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 we show that it is flawed. Building on Yu and integrating ideas from Burkett, we give a fixed scheme and provide a rigorous security analysis in a strenghtening of 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 containing *commitments* to the amounts of the inputs and outputs and giving proofs that the sum of the input values equals the sum of the output values, showing that 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 directly 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

[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

removed, the 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 by 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. 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. FOS 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 (denoted by "$\|$") of inputs and outputs, that is,

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

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

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

Outputs in one transaction that also appear as inputs in the other cancel out in Equation (0) for their aggregation tx. They can therefore 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] also remark that if the used signature scheme supports aggregation, then $\sigma_1 \| \sigma_2$ can be replaced by their aggregation to further save 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 that come from the same component transaction, because tx contains an excess value E_i 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) then contains $(E_1 \| E_2, t_1 + t_2)$. 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.

We stress that 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. The key $\sum \hat{\boldsymbol{q}} - \sum \boldsymbol{q}$ for the signature σ contained in a transaction depends on the openings of the receiver's and the sender's coins. Most implementations of MW therefore create new transactions via an *interactive* protocol between sender and receiver to produce σ.[8] FOS [FOS19] proposed a transaction protocol, where the sender creates all output coins, and so 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 outputs. The receiver 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 $R \coloneqq rG$, 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 protocol itself has no concept of addresses. Yu's proposal [Yu20] builds on top of Burkett's [Bur20], and after receiving feedback from the community, Yu further updated it with notes describing possible attacks and countermeasures. In essence, the idea is to extend a MW transaction output (C, π) by a one-time key P chosen by the sender, as well as an ephemeral key R that allows the receiver to compute the secret key for P (and when spending the output, a signature under signature-verification key P is required). To prevent the value P in a transaction from being modified (which would steal the output from the receiver), a signature ρ on P under key R (of which the sender knows the logarithm) is added. An output is thus of the form (C, π, R, ρ, P).

The mechanism for letting the receiver derive the secret key for P can also be used to let the receiver derive the opening q of the commitment C (and 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).

Commonly 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, which can therefore not be signed. Yu proposes to use the logarithms of the values P in the inputs and of the values R in the outputs, denoted $\hat{R}$ for outputs, to "authenticate" the spending by proving knowledge of the logarithm of $\sum \hat{\boldsymbol{R}} - \sum \boldsymbol{P}$ (similarly to how the openings of the input coins authenticate the output coins via Equation (0) in MW).

Yu proposes to simply arrange the $\hat{\boldsymbol{R}}$ values so that the above difference equals 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 adding one signature per transaction input, namely under the P value. The security and correctness of this proposed change are not argued 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]). As we show, cut-through enables attacks by miners, who could modify outputs of transactions (and violate our notion of *transaction-binding*, see below). This is however only relevant for *recent* transactions; once a transaction is in a block beyond the horizon, it is "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 the found issues. (In Section 4 we discuss further issues that emerged after the publication of [Yu20].) A variant of our protocol is now already being used by Litecoin in its MimbleWimble Extension Blocks.

Model. We propose a security model that strengthens the one from [FOS19], which did not protect against a malleability attack by miners and had weak privacy guarantees (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) stealing someone else's output in the ledger *(theft resistance)*
(iii) changing an output of a transaction not yet included in the ledger *(transaction-binding)*
(iv) 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 resistance* are straightforward adaptations of the notions from [FOS19, Def. 10 and 11], which become simpler for non-interactive transactions. *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 coins $\boldsymbol{C}$ and creating change $\boldsymbol{C}'$ for herself is guaranteed that $\boldsymbol{C}'$ are included in the ledger as soon as *any* of the coins from $\boldsymbol{C}$ 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 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

We let ε denote the empty string and $[a]$ the set $\{1, \ldots, a\}$ (for $a \in \mathbb{N}$). 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 literature [PS00,Seu12]. 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 $s \cdot G = R + c \cdot X + d \cdot Y$ (see also Figure 3, page 16).

We also use PoP to prove knowledge of a witness $x \in \mathbb{Z}_p$ for statements $(X = xG, m)$ by performing 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 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 prover's 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

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

A commitment is opened by sending v and q and verified by checking $C \stackrel{?}{=} vH + qG$. 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 its mind about a committed value, that is, find a commitment and two openings (v, q), (v', q') with $v \neq v'$ to it).

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

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

We assume a zero-knowledge proof-of-knowledge system RaP for the above language. (Note that we do not assume RaP to be simulation-sound [FOS19, Def. 8], whereas FOS required this in their proof of theft resistance [FOS19, Theorem 14]; our scheme could thus be potentially instantiated with a more efficient range proof system.) 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, and it is secure under DL in the ROM).

3 Proposal for MW with non-interactive transactions

We start with presenting the scheme and then discuss the rationale for our design choices, such as adding *stealth excesses* (Section 4.1), *doubling keys* (which prevent a sub-exponential-time attack; Section 4.2) and *epochs* (Section 4.4).

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 from $\mathbb{Z}_p$ as
    $$(\hat{k}, \hat{q}) \coloneqq \mathrm{H}(\hat{r}A)$$
    (note that $(\hat{k}, \hat{q})$ can be computed from $\hat{R}$ and the view key, since $\hat{r}A = a\hat{R}$)
  - a **commitment** $\hat{C} \coloneqq \hat{v}H + \hat{q}G$ to the output 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 $\hat{C} \| \hat{\pi} \| \hat{P} \| \hat{\chi}$ (i.e., a proof for statement $(\hat{R}, \hat{C} \| \hat{\pi} \| \hat{P} \| \hat{\chi})$)
  - 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

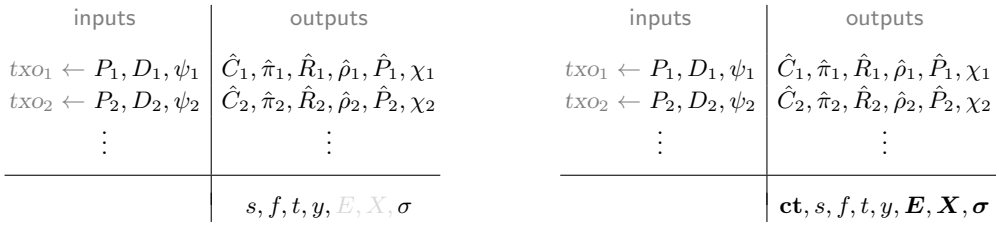

Fig. 1. Left: Visualization of a (simple) MW-NIT transaction. Inputs consist of a value P_i contained in a previous transaction output, a "doubling key" D_i and a proof of possession (PoP) ψ_i of P_i and D_i. Outputs consist of a commitment $\hat{C}_i$ to their value, an associated range proof $\hat{\pi}_i$, 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 $\mathbf{ct}$, and lists of excesses, stealth excesses and corresponding PoPs.

– 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 $p \coloneqq b' + k$.

To create a transaction that in epoch $\hat{\chi}$ spends transaction outputs txo_i of values v_i with 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 $\hat{v}_1, \ldots, \hat{v}_{\hat{n}}, s, f \in [0, v_{\max}]$ and $\sum \hat{\mathbf{v}} + f = \sum \boldsymbol{v} + s$ (if this is not the case, then abort), 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

- if $C_i \neq v_i H + q_i G$ (with C_i from txo_i) or if $P_i \neq p_i G$ then abort
- sample a random $d_i \leftarrow\!\!\$\ \mathbb{Z}_p$ and define $D_i \coloneqq d_i G$
- 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 $i \in [\hat{n}]$:
- sample a random $\hat{r}_i \leftarrow\!\!\$\ \mathbb{Z}_p$ and define ephemeral key $\hat{R}_i \coloneqq \hat{r}_i G$
- 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 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 a 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 a 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), their aggregation tx is computed 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 **ker** as the concatenation of their lists of excesses, stealth excesses and associated signatures σ.

– if the same value P appears in two entries of $\mathbf{txi} \,\|\, \mathbf{ct}$, or if the same value P appears in two entries of $\mathbf{txo} \,\|\, \mathbf{ct}$, then abort
– if a P value appears in an entry (P_i, D_i, ψ_i) of $\mathbf{txi}$ and in an entry txo_j of $\mathbf{txo}$, do the following: if $\mathsf{PoP.V}((P_i, D_i, txo_j), \psi_i) = \mathbf{true}$ then remove (P_i, D_i, ψ_i) from $\mathbf{txi}$, remove txo_j from $\mathbf{txo}$ and include $txo_j \,\|\, (P_i, D_i, \psi_i)$ in $\mathbf{ct}$; else abort.
– sort each list $\mathbf{txi}$, $\mathbf{txo}$, $\mathbf{ct}$ and $\mathbf{ker}$ lexicographically (required to hide which inputs and outputs come from which transaction)
– compute the aggregated supply, fee, offset, and stealth offset (from the supplies s_i, etc., of tx_i)

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

– 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 $(k, q) := \mathrm{H}(a\hat{R})$, and accept txo if it has not been previously received in epoch $\hat{\chi}$ and if

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

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 $\mathbf{txo}$ with $txo_i = (C_i, \pi_i, R_i, \rho_i, P_i, \chi_i)$, for $i \in [n]$, if

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

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

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

Aggregate transactions. An aggregate transaction

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

as in (8), with $\mathbf{txi} = (P_i, D_i, \psi_i)_{i\in[n]}$, $\widehat{\mathbf{txo}} = (\hat{C}_i, \hat{\pi}_i, \hat{R}_i, \hat{\rho}_i, \hat{P}_i, \hat{\chi}_i)_{i\in[\hat{n}]}$, $\mathbf{ct} = (txo'_i \| txi'_i)_{i\in[n']}$ is verified w.r.t. previous outputs $\mathbf{txo}$ as follows:

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

$$\mathbf{txo}^* := \mathbf{txo} \,\|\, \mathbf{txo}' = (C_i, \pi_i, R_i, \rho_i, P_i, \chi_i)_{i\in[n+n']}$$
$$\mathbf{txi}^* := \mathbf{txi} \,\|\, \mathbf{txi}' = (P_i, D_i, \psi_i)_{i\in[n+n']}$$
$$\widehat{\mathbf{txo}}^* := \widehat{\mathbf{txo}} \,\|\, \mathbf{txo}' = (\hat{C}_i, \hat{\pi}_i, \hat{R}_i, \hat{\rho}_i, \hat{P}_i, \hat{\chi}_i)_{i\in[\hat{n}+n']}$$

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

(v′) for all $i \in [\bar{n}] : \mathsf{PoP.V}((E_i, X_i, \varepsilon), \sigma_i) = \mathbf{true}$
(vi′) 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 $\hat{\boldsymbol{C}}$ only up to $\hat{n}$ and the $\boldsymbol{C}$ only 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) that has no inputs (as the inputs of any transaction added to Λ must spend existing outputs, these are moved to the cut-through list).

A transaction tx of the form (7) or (8) is included in Λ by aggregating (as defined in Section 3.3) Λ and tx to Λ', checking that Λ' has no inputs (thus all inputs of tx were spent/cut through) and checking validity (as defined in Section 3.5) of Λ' w.r.t. an empty list **txo**. If any of the checks fail, return $\perp$, otherwise Λ'. (If Λ is known to be valid, it suffices to identify, for every input (P_j, D_j, ψ_j) of tx, 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 discuss the main issues found with Yu's scheme, 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 made true 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 then set $\hat{R}_i := E + tG + \sum \boldsymbol{P} - \sum_{j \neq i} \hat{R}_j$ (for which she knows $\log \hat{R}_i$) and 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 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

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

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

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 spending one one output txo_1 and txo_2, resp., and creating one output each, transferring v_1 to Bob and $v_2 > v_1$ to Charlie (all supplies and fees are 0):

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

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

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

$$txo_2 \quad \leftarrow \quad \boxed{P_2, \psi_2 \;\middle|\; \begin{matrix} \dots, \hat{R}_1, \hat{\rho}_1, \dots \\ C^*, \pi^*, R^*, \rho^*, \dots \end{matrix}}_{t_1, y^*, E_1, X_1, \dots} \tag{tx^*}$$

It combines the input of tx_2 and the output and the excesses from tx_1. A second output is computed "honestly" by choosing $r^* \leftarrow\!\!\$\ \mathbb{Z}_p$, setting $R^* = r^*G$ and signing any value P^* (knowing $\log P^*$) as ρ^*. The miner also creates the corresponding coin C^*, so that tx^* satisfies Equation (11), 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 Al3ice's view key, or *by having sent the outputs* txo_1 *and* txo_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:

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

As per Equation (5), P_1 and P_2 are defined as $P_i = B + k_i G$, where k_i is obtained as in Equation (3). The crucial observation now is that if the miner knows the values k_1 and k_2 (which it can either obtain from Alice's view key or by being the creator txo_1 and txo_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 one-time key P^*.

Fixes. To prevent this attack, our first fix was to derive the one-time keys multiplicatively, that is, setting $P_i = k_i B$ 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, we showed that as long as the adversary cannot find distinct hash function outputs $k_{1,1}, \dots, k_{1,n_1}, k_{2,1}, \dots, k_{2,n_2}$ 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 for sufficiently large n_1 and n_2. To be protected against active adversaries ready to invest substantial computing power, a user would therefore need to limit the number of her pending transactions at any point in time (which could degrade scalability of the system). To overcome this downside, we follow David 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 satisfies transaction-binding in Section 6.5.

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 whether, 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 sum of all removed $\hat{R}$ values minus all removed 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. The result is 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. We illustrate the attack against a scheme that allows cut-through and includes a value Z to compensate for the removed terms. Consider two 1-input/1-output transactions tx_2 and tx_3 where tx_2 spends some output txo_1, creating one output, which is then spent by tx_3 (assume that all supplies and fees are 0).

$$txo_1 = (C_1, \dots) \quad \leftarrow \quad \boxed{P_1, D_1, \psi_1 \mid C_2, \pi_2, R_2, \rho_2, P_2, \chi_2}_{t_2, y_2, E_2, X_2, \sigma_2} \qquad (\mathsf{tx}_2)$$

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

(Note that 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, π_3, ρ_3, σ_2 and σ_3 are valid and the following holds:

$$C_3 - C_1 = E_2 + E_3 + (t_2 + t_3)G$$
$$R_3 - D_1 + Z = X_2 + X_3 + (y_2 + y_3)G$$

Then a miner could simply choose r^*, p^*, χ_3^* set $R_3^* := r^*G$, $P_3^* := p^*G$, create ρ_3^* honestly, define $Z^* := X_2 + X_3 + (y_2 + y_3)G - R_3^* + D_1$ and create the (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:

$$txo_1 = (C_1, \dots) \quad \leftarrow \quad \boxed{P_1, D_1, \psi_1 \mid C_3, \pi_3, R_3, \rho_3, P_3, \chi_3}_{\mathbf{ct}, t_2+t_3, y_2+y_3, (E_2,E_3), (X_2,X_3), (\sigma_2,\sigma_3)}$$

where $\mathbf{ct} = (R_2, P_2, D_2, \psi_2)$ and ψ_2 is a valid proof for P_2, D_2 (note that the tag for proofs ψ must change to account for removed information).

Intuitively, not having ρ_2 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 p_2^*, d_2^* and r_3^*, sets $D_2^* \coloneqq d_2^* G$ and $R_3^* \coloneqq r_3^* G$, computes ψ_2^* and ρ_3^* honestly and sets $X_2^* \coloneqq (r_3^* - d_2^* - y_2)G$ and computes σ_2^* honestly. It also chooses p_3^*, sets $P_3^* \coloneqq p_3^* G$, 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

$$txo_1 = (C_1, \dots) \quad \leftarrow \quad \boxed{P_1, D_1, \psi_1 \mid C_3, \pi_3, R_3^*, \rho_3^*, P_3^*, \chi_3^*}_{\mathbf{ct}^*, t_2+t_3, y_2+y_3, (E_2,E_3), (X_2^*,X_3^*), (\sigma_2^*,\sigma_3^*)}$$

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:

$$txo_1 = (C_1, \dots) \quad \leftarrow \quad \boxed{P_1, D_1, \psi_1 \mid C_3, \pi_3, R_3, \rho_3, P_3, \chi_3}_{\mathbf{ct}, t_2+t_3, y_2+y_3, (E_2,E_3), (X_2,X_3), (\sigma_2,\sigma_3)}$$

where $\mathbf{ct} = (C_2, \pi_2, R_2, \rho_2, P_2, \chi_2, D_2)$; note that now C_2, π_2 and χ_2 need to be kept for ρ_2 to be verifiable. Consider an honest transaction tx_2

$$txo_1 = (C_1, \dots) \quad \leftarrow \quad \boxed{P_1, D_1, \psi_1 \mid C_2, \pi_2, R_2, \rho_2, P_2, \chi_2}_{t_2, y_2, E_2, X_2, \sigma_2}$$

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 $D_2^* \coloneqq R_2^* + R_3 - D_1 - X_2 - X_3 - y_2 G$. 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 transaction

$$txo_1 = (C_1, \dots) \quad \leftarrow \quad \boxed{P_1, D_1 \mid C_3, \pi_3, R_3, \rho_3, P_3, \chi_3}_{\mathbf{ct}^*, t_2+t_3, y_2, (E_2,E_3), (X_2,X_3), (\sigma_2,\sigma_3)}$$

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

$$R_2^* + R_3 - D_1 - D_2^* = X_2 + X_3 + y_2 G$$

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.

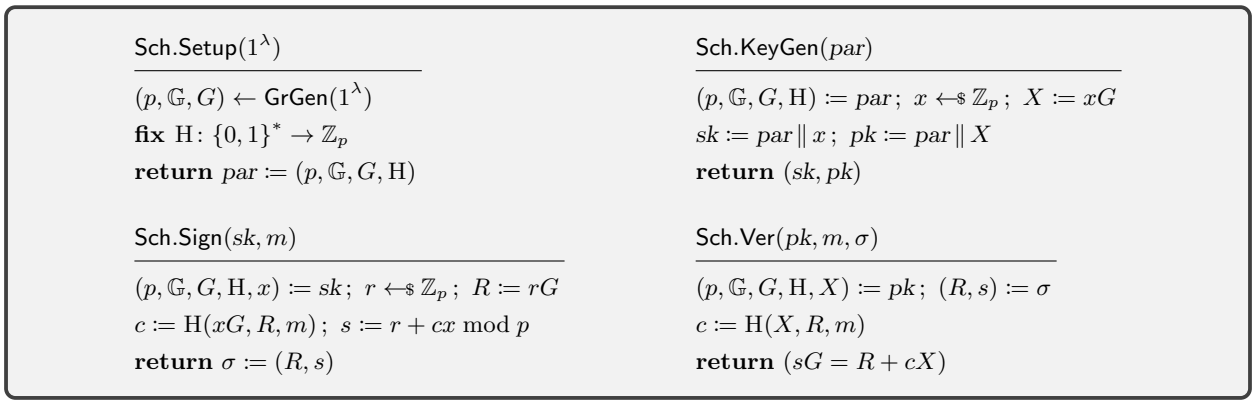

Fig. 2. Key-prefixed Schnorr signature scheme PoP[GrGen] based on a group generator GrGen

A simple defense against 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), 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, §2.9.2].

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 a 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 minimize 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 relation

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

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 (ideal models jointly used also 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 $(\log X^*, \log Y^*)$

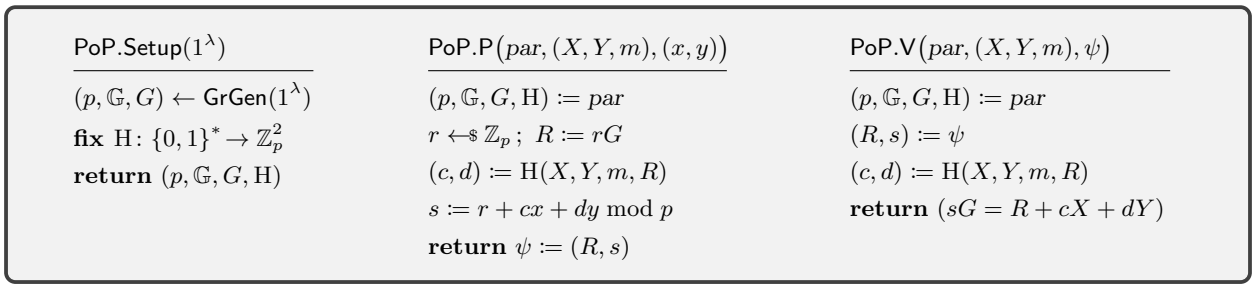

Fig. 3. Zero-knowledge simulation-extractable Schnorr proof of two logarithms PoP[GrGen] based on a group generator GrGen

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

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

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

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

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

A *transaction* tx has three attributes: a *supply* $\mathsf{tx.sply}$ specifying the amount of money it creates, the *fee* $\mathsf{tx.fees}$ paid to miners 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 $\perp$ 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 $\perp$. 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 $\Lambda' := \perp$;

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 $[0, v_{\max}]$, $\mathbf{vk}'_k$ a vector of elements from $\mathbf{vk}$ (i.e., $\mathbf{vk}'_k = (vk_{j_i})_i$ for some j_i) and let $\mathbf{sk}'_k$ be the corresponding spending keys (i.e., $\mathbf{sk}'_k = (sk_{j_i})_i$). For all $k \in [n]$, define $\mathbf{txo}_k \leftarrow \mathsf{Rcv}(pp, \mathsf{tx}'_k, \mathbf{v}'_k, \mathbf{vk}'_k)$.

Let $(txo_i)_i := \mathbf{txo}_1 \,\|\, \ldots \,\|\, \mathbf{txo}_n$ and let I be an index set s.t. $txo_i \neq \perp$ for all $i \in I$. Consider $\hat{\mathbf{v}} \in [0, v_{\max}]^*$, $\hat{\mathbf{pk}}$ consisting of elements from $\mathbf{pk}$, some $\boldsymbol{\chi}$ and $s, f \in [0, v_{\max}]$ such that $\sum_{i \in I} v'_i = \sum \hat{\mathbf{v}} + f - s$. Then for $\mathsf{tx} \leftarrow \mathsf{Send}(pp, ((txo_i)_{i \in I}, (v'_i)_{i \in I}, (sk'_i)_{i \in I}), (\hat{\mathbf{v}}, \hat{\mathbf{pk}}, \boldsymbol{\chi}), s, f)$ 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 $\perp \notin \mathbf{txo}$, 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. 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].

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 that accept the outputs of tx^*. Letting s denote the ledger supply, s^* the supply and f^* the fee of tx^*, the adversary wins if

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

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

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

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

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

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

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

Fig. 4. Game INFL for inflation resistance

The game is formalized in Figure 4.

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

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

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.

The reduction extracts all values v_i and openings q_i for all the output commitments C_i in Λ^* 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 $\hat{C}_i = \mathsf{Cmt}(\hat{v}_i, \hat{q}_i)$. Since Λ^* accepted tx^*, all values P'_i are in the ledger, that is, $\boldsymbol{P'} = (P_i)_{i\in I}$ for some $I \subseteq \mathbb{N}$. From validity of tx^* we further have

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

From σ^*_i contained in tx^*, the reduction can extract $e^*_i := \log E^*_i$ and, analogously to the above, from (26) we get

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

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

$(pp, \Lambda) \leftarrow \mathsf{Setup}(1^\lambda, v_{\max})$
$j := 0$; $Hon, Archv := ()$
$\mathcal{A}^{\textsc{Keygen},\textsc{Send},\textsc{Ledger}}(pp, \Lambda)$
return $(Hon \not\subseteq \Lambda.\mathsf{out})$

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

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 $\textsc{Keygen}()$

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

Oracle $\textsc{Ledger}(\mathsf{tx}, \hat{\mathbf{v}}, (i_j)_j)$

$\Lambda' := \mathsf{Ldgr}(pp, \Lambda, \mathsf{tx})$
if $\Lambda' = \perp$ **then return** $\perp$ **else** $\Lambda := \Lambda'$
// Check for new honest outputs in tx:
$\mathbf{txo} \leftarrow \mathsf{Rcv}(pp, \mathsf{tx}, \hat{\mathbf{v}}, (vk_{i_j})_j)$
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 THFT for theft resistance

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. □

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 following game, which is an adaptation of [FOS19, Fig. 8].

Honest users are simulated by the experiment and the adversary can create new users by calling $\textsc{Keygen}$; the adversary can instruct users spend outputs they own (stored in the list *Hon*) by calling $\textsc{Send}$; moreover it can submit any transactions to the ledger using its oracle $\textsc{Ledger}$. If the transaction contains outputs that are accepted by honest users, these are added to *Hon*. As

[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.

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*; if an output is already in *Archv*, then LEDGER does not add it to *Hon*. 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. The game is formalized in Figure 5.

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 particular, this involves an instruction for the honest users 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 view key (a, 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 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'.

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^*. By the definition of the security game, the honest user has never spent txo before. This means that the reduction has never simulated a proof for (P, D^*, txo). Therefore, by simulation-extractability (SE) of PoP, 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^*.

Note that if extraction fails, we can construct a reduction that breaks SE of PoP, which is algebraic if the adversary is: from the adversary's representation of (P, D^*), the reduction against PoP can derive a representation in basis (G, B^*), where B^* is the auxiliary group element input; likewise, it can give representations $P' = B^* + k'G$ and $D' = d'G$ of the elements for which it queries simulated proofs. □

6.5 Transaction-binding

While the previous notion states that once a user owns an output in the ledger it cannot be stolen, 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 (except for the owners of the outputs). Since transactions can be aggregated, further inputs and outputs can also be added to the original transaction. Thus, while *theft-resistance* protects the outputs of a transaction, *transaction-binding* protects the inputs.

Definition. We define transaction-binding via the following game. The experiment simulates all honest users, which the adversary can create by calling KEYGEN, which creates a new address,

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

$(pp, \Lambda) \leftarrow \mathsf{Setup}(1^\lambda, v_{\max})$
$j := 0$; $HTxs := ()$ // honest transactions
$STxo := ()$ // spent outputs
$(\Lambda, \mathsf{tx}^*, \widehat{txo}, \hat{v}, i) \leftarrow \mathcal{A}^{\textsc{Keygen},\textsc{Send},\textsc{Ledger}}(pp, \Lambda)$ // Λ should contain the TXOs spent by tx^*
$\Lambda^* := \mathsf{Ldgr}(pp, \Lambda, \mathsf{tx}^*)$
return $\big(\exists\, \mathsf{tx} \in HTxs : \mathsf{Ldgr}(pp, \Lambda, \mathsf{tx}) \neq \bot$ // tx accepted by Λ, ...
and $\Lambda^* \neq \bot$ **and** $\mathsf{Ldgr}(pp, \Lambda^*, \mathsf{tx}) = \bot$ // ... but not after tx^* was (accepted and) added
and $\widehat{txo} \in \mathsf{Rcv}(pp, \mathsf{tx}, \hat{v}, vk_i)$ // $\widehat{txo}$ belongs to an honest user, ...
and $\widehat{txo} \notin STxo$ **and** $\widehat{txo} \notin \mathsf{tx}^*.\mathsf{out}\big)$ // ... and was not spent, and it is not in tx^*

Oracle $\textsc{Keygen}()$

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

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

// Honest users never spend the same txo twice:
if $\mathbf{txo} \cap STxo \neq \emptyset$ **then return** $\bot$
$\mathsf{tx} \leftarrow \mathsf{Send}(pp, (\mathbf{txo}, \mathbf{v}, (sk_i)_{i \in I}), (\hat{\mathbf{v}}, \hat{\mathbf{pk}}, \hat{\boldsymbol{\chi}}), s, f)$
$HTxs := HTxs \,\|\, \mathsf{tx}$; $STxo := STxo \,\|\, \mathbf{txo}$
return tx

Fig. 6. Game TXBND for transaction-binding

for which the adversary receives the view key. The adversary can instruct honest users to spend outputs of the adversary's choice to addresses of the adversary's choice; 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 *one of the outputs in* tx *for an honest user is missing in* tx^* (and that user was not instructed to spend it).

We formalize this by requiring the adversary to 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, so that their view keys accept tx but do not accept tx^* (thus one of the outputs of tx is missing in tx^*). The game is formalized in Figure 6.

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 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 of tx have been replaced by new ones. Thus, we do 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 prove, we start with a simplified scenario. Consider an adversary $\mathcal{A}$ that creates two users via $\textsc{Keygen}$, computes an output $txo = (C, \pi, R, \rho, P, \chi)$ for User 1 and calls $\textsc{Send}$ so txo is spent to User 2, which creates a transaction

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

$\mathcal{A}$ makes no further oracle calls and returns a transaction $\mathsf{tx}^* = ((P, D^*, \psi^*), txo^*, (s^*, f^*, t^*, y^*, \sigma^*))$ that spends txo and creates an output $txo^* = (C^*, \pi^*, R^*, \rho^*, P^*, \chi^*) \neq \widehat{txo}$ (so tx^* is a *simple*, i.e., non-aggregate transaction).

We first show that conditions (I)–(III) below must hold with overwhelming probability, and in (IV) we show that when (I)–(III) hold, $\mathcal{A}$ can only win with negligible probability. We show all four claims by reducing the DL problem to them, assuming strong simulation-extractability (SE) of the proof-of-possession system PoP.

(I) $D = D^*$. Assume $D \neq D^*$. By SE of PoP, this can be used to break DL by simulating the game as in the proof of theft-resistance (Theorem 2):
Given a DL challenge B, the reduction embeds it into User 1's address (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 $\mathcal{A}$ outputs a transaction with input (P, D^*, ψ^*), the witness (p, d^*) can be extracted from ψ^*, as the only simulated proof was for a different statement (P, D, txo). Since $P = B + kG$, with k derived from User 1's view key per Equation (3), the reduction can return $\log B = p - k$.
(II) $R^* \neq \hat{R}$. Assume $R^* = \hat{R}$. Since $txo^* \neq \widehat{txo}$, 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 User 1, $\log R^*$ can be extracted by strong SE ("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 $(\hat{k}, \hat{q}) = H(a'\hat{R})$ from User 2's view key (a', B') (who 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 the definition of stealth excesses in Equation (2) we get $R^* - D^* - y^*G = \hat{R} - D - yG$. By (I) we have $D^* = D$ and 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 σ to compute tx. Since, $R^* \neq \hat{R}$ by (II), the statement of ρ^* is different from that of the simulated proof $\hat{\rho}$ and thus the reduction can extract $r^* = \log R^*$ from ρ^*. If $X^* = X$ then it can compute $\log \hat{R} = r^* + y - y^*$.
(IV) Finally, we show that $\mathcal{A}$ cannot win the game when (I)–(III) hold. 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 $r^* := \log R^*$, 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 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 simulated), and from σ^* the reduction extracts x^* (the simulated proof σ was under $X \neq X^*$) and returns $r^* - y^* - x^* = \log D$.

Proof (of Theorem 3). We generalize the above arguments, now considering the actual security game. In particular, there can now be many honest transactions created by the game and the adversary's output tx^* can be an aggregated transaction, whose stealth excesses might contain X values from different honest transactions. Moreover, tx^* can contain cut-through terms (in particular, the "stolen" output $\widehat{txo}$ could be in the cut-through of tx^*).

Let tx^* and $\widehat{txo}$ be the adversary's outputs and tx be the attacked transaction, that is, tx and tx^* have an input in common, and txo is an output of tx (accepted by an honest user) but not of tx^*. We proceed by a sequence of hybrid games, each one introducing new abort conditions which make the game return 0. The conditions in H_1 correspond to (I) above and $\widehat{txo}$ being in the cut-through of tx^*. The conditions in H_2 correspond to (II) and (III), which we combine in the same hybrid as 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^* spends the same txo as an honest transaction, i.e., $txo \in HTxs$, but the associated D value is different (abort condition (A1a)); or if $\widehat{txo} \in \mathbf{ct}^*$, the cut-through of tx^* (A1b).

H_2 In addition, the second hybrid aborts if either the value $\hat{R}$ in $\widehat{txo}$ appears in tx^* (that is, either in an output or in its cut-through) (A2a); or $\hat{R}$ does not appear in tx^* and among the stealth excesses of tx^* is the stealth excess X of tx (A2b).

Note to win H_2, tx^* must share an input with tx (the attacked transaction), and moreover the corresponding D values in tx^* and tx must be the same (otherwise the game would abort because of (A1a)). Also note that if tx^* contains the stealth excess X of tx then H_2 returns 0: if it does not abort in (A2b) then $\hat{R}$ must appear in tx^*, and thus it aborts in (A2a). Therefore, if the adversary wins H_2 then

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

(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 (A1a) or (A1b) occurring, assuming simulation-extractability (SE) of PoP.

Given a DL challenge $Z \in \mathbb{G}$, the reduction embeds it into the addresses of *all* honest users: it picks $a_j, b_j \leftarrow\!\!\$\ \mathbb{Z}_p$ and defines the j-th user's address as $(a_jG, B_j := Z + b_jG)$. Note that this is correctly distributed and the corresponding view key (a_j, B_j) can be given to the adversary. When the adversary calls SEND, 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 $(txo_i)_i = STxo$ be the outputs queried to SEND and let (P_i, D_i, ψ_i) be the corresponding inputs of the transactions created by SEND. Thus, ψ_i are the simulated proofs for (P_i, D_i, txo_i). Since SEND never accepts the same txo twice, all txo_i are distinct.

(A1a). Let tx^* be the transaction returned by the adversary, and assume (A1a) occurs, i.e., tx^* spends txo_{i^*} for some i^*, and the corresponding input is of the form (P_{i^*}, D^*, ψ^*) with $D^* \neq D_{i^*}$. Since $(P_{i^*}, D^*, txo_{i^*}) \neq (P_{i^*}, D_{i^*}, txo_{i^*})$ and all txo_i are distinct, we have $(P_{i^*}, D^*, txo_{i^*}) \neq (P_i, D_i, txo_i)$ for all i. This means that no proof for $(P_{i^*}, D^*, txo_{i^*})$ has been simulated and by SE of PoP, the reduction can extract the witness (p_{i^*}, d^*) from ψ^*. Let j be the index of the user that spent txo_{i^*} (i.e., the value in I associated to Tx_{i^*} in the call to SEND); then $P_{i^*} = B_j + kG$ for k derived the view key (a_j, B_j) as per (3). Since $B_j = Z + b_jG$, we get $\log Z = p_{i^*} - b_j - k$, the solution of the DL challenge.

(A1b). Now assume that (A1b) occurs, that is, $\widehat{txo}$, the value returned by the adversary, is an output of an honest transaction tx, and $\widehat{txo} \notin STxo$ (otherwise the adversary did not win), and the cut-through of tx^* contains $\widehat{txo} \,\|\, (\hat{P}, D^*, \psi^*)$ for $\hat{P}$ in $\widehat{txo}$ and some D^*, ψ^*. Since P values are derived from R values, which are uniform, the probability that $\hat{P}$ is contained in some $txo_i \in STxo$ is negligible. As the entries in $STxo$ are precisely those for which ψ proofs were simulated and $\widehat{txo} \notin STxo$, the reduction can extract from ψ^* the witness $(\hat{p}, d^*)$. Since the i-th user (with i from the adversary's output) accepted $\widehat{txo}$, we have $\hat{P} = B_i + \hat{k}G$ for $\hat{k}$ derived from (a_i, B_i) as per (3). Since $B_i = Z + b_iG$, the reduction can thus again compute $\log Z = \hat{p} - b_i - \hat{k}$.

Hybrid H_1 to H_2. We next show that in a winning run of H_2 the abort conditions (A2a) and (A2b) can only occur with negligible probability. If any of them occurs, then there exist i and $\hat{\imath}$ so that tx will be the i-th transaction and its $\hat{\imath}$-th output will be $\widehat{txo}$, and one of the following hold:
(A2a) $\hat{R}$ from $\widehat{txo}$ is contained in some txo^* in tx^* (either in an output or in its cut-through list $\mathbf{ct}^*$). (Note that we have $\widehat{txo} \neq txo^*$: otherwise, if $\widehat{txo}$ is an output then the adversary did not win, and if $\widehat{txo}$ is in $\mathbf{ct}^*$ then the game would have aborted because of (A1b)); or

(A2b) $\hat{R}$ from $\widehat{txo}$ does not appear in tx^* and the stealth excess X of tx is among the stealth excess list of tx^*.

We define a reduction that makes a *guess* $(i, \hat{\imath})$, so that if the adversary wins H_1 but not H_2 (i.e., either (A2a) or (A2b) occur) and the guess is correct, then it breaks DL.

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

$$(k_{\hat{\imath}}, q_{\hat{\imath}}) \coloneqq \mathrm{H}(a_j R_{\hat{\imath}})$$

and, from this, $C_{\hat{\imath}}$, $P_{\hat{\imath}}$ and $\pi_{\hat{\imath}}$ as prescribed. It simulates a proof $\rho_{\hat{\imath}}$ for $(R_{\hat{\imath}}, C_{\hat{\imath}} \| \pi_{\hat{\imath}} \| P_{\hat{\imath}} \| \chi_{\hat{\imath}})$, which completes $txo_{\hat{\imath}}$. To complete tx_i, it also simulates σ (whose witness depends on $\log R_{\hat{\imath}}$).

(A2a). Assume that the reduction guessed correctly and that (A2a) occurs, that is, tx^* contains $(C^*, \pi^*, R_{\hat{\imath}}, \rho^*, P^*, \chi^*) \neq \widehat{txo} = txo_{\hat{\imath}}$. Thus, ρ^* is valid for $(R_{\hat{\imath}}, C^* \| \pi^* \| P^* \| \chi^*)$ and $(C^* \| \pi^* \| P^* \| \chi^*, \rho^*) \neq (C_{\hat{\imath}} \| \pi_{\hat{\imath}} \| P_{\hat{\imath}} \| \chi_{\hat{\imath}}, \rho_{\hat{\imath}})$; otherwise we would have $txo^* = \widehat{txo}$. The statement for ρ^* (or ρ^* itself) is thus different from the statement for the simulated $\rho_{\hat{\imath}}$ (or $\rho_{\hat{\imath}}$ itself). By strong SE of PoP, the reduction can extract from ρ^* the witness $\log R_{\hat{\imath}} = \log Z$, the DL solution. (Recall that σ is also simulated, but that different types of proofs are assumed to be domain-separated, cf. Section 2)

(A2b). Now assume that the reduction guessed correctly and that (A2b) occurs. This means that

$$\mathsf{tx}^* = \big((\boldsymbol{P}^*, \boldsymbol{D}^*, \boldsymbol{\psi}^*), (\hat{\boldsymbol{C}}, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{R}}, \hat{\boldsymbol{\rho}}, \hat{\boldsymbol{P}}, \hat{\boldsymbol{\chi}}), ((\boldsymbol{C}', \boldsymbol{\pi}', \boldsymbol{R}', \boldsymbol{\rho}', \boldsymbol{P}', \boldsymbol{\chi}', \boldsymbol{P}', \boldsymbol{D}', \boldsymbol{\psi}'), s^*, f^*, t^*, y^*, \boldsymbol{E}, \boldsymbol{X}, \boldsymbol{\sigma})\big) \ ,$$

returned by the adversary, is such that $\boldsymbol{X}$ contains the stealth excess X of tx and neither $\hat{\boldsymbol{R}}$ nor $\boldsymbol{R}'$ contain $R_{\hat{\imath}}$. By validity of tx, it satisfies the balance equation (12) and since $R_{\hat{\imath}} = Z$, we have

$$X = \textstyle\sum_{j \neq \hat{\imath}} R_j + Z - \sum \boldsymbol{D} - yG \ . \tag{28}$$

All values $r_j = \log R_j$, for $j \neq \hat{\imath}$, and $d_j = \log D_j$ have been chosen by (and are thus known to) the reduction. Moreover, since (A2b) occurred, for some J^* with $|J^*| =: m \geq 1$ we have $X_j = X$ for all $j \in J^*$ (the adversary could have included X multiple times in $\boldsymbol{X}$). By validity of tx^* (recall that for aggregated transactions, Equation (12) also ranges over the cut-through terms) this yields

$$\textstyle\sum_{j \notin J^*} X_j + mX = \sum \hat{\boldsymbol{R}} + \sum \boldsymbol{R}' - \sum \boldsymbol{D}^* - \sum \boldsymbol{D}' - y^* G \ . \tag{29}$$

From the proofs σ_j in tx^*, for $j \notin J^*$, the reduction can extract the corresponding values $x_j = \log X_j$ (since only σ for X was simulated and $X \notin (X_j)_{j \notin J^*}$). From the proofs ψ_j^* and ψ_j' the reduction can extract the values $d_j^* = \log D_j^*$ and $d_j' = \log D_j'$, 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 $\hat{\rho}_j$ and ρ_j', the reduction can extract the values $\hat{r}_j = \log \hat{R}_j$ and $r_j' = \log R_j'$ (and it knows the logarithms of the ones copied from honest transactions). Recall that $R_{\hat{\imath}}$ (for which $\rho_{\hat{\imath}}$ was simulated) is not among $\hat{\boldsymbol{R}}$ nor $\boldsymbol{R}'$.

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

$$\log Z \equiv_p \tfrac{1}{m}\big(- \textstyle\sum_{j \notin J^*} x_j + \sum \hat{\boldsymbol{r}} + \sum \boldsymbol{r}' - \sum \boldsymbol{d}^* - \sum \boldsymbol{d}' - y^*\big) - \sum_{j \neq \hat{\imath}} r_j + \sum \boldsymbol{d} + y \ ,$$

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 D values chosen by the oracle SEND. Whenever the adversary

calls SEND, the reduction randomly samples $d \leftarrow\!\!\$\ \mathbb{Z}_p$, sets $D := dZ$ and simulates proofs ψ for (P, D, txo), as well as the excess proof σ (whose witnesses depend on $\log D$). Let

$$\mathsf{tx}^* = \big((\boldsymbol{P}, \boldsymbol{D}, \boldsymbol{\psi}), (\hat{\boldsymbol{C}}, \hat{\boldsymbol{\pi}}, \hat{\boldsymbol{R}}, \hat{\boldsymbol{\rho}}, \hat{\boldsymbol{P}}, \hat{\boldsymbol{\chi}}), ((\boldsymbol{C}', \boldsymbol{\pi}', \boldsymbol{R}', \boldsymbol{\rho}', \boldsymbol{P}', \boldsymbol{\chi}', \boldsymbol{P}', \boldsymbol{D}', \boldsymbol{\psi}'), s, f, t, y, \boldsymbol{E}, \boldsymbol{X}, \boldsymbol{\sigma})\big) \ ,$$

be the transaction returned by the adversary. By (W1) and (W2), $\boldsymbol{D}$ contains an input D from an honest transaction tx, whose stealth excess X is not contained in $\boldsymbol{X}$. In tx^* we distinguish two types of values in $\boldsymbol{D} \,\|\, \boldsymbol{D}'$ and two types of stealth-excess elements:

- let $\bar{D}_i$ be the values in $\boldsymbol{D}$ and $\boldsymbol{D}'$ that appear in honest transactions and D_i^* be those that don't
- let $\bar{X}_i$ be the stealth excesses that appear in honest transactions and X_i^* be those that don't

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 and 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 values $\hat{R}_i$ or R_i' in tx^* were copied from an honest transaction, the reduction knows their logarithms. For all other $\hat{R}_i$ and R_i', from the proofs $\hat{\rho}_i$ and ρ_i' contained in tx^*, the reduction can extract $\log \hat{R}_i$ and $\log R_i'$ (as no ρ proofs are simulated). The reduction thus knows the values

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

Since tx^* is valid, it satisfies the balance equation (12) for aggregated transactions, that is $\boldsymbol{X} = \sum \hat{\boldsymbol{R}} + \sum \boldsymbol{R}' - \sum \boldsymbol{D} - \sum \boldsymbol{D}' - yG$. Since the lists $\boldsymbol{X}$ and $\boldsymbol{X}^* \| \bar{\boldsymbol{X}}$ contain the same elements, as do $\boldsymbol{D} \,\|\, \boldsymbol{D}'$ and $\boldsymbol{D}^* \| \bar{\boldsymbol{D}}$, this yields

$$\textstyle\sum \boldsymbol{X}^* + \sum \bar{\boldsymbol{X}} \ = \ \sum \hat{\boldsymbol{R}} + \sum \boldsymbol{R}' - \sum \boldsymbol{D}^* - \sum \bar{\boldsymbol{D}} - yG \ .$$

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

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

We argue that with overwhelming probability we have

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

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 tx is not among the transactions (tx_i) whose stealth excesses were copied, and from which the values $(d_{i,j})_{i,j}$ come. The value $\bar{d}_{i^*}$ is thus not among $(d_{i,j})_{i,j}$ (except with negligible probability if the reduction chose the same value twice). Since all values are chosen at random by the reduction, the probability that (33) is not satisfied is thus $1/p$, with p the order of the group $\mathbb{G}$. Whenever (33) is satisfied, the reduction can compute $\log Z$ from (32). □

6.6 Transaction privacy

We now consider an attacker that passively observes blocks of the underlying blockchain and attempts to deduce 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 the initial version of Mimblewimble one can "disaggregate" transactions [Dev20a], that is, determine which inputs and outputs come from the same original transaction. As a consequence, one could infer how money was being transferred 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 in [FOS19]. Our notion 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 inputs and outputs, *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. The adversary has access to an oracle that creates honest users and a challenge oracle that produces transactions, taking as 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 list of 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 (for the i-th transaction)

In the "real" world, the oracle does the following: for all $i \in [\ell]$, it creates tx_i by running Send on the inputs referenced by $(ref_{i,j})_j$: these are either explicit output/value/spending-key triples; or $(ref_{i,j})_j = id_{i,j}$, where $id_{i,j}$ was associated to a previous output argument $(\hat{v}_{i,j}, \hat{pk}_{i,j}, \hat{\chi}_{i,j})$ for which $\hat{pk}_{i,j}$ was generated by the "honest-user" oracle; in this case the output $txo_{i,j}$ of the transaction previously generated for $id_{i,j}$ is used together with $\hat{v}_{i,j}$ and $\hat{sk}_{i,j}$, the spending key associated to $pk_{i,j}$. (If the adversary wants to spend outputs for an address it controls, it can provide an explicit output/value/spending-key triple.) The remaining arguments for Send are $(\hat{\mathbf{v}}_i, \hat{\mathbf{pk}}_i, \hat{\boldsymbol{\chi}}_i)$, s_i and f_i. Using Agg, the oracle then aggregates $\mathsf{tx}_1, \ldots, \mathsf{tx}_\ell$ to tx and returns tx.

Transaction privacy requires that the "real" oracle is indistinguishable from an oracle that creates an aggregate transaction tx using 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); moreover, $\mathbf{in}$ is sorted lexicographically; it contains neither values nor keys
- a (sorted) list $\mathbf{out} = (\hat{\mathbf{id}}, (\hat{\mathbf{v}}, \hat{\mathbf{pk}}, \hat{\boldsymbol{\chi}}))$ of output identifiers and triples which are either for adversarial keys or which are of the form $(\perp, \perp, \hat{\chi}_i)$ 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.

In addition to the simulated tx, the simulator also provides a list of the transaction outputs in tx for the honest users together with their respective identifiers (to be used by the oracle when compiling inputs in a subsequent call of the simulator). The game is formalized in Figure 7. The

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

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

Oracle KEYGEN()

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

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

// "real" game
// check for double and already used id's:
if $\exists\,(i,j) \neq (i',j') : \hat{id}_{i,j} = \hat{id}_{i',j'}$ **or**
$\quad Txos(\hat{id}_{i,j}) \neq \varepsilon$ **then abort**
for $i \in [\ell]$ // get ref's to previous txo's
$\quad \mathbf{in}_i := ()$
$\quad$ **for** $j \in [|\mathbf{ref}_i|]$
$\quad\quad$ **if** $ref_{i,j} =: id$ // $ref_{i,j}$ is an id
$\quad\quad\quad$ **if** $Txos(id) =: (txo, v, sk) \neq \varepsilon$
$\quad\quad\quad\quad \mathbf{in}_i := \mathbf{in}_i \,\|\, (txo, v, sk)$
$\quad\quad$ **if** $ref_{i,j} =: (txo, v, sk)$
$\quad\quad\quad \mathbf{in}_i := \mathbf{in}_i \,\|\, (txo, v, sk)$
$\quad\quad$ **else abort**
$\quad \mathsf{tx}_i \leftarrow \mathsf{Send}(pp, \mathbf{in}_i, (\hat{\mathbf{v}}_i, \hat{\mathbf{pk}}_i, \hat{\boldsymbol{\chi}}_i), s_i, f_i)$
$\quad$ // store outputs for honest users in $Txos$:
$\quad$ **for** $j \in [|\hat{\mathbf{id}}_i|]$
$\quad\quad$ **if** $Pks(\hat{pk}_{i,j}) =: (\hat{vk}_{i,j}, \hat{sk}_{i,j}) \neq \varepsilon$
$\quad\quad\quad$ // j-th recipient is honest
$\quad\quad\quad txo_{i,j} \leftarrow \mathsf{Rcv}(pp, \mathsf{tx}_i, \hat{v}_{i,j}, \hat{vk}_{i,j})$
$\quad\quad\quad$ // if multiple transactions are ...
$\quad\quad\quad$ // ... found, store the subsequent ...
$\quad\quad\quad$ // ... one at each iteration
$\quad\quad\quad Txos(\hat{id}_{i,j}) := (txo_{i,j}, \hat{v}_{i,j}, \hat{sk}_{i,j})$
$\mathsf{tx} \leftarrow \mathsf{Agg}(pp, \mathsf{tx}_1, \ldots, \mathsf{tx}_\ell)$
return tx

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

// "simulated" game
if $\exists\,(i,j) \neq (i',j') : \hat{id}_{i,j} = \hat{id}_{i',j'}$ **or**
$\quad Txos(\hat{id}_{i,j}) \neq \varepsilon$ **then abort**
$\mathbf{in} := ()\,;\ \mathbf{out} := ()$ // simulator gets "aggregated" lists
for $i \in [\ell]$
$\quad val_i := 0\,;\ \hat{val}_i := \sum_j \hat{v}_{i,j}$ // sums of in and out values
$\quad$ **for** $j \in [|\mathbf{ref}_i|]$
$\quad\quad$ **if** $ref_{i,j} =: id$
$\quad\quad\quad$ **if** $Txos(id) =: (txo, v) \neq \varepsilon$
$\quad\quad\quad\quad \mathbf{in} := \mathbf{in} \,\|\, txo\,;\ val_i := val_i + v$
$\quad\quad\quad$ **elseif** $\exists\, i' \in [i-1], j : id = \hat{id}_{i',j'}$
$\quad\quad\quad\quad \mathbf{in} := \mathbf{in} \,\|\, id\,;\ val_i := val_i + \hat{v}_{i',j'}$
$\quad\quad$ **if** $ref_{i,j} =: (txo, v, sk)$
$\quad\quad\quad \mathbf{in}_i := \mathbf{in}_i \,\|\, (txo, v, sk)\,;\ val_i := val_i + v$
$\quad\quad$ **else abort**
$\quad$ **if** $\hat{val}_i + f_i \neq val_i + s_i$: **return** $\mathsf{tx} := \bot$ // tx unbalanced
$\quad$ **for** $j \in [|\hat{\mathbf{id}}_i|]$ // remove honest keys and values:
$\quad\quad$ **if** $Pks(\hat{pk}_{i,j}) = \varepsilon$: $\mathbf{out} := \mathbf{out} \,\|\, (\hat{id}_{i,j}, (\hat{v}_{i,j}, \hat{pk}_{i,j}, \hat{\chi}_{i,j}))$
$\quad\quad$ **else** $\mathbf{out} := \mathbf{out} \,\|\, (\hat{id}_{i,j}, (\bot, \bot, \hat{\chi}_{i,j}))$
$\mathbf{in}' := \mathsf{Sort}(\mathbf{in})\,;\ \mathbf{out}' := \mathsf{Sort}(\mathbf{out})$
$s := \sum_i s_i\,;\ f := \sum_i f_i$
$(\mathsf{tx}, (\mathbf{id}', \mathbf{txo}')) \leftarrow \mathsf{Sim}(\mathbf{in}', \mathbf{out}', s, f, \ell)$
// store outputs for honest users in $Txos$:
let i', j' **such that** $id'_i = \hat{id}_{i',j'}$
$\quad Txos(id'_i) := (txo'_i, v_{i',j'})$
return tx

Fig. 7. Game TXPRV for transaction privacy

adversary gets access to an oracle that either returns real or simulated (aggregated) transactions, and the adversary has to decide which is the case. If in a "real" query, the input and output values are unbalanced, Send would return $\bot$ and after aggregation, the oracle response would be $\bot$. Since the simulator does not get any values, the oracle in the "simulated" world checks balancedness explicitly. The complexity of the formal game mainly stems from bookkeeping.

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 view key. Hence, for outputs that do not belong to honest users, 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. Finally, 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 Sim. It 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 elements of the form $(\hat{id}_i, (\hat{v}_i, \hat{pk}_i, \hat{\chi}_i))$ where possibly $(\hat{v}_i, \hat{pk}_i) = (\bot, \bot)$; 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 cut through; for $i \in [n] \setminus I_{id}$, the entries in_i are explicit TXO's. Further, let $\hat{I}_{\text{hon}}$ be the indices $i \in [\hat{n}]$ for which $(\hat{v}_i, \hat{pk}_i) = (\bot, \bot)$, that is, the indices of outputs for honest users.

(a) for each **output** index $i \in [\hat{n}] \setminus \hat{I}_{\text{hon}}$, sample $\hat{r}_i \leftarrow\!\$\ \mathbb{Z}_p$ and use it to 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
(b) for each $i \in \hat{I}_{\text{hon}}$, pick random values $\hat{C}_i, \hat{R}_i, \hat{P}_i \leftarrow\!\$\ \mathbb{G}$
(c) 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 $\widehat{txo}_i := (\hat{C}_i, \hat{\pi}_i, \hat{R}_i, \hat{\rho}_i, \hat{P}_i, \hat{\chi}_i)$
(d) define the honest outputs via $\mathbf{id}' := (\hat{id})_{i \in \hat{I}_{\text{hon}}}$ and $\mathbf{txo}' := (\widehat{txo}_i)_{i \in \hat{I}_{\text{hon}}}$

(e) for each **input** index $i \in [n]$, pick a random value $D_i \leftarrow\!\$\ \mathbb{G}$
(f) for each $i \in I_{\text{id}}$, let k_i be such that $in_i = \hat{id}_{k_i}$ (if there is none, abort); set $P_i := \hat{P}_{k_i}$ with $\hat{P}_{k_i}$ from (a) or (b); simulate ψ_i for the statement $(P_i, D_i, \widehat{txo}_{k_i})$ with $\widehat{txo}_{k_i}$ from (c)
(g) for each $i \in [n] \setminus I_{\text{id}}$, with in_i containing an explicit output $txo_i =: (C_i, \pi_i, R_i, \rho_i, P_i, \chi_i)$, simulate ψ_i for the statement (P_i, D_i, in_i)
(h) define the cut-through **ct** as the (lexicographical) ordering of $(\widehat{txo}_{k_i}, P_i, D_i, \psi_i)_{i \in I_{\text{id}}}$
(i) define $\mathbf{txi}'$ as the ordering of $(P_i, D_i, \psi_i)_{i \in [n] \setminus I_{\text{id}}}$ (i.e., cut-through terms removed)
(j) define $\mathbf{txo}'$ as the ordering of $(\widehat{txo}_i)_{i \in [\hat{n}] \setminus \{k_i\}_{i \in I_{\text{id}}}}$ (i.e., cut-through terms removed)

(k) compute the **kernel**: pick random values $E_2, \ldots, E_\ell, X_2, \ldots, X_\ell \leftarrow\!\$\ \mathbb{G}$, as well as $t, y \leftarrow\!\$\ \mathbb{Z}_p$
(l) set $E_1 := \sum \hat{\boldsymbol{C}} - \sum \boldsymbol{C} + (f - s)H - tG - \sum_{i=2}^{\ell} E_i$
and $X_1 := \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{D} - yG - \sum_{i=2}^{\ell} X_i$ (note that we could have $\ell = 1$)
(m) for each $i \in [\ell]$, simulate σ_i for (E_i, X_i) with tag ε

Set

$$\mathsf{tx} = \left(\mathbf{txi}', \mathbf{txo}', (\mathbf{ct}, s, f, t, y, \boldsymbol{E}, \boldsymbol{X}, \boldsymbol{\sigma})\right) ,$$

and remove **ct** if it is empty, and remove $\boldsymbol{E}$ and $\boldsymbol{X}$ if $\ell = 1$. Return $(\mathsf{tx}, (\mathbf{id}', \mathbf{txo}'))$.

We now show that in the random oracle model, assuming indistinguishability of simulated RaP and PoP proofs and the DDH assumption, a real transaction is indistinguishable from a simulated transaction. We start with a real transaction (as returned by Tx_0) and consider a sequence of hybrid games that turns it into a simulated transaction (as returned by Tx_1).

H_0 This is the "real" game, in which the adversary interacts with the oracle Tx_0. The component transactions are created as described (in Section 3.2) using the spending keys and 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 In this hybrid, when determining the outputs of the honest users to be added to $Txos$, instead of running Rcv, the oracle simply sets $txo_{i,j}$ as the j-th output of the i-th transaction. By the definition of the scheme, this does not change anything.
Note that in this hybrid, the part $a_{i,j}$ of the view key $vk_{i,j}$ of an honest user is only used to compute $(k_{i,j}, q_{i,j}) \coloneqq \mathrm{H}(a_{i,j}R_{i,j})$ when running Send on an honest input; moreover the value $\hat{r}_{i,j}$ for honest outputs is only used to compute $(\hat{k}_{i,j}, \hat{q}_{i,j}) \coloneqq \mathrm{H}(\hat{r}_{i,j}A_{i,j})$ (since H_1). Also, the spending key sk_ι is never used (since H_1).

H_3 For every output for an honest user 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 $Z_{i,j} \leftarrow\!\$\ \mathbb{G}$; when the output is later spent then analogously $a_{i,j}\hat{R}_{i,j}$ is replaced by the same $Z_{i,j}$. 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 not used anywhere else in the game (the former because $\rho_{i,j}$ and σ_i are simulated; the latter because of the modification in H_2 and since the address oracle does not reveal any secret keys).

H_4 The game aborts if the adversary queries $Z_{i,j}$ for any 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_4, the values $\hat{k}_{i,j}$ and $\hat{q}_{i,j}$, derived from $\mathrm{H}(Z_{i,j})$ 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_4, it remains to argue that a transaction generated in H_4 is distributed equivalently to a transaction computed by the simulator, when interacting with the oracle $\textsc{Tx}_1$. The transactions produced in H_4 and in $\textsc{Tx}_1$ (by the simulator) are both of the form

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

The supply and fee are defined as $s \coloneqq \sum s_i$ and fee $f \coloneqq \sum f_i$ both in H_4 (when running Agg) and in $\textsc{Tx}_1$. The values t, y are distributed uniformly at random both in H_4 (where $t \coloneqq \sum_i t_i$ with each $t_i \leftarrow\!\$\ \mathbb{Z}_p$) and by the simulator in (k) (where $t \leftarrow\!\$\ \mathbb{Z}_p$). All range proofs RaP and the proof of possession PoP are simulated both in H_4 and by the simulator. We are left proving that all the group elements in tx follow the same distribution.

Within a query to the transaction oracle, denote by I_{id} the set of indices (i, j) of the inputs given as *id* values (and not an explicit triple) and let I_{hon} be the set of indices (i, j) for which $\hat{pk}_{i,j}$ is honest, that is, it was generated by the oracle $\textsc{Keygen}$.

In a transaction produced in H_4, for honest outputs, the coin openings $\hat{q}_{i,j}$, for $(i, j) \in I_{\mathrm{hon}}$, are uniformly random and independent, and thus so are the elements $\hat{C}_{i,j}$ for $(i, j) \in I_{\mathrm{hon}}$, which is how the simulator creates them in (b). Similarly, the values $\hat{k}_{i,j}$, for $(i, j) \in I_{\mathrm{hon}}$ are uniformly random and independent and thus so are the corresponding values $\hat{P}_{i,j}$, which is how the the simulator produces them in (b). For adversarial outputs, the values $\hat{C}_{i,j}$ and $\hat{P}_{i,j}$ for $(i, j) \notin I_{\mathrm{hon}}$ are computed by the simulator as prescribed by Send, which is what H_4 does as well. Finally, all values $D_{i,j}$ and $\hat{R}_{i,j}$ are sampled uniformly random and independently both in H_4 and by the simulator.

By the definition of Agg, we have that in H_4 the excesses $E_1, \ldots, E_\ell$ are uniformly random elements (since all t_i's are uniformly random values) 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 in (k) and (l). Likewise, the values $X_1, \ldots, X_\ell$ are uniformly random conditioned on

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

which is exactly how the simulator picks them in (k) and (l).

Finally, the output/input pairs in the cut-through list of the simulator are those that Agg would put there, and the ordering (lexicographic) is the same used in input, outputs, and cut-through lists as per Agg. □

Acknowledgements. The first author is supported by the Vienna Science and Technology Fund (WWTF) through project VRG18-002. We 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.