3^* := r^*G$, $P_3^* := p^*G$, create ρ_3^* honestly, define $Z^* := X_2 + X_3 + (y_2 + y_3)G - R_3^* + P_1$ and create the following (valid!) transaction:

$$txo_1 \quad \leftarrow \quad \boxed{P_1, D_1, \psi_1 \mid C_3, \pi_3, R_3^*, \rho_3, P_3^*, \chi_3^*}_{t_2+t_3, y_2+y_3, (E_2, E_3), (X_2, X_3), (\sigma_2, \sigma_3), Z^*}$$

for which she knows the temporary spending key p^*.

On keeping ρ and ψ. One may wonder then if it is possible to keep the R and D values but elide the proofs ρ and ψ, or at least one of them? Again, each of these removals would lead to an attack. We start with considering **removing ρ but keeping ψ**, that is, an aggregated transaction in the example above looks as follows:

$$(\dots C_1) = \mathrm{Tx}_1 \quad \leftarrow \quad \boxed{P_1, D_1, \psi_1 \mid C_3, \pi_3, R_3, \rho_3, P_3, \chi_3}_{t_2+t_3, y_2+y_3, (E_2,E_3), (X_2,X_3), (\sigma_2,\sigma_3), \mathbf{ct}}$$

where $\mathbf{ct} = (R_2, P_2, D_2, \psi_2)$ and ψ_2 is a valid signature for P_2, D_2 (note that we elide the the transaction output for correctness). Intuitively, not having ρ means that P_2 is not bound to R_2 anymore, which the following attack leverages, where given tx_2 and tx_3, the miner replaces tx_3 by a transaction it owns: the miner chooses d_2^*, p_2^* and r_3^*, sets $D_2^* := d_2^* G$ and $R_3^* := r_3^* G$, computes ψ_2^* and ρ_3^* honestly and sets $X_2^* := (r_3^* - d_2^* - y_2)G$ and computes σ_2^* honestly. It also chooses p_3^*, d_3^*, sets $P_3^* := p_3^* G, D_3^* := d_3^* G$, and creates a proof ρ_3^* on it using r_3^*, and computes X_3^* and σ_3^* honestly. Then the following is a valid transaction for which the miner knows the key to spend the output:

$$(\dots C_1) \quad \leftarrow \quad \boxed{P_1, D_1, \psi_1 \mid C_3, \pi_3, R_3^*, \rho_3^*, P_3^*, \chi_3^*}_{t_2+t_3, y_2+y_3, (E_2,E_3), (X_2^*,X_3^*), (\sigma_2^*,\sigma_3^*), \mathbf{ct}^*}$$

where $\mathbf{ct}^* = (R_2, P_2^*, D_2^*, \psi_2^*)$. Finally, we show that **removing ψ but keeping ρ** also leads to attacks, similarly to the rogue key attack observed in [Yu20, §2.9.3]. In this scenario, the above aggregated transaction would look as follows:

$$(\dots C_1) \quad \leftarrow \quad \boxed{P_1, D_1, \psi_1 \mid C_3, \pi_3, R_3, \rho_3, P_3, \chi_3}_{t_2+t_3, y_2+y_3, (E_2,E_3), (X_2,X_3), (\sigma_2,\sigma_3), \mathbf{ct}}$$

where $\mathbf{ct} = (C_2, \pi_2, R_2, \rho_2, P_2, \chi_2, D_2)$ – note that C_2 and π_2 need to be present for ρ_2 to be verifiable. Consider an honest transaction tx_2

$$(\dots C_1) \leftarrow \boxed{P_1, D_1, \psi_1 \mid C_2, \pi_2, R_2, \rho_2, P_2, \chi_2}_{t_2, y_2, E_2, X_2, \sigma_2}$$

A miner can steal the output as follows (again, assuming it knows the view key of its holder and thus the opening of C_2).

The miner computes a fresh output $(C_3, \pi_3, R_3, \rho_3, P_3, \chi_3)$ honestly (i.e., knowing the logarithms of R_3 and P_3), picks random R_2^* and X_3 (knowing the corresponding logarithms) and sets $D_2^* := R_2^* + R_3 - D_1 - X_2 - X_3 - t_2 G$. It signs P_2 (as well as C_2, and π_2) under R_2^* as ρ_2^* and produces σ_3 under E_3, X_3. It then publishes the following transaction:

$$(\dots C_1) \quad \leftarrow \quad \boxed{P_1, D_1 \mid C_3, \pi_3, R_3, \rho_3, P_3, \chi_3}_{t_2+t_3, y_2, (E_2,E_3), (X_2,X_3), (\sigma_2,\sigma_3), \mathbf{ct}^*}$$

where $\mathbf{ct}^* := (C_2, \pi_2, R_2^*, \rho_2, P_2^*, \chi_2, D_2^*)$. Note that the transaction is valid, since we have

$$R_2^* + R_3 - D_1 - D_2^* = X_2 + X_3 + t_2 G$$

and the miner knows the key to spend the output.

4.4 Replay attacks

Yu [Yu20, §2.9.2] explains a replay attack for MW that is a result of non-interactive transactions: the adversary pays Alice via some output *txo*, which Alice later spends. Then the adversary pays her again, creating the exact same output; if Alice accepts it, the adversary can replay Alice's previous spend, making her lose the money.

A simple fix for replay attacks is requiring users to store all outputs ever received and never accept the same output a second time. A more viable method is using time stamps (named *epochs* and denoted χ in our notation), which Yu introduces in order to prevent rogue-key attacks. "Each *Input* must attach its own proof for $[P_i]$, as a second proof for the coin ownership" [Yu20, §2.9.3].

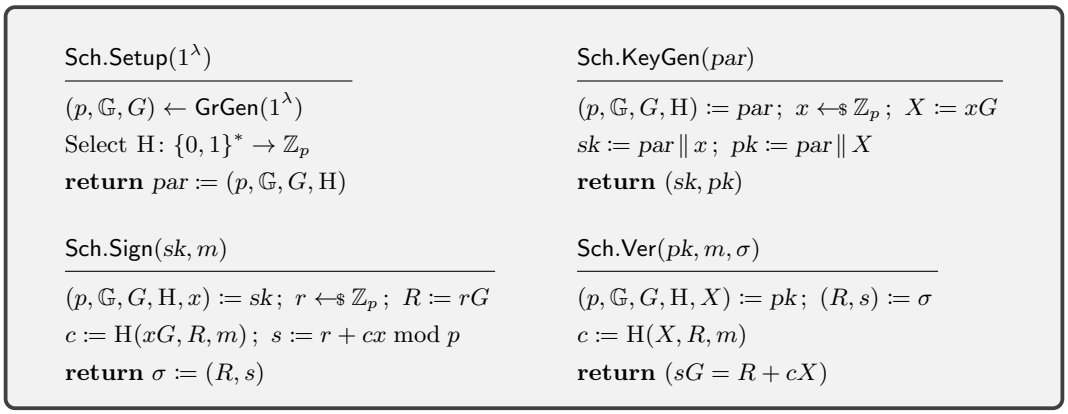

Fig. 2. Key-prefixed Schnorr signature scheme PoP[GrGen] based on a group generator GrGen.

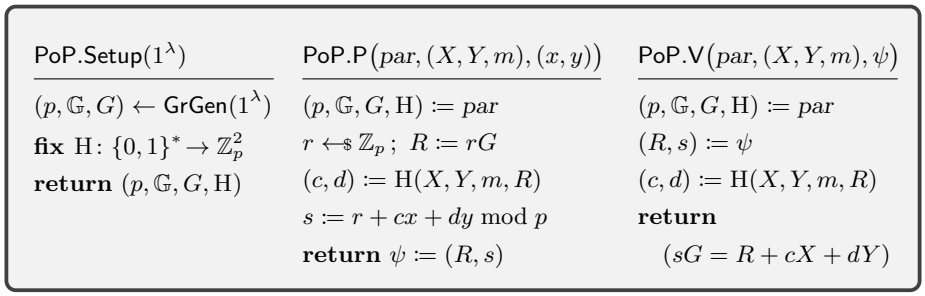

Fig. 3. Batch Schnorr zero-knowledge simulation-extractable proof of knowledge of two logarithms PoP[GrGen] w.r.t. a group generator GrGen.

While it is not specified which message is signed, it is crucial that the entire output specifically including the time stamp (and not just C) is signed. Otherwise, the above attack still works, as the adversary can change the time stamp in the replayed transaction (so the user accepts it) and recompute ρ, and send it to the user again. If the proof contained in the user's spendings did not authenticate the time stamp (or ρ), then the previous spend would still be valid on the replayed transaction. This is why in MW-NIT we define ψ as a proof that involves the *entire* output.

Epochs. If the user only accepts outputs that correspond to the current epoch, she only needs to compare a new output to those received in the same epoch; she can therefore delete all outputs from previous epochs. The duration of an epoch is a global parameter of the system, where short time intervals minimizes data storage, while larger intervals yield better privacy (as there are more transactions per epoch).

5 Simulation-extractability of Schnorr signatures

Before analyzing our scheme, we introduce and analyze its main building block. Key-prefixed Schnorr signatures, formally defined in Figure 2, can be reinterpreted as zero-knowledge proofs of knowledge of the secret key, with the statement also containing the message. To improve efficiency, we generalize this to a "batch" version that enables proving knowledge of the logarithms of two group elements, that is, proofs for the NP language defined w.r.t. a group description $(p, \mathbb{G}, G)$ by the following relation:

$$\big\{\big((X, Y, m), (x, y)\big) \,:\, X = xG \wedge Y = yG \wedge m \in \{0,1\}^*\big\}\ . \tag{16}$$

The proof system PoP is defined in Figure 3. We also use it to prove statements (X, m) with witness x by using standard Schnorr signatures, that is, PoP.P runs Sch.Sign and PoP.V runs Sch.Ver. The witness relation for PoP is thus the union of (16) and $\{((X, m), x) : X = xG\}$.

We show that PoP satisfies *strong simulation extractability* in the *algebraic group model* [FKL18] (see below) and the random oracle model (a tion of models also used in [FPS20] to show tight security of Schnorr signatures under the discrete-logarithm assumption).

Simulation-extractability. Strong simulation extractability for the above language means that from any adversary that returns a proof ψ^* for a statement (X^*, Y^*, m^*), the witness $(\log X^*, \log Y^*)$ can be extracted; and this holds even if the adversary gets access to an oracle that on inputs (X_i, Y_i, m_i) returns simulated proofs ψ_i for these statements. The only restriction is that the returned pair $((X^*, Y^*, m^*), \psi^*)$ must be different from all query/response pairs $((X_i, Y_i, m_i), \psi_i)$. Thus, forging a fresh proof ψ^* on a queried statement is considered a break of *strong* simulation-extractability if the extractor fails to extract a witness from ψ^*. (Note that this notion is stronger than forms of related-key-attack security for signature schemes (like UNF-CRO as defined and used in [FOS19]), where the adversary can only query signatures under keys for which it knows the difference in secret keys w.r.t. the challenge key.)

The algebraic group model. In the algebraic group model (AGM) [FKL18], adversaries are assumed to return a *representation* of any group element that they return. This means that, after having received input group elements $Z_1, \ldots, Z_n$, whenever the adversary returns a group element X, it must also return coefficients $\zeta_1, \ldots, \zeta_n$ so that $X = \sum \zeta_i Z_i$.

All our security proofs (except for the privacy notion) are reductions of solving the discrete logarithm (DL) problem to breaking the analyzed security notion of our scheme MW-NIT, assuming that PoP satisfies (strong) simulation-extractability (SE) in the AGM. The reduction thus receives a DL challenge Z and simulates the security game to an adversary $\mathcal{A}$, which we assume is algebraic. To leverage SE of PoP *in the AGM*, the reduction must construct an *algebraic* SE adversary, that is, one that accompanies each group-element output by their representations. However, the reduction can only return representations in basis (G, Z), its own group-element inputs. In particular, the reduction will run the adversary on some group elements $X_1, \ldots, X_n$, which it produces from its inputs G and Z in an "algebraic" way (i.e., knowing representations in basis (G, Z)). The adversary's group-element outputs will thus be in basis $X_1, \ldots, X_n$, which the reduction can then translate into the basis (G, Z).

For our reductions make use of simulation extractability, we must therefore strengthen the notion and consider auxiliary inputs. In the SE game, the adversary receives, besides a description of the underlying group, with generator G, and possible proof system parameters, an "auxiliary" uniform group element Z. At the end of its execution, the algebraic adversary must accompany each group element queried to the simulation oracle and output to the challenger (in particular, group elements in the statements (X_i, Y_i, m) for which the extractor must extract the witness) by a representation in basis (G, Z).

Security of PoP. Extending the techniques for showing tight security of Schnorr signatures in the AGM+ROM [FPS20], we show that in this model our proof system PoP is SE with auxiliary inputs.

Claim 1 *The proof system* PoP *in Figure 3 is strongly simulation-extractable with auxiliary group-element input in the algebraic group model and the random oracle model.*

Proof. The game is parametrized by a group $(p, \mathbb{G}, G)$ and a random oracle H, all provided to the adversary, who also receives a random group element $Z \leftarrow\!\!\$\ \mathbb{G}$. When the adversary queries

simulation of a proof for a statement $(X_i, Y_i, m_i) \in \mathbb{G}^2 \times \{0,1\}^*$, the simulator chooses uniform $c_i, d_i, s_i \leftarrow\!\!\$\ \mathbb{Z}_p$, sets $R_i := s_i G - c_i X_i - d_i Y_i$ and programs the random oracle so that $\mathrm{H}(X_i, Y_i, m_i, R_i) = (c_i, d_i)$. Since R_i is uniform and independent, the probability that the RO has already been defined for this value is negligible. If this happens then the simulator aborts and the adversary wins. As the adversary is algebraic, it needs to accompany its first query (X_1, Y_1, m_1) by coefficients $\alpha_1, \beta_1, \gamma_1, \delta_1$ with $X_1 = \alpha_1 G + \beta_1 Z$ and $Y_1 = \gamma_1 G + \delta_1 Z$. After receiving R_1, its second query is accompanied by $\gamma_X^{(2)}, \zeta_X^{(2)}, \rho_X^{(2)}, \gamma_Y^{(2)}, \zeta_Y^{(2)}, \rho_Y^{(2)}$ with $X_2 = \gamma_X^{(2)} G + \zeta_X^{(2)} Z + \rho_X^{(2)} R_1$ and $Y_2 = \gamma_Y^{(2)} G + \zeta_Y^{(2)} Z + \rho_Y^{(2)} R_1$. Since $R_1 = s_1 G - c_1 X_1 - d_1 Y_1$, this yields $X_2 = \alpha_2 G + \beta_2 Z$ with

$$\alpha_2 := \gamma_X^{(2)} + \rho_X^{(2)}(s_1 - c_1\alpha_1 - d_1\gamma_1) \quad \text{and} \quad \beta_2 := \zeta_X^{(2)} + \rho_X^{(2)}(-c_1\beta_1 - d_1\delta_1)\ ,$$

and analogous values γ_2 and δ_2 with $Y_2 = \gamma_2 G + \delta_2 Z$. In general, for every query answered with (R_i, s_i) for $(c_i, d_i) = \mathrm{H}(X_i, Y_i, m_i, R_i)$ we thus have

$$\begin{aligned} R_i = s_i G - c_i X_i - d_i Y_i &= (s_i - c_i\alpha_i - d_i\gamma_i)G - (c_i\beta_i + d_i\delta_i)Z \\ &= \Gamma_i G - \Delta_i Z \quad \text{with } \Gamma_i := s_i - c_i\alpha_i - d_i\gamma_i \ \text{ and } \ \Delta_i := c_i\beta_i + d_i\delta_i \end{aligned} \tag{17}$$

and from any representation

$$X_i = \gamma_X^{(i)} G + \zeta_X^{(i)} Z + \sum_{j=1}^{i-1} \rho_X^{(i,j)} R_j$$

we can recursively derive α_i, β_i so that $X_i = \alpha_i G + \beta_i Z$ and similarly γ_i, δ_i for $Y_i = \gamma_i G + \delta_i Z$.

Consider a proof (R^*, s^*) for a statement (X^*, Y^*, m^*) output by the adversary together with representations (α^*, β^*) for X^* and (γ^*, δ^*) for Y^* so that

$$(X^*, Y^*, m^*, R^*, s^*) \neq (X_i, Y_i, m_i, R_i, s_i) \quad \text{for all } i. \tag{18}$$

Validity means

$$R^* + c^* X^* + d^* Y^* = s^* G \quad \text{with} \quad (c^*, d^*) = \mathrm{H}(X^*, Y^*, m^*, R^*)\ . \tag{19}$$

Consider the point when $\mathrm{H}(X^*, Y^*, m^*, R^*)$ gets defined. This must be during a random-oracle query by the adversary, since a successful adversary cannot have made a simulation query for (X^*, Y^*, m^*) answered with R^*: as there is only one valid value s^*, this would mean that the adversary returned the oracle's response, i.e., (18) does not hold.

Let q be the number of simulation queries made before the random-oracle query (X^*, Y^*, m^*, R^*). Since the adversary is algebraic, it must accompany X^*, Y^* and R^* by representations $(\gamma_X, \zeta_X, \boldsymbol{\xi})$, $(\gamma_Y, \zeta_Y, \boldsymbol{\upsilon})$ and $(\gamma_R, \zeta_R, \boldsymbol{\rho})$, respectively with

$$\begin{aligned} X^* &= \gamma_X G + \zeta_X Z + \sum_{i=1}^{q} \xi_i R_i \overset{(17)}{=} \Big(\gamma_X + \sum_{i=1}^{q} \xi_i \Gamma_i\Big) G + \Big(\zeta_X - \sum_{i=1}^{q} \xi_i \Delta_i\Big) Z\ , \\ Y^* &= \Big(\gamma_Y + \sum_{i=1}^{q} \upsilon_i \Gamma_i\Big) G + \Big(\zeta_Y - \sum_{i=1}^{q} \upsilon_i \Delta_i\Big) Z \quad \text{and} \\ R^* &= \Big(\gamma_R + \sum_{i=1}^{q} \rho_i \Gamma_i\Big) G + \Big(\zeta_R - \sum_{i=1}^{q} \rho_i \Delta_i\Big) Z \end{aligned} \tag{20}$$

where the equalities follow from (17). Substituting X^*, Y^* and R^* in (19) by the above RHSs and grouping the coefficients of Z and G yields

$$\begin{aligned} &\Big(\zeta_R - \sum_{i=1}^{q} \rho_i \Delta_i + c^*\Big(\zeta_X - \sum_{i=1}^{q} \xi_i \Delta_i\Big) + d^*\Big(\zeta_Y - \sum_{i=1}^{q} \upsilon_i \Delta_i\Big)\Big) Z \\ &\qquad = \Big(s^* - \Big(\gamma_R + \sum_{i=1}^{q} \rho_i \Gamma_i\Big) - c^*\Big(\gamma_X + \sum_{i=1}^{q} \xi_i \Gamma_i\Big) - d^*\Big(\gamma_Y + \sum_{i=1}^{q} \upsilon_i \Gamma_i\Big)\Big) G\ . \end{aligned} \tag{21}$$

First consider the case where the representations of X^* and Y^* in (20) are independent of Z, that is

$$\zeta_X - \sum_{i=1}^{q} \xi_i \Delta_i \equiv_p 0 \equiv_p \zeta_Y - \sum_{i=1}^{q} \upsilon_i \Delta_i \ . \tag{22}$$

The extractor can thus output the witness $\log X^* = \gamma_X + \sum_{i=1}^{q} \xi_i \Gamma_i$ and $\log Y^* = \gamma_Y + \sum_{i=1}^{q} \upsilon_i \Gamma_i$.

Otherwise, at least one of the terms in (22), which are the coefficients of c^* and d^* in (21), is non-zero. The adversary chose the values $\alpha_i, \beta_i, \delta_i, \gamma_i$ (which define Γ_i and Δ_i) for all $i \in [q]$ when making simulation (or random-oracle) queries *before* making the query $\mathrm{H}(X^*, Y^*, m^*, R^*)$. Likewise, it must have chosen the values $\zeta_X, \zeta_Y, \zeta_R$ and $\xi_i, \upsilon_i, \rho_i$ for all $i \in [q]$ before making this query. Therefore, (c^*, d^*) is chosen uniformly at random *after* all other values in (21) are defined, and moreover at least one of c^* and d^* is not multiplied by 0. The probability that the coefficient of Z in (21) is congruent to 0 modulo p is thus $\frac{1}{p}$. From (21), the reduction can then efficiently compute $\log Z$ with overwhelming probability, and from the representations of X^* and Y^* in (20), it can compute the witness $(\log X^*, \log Y^*)$. □

Note that for readability we assumed all queried statements and the one returned are of the form (X, Y, m). However, the proof is easily extended to simultaneously allow for statements of the form (X, m), noting that (a) the inputs to the random oracle of type (X, Y, m, R) and (X, m, R) are disjoint and (b) extraction from a "simple" proof is done as from a proof for two elements by setting $d^* = 0$. This immediately yields the following:

Corollary 1. *In the AGM and the ROM, Schnorr signatures (Figure 2) are proofs of secret keys that are strongly simulation-extractable with auxiliary group-element input.*

6 Security analysis of MW-NIT

6.1 Assumptions

In our security analysis of the protocol from Section 3, we assume that range proofs in RaP prove knowledge of the committed value v and the opening q. (Note that for the employed Pedersen commitment, a proof of language membership, that is not "of knowledge" is vacuous, as for any C there always exists an opening e.g. $(v = 0, q = \log C)$.) We thus assume that there exists an extractor that from (an adversary outputting) a range proof π for $C \in \mathbb{G}$ can extract the values $v \in [0, v_{\max}]$ and $q \in \mathbb{Z}_p$.

We assume the existence of strongly simulation-sound (sSS) zero-knowledge (zk) proofs of knowledge (PoK) of the discrete logarithm of group elements with tags. In Section 5, we showed that in the combination of the random-oracle model and the algebraic group model [FKL18], Schnorr signatures are *adaptive* sSS zk-PoKs of the logarithm of the public key, for which the message acts as a tag. We furthermore extended this to proofs of knowledge of two logarithms, so that the proofs are of the same size as Schnorr signatures.

Finally, we assume that the discrete logarithm (DL) problem is hard in the group underlying the system, and for transaction privacy that the decisional Diffie-Hellman (DDH) assumption holds.

6.2 Syntax

We briefly review the syntax of an *aggregate cash system* [FOS19] and describe the adaptations required to capture addresses and non-interactive transactions.

The public parameters and an empty ledger are set up by $(pp, \Lambda) \leftarrow \mathsf{Setup}(1^\lambda, v_{\max})$, which takes as input the security parameter λ in unary and a maximal coin value $v_{\max}$. A *ledger* Λ specifies a *supply* $\Lambda.\mathsf{sply}$ (also denoted s) representing the value stored in Λ and a list of *transaction outputs*

Game $\mathrm{INFL}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max})$

$(pp, \Lambda) \leftarrow \mathsf{Setup}(1^\lambda, v_{\max})$

$(\Lambda^*, \mathsf{tx}^*, \hat{\mathbf{v}}, \mathbf{vk}) \leftarrow \mathcal{A}(pp, \Lambda)$

return $(\bot \notin \mathsf{Rcv}(pp, \mathsf{tx}^*, \hat{\mathbf{v}}, \mathbf{vk})$ // view keys accept outputs of tx^*

and $\mathsf{Ldgr}(pp, \Lambda^*, \mathsf{tx}^*) \neq \bot$ // tx^* is accepted by the ledger

and $\Lambda^*.\mathsf{sply} < \sum \hat{\mathbf{v}} + \mathsf{tx}^*.\mathsf{fees} - \mathsf{tx}^*.\mathsf{sply})$ // tx^* spends more than there is in the ledger

Fig. 4. Game for inflation resistance $\mathrm{INFL}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max})$.

(TXOs) $\Lambda.\mathsf{out}$. Users create addresses (or "wallets") by running $(pk, vk, sk) \leftarrow \mathsf{KeyGen}(pp)$, which returns a public key (*address*), a *view key* and a *spending key* sk.

A *transaction* tx has three attributes: a *supply* $\mathsf{tx.sply}$ specifying the amount of money it creates, a list $\mathsf{tx.in}$ of *inputs* and a list $\mathsf{tx.out}$ of *outputs* txo_i. A transaction is created by running $\mathsf{tx} \leftarrow \mathsf{Send}(pp, (\mathbf{txo}, \mathbf{v}, \mathbf{sk}), (\hat{\mathbf{v}}, \mathbf{pk}, \boldsymbol{\chi}), s, f)$ on vectors of transaction outputs $\mathbf{txo} = (txo_i)_i$, corresponding values $\mathbf{v} = (v_i)_i$ and spending keys $\mathbf{sk} = (sk_i)_i$, and vectors of output values $\hat{\mathbf{v}}$, destination addresses $\mathbf{pk}$, and epochs $\boldsymbol{\chi}$, as well as a supply s and a fee f. To aggregate transactions $\mathsf{tx}_1, \ldots, \mathsf{tx}_n$, run $\mathsf{tx} \leftarrow \mathsf{Agg}(pp, (\mathsf{tx}_1, \ldots, \mathsf{tx}_n))$ (which returns $\bot$ if they are incompatible, e.g. having an input in common).

Using a vector of view keys $\mathbf{vk}$ and a list of values $\hat{\mathbf{v}}$, one obtains the list of outputs of a transaction tx belonging to these keys by running $\mathbf{txo} \leftarrow \mathsf{Rcv}(pp, \mathsf{tx}, \hat{\mathbf{v}}, \mathbf{vk})$. If tx does not spend $\hat{v}_i \in \hat{\mathbf{v}}$ to $vk_i \in \mathbf{vk}$, then $txo_i \in \mathbf{txo}$ is set to $\bot$. Finally, $\Lambda' \leftarrow \mathsf{Ldgr}(pp, \Lambda, \mathsf{tx})$ returns an updated ledger Λ' including tx if it is valid and spends outputs present in the ledger, otherwise $\Lambda' := \bot$;

Correctness. We require the following straightforward correctness condition. Let $(pk_j, vk_j, sk_j)_j$ be key triples generated by KeyGen, and for $k \in [n]$ let tx'_k be a transaction, $\mathbf{v}'_k$ a vector of elements in $[0, v_{\max}]$, $\mathbf{vk}'_k \subseteq (vk_j)_j$ and let $\mathbf{sk}'_k$ be the corresponding spending keys (that is, if $\mathbf{vk}'_k = (vk_{j_i})_i$ for some j_i, then $\mathbf{sk}'_k = (sk_{j_i})_i$). For all $k \in [n]$, define $\mathbf{txo}_k \leftarrow \mathsf{Rcv}(pp, \mathsf{tx}'_k, \mathbf{v}'_k, \mathbf{vk}'_k)$.

Let indices i_j be so that $txo_{i_j} \neq \bot$ (where we let $(txo_\iota)_\iota := \mathbf{txo}_1 \,\|\, \ldots \,\|\, \mathbf{txo}_n$), let $\hat{\mathbf{v}} \in [0, v_{\max}]^*$, $\hat{\mathbf{pk}} \subseteq (pk_j)_j$, and s and f such that $\sum v'_{i_j} = \sum \hat{v}_i + f - s$. Then for $\mathsf{tx} \leftarrow \mathsf{Send}(pp, ((txo_{i_j})_j, (v'_{i_j})_j, (sk'_{i_j})_j), (\hat{\mathbf{v}}, \hat{\mathbf{pk}}, \boldsymbol{\chi}), s, f)$ and $\mathbf{txo} \leftarrow \mathsf{Rcv}(pp, \mathsf{tx}, \hat{\mathbf{v}}, \hat{\mathbf{vk}})$ where $\hat{\mathbf{vk}}$ corresponds to $\hat{\mathbf{pk}}$, we have $\bot \notin \mathbf{txo}$, that is, all outputs are accepted.

Comparison to FOS. The syntax of Send and Rcv differs from the one in [FOS19] due to the inclusion of addresses (as well as fees and epochs) and transactions being non-interactive. We moreover simplified notation by merging their algorithm Mint, used for creating money, with Send (which is now non-interactive and takes a supply as input). That is, $\mathsf{Mint}(pp, \hat{\mathbf{v}}, \mathbf{pk}, \boldsymbol{\chi})$ is an alias for $\mathsf{Send}(pp, (), (\hat{\mathbf{v}}, \mathbf{pk}, \boldsymbol{\chi}), \sum \hat{\mathbf{v}}, 0)$.

6.3 Inflation resistance

Definition. Informally, inflation resistance guarantees that the only way to create money in an *aggregate cash system*, such as Mimblewimble, is explicitly via the supply contained in transactions. The notion is defined by the following game, adapted from [FOS19, Def. 10] and formalized in Figure 4.

The adversary is given the system parameters (for MW-NIT they contain the elements G and H and potential parameters for the range proof), and its task is to produce a (valid) ledger and a transaction tx^* (accepted by the ledger) that spends an amount that exceeds the supply of the ledger (plus its own supply).

In addition to the output amounts $\hat{\mathbf{v}}$ of tx^*, the adversary must also return view keys, which accept the outputs of tx^*. Letting s denote the ledger supply, s^* the supply and f^* the fee of tx^*, the adversary wins if

$$s < \sum \hat{\mathbf{v}} + f^* - s^* \ . \tag{23}$$

Theorem 1. *If the range-proof system* RaP *and the proof-of-possession system* PoP *are extractable and if the discrete-logarithm assumption holds in the underlying group, then MW-NIT satisfies inflation resistance.*

Proof. Our analysis follows closely that of [FOS19, Theorem 13] for MW-FOS, since a ledger (or transaction) in MW-NIT contains an MW-FOS ledger (or transaction). A small difference is that FOS did not consider fees and kernel offsets, but these are easily added to the argument.

Consider a successful adversary that returns a ledger and a transaction

$$\begin{aligned}\Lambda^* &= \big(\boldsymbol{C}, \boldsymbol{\pi}, \boldsymbol{R}, \boldsymbol{\rho}, \boldsymbol{P}, \boldsymbol{\chi}, (\mathbf{ct}, s, f, t, y, \boldsymbol{E}, \boldsymbol{X}, \boldsymbol{\sigma})\big)\\ \mathsf{tx}^* &= \big((P'_i, D_i, \psi_i)_{i\in[n]}, (\hat{C}_i, \hat{\pi}_i, \hat{R}_i, \hat{\rho}_i, \hat{P}_i, \chi_i)_{i\in[\hat{n}]}, (s^*, f^*, t^*, y^*, \boldsymbol{E}^*, \boldsymbol{X}^*, \boldsymbol{\sigma}^*)\big)\end{aligned}$$

together with output values $(\hat{v}_i)_{i\in[\hat{n}]}$ and view keys $(a_i, B_i)_{i\in[\hat{n}]}$ that accept the outputs. For Λ^* to accept tx^*, all values P'_i must be in the ledger, that is, $\boldsymbol{P'} = (P_i)_{i\in I}$ for some $I \subseteq \mathbb{N}$. The reduction extracts all values v_i and openings q_i for all the output commitments C_i in the ledger from the respective range proofs π_i; thus $C_i = \mathsf{Cmt}(v_i, q_i)$. Since the ledger is valid, from (11) we have

$$\sum \boldsymbol{E} = \sum \mathsf{Cmt}(v_i, q_i) + \mathsf{Cmt}(f - s, 0) - \mathsf{Cmt}(0, t) \ . \tag{24}$$

We first show that the ledger must be balanced w.r.t. the values v_i extracted from the output commitments C_i, that is

$$s = \sum \boldsymbol{v} + f \ . \tag{25}$$

From the kernel proofs σ_i in Λ^*, the reduction can extract $e_i := \log E_i$. Since $\sum \boldsymbol{E} = \mathsf{Cmt}\big(0, \sum \boldsymbol{e}\big)$, we have that $(0, \sum \boldsymbol{e})$ is an opening of the right-hand side of (24). On the other hand, since Cmt is homomorphic, $\big(\sum \boldsymbol{v} + f - s, \sum \boldsymbol{q} - t\big)$ is also an opening. Thus if (25) did not hold then we would have two openings for two different values. This represents a break of the binding property of Pedersen commitments, which holds under the DL assumption. (This is formally proven by a reduction that obtains a DL challenge H, which then plays the role of Z in the AGM proof of extractability of PoP in Section 5.)

Let $\hat{q}_i$ be openings of the commitments in the outputs of tx^*, derived from $\hat{R}_i$ and the view key (a_i, B_i). Since the latter accepted the i-th output of tx^*, we have $\hat{C}_i = \mathsf{Cmt}(\hat{v}_i, \hat{q}_i)$. From validity of tx^* we have

$$\sum \boldsymbol{E}^* = \sum \hat{\boldsymbol{C}} - \textstyle\sum_{i\in I} C_i + \mathsf{Cmt}(f^* - s^*, 0) - \mathsf{Cmt}(0, t^*) \ , \tag{26}$$

where I is the set of indices of the outputs in the ledger that are spent by tx^*. From σ^*_i contained in tx^*, the reduction can extract $e^*_i := \log E^*_i$ and, analogously to the above, from (26) we get

$$\mathsf{Cmt}\big(0, \textstyle\sum e^*\big) = \mathsf{Cmt}\big(\sum \hat{\mathbf{v}} - \sum_{i\in I} v_i + f^* - s^*, \sum \hat{\boldsymbol{q}} - \sum_{i\in I} q_i - t^*\big) \ . \tag{27}$$

If the adversary outputs tx^* satisfying (23), then we have

$$\textstyle\sum_{i\in I} v_i \ \le\ \sum \boldsymbol{v} + f \overset{(25)}{=} s \overset{(23)}{<} \sum \hat{\mathbf{v}} + f^* - s^* \ ,$$

and thus $\sum \hat{\mathbf{v}} + f^* - s^* - \sum_{i\in I} v_i \neq 0$ (since all terms are bounded by $v_{\max}$ and thus never wrap around the modular representation). This means that the two sides in (27) are two different openings of the commitment $\sum \boldsymbol{E}^*$, which represents a break of commitment-binding. □

Game $\text{STEAL}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max})$

$(pp, \Lambda) \leftarrow \mathsf{Setup}(1^\lambda, v_{\max})$
$Hon, Archv := () \ ; \ j := 0$
$\mathcal{A}^{\text{KEYGEN},\text{SEND},\text{LEDGER}}(pp, \Lambda)$
return $(Hon \not\subseteq \Lambda.\mathsf{out})$

Oracle $\text{KEYGEN}()$

$j := j + 1$
$(pk_j, vk_j, sk_j) \leftarrow \mathsf{KeyGen}(pp)$
return (j, vk_j, pk_j)

Oracle $\text{SEND}((\mathbf{txo}, \mathbf{v}, I), (\hat{\mathbf{v}}, \mathbf{pk}, \boldsymbol{\chi}), s, f)$

if $\mathbf{txo} \not\subseteq Hon$ **then return** $\perp$
$\mathsf{tx} \leftarrow \mathsf{Send}\big(pp, (\mathbf{txo}, \mathbf{v}, (sk_i)_{i\in I}), (\hat{\mathbf{v}}, \mathbf{pk}, \boldsymbol{\chi}), s, f\big)$
if $\mathsf{tx} = \perp$ **then return** $\perp$
$Hon := Hon - \mathbf{txo}$ // remove spent outputs
return tx

Oracle $\text{LEDGER}(\mathsf{tx}, \hat{\mathbf{v}}, I)$

$\Lambda' := \mathsf{Ldgr}(pp, \Lambda, \mathsf{tx})$
if $\Lambda' = \perp$ **then return** $\perp$ **else** $\Lambda := \Lambda'$
// Check for new honest outputs in tx:
$\mathbf{txo} \leftarrow \mathsf{Rcv}(pp, \mathsf{tx}, \hat{\mathbf{v}}, (vk_i)_{i\in I})$
for $i \in [|\mathbf{txo}|]$:
  if $txo_i \neq \perp$ **and** $txo_i \notin Archv$
    $Hon := Hon \,\|\, txo_i$
    $Archv := Archv \,\|\, txo_i$
return Λ

Fig. 5. Game for theft resistance $\text{STEAL}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max})$

6.4 Theft resistance

We define two notions that protect users from losing money. The first one is an adaptation of the notion from [FOS19] to a scheme with non-interactive transactions. It guarantees that outputs in the ledger belonging to a user can only be spent by that user.

The main difference between MW-FOS and MW-NIT is that the former relies on the coin keys (the opening of the commitments) being kept secret, while in MW-NIT, the spender knows (and defines) the openings of the receivers' commitments.[12] In MW-NIT, the security relies on the secrecy of the "*spend key*" for the user's stealth address. We assume that the *view key* is known to the adversary (as delegating scanning for transactions should not endanger the security of these transactions).

Definition. Resistance to theft means that for any output belonging to a user in the ledger (that is, it was accepted by the user's view key), no matter how it was received (e.g., sent by the adversary), as long as the output has never been spent before and the user keeps her spend key safe, no one except her can spend it (even if her view key is publicly known). This is formalized via the game in Figure 5, which is an adaptation of [FOS19, Fig. 8]. Honest users are simulated by the experiment and the adversary can create new users by calling KEYGEN; the adversary can make users spend outputs they own (stored in the list *Hon*) by calling SEND; moreover it can submit any transactions to the ledger using it oracle LEDGER. If the transaction contains outputs that are accepted by honest users, it adds these to *Hon*. As users are not supposed to accept outputs they have already owned (cf. Section 4.4), the experiment stores all outputs ever received in a list *Archv*. The adversary wins the game if it spends any of the honest user's outputs, that is, if some output in *Hon* is not in the ledger.

Remark. Note that the game in Figure 5 for a scheme with non-interactive transactions is a lot simpler than [FOS19, Fig. 8], which had to take care of the interactive spending protocol. In

[12] Note that this is unavoidable for non-interactive transactions: knowing the (sum of) the receivers' keys is necessary to compute the excess proof σ.

particular, this involves an instruction for the honest user to *receive* coins, and the definition of the coins an honest user owns in [FOS19] is cumbersome, whereas for non-interactive transactions, anything accepted by the honest user's view key (and not previously owned) is considered belonging to her. Note also that we do not require an oracle MINT, since the adversary can run SEND with an empty input list.

Theorem 2. *If* PoP *is a simulation-sound proof of knowledge of discrete logarithms (cf. Section 5) and the DL assumption holds in the underlying group, then MW-NIT satisfies theft-resistance.*

Proof. Consider a user owning an output $txo = (C, \pi, R, \rho, P, \chi)$ with amount v in the ledger, that is, txo is accepted (see Section 3.4) by her stealth address $(A = aG, B)$, meaning $P = B + kG$ where $(k, q) = \mathrm{H}(aR)$. Spending this coin requires proving possession of P with tag txo, but an honest user, unless she spends that output, never proves possession of P with tag txo.

We formally use a theft to break the DL assumption assuming simulation-extractability of PoP. The reduction receives a DL challenge $B^* \in \mathbb{G}$, chooses $a^* \leftarrow\!\!\$\ \mathbb{Z}_p$, sets a random honest user's stealth address to $(A^* := a^*G, B^*)$ and gives the adversary the view key (a^*, B^*).

Whenever the user is asked to spend an output $txo' = (C', \pi', R', \rho', P', \chi')$ belonging to the user, the reduction computes the transaction as specified, choosing a doubling key $D' := d'G$ for a $d' \leftarrow\!\!\$\ \mathbb{Z}_p$. As it does not know the logarithm of P' (since this requires knowledge of $\log B^*$), it runs the zero-knowledge simulator for a proof of possession ψ for (P', D') with tag txo'. (Note that the reduction is algebraic w.r.t. its DL challenge B^*, in that it can give representations of all queried elements in basis (G, B^*), in particular $P' = B^* + k'G$, $D' = d'G$.)

Assume an output $txo = (C, \pi, R, \rho, P, \chi)$ belonging to the user is spent by the adversary. (If the adversary attacked a different user, the reduction aborts.) Then the corresponding transaction input must contain a proof ψ^* of possession of (P, D^*) with tag txo for some D^* (for which the reduction can derive a representation in basis (G, B^*) from the algebraic adversary's representation). By the definition of the security game, the honest user has never spent txo before. This means that the reduction has never queried a simulated proof for (P, D^*, txo). Therefore, by simulation-extractability in the AGM, the reduction can extract $p = \log P$, and since $P = B^* + kG$, for a value k known to the reduction, it can compute the solution $p - k$ to its DL challenge B^*. □

6.5 Transaction-binding

While the previous notion states that once a user owns an output in the ledger it cannot be purloined, we also need to guarantee that nothing can be "stolen" from a transaction *before* it is even added to the ledger (this protects against malicious miners, for example). In particular, if a user produces a transaction tx then no one should be able to create a transaction that contains *one* of the inputs of tx while not containing *all* its outputs (since transactions can be aggregated, further inputs and outputs could have been added to the original transaction). Thus, while *theft-resistance* protects the outputs of a transaction, *transaction-binding* in some sense protects the inputs.

Definition. We define transaction-binding via the following game, formalized in Figure 6. The experiment simulates all honest users, which the adversary can create by calling KEYGEN, which creates a new address, for which the adversary receives the view key. The adversary can instruct honest users to spend outputs to addresses of the adversary's choice, and the experiment computes the corresponding transaction using the users' spending keys and gives it to the adversary. The adversary's goal is to create a transaction tx^* which *spends the same input as an honest transaction* tx, but *does not contain all the outputs of* tx *that belong to honest users.*

We formalize this by having the adversary return a ledger Λ that must accept the attacked honest transaction tx, but after tx^* is included in Λ, the latter does not accept tx anymore (thus tx and tx^* have an input in common); the adversary also returns values and indices of honest users,

Game $\mathrm{TXBND}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max})$

$(pp, \Lambda) \leftarrow \mathsf{Setup}(1^\lambda, v_{\max})$
$j := 0\,;\ HTxs := ()$ // honest transactions
$(\Lambda, \mathsf{tx}^*, \hat{\mathbf{v}}, I) \leftarrow \mathcal{A}^{\textsc{Keygen},\textsc{Send}}(pp)$
$\Lambda^* := \mathsf{Ldgr}(pp, \Lambda, \mathsf{tx}^*)$
return $\big(\exists\, \mathsf{tx} \in HTxs : \mathsf{Ldgr}(pp, \Lambda, \mathsf{tx}) \neq \bot$ // tx accepted by $\Lambda\ldots$
and $\Lambda^* \neq \bot$ **and** $\mathsf{Ldgr}(pp, \Lambda^*, \mathsf{tx}) = \bot$ // …but not after tx^* was added
and $\bot \notin \mathsf{Rcv}(pp, \mathsf{tx}, \hat{\mathbf{v}}, (vk_i)_{i\in I})$ // tx sends values $\hat{\mathbf{v}}$ to $\mathbf{vk}\ldots$
and $\bot \in \mathsf{Rcv}(pp, \mathsf{tx}^*, \hat{\mathbf{v}}, (vk_i)_{i\in I})\big)$ // …but tx^* does not

Oracle $\textsc{Keygen}()$

$j := j + 1$
$(pk_j, vk_j, sk_j) \leftarrow \mathsf{KeyGen}(pp)$
return (j, vk_j, pk_j)

Oracle $\textsc{Send}((\mathbf{txo}, \mathbf{v}, I), (\hat{\mathbf{v}}, \mathbf{pk}, \boldsymbol{\chi}), s, f)$

$\mathsf{tx} \leftarrow \mathsf{Send}(pp, (\mathbf{txo}, \mathbf{v}, (sk_i)_{i\in I}), (\hat{\mathbf{v}}, \mathbf{pk}, \boldsymbol{\chi}), s, f)$
if $\mathsf{tx} = \bot$ **then return** $\bot$
$HTxs := HTxs \,\|\, \mathsf{tx}$
return tx

Fig. 6. Game for transaction-binding $\mathrm{TXBND}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max})$.

so that their view keys accept tx but do not accept tx^* (thus one of the outputs of tx is missing in tx^*).

Remark. While it might seem restrictive that only stealing outputs of honest users is considered a break of the notion, it is not. In aggregate cash systems like Mimblewimble (with cut-through), an adversary can always "steal" outputs it owns from any transaction tx: simply create a transaction that spends these outputs and then merge it with tx; the result is a transaction in which some of the outputs have been replaced by new ones. We do thus not consider this an attack and transaction-binding gives no guarantees against it.

Theorem 3. *If* PoP *is a strongly simulation-sound proof of knowledge of discrete logarithms (cf. Section 5) and the DL assumption holds in the underlying group, then MW-NIT satisfies transaction-binding.*

Proof intuition. As this is the most complex notion to analyze, we start with some intuition in a simplified scenario. Consider an adversary that sends Alice a transaction with one output $txo = (C, \pi, R, \rho, P, \chi)$ and that Alice creates a transaction

$$\mathsf{tx} = \big((P, D, \psi), (\hat{C}, \hat{\pi}, \hat{R}, \hat{\rho}, \hat{P}, \hat{\chi}), (s, f, t, y, \sigma)\big)$$

that spends txo to Bob. Moreover, assume this is the only transaction Alice ever makes. The adversary's goal is now to create a "forged" transaction $\mathsf{tx}^* = ((P, D^*, \psi^*), txo^*, (s^*, f^*, t^*, y^*, \sigma^*))$ that has P as its input, but with a different output $txo^* = (C^*, \pi^*, R^*, \rho^*, P^*, \chi^*)$. We assume tx^* to only have one output (and therefore be a *simple*, i.e., non-aggregate transaction).We show that conditions (I)–(III) must hold with overwhelming probability, by reducing the DL problem to it, assuming strong simulation-extractability (SE) of the proofs of possession. In (IV) we show that even then the game can only be won with negligible probability.

(I) $D = D^*$. Assume this is not the case. Under SE of PoP, this can be used to break DL by simulating the game as in the proof of theft-resistance.

Given a DL challenge B, the reduction embeds it into Alice's wallet (aG, B) for $a \leftarrow\!\!\$\, \mathbb{Z}_p$. It computes tx as prescribed, except that it simulates the proof ψ (as its witness depends on $\log B$). When the adversary outputs a transaction with input (P, D^*, ψ^*), it can extract the witness (p, d^*) from ψ^*, as the only simulated proof was for a different statement (P, D). Since $P = B + kG$, where k is computed from Alice's view key according to (3), the reduction can return $\log B = p - k$.

(II) $R^* \neq \hat{R}$. Assume $R^* = \hat{R}$. Since txo^* must be different, either the tag $C^*\|\pi^*\|P^*\|\chi^*$ for ρ^*, or ρ^* itself is different. As the statement/proof pair is different from the one created by Alice, one can extract $\log R^*$ by strong simulation extractability ("strong", since possibly only the proof differs). Given a DL challenge $\hat{R}$, the reduction embeds it in the output of tx. Not knowing $\log \hat{R}$, the reduction can still compute $(\hat{k}, \hat{q}) = H(a'\hat{R})$ using Bob's view key (a', B') (recall that Bob must be honest). From this, the reduction computes $\hat{C}$, $\hat{\pi}$ and $\hat{P}$. The proofs $\hat{\rho}$ and σ, whose witnesses depend on $\log \hat{R}$, are simulated.
If the adversary returns a transaction tx^* with $R^* = \hat{R}$, then from ρ^* the reduction can extract $\log \hat{R}$ (since the tag or ρ^* must be different), solving the DL challenge.

(III) $X^* \neq X$, that is, the stealth excess of tx^* is different from that of tx. Assume $X^* = X$. Then from their definitions in (2) we get $R^* - D - y^*G = \hat{R} - D - yG$, where we used $D^* = D$ from (I). Thus $\hat{R} = R^* + (y - y^*)G$. Since ρ^* proves knowledge of $\log R^*$, this means the adversary must also know $\log \hat{R}$.
Formally, as in case (II), the reduction embeds a DL challenge as $\hat{R}$ and simulates the proofs $\hat{\rho}$ and $\hat{\sigma}$ to compute tx. If the adversary's forgery violates (III), then from ρ^* the reduction can extract $r^* = \log R^*$ (since, by (II), ρ^* is for $R^* \neq \hat{R}$, and the simulated proof $\hat{\rho}$ was for $\hat{R}$) and thus compute the solution $r^* + y - y^*$.

(IV) Finally, we show that the adversary cannot win the game when (I)–(III) hold either. From (I) and (2), we have $X^* := R^* - D - y^*G$. Since σ^* contained in tx^* proves knowledge of $x^* := \log X^*$ (since $X^* \neq X$ by (III)) and ρ^* proves knowledge of $r^* := \log R^*$ (since $R^* \neq \hat{R}$ by (II)), this means that the adversary must know $\log D = r^* - y^* - x^*$.
The reduction embeds its DL challenge as D, the value in the input of tx, and simulates proofs ψ and σ (whose DL are or depend on $\log D$). All other components of tx are computed as prescribed. If the adversary is successful, then from ρ^* the reduction extracts r^* (no ρ proofs were ever simulated), and from σ^* the reduction extracts x^* (the simulated proof σ was under $X \neq X^*$) and computes $d := r^* - y^* - x^*$ as above.

Security proof. We generalize the above arguments, now considering the actual security game. Let tx^* be the adversary's forgery and tx be the attacked transaction, that is, tx^* contains one input of tx but not all its outputs. We proceed in a sequence of hybrid games, each one introducing new abort conditions which make the game return 0. The condition in H_1 corresponds to (I) above and the ones in H_2 correspond to (II) and (III), which we combine in the same hybrid is they can be proved in one reduction.

- H_0 This is the original transaction-binding game.
- H_1 The first hybrid game aborts if one of the inputs in tx^* contains a P value from an honest transaction input but the associated D value is different (abort condition (A1)).
- H_2 The second hybrid aborts if (A2) all the R values in tx also appear in tx^*, or if (A3) there is an output in tx for an honest user whose R value does not appear in tx^* and tx^* contains the stealth excess X of tx.

Note that if tx^* contains the stealth excess X of tx then Hybrid H_2 returns 0: if it does not abort in (A3) then all R values from tx must be in tx^*, and thus it aborts in (A2). Therefore, if the adversary wins H_2 then

(W1) one of the D values in tx^* is from an input of tx (otherwise (A2) would occur)

(W2) the stealth excess list $\boldsymbol{X}$ of tx^* does not contain the stealth excess X of tx

Hybrid H_0 to H_1. We start by showing that the probability that H_0 returns 1 but H_1 does not (because it aborts) is negligible by reducing the DL problem to the event (A1) occurring, assuming simulation-extractability (SE) of PoP.

Given a DL challenge $Z \in \mathbb{G}$, the reduction embeds it into the wallets of *all* honest users: it picks $a_j, b_j \leftarrow\!\$\, \mathbb{Z}_p$ and defines user U_j's wallet as $(a_j G, B_j := Z + b_j G)$. Note that this is correctly distributed and that the reduction can give the adversary the corresponding view key (a_j, B_j). When U_j is asked to create a transaction, the reduction proceeds as prescribed, except that runs the zero-knowledge simulator for the proof ψ (the only value that depends on $\log B_j$). Let (P_i, D_i, ψ_i) be the inputs of all transactions by honest users (and thus ψ_i are the simulated proofs). Since honest users only spend outputs that are in the ledger and the ledger only accepts every P value once, all P_i's are distinct.[13]

Consider the transaction tx^* returned by the adversary, and assume (A1) occurs, i.e., one of its inputs is of the form (P_{i^*}, D^*, ψ^*) with $D^* \neq D_{i^*}$. As all the P_i are different, we have $(P_{i^*}, D^*) \neq (P_i, D_i)$ for all i. This means that no proof for (P_{i^*}, D^*) has been simulated and by SE of PoP, the reduction can extract the witness (p_{i^*}, d^*) from ψ^*. Let U_j be the user that spent P_{i^*}; then $P_{i^*} = B_j + kG$ for k derived from U_j's view key (a_j, B_j) as in (3). Plugging in the definition of B_j yields $\log Z = p_{i^*} - b_j - k$, the solution of the DL challenge.

Hybrid H_1 to H_2. We next show that in a winning run of H_2 both abort conditions (A2) and (A3) can only occur with negligible probability. If any of them occurs, then there exist i and ι so that tx will be the i-th transaction and the ι-th output txo_ι of tx will be for an honest user and such that $\hat{R}_\iota$ from txo_ι will either (A2) be contained in some output txo^* of tx^* but $txo_\iota \neq txo^*$ (otherwise H_2 is not won when (A2) occurs) or (A3) $\hat{R}_\iota$ will not be contained in tx^* and X will be in the stealth excess list of tx^*. We will construct a reduction that makes a *guess* (i, ι), so that if the adversary wins H_1 but not H_1 (i.e., either (A2) or (A3) occur) and the guess is correct, then it breaks DL.

The reduction receives a DL instance $Z \in \mathbb{G}$ and guesses i and ι. The reduction simulates H_1 as prescribed, except when creating the i-th transaction it sets $R_\iota := Z$ (and thus does not know its discrete logarithm). It completes the transaction as follows: if the receiver of txo_ι is not honest, it aborts (and the guess (i, ι) was wrong). Otherwise, let (a_j, B_j) be receiver's view key. The reduction computes

$$(k_\iota, q_\iota) := \mathrm{H}(a_j R_\iota)$$

and, from this, C_ι, P_ι and π_ι as prescribed. It runs the zero-knowledge simualtor to obtain a proof ρ_ι for $(R_\iota, C_\iota \| \pi_\iota \| P_\iota \| \chi_\iota)$, which completes txo_ι. To complete the transaction, it also queries a simulated σ (whose witness depends on $\log R_\iota$).

Assume that the reduction guessed correctly and that (A2) occurs. Then some output txo^* of tx^* contains a proof ρ^* for $(R_\iota, C^* \| \pi^* \| P^* \| \chi^*)$ so that $(C^* \| \pi^* \| P^* \| \chi^*, \rho^*) \neq (C_\iota \| \pi_\iota \| P_\iota \| \chi_\iota, \hat{\rho}_\iota)$; otherwise we would have $txo_\iota = txo^*$. The statement for ρ^* (or ρ^* itself) is thus different from the statement for the simulated $\hat{\rho}_\iota$ (or $\hat{\rho}_\iota$ itself). By strong SE of PoP, the reduction can thus extract from ρ^* the witness $\log R_\iota = \log Z$, the DL solution. (Recall that σ is also simulated, but that different types of proofs are assumed to be domain-separated, cf. Section 2)

Now assume that the reduction guessed correctly and that (A3) occurs. This means that the adversary's transaction

$$\mathsf{tx}^* = \big((\boldsymbol{P}^*, \boldsymbol{D}^*, \boldsymbol{\psi}^*), (\hat{\boldsymbol{C}}, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{R}}, \hat{\boldsymbol{\rho}}, \hat{\boldsymbol{P}}, \hat{\boldsymbol{\chi}}), (s, f, t, y, \boldsymbol{E}, \boldsymbol{X}, \boldsymbol{\sigma})\big)$$

[13] We identify outputs univoquely by their P value. Despite a slight change in the game for transaction-binding would still alllow us to prove security in the eventuality that tx contained duplicated P's, we opeted for this simplified exposition.

is such that $\boldsymbol{X}$ contains the stealth excess X of tx and $\hat{\boldsymbol{R}}$ does not contain R_ι. By validity of tx, it satisfies the balance equation (12) and since $R_\iota = Z$, we have

$$X = \textstyle\sum_{i\neq\iota} R_i + Z - \sum \boldsymbol{D} - yG \ . \tag{28}$$

Note that all values $r_i = \log R_i$, for $i \neq \iota$, and $d_i = \log D_i$ have been chosen by (and are thus known to) the reduction. Moreover, since (A3) occurred, for some I^* with $|I^*| =: m \geq 1$ we have $X_i = X$ for all $i \in I^*$ (the adversary could have included X multiple times in $\boldsymbol{X}$). By validity of tx^* this yields

$$\textstyle\sum_{i\notin I^*} X_i + mX = \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{D}^* - yG \ . \tag{29}$$

From the proofs σ_i in tx^*, for $i \notin I^*$, the reduction can extract the corresponding values $x_i = \log X_i$ (since only σ for X was simulated and $X \notin (X_i)_{i\notin I^*}$). From the proofs ψ_i^*, the reduction can extract the values $d_i^* = \log D_i^*$, since no ψ proofs are simulated (if the adversary copied D and ψ from an honest transaction, the reduction knows $\log D$). From the proofs $\hat{\rho}_i$, the reduction can extract the values $\hat{r}_i = \log \hat{R}_i$ (and knows the logarithms of the ones copied from honest transactions). Note that R_ι (for which ρ_ι was simulated) is not among $\hat{\boldsymbol{R}}$.

Plugging (28) into (29), taking the logarithm to base G and substituting all logarithms of group elements known to the reduction thus yields:

$$\log Z \equiv_p \tfrac{1}{m}\big(-\textstyle\sum_{i\notin I^*} x_i + \sum \hat{\boldsymbol{r}} - \sum \boldsymbol{d}^* - y\big) - \sum_{i\neq\iota} \boldsymbol{r}_i + \sum \boldsymbol{d} + y \ ,$$

meaning the reduction can solve the DL challenge.

Hybrid H_2. It remains to show that H_2 can only be won with negligible probability, where we leverage that winning the game implies (W1) and (W2).

Again, we reduce solving DLs to winning H_2, assuming SE of PoP. The reduction now embeds a DL challenge $Z \in \mathbb{G}$ into *all* the doubling keys D in inputs of the honest users. Whenever the adversary asks an honest user to create a transaction, the reduction randomly samples $d \leftarrow\!\$\ \mathbb{Z}_p$, sets $D := dZ$ and queries simulated proofs ψ for (P, D), as well as the excess proof σ (whose witnesses all depend on $\log D$). At the end of the execution, the adversary returns a forged transaction tx^* of the form

$$\mathsf{tx}^* = \big((\boldsymbol{P}, \boldsymbol{D}, \boldsymbol{\psi}), (\hat{\boldsymbol{C}}, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{R}}, \hat{\boldsymbol{\rho}}, \hat{\boldsymbol{P}}, \hat{\boldsymbol{\chi}}), (s, f, t, y, \boldsymbol{E}, \boldsymbol{X}, \boldsymbol{\sigma})\big) \ ,$$

where, by (W1) and (W2), $\boldsymbol{D}$ contains an input D from an honest transaction tx and X is not contained in $\boldsymbol{X}$. In tx^* we distinguish two types of inputs and two types of stealth excesses:

- let $\bar{D}_i$ be the values that appear in honest transactions, and D_i^* be those that do not
- let $\bar{X}_i$ be the stealth excesses that appear in honest transactions and X_i^* be those that do not

For all $\bar{X}_i$ appearing in an honest transaction tx_i, let $D_{i,j}$ and $R_{i,j}$ denote the values contained in the inputs and outputs of tx_i and let y_i be its stealth offset. By validity of tx_i we have:

$$\bar{X}_i = \textstyle\sum_j R_{i,j} - \sum_j D_{i,j} - y_i G \ .$$

As these values $D_{i,j}$ and the values $\bar{D}_i$ contained in tx^* were created by the reduction, we have $D_{i,j} = d_{i,j} \cdot Z$ and $\bar{D}_i = \bar{d}_i \cdot Z$ for values $d_{i,j}$ and $\bar{d}_i$ known to the reduction. Moreover, the reduction knows the values $r_{i,j} := \log R_{i,j}$, as it chose them. From the above, we thus have:

$$\log \bar{X}_i = \textstyle\sum_j r_{i,j} - \sum_j (d_{i,j} \cdot \log Z) - y_i \qquad\qquad \log \bar{D}_i = \bar{d}_i \cdot \log Z \tag{30}$$

From the proofs ψ_i in tx^* corresponding to the values D_i^* and from σ_i corresponding to X_i^*, the reduction can extract $\log D_i^*$ and $\log X^*$ (since, by definition, D_i^* and X_i^* are values that have not been created by the reduction and thus no proofs have been simulated for them). Moreover, if any $\hat{R}_i$ in tx^* was copied from an honest transaction, the reduction knows $\log \hat{R}_i$. For all other $\hat{R}_i$, from

the proof $\hat{\rho}_i$ also contained in tx^*, the reduction can extract $\log \hat{R}_i$ (as no ρ proofs are simulated). The reduction thus knows the values

$$\log D_i^* =: d_i^* \qquad \log X_i^* =: x_i^* \qquad \log \hat{R}_i =: \hat{r}_i \tag{31}$$

Since tx^* is valid, it satisfies the balance equation (12), that is:

$$\sum \boldsymbol{X}^* + \sum \bar{\boldsymbol{X}} \;=\; \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{D}^* - \sum \bar{\boldsymbol{D}} - y^* G \;.$$

Taking the logarithm to base G, and using (30) and (31) this yields:

$$\sum x_i^* + \sum\sum r_{i,j} - \sum\sum (d_{i,j} \cdot \log Z) - \sum y_i \;\equiv_p\; \sum \hat{r}_i - \sum d_i^* - \sum (\bar{d}_i \cdot \log Z) - y^* \;. \tag{32}$$

We show that with overwhelming probability:

$$\sum \bar{d}_i - \sum\sum d_{i,j} \not\equiv_p 0 \;. \tag{33}$$

Indeed, by (W1), at least one value $\bar{d}_{i^*}$ comes from tx. Moreover, by (W2) the stealth excess of tx does not appear in tx^*, and thus the value $\bar{d}_{i^*}$ is not among $(d_{i,j})_{i,j}$ (except with negligible probability if the reduction chose the same value twice). Whenever (33) is satisfied, the reduction can thus compute $\log Z$ from (32). □

6.6 Transaction privacy

This section considers an attacker that passively observes blocks and attempts to recover information about the transaction graph or the transacted amounts. In Jedusor's original proposal [Jed16], no guarantee of privacy was given besides hiding transaction amounts, and this was reflected in prior definitions [FOS19, Def. 12]. In particular, in the initial version of Mimblewimble one can "disaggregate" transactions [Dev20a], that is, link inputs and outputs that come from the same original transaction. As a consequence, one could infer how money was being spent across the network.

Grin introduced *transaction offsets*, which enable stronger anonymity guarantees by preventing disaggregation. To reflect this, we present a stronger privacy notion than the one provided in [FOS19], which is tailored to non-interactive transactions. We stress that our analysis is limited to the cryptographic properties of the scheme and it does not provide network-level privacy guarantees.

Definition. Our scheme provides three basic anonymity guarantees:

- a transaction hides the amounts contained in its inputs and outputs, as well as
- the destination addresses of the outputs (and inputs), *except to the receivers of the transaction*;
- in an aggregated transaction, it is not possible to tell which inputs and outputs belonged to the same component transaction *except for what can be deduced via the receiver's keys and the epochs of the outputs.*

The above are implied by the following simulation-based definition, formalized in Figure 7. The adversary has access to an oracle that creates honest users and a challenge oracle that produces and aggregates transactions, taking input:

- lists $\mathbf{ref}_i = (ref_{i,j})_j$, for $i \in [\ell]$, where $ref_{i,j}$ is either a tuple $(txo_{i,j}, v_{i,j}, sk_{i,j})$ or an identifier $id_{i,j}$ of a previously generated output, within the same or a previous oracle call ($\mathbf{ref}_i$ thus specifies the inputs of the i-th transaction)
- lists $(\mathbf{id}_i, (\hat{\mathbf{v}}_i, \hat{\mathbf{pk}}_i, \hat{\boldsymbol{\chi}}_i))$, where $id_{i,j}$ serves as the unique identifier of the corresponding triple value/address/epoch (this specifies the outputs of the i-th transaction)
- supplies s_i and fees f_i

Game $\text{TXPRV}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max})$

$b \leftarrow\!\!\$\ \{0,1\}$
$Pks, Txos := ()$ // hash tables
$(pp, \Lambda) \leftarrow \mathsf{Setup}(1^\lambda, v_{\max})$
$b' \leftarrow \mathcal{A}^{\text{Tx}_b}(pp, \Lambda)$
return $(b = b')$

OracleTx$_0((\mathbf{ref}_i, (\hat{\mathbf{id}}_i, (\hat{\mathbf{v}}_i, \hat{\mathbf{pk}}_i, \hat{\boldsymbol{\chi}}_i)), s_i, f_i))_{i\in[\ell]})$

if $\exists\,(i,j) \neq (i',j') : \hat{id}_{i,j} = \hat{id}_{i',j'}$ **or**
  $Txos(\hat{id}_{i,j}) \neq \bot$ **then abort**
for $i \in [\ell]$ // get ref's to previous txo's
  $\mathbf{in}_i := \text{GET}_0(\mathbf{ref}_i)$
  $\mathsf{tx}_i \leftarrow \mathsf{Send}(pp, \mathbf{in}_i, (\hat{\mathbf{v}}_i, \hat{\mathbf{pk}}_i, \hat{\boldsymbol{\chi}}_i), s_i, f_i)$
  // store outputs for honest users in $Txos$
  UpdTxos$(\mathsf{tx}_i, (\hat{\mathbf{id}}_i, (\hat{\mathbf{v}}_i, \hat{\mathbf{pk}}_i)))$
$\mathsf{tx} \leftarrow \mathsf{Agg}(pp, \mathsf{tx}_1, \ldots, \mathsf{tx}_\ell)$
return tx

Subprocedure $\text{GET}_b(\mathbf{ref}_i)$

$\mathbf{in} := ()$
for $j \in [|\mathbf{ref}_i|]$
  if $ref_{i,j} =: id$ // $ref_{i,j}$ is an id
    if $Txos(id) = (txo, v, sk) \neq \bot$
      if $b = 0 : \mathbf{in} := \mathbf{in} \,\|\, (txo, v, sk)$
      if $b = 1 : \mathbf{in} := \mathbf{in} \,\|\, txo$
    else
      if $b = 0$: **abort**
      if $b = 1 : \mathbf{in} := \mathbf{in} \,\|\, id$
  if $ref_{i,j} =: (txo, v, sk)$
    $\mathbf{in} := \mathbf{in} \,\|\, (txo, v, sk)$
return in

Oracle Keygen()

$(pk, vk, sk) \leftarrow \mathsf{KeyGen}(pp)$
// add values to table Pks
$Pks(pk) := (sk, vk)$
return pk

OracleTx$_1((\mathbf{ref}_i, (\hat{\mathbf{id}}_i, (\hat{\mathbf{v}}_i, \hat{\mathbf{pk}}_i, \hat{\boldsymbol{\chi}}_i)), s_i, f_i))_{i\in[\ell]})$

if $\exists\,(i,j) \neq (i',j') : \hat{id}_{i,j} = \hat{id}_{i',j'}$ **or**
  $Txos(\hat{id}_{i,j}) \neq \bot$ **then abort**
for $i \in [\ell]$
  $\mathbf{in}_i := \text{GET}_1(\mathbf{ref}_i)$
  if $\exists\, id \in \mathbf{in}_i : id \notin \hat{\mathbf{id}}_1 \,\|\, \ldots \,\|\, \hat{\mathbf{id}}_{i-1}$
    abort // id not a previous output
foreach (i,j) // remove honest keys ...
  if $Pks(\hat{pk}_{i,j}) = \bot$ // ... and values
    $out_{i,j} := (\hat{id}_{i,j}, (\hat{v}_{i,j}, \hat{pk}_{i,j}, \hat{\chi}_{i,j}))$
  else $out_{i,j} := (\hat{id}_{i,j}, (\bot, \bot, \hat{\chi}_{i,j}))$
$\mathbf{in}' := \mathsf{Sort}(\mathbf{in})\,;\ \mathbf{out}' := \mathsf{Sort}(\mathbf{out})$
$s := \sum_i s_i\,;\ f := \sum_i f_i$
$\mathsf{tx} \leftarrow \mathsf{Sim}(\mathbf{in}', \mathbf{out}', s, f, \ell)$
for $i \in [\ell]$
  UpdTxos$(\mathsf{tx}, (\hat{\mathbf{id}}_i, (\hat{\mathbf{v}}_i, \hat{\mathbf{pk}}_i)))$
return tx

Subproc. UpdTxos$(\mathsf{tx}_i, (\hat{\mathbf{id}}_i, (\hat{\mathbf{v}}_i, \hat{\mathbf{pk}}_i)))$

for $j \in [|\hat{\mathbf{id}}_i|]$
  if $(\hat{sk}_{i,j}, \hat{vk}_{i,j}) := Pks(\hat{pk}_{i,j}) \neq \bot$
    $txo_{i,j} \leftarrow \mathsf{Rcv}(pp, \mathsf{tx}_i, \hat{v}_{i,j}, \hat{vk}_{i,j})$
    $Txos(\hat{id}_{i,j}) := (txo_{i,j}, \hat{v}_{i,j}, \hat{sk}_{i,j})$

Fig. 7. Game for transaction privacy $\text{TXPRV}_{\mathsf{CASH},\mathcal{A}}(\lambda, v_{\max})$

Consider transactions tx_i produced by Send on input the i-th elements of the above lists, and tx, produced by Agg on the tx_i's. Transaction privacy requires that tx is indistinguishable from a transaction created by a *simulator* Sim, which is only given:

- a list $\mathbf{in}$ of length the total number of inputs, whose elements are either an output txo_i or an identifier id_i where $id_i \in \hat{\mathbf{id}}$ from $\hat{\mathbf{id}}$ below (that is, it points to an output of a component transaction); in particular, it does not receive values nor keys
- a list $(\hat{\mathbf{id}}, (\hat{\mathbf{v}}, \hat{\mathbf{pk}}, \hat{\boldsymbol{\chi}}))$ of output identifiers and triples which are either for adversarial keys or which are $(\bot, \bot, \hat{\chi}_i)$ if the output is for an honest user
- the total supply and fee $s, f \in [0, v_{\max}]$ and the number of transactions ℓ.

Remark. Since owners of addresses must be able to identify payments to them, privacy can only hold for addresses for which the adversary does not know the secret key (in particular, not even the view key), hence the simulator receives the output addresses (and the corresponding values) created by

the adversary. The simulator does not receive any amounts related to inputs or outputs for honest users, therefore the transaction (which is indistinguishable from a simulated one) does not reveal any such amounts (to a computationally bounded adversary); the simulator does not receive any destination address apart from the ones owned by the adversary, the transaction can therefore not reveal any honest recipients, moreover the simulator does not get the input/output matching of the constituent transactions, thus the aggregated transaction cannot be "disaggregated".

Theorem 4. *If the proof systems* RaP *and* PoP *are zero-knowledge and if DDH is hard in* $\mathbb{G}$*, then MW-NIT satisfies transaction privacy in the random oracle model.*

Proof. We first define the simulator. The simulator Sim takes as input: a list **in** of length n, whose elements are of the form txo_i or id_i; an $\hat{n}$-element list for the outputs with identifiers $\hat{\mathbf{id}}$ and elements the form $(\hat{v}_i, pk_i), \hat{\chi}_i)$ or $(\bot, \bot, \chi_i)$ together with their identifiers; total supply and fees s, f and the number of transactions ℓ.

Let $I_{\text{id}} \subseteq n$ denote the set of indices for which $in_i = id_i$, that is, the references to outputs of the transaction to be simulated (which will thus be cut through); for $i \in [n] \setminus I_{id}$, the entries in_i are explicit output.

- for each output index $i \in [\hat{n}]$ for which $(\hat{v}_i, \hat{pk}_i) \neq (\bot, \bot)$, sample $\hat{r}_i \leftarrow\!\$\ \mathbb{Z}_p$ and from them compute $\hat{R}_i = \hat{r}_i G$ and $\hat{P}_i$ (using $\hat{pk}_i = (\hat{A}_i, \hat{B}_i)$) and $\hat{C}_i$ (using $\hat{v}_i$) as prescribed in Send.
- for each $i \in [\hat{n}]$ with $(\hat{v}_i, \hat{pk}_i) = (\bot, \bot)$, pick random values $\hat{C}_i, \hat{R}_i, \hat{P}_i \leftarrow\!\$\ \mathbb{G}$
- for each $i \in [\hat{n}]$, simulate the range proofs $\hat{\pi}_i$ for statements $\hat{C}_i$, and the proofs ρ_i for $\hat{R}_i$ with tag $\hat{C}_i \| \hat{\pi}_i \| \hat{P}_i \| \chi_i$. Let $txo_i \coloneqq (\hat{C}_i, \hat{\pi}_i, \hat{R}_i, \hat{\rho}_i, \hat{P}_i, \hat{\chi}_i)$.
- for each input index $i \in [n]$, pick a random value $D_i \leftarrow\!\$\ \mathbb{G}$
- for each $i \in I_{\text{id}}$, let k_i be such that $in_i = id_{k_i} \in \hat{\mathbf{id}}$ (if there is none, abort); simulate ψ_i for the statement $(\hat{P}_{k_i}, D_i, txo_{k_i})$ where $\hat{P}_{k_i}, txo_{k_i}$ are defined above; let $I_{\text{ct}} \coloneqq \{k_i\}_{i \in I_{\text{id}}}$ be the indices of the cut-through outputs
- for each $i \in [n] \setminus I_{\text{id}}$, with $in_i =: (C_i, \pi_i, R_i, \rho_i, P_i, \chi_i)$, simulate the proof ψ_i for the statement (P_i, D_i, in_i)
- the input list of tx is $(P_i, D_i, \psi_i)_{i \in [n] \setminus I_{\text{id}}}$, ordered lexicographically
- the output list of tx is $\mathbf{txo} = (txo_i)_{i \in [\hat{n}] \setminus I_{\text{ct}}}$, ordered lexicographically
- the cut-through list **ct** are the (ordered) elements $(txo_{k_i}, \hat{P}_{k_i}, D_i, \psi_i)_{i \in I_{\text{id}}}$
- pick random values $E_2, \ldots, E_\ell, X_2, \ldots, X_\ell \leftarrow\!\$\ \mathbb{G}$, as well as $t, y \leftarrow\!\$\ \mathbb{Z}_p$
- set $E_1 \coloneqq \sum \hat{\boldsymbol{C}} - \sum \boldsymbol{C} + (f - s)H - tG - \sum_{i=2}^{\ell} E_i$
  and $X_1 \coloneqq \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{D} - yG - \sum_{i=2}^{\ell} X_i$ (note that we could have $\ell = 1$)
  For $i \in [\ell]$: simulate σ_i for (E_i, X_i) with tag ε
- set tx.sply $\coloneqq s$ and tx.fees $= f$

We now show that assuming indistinguishability of simulated RaP and PoP proofs and under the DDH assumption, a real transaction is indistinguishable from a simulated transaction. We start with a real transaction and consider a sequence of hybrid games that turns it into a simulated transaction.

H_0 In the "real" game, the component transactions are created as described (in Section 3.2) using the spending keys; they are then aggregated (Section 3.3).

H_1 In the first hybrid, all RaP proofs π and PoP proofs (cf. Section 5) ψ, ρ and σ_i are simulated. By the zero-knowledge property of both primitives, this change is indistinguishable from the honest computation of the range proofs and the proofs of possession.

H_2 For every output for an address that was generated by the address oracle, in the generation of the key shares in Equation (3), the argument $\hat{r}_{i,j} A_{i,j}$ of the hash function is replaced by a random value $Z_{i,j} \leftarrow\!\$\ \mathbb{G}$. This is indistinguishable by the DDH assumption, noting that all $\hat{R}_{i,j}$ and $A_{i,j}$ for

such outputs are created by the reduction and their logarithms are never used in the simulation of the game (the former because $\rho_{i,j}$ and σ_i are simulated; the latter because $\psi_{i,j}$ is simulated and the address oracle does not reveal any secret keys).

H_3 The game aborts if the adversary at some point queries $Z_{i,j}$ for some i, j to the random oracle. Since the adversary has no information on $Z_{i,j}$, the probability of aborting is negligible. Note that in H_3, the values $\hat{k}_{i,j}$ and $\hat{q}_{i,j}$ are uniformly random and independent.

Now that we have showed that the adversary's return value can only change negligibly between H_0 and H_3, it remains to argue that a transaction generated in H_3 is distributed equivalently to a transaction computed by the simulator.

In a call to Tx_0, let I_{ex} be the set of indices (i, j) for which $ref_{i,j}$ is an explicit input (and not an *id*) and let I_{adv} be the set of indices (i, j) for which $\hat{pk}_{i,j}$ was not generated by the oracle KEYGEN. In a transaction produced in H_3, for honest outputs, the coin openings $\hat{q}_{i,j}$, for $(i, j) \notin I_{\text{adv}}$, are uniformly random and independent. Thus, by the definition of $E_1, \ldots, E_\ell$, we have that for fixed values $(C_{i,j})_{(i,j)\in I_{\text{ex}}}$ (i.e., the inputs defined by the adversary) and $(\hat{C}_{i,j})_{(i,j)\in I_{\text{adv}}}$ (i.e., outputs for adversarial addresses), and $s \coloneqq \sum s_i$ and $f \coloneqq \sum f_i$, the tuple $\big((\hat{C}_{i,j})_{(i,j)\notin I_{\text{adv}}}, (E_i)_{i=1}^{\ell}, \sum t_i\big)$ is uniformly random conditioned on

$$\sum E_i = \sum\sum \hat{C}_{i,j} - \sum\sum C_{i,j} + (f - s)H - (\sum t_i)G \ .$$

This is exactly how the simulator produces these values.

Since for honest outputs the values $\hat{k}_{i,j}$, for $(i, j) \notin I_{\text{adv}}$, are uniformly random and independent, the corresponding values $\hat{P}_{i,j}$ are uniform and independent, as the simulator produces them. Finally, by the definition of the values $D_{i,j}$, $\hat{R}_{i,j}$ and X_i, the tuple $\big((\boldsymbol{D}_i)_{i=1}^{n}, (\hat{\boldsymbol{R}}_i)_{i=n^*+1}^{\hat{n}}, (X_i)_{i=1}^{\ell}, \sum y_i\big)$ is uniformly random conditioned on

$$\sum X_i = \sum\sum \hat{R}_{i,j} - \sum\sum D_{i,j} - (\sum y_i)G \ .$$

Again, this is how the simulator generates these values. Finally, in a transaction produced in H_3, all RaP and PoP proofs are simulated, which is how the simulator generates them, and the outputs/input pairs in the cut-through list of the simulator are those that Agg would put there. □

Acknowledgements. The first author is supported by the Vienna Science and Technology Fund (WWTF) through project VRG18-002. We would like to thank MWC and David Burkett for the fruitful collaboration; we are also grateful to the anonymous reviewers of ASIACRYPT'22 for their helpful comments.

References

BBB+18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE S&P 2018*, pages 315–334. IEEE, 2018

Bur20. David Burkett. Offline transactions in Mimblewimble, 2020. https://gist.github.com/DavidBurkett/32e33835b03f9101666690b7d6185203.

Bur21. David Burkett. One-sided transactions in Mimblewimble (consensus layer), 2021. https://github.com/DavidBurkett/lips/blob/master/lip-0004.mediawiki.

Dev20a. Grin Developers. Grin documentation: Intro, 2020. Availalable at: https://github.com/mimblewimble/grin/blob/master/doc/intro.md.

Dev20b. Grin Developers. Grin documentation: Mimblewimble, 2020. Availalable at: https://docs.grin.mw/wiki/introduction/mimblewimble/mimblewimble/#kernel-offsets.

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. *CRYPTO 2018, Part II*, *LNCS* 10992, pages 33–62. Springer, 2018.

FOS19. Georg Fuchsbauer, Michele Orrù, and Yannick Seurin. Aggregate cash systems: A cryptographic investigation of Mimblewimble. *EUROCRYPT 2019, Part I*, *LNCS* 11476, pages 657–689. Springer, 2019.

FPS20. Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind Schnorr signatures and signed ElGamal encryption in the algebraic group model. *EUROCRYPT 2020, Part II*, *LNCS* 12106, pages 63–95. Springer, 2020.

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

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.

Nak08. Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2008. Available at http://bitcoin.org/bitcoin.pdf.

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

PS00. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.

Seu12. Yannick Seurin. On the exact security of Schnorr-type signatures in the random oracle model. *EUROCRYPT 2012*, *LNCS* 7237, pages 554–571. Springer, 2012.

Tod14. Peter Todd. Stealth addresses, 2014. http://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg03613.html.

VFV17. Shaileshh Bojja Venkatakrishnan, Giulia C. Fanti, and Pramod Viswanath. Dandelion: Redesigning the bitcoin network for anonymity. *CoRR*, abs/1701.04439, 2017.

vS13. Nicolas van Saberhagen. CryptoNote v 2.0, 2013. https://cryptonote.org/whitepaper.pdf.

Wag02. David Wagner. A generalized birthday problem. *CRYPTO 2002*, *LNCS* 2442, pages 288–303. Springer, 2002.

Yu20. Gary Yu. Mimblewimble non-interactive transaction scheme. Cryptology ePrint Archive, Report 2020/1064, 2020. https://ia.cr/2020/1064.