Non-interactive Mimblewimble transactions, revisited

Georg Fuchsbauer[1] and Michele Orr[2]

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

Abstract. Mimblewimble is a cryptocurrency protocol promising to overcome notorious blockchain scalability issues. To this day, one of the major factors hindering its wider adoption is the lack of non-interactive transactions, that is, payments where only the sender needs to be online. We start from a proposal by Yu that adds non-interactive transactions to stealth addresses to Mimblewimble, but which turned out to be flawed. Using ideas from Burkett, we propose a modified scheme and analyze it using the provable-security methodology.

1 Introduction

Mimblewimble (MW) was first proposed by an anonymous author in 2016 [Jed16]. It was built on the top of three ideas initially envisioned for Bitcoin [Nak08]: confidential transactions [Max15], CoinJoin [Max13a], and transaction cut-through [Max13b]. Initially investigated by Poelstra [Poe16], a formal model and an analysis of MW were provided by Fuchsbauer, Orr and Seurin (FOS) [FOS19] in 2019. In 2020, Burkett [Bur20] proposed an extension of Mimblewimble supporting non-interactive transactions, later described in more detail by Yu [Yu20]. We will refer to this extension as MW-Yu. In this work, we study and asses the security of MW-Yu [Yu20, §2.1], in the definitional framework of [FOS19] and fix discovered vulnerabilities, integrating some ideas from a more recent proposal from Burkett [Bur21].

MW and its variants use 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 for the commitment enables spending the coin. Similarly to Bitcoin, a transaction in MW is a list of output coins $\hat{\boldsymbol{C}} \in \mathbb{G}^{\hat{n}}$ and input coins $\boldsymbol{C} \in \mathbb{G}^n$, where

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

Leaving fees and 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

$$\sum \hat{\boldsymbol{C}} - \sum \boldsymbol{C} = \left(\sum \hat{\boldsymbol{q}} - \sum \boldsymbol{q}\right) G \ , \tag{0}$$

a quantity called the *kernel excess* $E \in \mathbb{G}$ in MW. If the transaction is balanced, then knowledge of the openings $\hat{\boldsymbol{q}}, \boldsymbol{q}$ of all involved coins implies knowledge of the discrete logarithm $\log E$ of the excess E to base G. 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 over $\mathbb{G}$, balancedness follows from the hardness of computing discrete logarithms in $\mathbb{G}$ in the random-oracle model. They moreover show that as long as a user owning coin C in the ledger keeps the opening private, no one can steal C (i.e., creating a transaction that spends C).

Transactions in Mimblewimble can be easily merged non-interactively, in a similar way to CoinJoin [Max13a]. Consider two transactions $\mathsf{tx}_0 = (\hat{\boldsymbol{C}}_0, \boldsymbol{C}_0, \boldsymbol{\pi}_0, E_0, \sigma_0)$ and $\mathsf{tx}_1 = (\hat{\boldsymbol{C}}_1, \boldsymbol{C}_1, \boldsymbol{\pi}_1, E_1, \sigma_1)$. The *aggregate transaction* tx results from the concatenation of inputs and outputs, that is,

$$\mathsf{tx} = \left(\hat{\boldsymbol{C}}_0 \| \hat{\boldsymbol{C}}_1, \boldsymbol{C}_0 \| \boldsymbol{C}_1, \boldsymbol{\pi}_0 \| \boldsymbol{\pi}_1, E_0 \| E_1, \sigma_0 \| \sigma_1\right) \ ,$$

where we let "$\|$" denote concatenation. As outputs in one transaction that also appear as inputs in the other cancel out in Equation (0) for tx, they can simply be removed from the input and output list (together with their range proofs), while validity of tx will be maintained. This has been called *transaction cut-through* in the literature [Max13b]. In a cryptocurrency based on 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. In other cryptocurrencies, instead, the history of all transactions needs to be stored to ensure that the UTXO set is cryptographically valid. FOS [FOS19] further remarked that if the signature scheme supports aggregation, then $\sigma_0 \| \sigma_1$ can be replaced by their aggregation to save space. Thus, the only trace of a transaction whose outputs have been spent in the ledger is the value E.

Even if the inputs and outputs in an aggregate transaction tx are ordered lexicographically, one can still determine which inputs and outputs come from the same component transaction, since tx will contain an excess value E which 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_0, t_0) and (E_1, t_1) will then contain $(E_0 \| E_1, t_0 + t_1)$.

Most implementations of MW create new transactions via an *interactive* protocol between sender and receiver in on order to produce the Schnorr signature σ, whose secret key depends on the openings of the sender's and the receiver's coins.[1] FOS [FOS19] proposed a transaction protocol, where the sender creates all output coins, and can thus complete the transaction by computing σ on its own. The sender then shares (through a separate private channel) a transaction along with the secret key associated to one of the output coins. The receiver can then create a transaction spending this coin, merge it with the received transaction and then broadcast 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.

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

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

[1] For instance, in Grin this is documented in the grin-wallet documentation. In Beam this is documented in the developer documentation.

signing key R (of which the sender knows the logarithm) is added. An output is thus of the form (C, π, R, ρ, P).

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

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

Yu proposes to simply arrange the $\hat{\boldsymbol{R}}$ values so that the above results in the excess E (which is defined by (0)). However, this is only possible if one of the outputs goes back to the sender (who can choose the q value of the corresponding C arbitrarily); for all other coins, $\hat{R}$ (together with the stealth address) defines q, which defines E, for which $\hat{R}$ has to be chosen. We therefore modify the scheme and introduce a *stealth excess* $X := \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{P}$, under which we add (as for E) a signature to the transaction. Our scheme then supports transactions for which all outputs are sent to destination addresses.

At the time of writing, the core proposal in [Yu20, §2.1] is still affected by further issues. The ones known before our analysis are the following:

– As illustrated in [Yu20, §2.9.1], MW-Yu is susceptible to a rogue-key attack [Yu20, §2.9.3]. A fix was also proposed, which requires the addition of one signature per transaction input, namely a signature proving knowledge of the logarithm of P. The security and correctness analysis of this proposed change are not detailed further.
– Mixing NIT with non-NIT transactions, as envisaged in [Yu20], leads to correctness issues within the balance equations.[2] 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.)

Our contributions. In Section 4 we discuss further issues that emerged after the publication of [Yu20]. We propose and formally define a new protocol for non-interactive transactions, greatly inspired by [Yu20] and using a idea by Burkett [Bur21] that overcomes the found issues (Section 3). We then provide arguments for the security of our protocol against specific types of attacks by following the provable-security methodology and sketching security reductions to hardness assumptions in idealized models. To do so, we use multiple security experiments that capture relevant attack scenarios. We then analyze our protocol with respect to them them. In particular, we consider the following types of attacks:

(i) creating money other than via "minting" transactions *(inflation resistance)*;
(ii) stealing money from the ledger *(theft prevention)*;
(iii) stealing money from a transaction not yet merged with the ledger *(transaction-binding)*;
(iv) breaking privacy by decomposing an aggregate transaction into its components or learning anything about the transacted amounts or the destination addresses *(transaction privacy)*.

Inflation resistance and *theft* prevention are straightforward adaptations of the notions from [FOS19, Def. 10 and 11]; *transaction-privacy* is stronger than FOS's privacy notion [FOS19, Def. 11], which only requires that amounts are hidden (in MW there are no addresses and FOS do not consider

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

kernel offsets). *Transaction-binding* prevents attacks specific to MW with non-interactive transactions (it appears that for MW, as analyzed in [FOS19], attacks of this type are covered already by theft-prevention).

In our security arguments, 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*. We moreover assume a *zero-knowledge* proof system that satisfies *strong simulation-extractability*. In Section 6 we show that the Schnorr signature scheme, and a variant thereof, satisfy the required notion in an idealized model, namely the combination of the *algebraic group model* [FKL18] and the *random oracle model*.

2 Preliminaries

Let ε denote the empty string. 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, we assume 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 $X = xG$.

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

$$C := vH + qG\ .$$

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

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

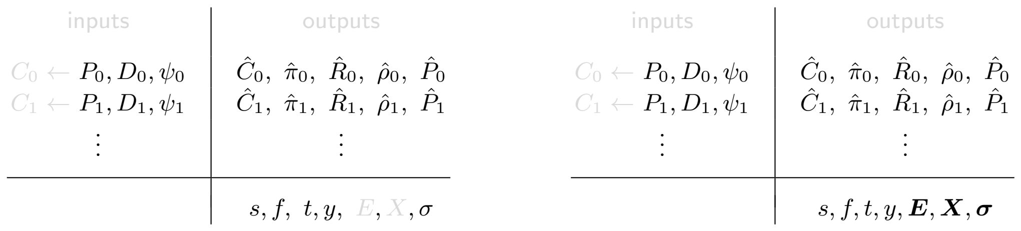
$$\big\{\big(C, (v,r)\big) \ : \ C = \mathsf{Cmt}(v,r) \ \wedge \ v \in [0, v_{\max}]\big\}$$

We assume a zero-knowledge proof system RaP for proofs of knowledge of a witness (v, r) for a statement C such that $0 \le v \le v_{\max}$. (Note that we do not assume RaP to be simulation-sound [FOS19, Def. 8], whereas FOS required this in their proof of theft prevention. Our scheme could thus be potentially instantiated with a more efficient range proof system than theirs.) We denote the prover algorithm by $\pi \leftarrow \mathsf{RaP.P}(C,(v,r))$ and the verifier by $b \leftarrow \mathsf{RaP.V}(C,\pi)$.

Proofs of possession. We model the employed cryptographic hash function as a random oracle and denote it by $\mathrm{H}(\cdot)$. We use (key-prefixed) Schnorr signatures (defined in Figure 2 on page 22), which are unforgeable under the DL assumption in the random oracle model (ROM), and whose security has been extensively studied in the past literature [PS00,Seu12]. Here we interpret Schnorr signatures as (zero-knowledge) proofs of knowledge of the secret key, that is, if $X = xG \in \mathbb{G}$ is the public key then a Schnorr signature is a proof of knowledge of $x = \log X$.

We generalize this to a proof of knowledge of *two* logarithms that is as efficient as a Schnorr signature. Interpreting knowledge of $\log X$ as "possessing" X, we call the proof system PoP for "proof of possession". More formally, we define a proof system PoP for the following NP-relation, defined w.r.t. a group description $(p, \mathbb{G}, G)$:

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



inputs
outputs
$C_0 \leftarrow P_0, D_0, \psi_0$
$C_1 \leftarrow P_1, D_1, \psi_1$
$\hat{C}_0,\ \hat{\pi}_0,\ \hat{R}_0,\ \hat{\rho}_0,\ \hat{P}_0$
$\hat{C}_1,\ \hat{\pi}_1,\ \hat{R}_1,\ \hat{\rho}_1,\ \hat{P}_1$
$s, f,\ t, y,\ E, X, \sigma$
inputs
outputs
$C_0 \leftarrow P_0, D_0, \psi_0$
$C_1 \leftarrow P_1, D_1, \psi_1$
$\hat{C}_0,\ \hat{\pi}_0,\ \hat{R}_0,\ \hat{\rho}_0,\ \hat{P}_0$
$\hat{C}_1,\ \hat{\pi}_1,\ \hat{R}_1,\ \hat{\rho}_1,\ \hat{P}_1$
$s, f, t, y, \boldsymbol{E}, \boldsymbol{X}, \boldsymbol{\sigma}$

Fig. 1. Left: Visualization of a (simple) MW-NIT transaction. Inputs consist of a value P_i form a previous transaction output and a new one-time input D_i, together with a proof of possession ψ_i authenticating the previous output. Outputs consist of a commitment $\hat{C}_i$ to the value of the output, an associated range proof $\hat{\pi}_i$, an ephemeral key $\hat{R}_i$, a signature $\hat{\rho}_i$ under $\hat{R}_i$, and the destination address $\hat{P}_i$. The kernel consists of the supply s, the fee f, the offsets t and y, the proof of knowledge of the excess σ (valid under E, X). The excess E and the stealth excess X can be computed as in (1) and (2). **Right:** Visualization of an aggregated non-interactive Mimblewimble transaction. The excesses and their signatures are lists of the excesses and signatures from the composing transactions.

A statement thus contains a part m, which is sometimes called a "tag". A proof for a statement $(X, Y, m) \in \mathbb{G}^2 \times \{0,1\}^*$ is computed via $\mathsf{PoP.P}$ using the witness (x, y) by picking a uniform $r \leftarrow_\$ \mathbb{Z}_p$, defining $R := rG$, computing $(c, d) := \mathrm{H}(X, Y, m, R)$ and returning a proof $(R, s) \in \mathbb{G} \times \mathbb{Z}_p$ with $s := r + c \cdot x + d \cdot y \bmod p$. The verification algorithm $\mathsf{PoP.V}((X, Y, m), (R, s))$ computes $(c, d) := \mathrm{H}(X, Y, m, R)$ and checks whether $sG = R + c \cdot X + d \cdot Y$ (see Figure 3 on page 22).

The system PoP can also be used to prove knowledge of a witness $x \in \mathbb{Z}_p$ for statements $(X = xG, m)$ by using Schnorr signing and verification (defined like above but setting $d := 0$). 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 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 6 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 we 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 6) that produces a valid proof ψ for a statement (X, m) (or (X, Y, m)) can extract the witness $\log X$ (or $(\log X, \log Y)$). Proofs are *simulation-sound* (also called *simulation-extractable* (SE) for PoKs) if a witness can be extracted from a prover even if the prover can obtain simulated proofs ψ_i for statements (X_i, m_i) of its choice (except the one it is proving). For *strong* SE the only restriction is that the pair $((X, m), \psi)$ must be different from all query/response pairs $((X_i, m_i), \psi_i)$.

3 A new proposal for MW with non-interactive transactions

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 $\mathsf{out} = (\hat{C}, \hat{\pi}, \hat{R}, \hat{\rho}, \hat{P})$, each implicitly associated to an **output address** (A, B), composed of:

- an **ephemeral key** $\hat{R} = \hat{r}G \in \mathbb{G}$, chosen by the sender, which defines two keys as:

$$(\hat{k}, \hat{q}) := \mathrm{H}(\hat{r}A, (A, B))$$

  (note that $\hat{k}$ and $\hat{q}$ can be computed from the view key and $\hat{R}$ since $\hat{r}A = a\hat{R}$)
- a **commitment** $\hat{C} := \hat{v}H + \hat{q}G$ to the coin value $\hat{v}$, using randomness $\hat{q}$
- a **range proof** $\hat{\pi}$ proving knowledge of an opening (v, q) of $\hat{C}$, with $v \in [0, v_{\max}]$
- a **one-time output public key** $\hat{P} \in \mathbb{G}$, computed from $\hat{k}$ as $\hat{P} := \hat{B} + kG$ (note that the *spend* key is required to compute $\log \hat{P}$)
- a **proof of possession** $\hat{\rho}$ of $\hat{R}$ with tag $\hat{C}\|\hat{\pi}\|\hat{P}$ (and possibly a time stamp)

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

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

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

  which can be seen as $E := E' - tG$ in terms of the *true excess* $E' := \sum \hat{\boldsymbol{C}} + fH - \sum \boldsymbol{C} - sH$
- 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

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

- 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:

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

3.2 Transaction creation

Consider a transaction output $\mathsf{out} = (C, \pi, R, \rho, P)$ spent to a stealth address (A', B').

– Given the corresponding view key (a', B'), one can compute the shared keys as:

$$(k, q) := \mathrm{H}(a'R, (a'G, B'))$$

  (where q is the opening for the commitment C).

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

To create a transaction that spends transaction outputs out_i of values v_i from source addresses 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 $\sum \hat{\boldsymbol{v}} + f = \sum \boldsymbol{v} + s$, do the following:

– for each input index $i \in [n]$:
- compute all values q_i and $p_i := \log P_i$, for $i \in [n]$, as described above

- select a random $d_i \leftarrow_{\$} \mathbb{Z}_p$ and set $D_i := d_i G$
- compute a proof of possession

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

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

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

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

$$\hat{C}_i := \hat{v}_i H + \hat{q}_i G \tag{4}$$
$$\hat{P}_i := \hat{B}_i + \hat{k}_i G \tag{5}$$

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

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

– sample uniformly at random $t \leftarrow_{\$} \mathbb{Z}_p$ and compute

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

with E as in (1)

– sample uniformly at random $y \leftarrow_{\$} \mathbb{Z}_p$ and compute

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

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

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

3.3 Transaction aggregation

Aggregate transactions are essentially a concatenation 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. Given transactions $\mathsf{tx}_0, \mathsf{tx}_1$, the aggregate transaction tx is computed as follows:

– if a P value in the input of one transaction appears in the inputs of the other, or a P value in the output of one transaction appears in the outputs of the other, abort

– concatenate the inputs of tx_0 and tx_1 as well as their outputs, and sort them lexicographically (this is required to hide which inputs and outputs comes from which transaction)

– compute the supply, the fee, offset, and stealth offset (from the supplies s_i, etc., of tx_i) of tx:

$$s := s_0 + s_1 \qquad f := f_0 + f_1 \qquad t := t_0 + t_1 \qquad y := y_0 + y_1$$

– concatenate the lists of excesses, and the lists of stealth excesses, together with the associated signatures σ. Sort each list lexicographically by the E values

The aggregate transaction is of the form

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

Note that simple transactions do not (need to) contain the (stealth) excesses, as they can be computed from other values of the transaction via Equations (1) and (2) (therefore displayed in light gray in Figure 1). In contrast, aggregate transactions (also displayed in Figure 1) will contain a list of excesses $\boldsymbol{E}$ and stealth excesses $\boldsymbol{X}$ together with lists for the associated proofs $\boldsymbol{\sigma}$.

3.4 Transaction verification

Simple transactions. A transaction

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

for which each P_i appears in an output $\mathsf{out}_i = (C_i, \pi_i, R_i, \rho_i, P_i)$, is valid if all of the following hold:

(i) all input proofs are valid: for all $i \in [n]$: $\mathsf{PoP.V}((P_i, D_i, \mathsf{out}_i), \psi_i) = \mathbf{true}$

(ii) all range proofs are valid: for all $i \in [\hat{n}]$: $\mathsf{RaP.V}(\hat{C}_i, \hat{\pi}) = \mathbf{true}$

(iii) all output signatures are valid: for all $i \in [\hat{n}]$: $\mathsf{PoP.V}(\hat{R}_i, \hat{C}_i \| \hat{\pi}_i \| \hat{P}_i, \hat{\rho}_i) = \mathbf{true}$ (where the message could also contain a time stamp, which is checked for legitimacy)

(iv) the excess proof of possession is valid: $\mathsf{PoP.V}((E, X, \varepsilon), \sigma) = \mathbf{true}$, for

$$E := \sum \hat{\boldsymbol{C}} - \sum \boldsymbol{C} + (f - s)H - tG$$
$$X := \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{D} - yG$$

Aggregate transactions. An aggregate transaction of the form (8) for which P_i appears in $\mathsf{out}_i = (C_i, \pi_i, R_i, \rho_i, P_i)$ is verified by checking that inputs are sorted lexicographically, verifying (i)–(iii) as for simple transactions, and checking the following variant of (iv):

(iv′) for all $i \in [\bar{n}]$: $\mathsf{PoP.V}((E_i, X_i, \varepsilon), \sigma_i) = \mathbf{true}$

(v′) additionally, the following "balance equations" are checked:

$$\sum \boldsymbol{E} = \sum \hat{\boldsymbol{C}} - \sum \boldsymbol{C} + (f - s)H - tG \tag{9}$$
$$\sum \boldsymbol{X} = \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{D} - yG \tag{10}$$

3.5 Output verification

We define when a holder of a view key should accept a given transaction output as payment to his stealth address. Given a view key (a, B), an amount v and a transaction output $\mathsf{out} = (\hat{C}, \hat{\pi}, \hat{R}, \hat{\rho}, \hat{P})$, compute $(k, q) := \mathrm{H}(a\hat{R}, (aG, B))$, and accept the payment if

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

3.6 Inclusion of transactions in the ledger

A ledger Λ is an aggregated transaction, whose inputs will also appear as outputs (since all inputs, in order to be spent, must previously appear as outputs). For this reason, we rearrange the terms in the ledger as:

$$\Lambda = \big(\boldsymbol{C}, \boldsymbol{\pi}, \boldsymbol{R}, \boldsymbol{\rho}, \boldsymbol{P}, (\mathsf{CT}, s, f, t, y, \boldsymbol{E}, \boldsymbol{X}, \boldsymbol{\sigma})\big)$$

where $\boldsymbol{C}, \boldsymbol{\pi}, \boldsymbol{R}, \boldsymbol{\rho}, \boldsymbol{P}$ is the list of unspent outputs, and $\mathsf{CT} := \big(C'_i, D'_i, R'_i, P'_i, \pi'_i, \psi'_i, \rho'_i\big)_{i\in[m]}$ are the *cut-through* terms: terms that would appear both as inputs and outputs. Note that the excess $\sum \boldsymbol{E}$ can be checked only using the output commitments, since the terms in CT cancel out.

Given a transaction tx of the form (7) or (8) and a ledger Λ, tx is included in the Λ as follows:

- check that every P_i, for $i \in [n]$, in the list of inputs of tx is contained in an output of Λ (this ensures that only coins in the ledger can be spent)
- aggregate Λ and tx as defined in Section 3.3, ignoring the cut-through list CT and considering Λ as an aggregate transaction with empty inputs. If this aborts, then return Λ. Return the merged transaction as the updated ledger with cut-through terms

  $$\mathsf{CT} := \mathsf{CT} + (\Lambda.\mathsf{out} - \mathsf{tx.in})$$

  where in the above "$\mathsf{a} + \mathsf{b}$" denotes the list of elements that appear either in a or in b sorted in lexicographic order, and "$\mathsf{a} - \mathsf{b}$" denotes the list of elements that appear in a but not in b, sorted in lexicographic order.

4 Fallacies in the initial proposal

We list below the main attacks that were found during the review of Yu's scheme and which motivated most of the design decisions for our scheme in Section 3. In Yu's protocol [Yu20, §2.2.2], the transaction does not include D, nor the stealth excess X and the respective offset y. Instead, a valid transaction must satisfy

$$E = \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{P} \ , \tag{11}$$

instead of the second equation in Item (iv) in Section 3.4. The proof of possession ψ proves only knowledge of the discrete logarithm of the one-time key P.

4.1 Correctness

Equation (11) can be achieved if one of the outputs, say the i-th, goes back to the creator of the transaction (e.g., because it is a "change output"). She can just sample q_i uniformly and set $\hat{R}_i := E + \sum \boldsymbol{P} - \sum_{j \neq i} \hat{R}_j$ (for which she knows $\log R_i$). However, one cannot create a transaction whose outputs are all linked to destination addresses, e.g., a transaction with a single output: R (together with the address) determines the coin opening q, which defines the value E; but (re-)defining R so that (11) holds would lead to a new value E. (Using 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 require that there is always a change output (and because it enables us to give a rigorous security analysis of our scheme), we introduced stealth excesses X, that along with the proof σ accounts for the "excess" in stealth addresses. We also introduce a *stealth offset*, so privacy of aggregated transactions is preserved.

4.2 Security: the feed-me attack

Analogously to how the stealth addresses were originally defined [vS13], and how they are currently used (e.g. in Monero[3]), in Yu's scheme [Yu20, §2.2.2], the one-time output key is defined "additively" as:

$$\hat{P}_i := \hat{k}_i G + B.$$

It turns out that computing the balance equation on the one-time keys as $X = \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{P}$ leads to an attack that affects the *transaction binding* property (which we introduce in Section 5.4). This vulnerability was originally found by `@south_lagoon77`, alias `kurt`.[4] Alice, an honest user with address (aG, B), creates two transactions tx_0 and tx_1, both with one input and one output, transferring respectively v_0 to Bob and $v_1 > v_0$ to Charlie:

$$(\ldots, C_0) \quad \leftarrow \quad \boxed{P_0, \psi_0 \mid \ldots, \hat{R}_0, \hat{\rho}_0, \ldots}_{\,t_0, y_0, E_0, X_0, \ldots} \tag{tx_0}$$

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

Both transaction are broadcast to the miners. A malicious miner can now forge a new transaction, transferring the amount $v_1 - v_0$ to himself, as follows:

$$(\ldots, C_1) \quad \leftarrow \quad \boxed{P_1, \psi_1 \mid \ldots, \hat{R}_0, \hat{\rho}_0, \ldots \| \ldots R^*, \rho^*, \ldots}_{\,t_0, y^*, E_0, X_0, \ldots} \ , \tag{tx^*}$$

copying the input of tx_1 and the output and the excesses from tx_0. It computes a second output "honestly", choosing $r^* \leftarrow_{\$} \mathbb{Z}_p$, setting $R^* = r^* G$ and signing any P value (knowing $\log P$) via ρ^*. (The miner can also create the corresponding coin C^*, so that Equation (9) is satisfied for tx^*,

[3] See `https://www.getmonero.org/resources/moneropedia/stealthaddress.html`

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

by setting $C^* := \mathsf{Cmt}(v_1 - v_0, q_1 - q_0)$, where q_0 and q_1 are the openings of C_0 and C_1, which the miner can either obtain by knowing Alice's view key, or *by having sent the outputs that are now being spent by* tx_0 *and* tx_1 *to Alice in the first place.*) Finally, the miner computes y^* so that Equation (10) is also satisfied for tx^*, that is

$$X_0 + y^*G = R^* + R_0 - P_1 \ .$$

By validity of tx_0, we have $X_0 + y_0G = R_0 - P_0$ and thus

$$y^*G = R^* - P_1 + P_0 + y_0G \ . \tag{12}$$

The crucial observation now is that if the miner knows the values k_0 and k_1 that define P_0 and P_1, resp. (which it can either obtain by knowing Alice's view key or by being the creator of the outputs spent by tx_0 and tx_1), then it knows the discrete logarithm of $P_0 - P_1 = (k_0 - k_1)G$, as the value B from Alice's address *cancels out*. The miner can thus compute y^* satisfying (12).

Possible fixes. To prevent these forms of attacks, our first attempt was to derive the one-time key multiplicatively, as $P_i = k_iB$ in Equation (5). The term $P_0 - P_1$ then becomes $(k_0 - k_1)B$, which is non-zero with overwhelming probability and therefore it becomes hard to compute y^*. More generally, as long as the adversary cannot find distinct values $k_{0,1}, \ldots, k_{0,n_0}, k_{1,1}, \ldots, k_{1,n_1}$ so that

$$\sum \boldsymbol{k}_0 - \sum \boldsymbol{k}_1 = 0 \ , \tag{13}$$

our scheme satisfies transaction binding.

However, Wagner's k-list tree algorithm [Wag02] can be used to find those $n_0 + n_1$ values in sub-exponential time. In order to be protected against active adversaries ready to invest substantial computing power, a user therefore would need to limit the number of its pending transactions at any point in time. In order to prevent this attack, which could impact scalability of the system, we follow the approach of Burkett [Bur21] and introduce an additional group element D, which replaces P in the balance equation. As it is chosen by the attacked spender (whereas P is chosen by the previous spender, who in this types of attacks is malicious), the values corresponding to the $k_{i,j}$ above are random, and thus the probability of (13) being satisfied is negligible.

4.3 Security: on transaction cut-through

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

While validity of the coin-balance equation (9) would be maintained, this is not the case for the "address equation" (10). One may thus consider (as Yu does [Yu20, §2.1.1] for (sufficiently old parts of) the ledger) to add a value Z defined as the difference of the sum of all removed $\hat{R}$ values and all removed $\hat{D}$ values to the aggregated transaction. The check in Equation (10) 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 signature $\hat{\rho}$ and defines Z as $\sum \boldsymbol{X} + yG + \sum \boldsymbol{D}$ minus the sum of the $\hat{\boldsymbol{R}}$ values including her own. This results in a valid transaction in 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, as in [Yu20, Eq. ③], it says:

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

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

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

$$(\ldots C_0) \quad \leftarrow \quad \boxed{P_0, D_0, \psi_0 \mid C_1, \pi_1, R_1, \rho_1, P_1}_{\,t_1, y_1, E_1, X_1, \sigma_1} \qquad (\mathsf{tx}_1)$$

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

where tx_1 spends some coin C_0 creating one output, which is then spent by tx_2 (that is, we have $\mathsf{PoP.V}((P_1, D_1), \psi_1) = \mathbf{true}$). If tx_1 and tx_2 could be merged as

$$(\ldots C_0) \quad \leftarrow \quad \boxed{P_0, D_0, \psi_0 \mid C_2, \pi_2, R_2, \rho_2, P_2}_{\,t_1+t_2, y_1+y_2, (E_1, E_2), (X_1, X_2), (\sigma_1, \sigma_2), Z}$$

which is valid if ψ_0 and ρ_2 (and π_2), as well as σ_1, σ_2, are valid on their respective messages, and the following holds:

$$C_2 - C_0 = E_1 + E_2 + (t_1 + t_2)G$$
$$R_2 - D_0 + Z = X_1 + X_2 + (y_1 + y_2)G$$

Then a miner could simply choose r^*, p^*, set $R_2^* := r^*G$, $P_2^* := p^*G$, create ρ_2^* honestly, define $Z^* := X_1 + X_2 + (y_1 + y_2)G - R_2^* + P_0$ and create the following (valid!) transaction:

$$(\ldots C_0) \quad \leftarrow \quad \boxed{P_0, D_0, \psi_0 \mid C_2, \pi_2, R_2^*, \rho_2, P_2^*}_{\,t_1+t_2, y_1+y_2, (E_1, E_2), (X_1, X_2), (\sigma_1, \sigma_2), Z^*}$$

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 P 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:

$$(\ldots C_0) \quad \leftarrow \quad \boxed{P_0, D_0, \psi_0 \mid R_1, P_1 \parallel \psi_1 \mid C_2, \pi_2, R_2, \rho_2, P_2}_{\,t_1+t_2, y_1+y_2, (E_1, E_2), (X_1, X_2), (\sigma_1, \sigma_2)}$$

Intuitively, not having ρ means that P_1 is not bound to R_1 anymore, which the following attack leverages, where given tx_1 and tx_2, the miner replaces tx_2 by a transaction it owns: The miner chooses p_1^* and r_2^*, sets $P_1^* := p_1^*G$ and $R_2^* := r_2^*G$, computes ψ_1^* and ρ_2^* honestly and sets $X_1^* := (r_2^* - p_1^* - y_1)G$ and computes σ_1^* honestly. It also chooses p_2^*, sets $P_2^* := p_2^*G$, creates a proof ρ_2^* on it using r_2^*, and computes X_2^* and σ_2^* honestly. Then the following is a valid transaction for which the miner knows the key to spend the output:

$$(\ldots C_0) \quad \leftarrow \quad \boxed{P_0, D_0, \psi_0 \mid R_1, P_1^* \parallel \psi_1^* \mid C_2, \pi_2, R_2^*, \rho_2^*, P_2^*}_{\,t_1+t_2, y_1+y_2, (E_1, E_2), (X_1^*, X_2^*), (\sigma_1^*, \sigma_2^*)}$$

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:

$$(\ldots C_0) \quad \leftarrow \quad \boxed{P_0, D_0, \psi_0 \mid C_1, \pi_1, R_1, \rho_1, P_1 \parallel C_2, \pi_2, R_2, \rho_2, P_2}_{\,t_1+t_2, y_1+y_2, (E_1, E_2), (X_1, X_2), (\sigma_1, \sigma_2)}$$

(Note that C_1 and π_1 need to be present for ρ_1 to be verifiable.) Consider an honest transaction tx_1

$$(\ldots C_0) \leftarrow \boxed{P_0, D_0, \psi_0 \mid C_1, \pi_1, R_1, \rho_1, P_1}_{\,t_1, y_1, E_1, X_1, \sigma_1}$$

A miner can steal the output as follows (again, assuming it knows the view key of its holder and thus the opening of C_1).

The miner computes a fresh output $(C_2, \pi_2, R_2, \rho_2, P_2)$ honestly (i.e., knowing the logarithms of R_2 and P_2), picks random R_1^* and X_2 (knowing the corresponding logarithms) and sets $D_1^* := R_1^* + R_2 - D_0 - X_1 - X_2 - t_1 G$. It signs P_1 (as well as C_1, and π_1) under R_1^* as ρ_1^* and produces σ_2 under E_2, X_2. It then publishes the following transaction:

$$(\ldots C_0) \quad \leftarrow \quad \boxed{P_0, D_0 \mid C_1, \pi_1, R_1^*, \rho_1, P_1^* \parallel C_2, \pi_2, R_2, \rho_2, P_2}_{\,t_1+t_2, y_1, (E_1, E_2), (X_1, X_2), (\sigma_1, \sigma_2)}$$

Note that the transaction is valid, since we have

$$R_1^* + R_2 - D_0 - D_1^* = X_1 + X_2 + t_1 G$$

and the miner knows the key to spend the output.

4.4 Security: the modified replay attack

Recall the replay attack for MW with non-interactive transactions mentioned by Yu [Yu20, §2.9.2]: a user is paid via some output out, which the user later spends. Then an adversary pays the user again, creating the exact same output; if the user accepts it, the adversary can replay the user's previous spend, making the latter lose the money.

The attack is prevented (without having the user store all previously spent outputs) by using time stamps and signatures (ψ, in our notation), which Yu [Yu20, §2.9.3] introduces in order to prevent rogue-key attacks. In his words, "Each *Input* must attach its own signature for $[P_i]$, as a second proof for the coin ownership". While it is not specified which message is signed, it is crucial that the entire output (and not just C) is signed, in particular, including the time stamp.

Otherwise, the above replay attack can be adapted: if the adversary created the output out then it can later simply change the time stamp (so the user accepts it), recompute ρ, and send it to the user again. If the signature contained in the user's spendings did not sign the time stamp (or ρ), then the previous spend would be valid again and the replay attack can still be mounted. This is why in MW-NIT we define ψ as a signature on the *entire* output.

5 Security analysis

5.1 Assumptions

In our security analysis, 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 6, we show that in an idealized model, namely 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 show how to extend 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.

5.2 Inflation resistance

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

> **Inflation-resistance game.** 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 the supply of tx^*).
> In addition to the amount $\hat{\boldsymbol{v}}$ of the outputs of tx^*, for MW-NIT the adversary must also return view keys (a_i, B_i), which have to accept the outputs out_i of tx^*. Letting s denote the ledger supply, s^* the supply and f^* the fee of tx^*, the adversary wins if
> $$s < \sum \hat{\boldsymbol{v}} + f^* - s^* \ . \tag{14}$$

Below we argue the following:

> *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.*

Security argument. 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

$$\Lambda^* = \big(\boldsymbol{C}, \boldsymbol{\pi}, \boldsymbol{R}, \boldsymbol{\rho}, \boldsymbol{P}, (\mathsf{CT}, s, f, t, y, \boldsymbol{E}, \boldsymbol{X}, \boldsymbol{\sigma})\big)$$

and a transaction

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

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. For Λ^* to accept tx^*, all values P_i must be in the ledger, that is, $\boldsymbol{P'} = (P_i)_{i\in I}$ for some $I \subseteq \mathbb{N}$. The reduction extracts all values v_i and openings q_i for all the output coins C_i contained in the ledger from the respective range proofs π_i; thus $C_i = \mathsf{Cmt}(v_i, q_i)$. Since the ledger is valid, from (9) we have

$$\sum \boldsymbol{E} = \sum \mathsf{Cmt}(v_i, q_i) + \mathsf{Cmt}(f - s, 0) - \mathsf{Cmt}(0, t) \ . \tag{15}$$

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

$$s = \sum \boldsymbol{v} + f \ . \tag{16}$$

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 (15). 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 (16) 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 6.)

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

$$\sum \boldsymbol{E}^* = \sum \hat{\boldsymbol{C}} - \textstyle\sum_{i\in I} C_i + \mathsf{Cmt}(f^* - s^*, 0) - \mathsf{Cmt}(0, t^*) \tag{17}$$

where I is the set of indices of the outputs in the ledger that are spent by tx^*. From σ_i^* contained in tx^*, the reduction can extract $e_i^* := \log E_i^*$ and, analogously to the above, from (17) we get

$$\mathsf{Cmt}\big(0, \sum e^*\big) = \mathsf{Cmt}\big(\sum \hat{\boldsymbol{v}} - \sum_{i\in I} v_i + f^* - s^*, \sum \hat{\boldsymbol{q}} - \sum_{i\in I} q_i - t^*\big) \ . \tag{18}$$

And thus If the adversary outputs tx^* satisfying (14), then we have

$$\sum_{i\in I} v_i \ \leq \ \sum \boldsymbol{v} + f \overset{(16)}{=} s \overset{(14)}{<} \sum \hat{\boldsymbol{v}} + f^* - s^* \ ,$$

and thus $\sum \hat{\boldsymbol{v}} + f^* - s^* - \sum_{i\in I} v_i \neq 0$ (since all terms are bounded by $v_{\max}$ and thus never wrap around the modular representation). This means that the two sides in (18) are two different openings of the commitment $\sum \boldsymbol{E}^*$, which represents a break of binding.

5.3 Theft resistance

We define two notions that protect users from losing coins. The first one is an adaptation of the notion from [FOS19] to a scheme with non-interactive transactions. It guarantees that coins 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' coins.[5] Instead, in MW-NIT, the security relies on the secrecy of the "*spend key*" for the user's stealth address. We strengthen the notion by assuming that the *view key* is known to the adversary (as delegating scanning for transactions should not endanger the security of these transactions).

> **Theft-resistance (from ledger).** *For any coin belonging to a user in the ledger (that is, it was accepted w.r.t. the user's view key), no matter how it was received (e.g., sent by the adversary), as long as the coin 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 a game akin to [FOS19, Fig. 8], where there is one honest user simulated by the experiment and the adversary can command the user's actions (create minting transactions for that user and make the user spend coins of hers which she accepted using her view key) and the user never accepts a coin she has already owned. The adversary wins the game if it spends any of the honest user's coins.

Remark. Note that this notion *does not prevent (a pure) replay attack*, where the user, after spending some output, is sent the exact same output again by the adversary, who can then replay the user's previous spending. Formally speaking, when an output was spent and is received a second time, then it *has already been spent*, so it is outside the scope of this definition. (Such replay attacks are protected against by *transaction-binding*, the next security notion described below.)

Further, note that the formal game for theft resistance for a scheme with non-interactive transactions would be a lot simpler than the one from [FOS19], which had to take care of the interactive spending protocol. In particular, this involves an instruction for the honest user to *receive* coins, and the definition of the coins an honest user owns in [FOS19] is cumbersome, whereas for non-interactive transactions, anything which the honest user accepts is considered an output belonging to her.

Below we argue the following:

> *If* PoP *is a simulation-sound proof of knowledge of discrete logarithms (cf. Section 6) and the DL assumption holds in the underlying group, then MW-NIT satisfies theft-resistance.*

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

Security argument. Consider a user with stealth key $(A = aG, B)$ owning an output $\mathsf{out} = (C, \pi, R, \rho, P)$ with amount v in the ledger, that is, out is accepted (see Section 3.5), meaning $P = B + kG$ where $(k, q) = \mathrm{H}(aR, (aG, B))$. Spending this coin requires proving possession of P with tag out, but an honest user, unless she spends that output, never proves possession of P.

We show that a theft can be used 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 the honest user's stealth address to $(A^* := a^*G, B^*)$ and gives the adversary the view key (a^*, B^*).

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

When an output $\mathsf{out} = (C, \pi, R, \rho, P)$ belonging to the user is spent by the adversary, then the transaction contains in the corresponding input a proof ψ^* of possession of (P, D^*) with tag out for some D^* (for which the algebraic adversary also returns a representation in basis (G, B^*)). By the definition of the security game, the honest user has never spent out before. This means that the reduction has never queried a simulated proof for (P, D^*, out). Therefore, by simulation-extractability in the AGM, the reduction can extract $p = \log P$, and since $P = B^* + kG$, for a value k known to the reduction, from which it computes the solution $p - k$ to its DL challenge.

5.4 Transaction-binding

Definition. While the previous notion states that once a user's coin is in the ledger it cannot be purloined, we define an additional notion, which guarantees that nothing can be "stolen" from a transaction *before* it is added to the ledger (this protects against malicious miners, for example). In particular, we want to guarantee that if a user produces a transaction tx then no one can create a transaction that contains *one* of the inputs of tx while not containing *all* its outputs.

Thus, while *theft-resistance* protects the outputs of a transaction, *transaction-binding* in some sense protects the inputs: even if an honest user spends a coin by putting it as an input into a transaction (and broadcasts it to be included in a block), then this input cannot later appear in another transaction, except for a transaction that contains *all* the outputs of the original transaction: since transactions can be aggregated, further inputs and outputs can be added to the original transaction. We formalize transaction-binding via the following game:

> **Transaction-binding.** The ledger is controlled by the experiment (which in reality is taken care of by the consensus mechanism). The experiment also simulates all honest users, and the adversary can ask for spawning new users, for which the experiment will create stealth addresses and give the corresponding view keys to the adversary.
> The adversary can instruct honest users to spend coins (which are in the ledger and owned by the corresponding user) to addresses of the adversary's choice (belonging to honest users or the adversary), and the experiment computes the corresponding transaction and gives it to the adversary. The adversary can commit any transactions (computed by itself or the experiment) to the ledger.
> The adversary's goal is to create a transaction (which is accepted by the ledger) which *contains input coins from an honest transaction*, but *does not contain all the output coins of that transaction that belong to honest users.*

Two remarks are in order.

(1) About honest users. While it might seem restrictive that only stealing output coins of honest users is considered a break of the notion, it is not. In aggregate cash systems like Mimblewimble

(with cut-through), an adversary can always "steal" output coins it owns from a transaction tx: it can simple create a transaction that spends these coins and then merge this transaction with tx; the result is a transaction in which some of the output coins have been replaced by new coins. We do thus not consider this an attack and transaction-binding gives no guarantees against it.

(2) Replay attacks. As users are not supposed to keep track of all the coins they have ever owned, they could be tricked into accepting a coin they had already owned and spent (the spender just needs to select the same r value). After receiving the coin again, the adversary could *replay* the transaction that spent the first instance of the coin and the user would lose her coin. This is prevented by including a time stamp [Yu20, §2.1.1] in the output (which is part of the message signed under the key $\hat{R}$). In order to be successful, the adversary would thus have to change the time stamp of the replayed attack, and this would also be considered a break of transaction-binding because the original output must be replaced by a new output with a different time stamp. The precision of the timestamp determines also the anonymity set of the transaction output.

Below we argue the following:

> *If* PoP *is a strongly simulation-sound proof of knowledge of discrete logarithms (cf. Section 6) and the DL assumption holds in the underlying group, then MW-NIT satisfies transaction-binding.*

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

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

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

(I) $D = D^*$. Assume this is not the case. We show that under simulation-extractability of PoP, this can be used to break DL by simulating the game as in the proof of theft-resistance.
Given a DL challenge B, the reduction embeds it into Alice's wallet (aG, B) for $a \leftarrow\!\!\$\, \mathbb{Z}_p$. It computes tx as prescribed, except that it queries its simulation oracle for the proof ψ (as its witness depends on $\log B$). When the adversary outputs a transaction with input (P, D^*, ψ^*), it can extract the witness (p, d^*) from ψ^* as the only simulated proof was for a different statement (P, D). Since $P = B + kG$, where k is computed from Alice's view key according to (3), the reduction can return $\log B = p - k$.

(II) $R^* \neq \hat{R}$. Assume $R^* = \hat{R}$. Since out^* must be different, either the tag $C^* \| \pi^* \| P^*$ (possibly including the time stamp) for ρ^*, or ρ^* itself is different. This means that the statement/proof pair is different from the one created by Alice, and one can thus extract $\log R^*$ by strong simulation extractability ("strong", since possibly only the proof is different).
Given a DL challenge $\hat{R}$, the reduction embeds it in the output of tx. Even without knowing $\log \hat{R}$, the reduction can compute $(\hat{k}, \hat{q}) = H(a'\hat{R}, (a'G, B'))$ using Bob's view key (a', B') (recall that Bob must be honest). From this, the reduction can compute $\hat{C}$, $\hat{\pi}$ and $\hat{P}$. For the proofs $\hat{\rho}$ and σ, whose witnesses require knowledge of $\log \hat{R}$, the reduction queries its simulation oracle. If the adversary outputs 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) and thus solve 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 definitions of the stealth excesses in (2) we get $R^* - D - y^*G = \hat{R} - D - yG$, where we used $D^* = D$ from (I). Thus $\hat{R} = R^* + (y - y^*)G$. Since ρ^* proves knowledge of $\log R^*$, this means the adversary must also know $\log \hat{R}$. As in case (II), the reduction embeds a DL challenge as $\hat{R}$ and uses its oracle to obtain the proofs $\hat{\rho}$ and $\hat{\sigma}$ to compute tx. If the adversary returns a forgery violating (III), then from ρ^* the reduction can extract $r^* = \log R^*$ (since, by (II), ρ^* is for $R^* \neq \hat{R}$, and the queried proof $\hat{\rho}$ was for $\hat{R}$) and thus compute the DL solution $r^* + y - y^*$.

(IV) Finally, we show that the adversary cannot win the game when satisfying (I)–(III) either. From (I), we have $X^* := R^* - D - y^*G$. Since σ^* contained in tx^* proves knowledge of $x^* := \log X^*$ (since $X^* \neq X$ by (III)) and ρ^* proves knowledge of $r^* := \log R^*$ (since $R^* \neq \hat{R}$ by (II)), this means that the adversary must know $\log D = r^* - y^* - x^*$. The reduction embeds its DL challenge as D, the value in the input of tx, and queries its simulation oracle for proofs ψ and σ (whose DL are or depend on 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 have ever been simulated), and from σ^* the reduction extracts x^* (the simulated proof σ was under $X \neq X^*$) and computes $d := r^* - y^* - x^*$ as above.

While the above provide a general intuition, the actual argument is made more complex by considering the "complete" scenario:

– Alice may create other transactions, of which the adversary might reuse parts in its tx^*
– The transaction tx^* might be an aggregated transaction, meaning that it has a list of stealth excesses, of which parts can be reused from transactions by Alice and others may be "fresh".

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

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

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

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

Hybrid H_1 to H_2. We next show that both added abort conditions in H_2 can only be satisfied with negligible probability, again by reduction of DL assuming strong SE of PoP.

The reduction receives a DL instance $Z \in \mathbb{G}$ makes a *guess* (i', ι) so that tx' will be the i'-th transaction and the ι-th output out_ι of tx' will be for an honest user and such that $\hat{R}_\iota$ from out_ι will either (a) be contained in some output out^* of tx^* but $\mathsf{out}_\iota \neq \mathsf{out}^*$ or (b) $\hat{R}_\iota$ will not be contained in tx^* and X' will be in the stealth excess list of tx^*.

Assume that the adversary wins H_1. If (A2) occurs then there exists i' and ι so that (a) occurs, otherwise all outputs for honest users in tx' would be contained in tx^* and the adversary would not win. Moreover (A3) implies (b). Thus if H_1 returns 1 then H_2 does so too, except if (a) or (b) occur. We show that in both cases, the reduction can compute $\log Z$ if its guess was correct.

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

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

and, from this, C_ι, P_ι and π_ι as prescribed. It queries its oracle for a simulated proof ρ_ι for $(R_\iota, C_\iota \| \pi_\iota \| P_\iota)$ (possibly including a time stamp), which completes out_ι. To complete the transaction, it also queries a simulated σ' (whose witness depends on $\log R_\iota$).

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

Now assume that the reduction guessed correctly and that (b) occurs. Recall that this means that the adversary's transaction

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

is such that $\boldsymbol{X}$ contains a stealth excess X' of tx' and $\hat{\boldsymbol{R}}$ does not contain R_ι. By validity of tx', it satisfies the balance equation (10) and since $R_\iota = Z$, we have

$$X' = \textstyle\sum_{i \neq \iota} R_i + Z - \sum \boldsymbol{D} - y'G \ . \tag{19}$$

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

$$\textstyle\sum_{i \notin I^*} X_i + mX' = \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{D}^* - yG \ . \tag{20}$$

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

Plugging (19) into (20), taking the logarithm to base G and substituting all logarithms of group elements known to the reduction thus yields:

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

meaning the reduction can solve the DL challenge.

Hybrid H_2. It remains to show that H_2 can only be won with negligible probability. From abort conditions (A1)–(A3), it follows that the adversary can only win H_2 if

(W1) one of the D values in tx^* is from an input of tx' (otherwise (A1) would occur)
(W2) the stealth excess list $\boldsymbol{X}$ of tx^* does not contain the stealth excess X' of tx' (otherwise, either (A2) or (A3) must would occur).

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

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

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

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

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

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

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

$$\log \bar{X}_i = \textstyle\sum_j r_{i,j} - \sum_j (d_{i,j} \cdot \log Z) - y_i \qquad\qquad \log \bar{D}_i = \bar{d}_i \cdot \log Z \tag{21}$$

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

$$\log D_i^* =: d_i^* \qquad\qquad \log X_i^* =: x_i^* \qquad\qquad \log \hat{R}_i =: \hat{r}_i \tag{22}$$

Since tx^* is valid, it satisfies the balance equation (10), that is:

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

Taking the logarithm to base G, and using (21) and (22) this yields:

$$\textstyle\sum x_i^* + \sum\sum r_{i,j} - \sum\sum (d_{i,j} \cdot \log Z) - \sum y_i \ \equiv_p \ \sum \hat{r}_i - \sum d_i^* - \sum (\bar{d}_i \cdot \log Z) - y^* \ . \tag{23}$$

We show that with overwhelming probability:

$$\textstyle\sum \bar{d}_i - \sum\sum d_{i,j} \not\equiv_p 0 \ . \tag{24}$$

Indeed, by (W1), at least one value $\bar{d}_{i^*}$ comes from tx'. Moreover, by (W2) the stealth excess of tx' does not appear in tx^*, and thus the value $\bar{d}_{i^*}$ is not among $(d_{i,j})_{i,j}$ (except with negligible probability if the reduction chose the same value twice). Whenever (24) is satisfied, the reduction can thus compute $\log Z$ from (23).

5.5 Transaction privacy

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

Grin introduced *transaction offsets* and network protocol changes such as *Dandelion* [VFV17] that provide stronger anonymity guarantees. To accommodate for this new privacy feature, and for the new features of non-interactive transactions, we present a stronger privacy notion than the one provided in [FOS19], which cannot be achieved without offsets. We stress that our analysis is limited to the cryptographic properties of the scheme and that we do not consider possible external attack vectors, e.g. network adversaries.

Definition. Informally, our scheme provides three basic anonymity guarantees:

- a transaction hides the amounts contained in its inputs and outputs as well as
- the destination addresses of the outputs (and inputs), except to the receivers of the transaction;
- in an aggregated transaction, it is not possible to tell which inputs and outputs belonged to the same component transaction.

The above follow immediately from the following simulation-based definition:

Transaction privacy. There exists a simulator which can *simulate* a transaction tx when given the following:
- transaction outputs $\mathsf{out}_1, \ldots, \mathsf{out}_n$ which will be spent by tx
- the number ℓ of transactions aggregated in tx, and the total number $\hat{n}$ of outputs
- the supply and the fee

so that no adversary, given an *address oracle*, which creates and returns stealth addresses (A_j, B_j), can distinguish whether a *challenge oracle*, on inputs
- the number of component transactions ℓ
- for every component transaction i:
  - a vector of transaction outputs $\mathbf{out}_i$ together with associated values $\boldsymbol{v}_i$ and spending keys $(\boldsymbol{a}_i, \boldsymbol{b}_i)$, so that for all $j \in [|\mathbf{out}|]$, the pair $(\mathsf{out}_{i,j}, v_{i,j})$ is accepted by the view key $(a_{i,j}, b_{i,j}G)$
  - output values $\hat{\boldsymbol{v}}_i$ and associated addresses $(\boldsymbol{A}_i, \boldsymbol{B}_i)$ returned by the address oracle
  - the supply s_i and fee f_i, so that $\sum \hat{\boldsymbol{v}}_i + f_i = \sum \boldsymbol{v}_i + s$ (and all values are in $[0, v_{\max}]$)

either (1) creates transactions $\mathsf{tx}_1, \ldots, \mathsf{tx}_\ell$, where tx_i spends $\mathbf{out}_i$ using keys $(\boldsymbol{a}_i, \boldsymbol{b}_i)$ creating outputs of values $\hat{\boldsymbol{v}}_i$ for recipient addresses $(\boldsymbol{A}_i, \boldsymbol{B}_i)$,
or (2) runs the simulator on input the list $\mathbf{out}_1 \| \ldots \| \mathbf{out}_\ell$, ordered lexicographically, the numbers ℓ and $\sum_{i=1}^{\ell} |\hat{\boldsymbol{v}}_i|$ (i.e., the total number of outputs), as well as $\sum_{i=1}^{\ell} s_i$ and $\sum_{i=1}^{\ell} f_i$.

Below, we argue the following:

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.*

Security argument. We first define the simulator. On input $\mathbf{out} = (C_i, \pi_i, R_i, \rho_i, P_i)_{i=1}^{n}$, ℓ, $\hat{n}$, s and f, it does the following:

- pick random values $D_1, \ldots, D_n$, $\hat{C}_1, \ldots, \hat{C}_{\hat{n}}, \hat{R}_1, \ldots, \hat{R}_{\hat{n}}, \hat{P}_1, \ldots, \hat{P}_{\hat{n}} \leftarrow\$\, \mathbb{G}$
- pick random values $E_2, \ldots, E_\ell, X_2, \ldots, X_\ell \leftarrow\$\, \mathbb{G}$, as well as $t, y \leftarrow\$\, \mathbb{Z}_p$

– set $E_1 := \sum \hat{\boldsymbol{C}} - \sum \boldsymbol{C} + (f-s)H - tG - \sum_{i=2}^{\ell} E_i$
  and $X_1 := \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{D} - yG - \sum_{i=2}^{\ell} X_i$ (note that we could have $\ell = 1$)
– using the respective simulators guaranteed by zero-knowledge, simulate the range proofs $\hat{\pi}_i$ for statements $\hat{C}_i$, for all $i \in [\hat{n}]$, as well as the following: for all $i \in [n]$: ψ_i for (P_i, D_i) with tag out_i; for $i \in [\hat{n}]$: ρ_i for $\hat{R}_i$ with tag $\hat{C}_i \| \hat{\pi}_i \| \hat{P}_i$; for $i \in [\ell]$: σ_i for (E_i, X_i) with tag ε.

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

H_0 In the "real" game, the component transactions are created as described (in Section 3.2) using the spending keys; they are then aggregated (Section 3.3).
H_1 In the first hybrid, all RaP proofs π and PoP proofs ψ, ρ and σ (cf. Section 6) in the transaction are simulated. By the zero-knowledge property of both primitives, this change is indistinguishable from the honest computation of the range proofs and the signatures.
H_2 For every output, in the generation of the key shares in Equation (3), the first argument $\hat{r}_{i,j} A_{i,j}$ of the hash function is replaced by a random value $Z_{i,j} \leftarrow_{\$} \mathbb{G}$. This is indistinguishable by the DDH assumption, noting that all $\hat{R}_{i,j}$ and $A_{i,j}$ are created by the reduction and their logarithms are never used in the simulation of the game (the former because $\rho_{i,j}$ and σ_i are simulated; the latter because the address oracle never reveals any secret keys).
H_3 The game aborts if the adversary at some point queries $(Z_{i,j}, (A_{i,j}, B_{i,j}))$ for some i, j to the random oracle. Since the adversary has no information on $Z_{i,j}$, the probability of aborting is negligible. Note that in H_3, the values $\hat{k}_{i,j}$ and $\hat{q}_{i,j}$ are uniformly random and independent.

We now argue that a transaction generated in H_3 is distributed equivalently to a transaction computed by the simulator, which concludes the proof.

In a transaction produced in H_3, since the coin openings $\hat{q}_{i,j}$ are uniformly random and independent, and by the definition of $E_1, \ldots, E_\ell$, we have that for fixed values $s := \sum s_i$ and $f := \sum f_i$, the tuple $\big((\hat{\boldsymbol{C}}_i)_{i=1}^{\ell}, (E_i)_{i=1}^{\ell}, \sum t_i\big)$ is uniformly random conditioned on

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

This is exactly how the simulator produces these values.

Since the values $\hat{k}_{i,j}$ are uniformly random and independent, the values $P_{i,j}$ are uniform and independent, as the simulator produces them. Finally, by the definition of the values $D_{i,j}$, $\hat{R}_{i,j}$ and X_i, the tuple $\big((\boldsymbol{D}_i)_{i=1}^{\ell}, (\hat{\boldsymbol{R}}_i)_{i=1}^{\ell}, (X_i)_{i=1}^{\ell}, \sum y_i\big)$ is uniformly random conditioned on $\sum X_i = \sum \sum \hat{R}_{i,j} - \sum \sum D_{i,j} - (\sum y_i)G$. Again, this is how the simulator generates this tuple. Finally, in a transaction produced in H_3, all RaP and PoP proofs are simulated, which is how the simulator generates them.

6 Simulation-extractability of Schnorr signatures in the AGM

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

$$\big\{\big((X, Y, m), (x, y)\big) \;:\; X = xG \wedge Y = yG \wedge m \in \{0,1\}^*\big\} \;. \qquad (25)$$

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 (25) and $\{((X, m), x) \;:\; X = xG\}$.

```
$\mathsf{Sch.Setup}(1^\lambda)$
$(p, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda)$
Select $\mathrm{H}\colon \{0,1\}^* \to \mathbb{Z}_p$
**return** $par := (p, \mathbb{G}, G, \mathrm{H})$

$\mathsf{Sch.Sign}(sk, m)$
$(p, \mathbb{G}, G, \mathrm{H}, x) := sk\,;\ r \leftarrow\!\!\$\, \mathbb{Z}_p\,;\ R := rG$
$c := \mathrm{H}(xG, R, m)\,;\ s := r + cx \bmod p$
**return** $\sigma := (R, s)$

$\mathsf{Sch.KeyGen}(par)$
$(p, \mathbb{G}, G, \mathrm{H}) := par\,;\ x \leftarrow\!\!\$\, \mathbb{Z}_p\,;\ X := xG$
$sk := par \,\|\, x\,;\ pk := par \,\|\, X$
**return** $(sk, pk)$

$\mathsf{Sch.Ver}(pk, m, \sigma)$
$(p, \mathbb{G}, G, \mathrm{H}, X) := pk\,;\ (R, s) := \sigma$
$c := \mathrm{H}(X, R, m)$
**return** $(sG = R + cX)$
```

Fig. 2. Key-prefixed Schnorr signature scheme $\mathsf{PoP}[\mathsf{GrGen}]$ based on a group generator GrGen.

```
$\mathsf{PoP.Setup}(1^\lambda)$
$(p, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda)$
Select $\mathrm{H}\colon \{0,1\}^* \to \mathbb{Z}_p$
**return** $par := (p, \mathbb{G}, G, \mathrm{H})$

$\mathsf{PoP.V}\big(par, (X, Y, m), \psi\big)$
$(p, \mathbb{G}, G, \mathrm{H}) := par\,;\ (R, s) := \psi$
$(c, d) := \mathrm{H}(X, Y, m, R)$
**return** $(sG = R + cX + dY)$

$\mathsf{PoP.P}\big(par, (X, Y, m), (x, y)\big)$
$(p, \mathbb{G}, G, \mathrm{H}) := par$
$r \leftarrow\!\!\$\, \mathbb{Z}_p\,;\ R := rG$
$(c, d) := \mathrm{H}(X, Y, m, R)$
$s := r + cx + dy \bmod p$
**return** $\psi := (R, s)$
```

Fig. 3. Batch Schnorr zero-knowledge simulation-extractable proofs of knowledge of two logarithms $\mathsf{PoP}[\mathsf{GrGen}]$ w.r.t. a group generator GrGen.

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

Simulation-extractability. Strong simulation extractability for the above language means that from any adversary that returns a proof ψ^* for a statement (X^*, Y^*, m^*), the witness $(\log X^*, \log Y^*)$ can be extracted; and this holds even if the adversary gets access to an oracle that on inputs (X_i, Y_i, m_i) returns simulated proofs ψ_i for these statements. The only restriction is that the 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 (akin to UNF-CRO as defined 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 [FKL18], adversaries are assumed to return a *representation* of any group element that they return. This means that, after being given input group elements $Z_1, \ldots, Z_n$, whenever the adversary returns a group element X, it must also return coefficients $\zeta_1, \ldots, \zeta_n$ with $X = \sum \zeta_i Z_i$.

All our proofs (except for the privacy notion) are reductions of solving the discrete logarithm (DL) problem to breaking some security notion of our scheme MW-NIT, assuming that PoP satisfies (strong) simulation-extractability (SE). The reduction thus receives a DL challenge Z and simulates the security game to the adversary, which is assumed to be algebraic. Since the reduction relies on SE of PoP in the AGM, it must itself be algebraic; however, it can only return representations in basis (G, Z), which are its own group-element inputs. All our reductions will run the adversary on some group elements $X_1, \ldots, X_n$, which the reductions produce from G and Z in an "algebraic" way (i.e., they know 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).

We thus assume that there is some group element Z in addition to the generator G, and that the algebraic adversary in the (strong) SE game for PoP must accompany all its group-element outputs (in particular, those part of the statements (X_i, Y_i, m) queried to the simulation oracle and those part of the statement(s) from which the extractor must extract the witness) by representations in basis (G, Z). We refer to this as the *AGM with discrete-logarithm challenge*.

Security of PoP. In this "weakening" of the AGM we now show the following.

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

Proof. The game is parametrized by a group $(p, \mathbb{G}, G)$ and a "DL challenge" $Z \in \mathbb{G}$, and a random oracle H, all provided to the adversary. 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 $\mathrm{H}(X_i, Y_i, m_i, R_i) = (c_i, d_i)$. Sience 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 query by values $\alpha_i, \beta_i, \gamma_i, \delta_i \in \mathbb{Z}_p$ so that $X_i = \alpha_i G + \beta_i Z$ and $Y_i = \gamma_i G + \delta_i Z$. 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

$$\begin{aligned} R_i = s_i G - c_i X_i - d_i Y_i &= (s_i - c_i\alpha_i - d_i\gamma_i)G - (c_i\beta_i + d_i\delta_i)Z \\ &= \mathit{\Gamma}_i G - \mathit{\Delta}_i Z \quad \text{with } \mathit{\Gamma}_i := s_i - c_i\alpha_i - d_i\gamma_i \text{ and } \mathit{\Delta}_i := c_i\beta_i + d_i\delta_i \end{aligned} \tag{26}$$

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

$$(X^*, Y^*, m^*, R^*, s^*) \neq (X_i, Y_i, m_i, R_i, s_i) \quad \text{for all } i. \tag{27}$$

Validity means

$$R^* + c^* X^* + d^* Y^* = s^* G \quad \text{with} \quad (c^*, d^*) = \mathrm{H}(X^*, Y^*, m^*, R^*)\ . \tag{28}$$

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., (27) 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

$$\begin{aligned} X^* &= \gamma_X G + \zeta_X Z + \sum_{i=1}^{q} \xi_i R_i \overset{(26)}{=} \Big(\gamma_X + \sum_{i=1}^{q} \xi_i \varGamma_i\Big) G + \Big(\zeta_X - \sum_{i=1}^{q} \xi_i \varDelta_i\Big) Z \;, \\ Y^* &= \Big(\gamma_Y + \sum_{i=1}^{q} \upsilon_i \varGamma_i\Big) G + \Big(\zeta_Y - \sum_{i=1}^{q} \upsilon_i \varDelta_i\Big) Z \quad \text{and} \qquad (29) \\ R^* &= \Big(\gamma_R + \sum_{i=1}^{q} \rho_i \varGamma_i\Big) G + \Big(\zeta_R - \sum_{i=1}^{q} \rho_i \varDelta_i\Big) Z \end{aligned}$$

where the equalities follow from (26). Substituting X^*, Y^* and R^* in (28) by the above RHSs and grouping the coefficients of Z and G yields

$$\begin{aligned} &\Big(\zeta_R - \sum_{i=1}^{q} \rho_i \varDelta_i + c^*\Big(\zeta_X - \sum_{i=1}^{q} \xi_i \varDelta_i\Big) + d^*\Big(\zeta_Y - \sum_{i=1}^{q} \upsilon_i \varDelta_i\Big)\Big) Z \\ &\qquad = \Big(s^* - \Big(\gamma_R + \sum_{i=1}^{q} \rho_i \varGamma_i\Big) - c^*\Big(\gamma_X + \sum_{i=1}^{q} \xi_i \varGamma_i\Big) - d^*\Big(\gamma_Y + \sum_{i=1}^{q} \upsilon_i \varGamma_i\Big)\Big) G \;. \qquad (30) \end{aligned}$$

First consider the case where the representations of X^* and Y^* in (29) are independent of Z, that is

$$\zeta_X - \sum_{i=1}^{q} \xi_i \varDelta_i \equiv_p 0 \equiv_p \zeta_Y - \sum_{i=1}^{q} \upsilon_i \varDelta_i \;. \qquad (31)$$

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

Otherwise, at least one term in (31), which are the coefficients of c^* and d^* in (30), is non-zero. The adversary chose the values $\alpha_i, \beta_i, \delta_i, \gamma_i$ (which define $\varGamma_i$ and $\varDelta_i$) 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 (30) are defined, and moreover at least one of c^* and d^* is not multiplied by 0. The probability that the coefficient of Z in (30) is congruent to 0 modulo p is thus $\frac{1}{p}$. From (30), the reduction can then efficiently compute $\log Z$ with overwhelming probability, and from the representations of X^* and Y^* in (29), it can compute the witness $(\log X^*, \log Y^*)$. $\square$

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. *Schnorr signatures (Figure 2) are strongly simulation-extractable proofs of secret keys in the AGM-with-DL-challenge and the ROM.*

References

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

Bur21. David Burkett. One-sided transactions in Mimblewimble (consensus layer), 2021. Available at: `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*, volume 10992 of *LNCS*, 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*, volume 11476 of *LNCS*, 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*, volume 12106 of *LNCS*, 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*, volume 7237 of *LNCS*, pages 554–571. Springer, 2012.

Tod14. Peter Todd. Stealth addresses, 2014. Available at: 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. Manuscript available at https://cryptonote.org/whitepaper.pdf.

Wag02. David Wagner. A generalized birthday problem. *CRYPTO 2002*, volume 2442 of *LNCS*, 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.