# 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 analyze and fix a proposal by Yu, which, inspired by stealth addresses, introduces non-interactive transactions to Mimblewimble.

## 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 was 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 Gary 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.

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* $\pi$ 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 $\sigma$ 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* $\mathsf{tx}$ 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 $\mathsf{tx}$, they can simply be removed from the input and output list (together with their range proofs), while validity of $\mathsf{tx}$ will be maintained. This has been called *transaction cut-through* in the literature [Max13b]. In a cryptocurrency based on MW, ledger is defined as the (cut-through) of the aggregation of all transactions. Since every spent coin ("spent 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 $\mathsf{tx}$ are ordered lexicographically, one can still determine which inputs and outputs come from the same component transaction, since $\mathsf{tx}$ 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 $\sigma$,[1] whose secret key depends on the openings of the sender's and the receiver's coins. FOS [FOS19] proposed a transaction protocol, where the sender creates all output coins, and can thus complete the transaction by computing $\sigma$ 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 $\mathrm{H}$ 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 $(C, \pi)$ 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 one-time secret key. To prevent the value $P$ from being modified (which would mean stealing it from the receiver), a signature $\rho$ on $P$ under signing key $R$ (of which the sender knows the logarithm) is added. An output is thus of the form $(C, \pi, R, \rho, P)$.

Moreover, the mechanism for letting the receiver derive the secret key for $P$ can now 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 view the so-called "view key" $(a, B)$ of stealth

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

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 3 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], that overcomes the found issues (Section 2). 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 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).

For inflation resistance and theft prevention, 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*. For transaction-binding, we introduce a new assumption (see Section 4.1), which states that an adversary that can ask for random values in $\mathbb{Z}_p$ (with $p = |\mathbb{G}|$) cannot arrange these values (using every value at most once) into two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ so that

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

$\sum \boldsymbol{x} = \sum \boldsymbol{y} \pmod{p}$. Concretely, for the security of our scheme it suffices to require the size of these vectors to be upper-bounded by the maximum number inputs of pending transactions an attacked user creates (i.e., transactions broadcast to miners that are not yet included in the ledger). We note that this problem can be solved in sub-exponential time thanks to the $k$-list tree algorithm [Wag02]. In order to be protected against active adversaries ready to invest substantial computing power, a user therefore needs to limit the number of its pending transactions at any point in time.

# 2 A new proposal for MW with non-interactive transactions

## 2.1 Preliminaries

Let $\varepsilon$ 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 $\lambda$. (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 model the employed cryptographic hash function as a random oracle and denote it by $\mathrm{H}(\cdot)$. We employ a homomorphic commitment scheme, specifically Pedersen commitments, for which 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 $C \overset{?}{=} vH + qG$. 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).

**Schnorr signatures.** We use (key-prefixed) Schnorr signatures. A signature on a message $m \in \{0,1\}^*$ under public key $X = xG \in \mathbb{G}$, is computed via $\mathsf{Sch.Sign}(x, m)$, which picks a uniform $r \leftarrow\!\!\$\, \mathbb{Z}_p$ and returns a signature $(R, s) \in \mathbb{G} \times \mathbb{Z}_p$ with $R := rG$ and $s := r + \mathrm{H}(X, R, m) \cdot x \bmod p$. The verification algorithm $\mathsf{Sch.Vf}(X, m, (R, s))$ checks whether $sG = R + \mathrm{H}(X, R, m) \cdot X$ (see Fig. 2). Schnorr signatures are unforgeable under the DL assumption in the random oracle model (ROM), and their security has been extensively studied in the past literature [Seu12].

We use Schnorr signatures for different *types* of values (e.g., signatures $\sigma$ under excesses $E$ or signatures $\rho$ under $R$). We assume that these are "domain-separated", which can easily achieved by including the type in the message that is being signed. 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 Schnorr signatures.

In Section 5 we show that in the *algebraic group model* [FKL18] combined with the ROM, Schnorr signatures are *simulation-sound zero-knowledge proofs of knowledge* of the secret key, a property that we will be central in the security analysis of our protocol. The statements for the proof system are public-key/message pairs $(X, m)$ for which the witness is $x = \log X$. *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) that produces a valid proof for a statement $(X, m)$ can extract the witness $\log X$. Proofs are *simulation-sound* (also called *simulation-extractable* for PoKs) if a witness can be extracted from a prover even if the prover can obtain simulated proofs for statements $(X_i, m_i)$ of its choice (except the one it is proving).

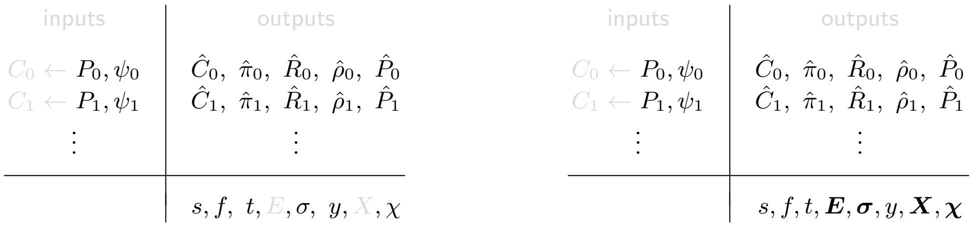
inputs
outputs
$C_0 \leftarrow P_0, \psi_0$
$C_1 \leftarrow P_1, \psi_1$
$\hat{C}_0,\ \hat{\pi}_0,\ \hat{R}_0,\ \hat{\rho}_0,\ \hat{P}_0$
$\hat{C}_1,\ \hat{\pi}_1,\ \hat{R}_1,\ \hat{\rho}_1,\ \hat{P}_1$
$s, f,\ t,\ E, \sigma,\ y,\ X, \chi$
inputs
outputs
$C_0 \leftarrow P_0, \psi_0$
$C_1 \leftarrow P_1,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

**Fig. 1. Left:** Visualization of a (simple) MW-NIT transaction. Inputs consist of a value $P_i$ form a previous transaction output and a signature $\psi_i$. 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 excess signature $\sigma$ (valid under $E$) and the stealth excess signature $\chi$ (valid under $X$). 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.

**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:

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

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

## 2.2 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:

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

    (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 of $\hat{C}$ to a value $v \in [0, v_{\max}]$
  - a **one-time output public key** $\hat{P} \in \mathbb{G}$, computed from $\hat{k}$ as $\hat{P} := \hat{k}B$ (note that the *spend* key is required to compute $\log \hat{P}$)
  - a **signature** $\hat{\rho}$ under verification key $\hat{R}$ on message $\hat{C} \| \hat{\pi} \| \hat{P}$ (and possibly a time stamp)
- A list of **inputs** of the form $(P, \psi)$ 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) and
  - $\psi$ is a signature under verification key $P$ on the spent output
- The **kernel**, which is composed of:
  - the **supply** $s \in [0, v_{\max}]$, indicating the amount of money created in the transaction
  - the **fee** $f \in [0, v_{\max}]$, indicating the fee paid for the current transaction

- the **offset** $t \in \mathbb{Z}_p$
- the **excess** $E \in \mathbb{G}$, defined as the difference between the commitments in the outputs (including the fee) and the inputs (including the supply), shifted by the offset. If $C_i$ is the $i$-th input commitment, that is, the value contained in the output in which $P_i$ appears, then

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

which can be seen as $E \coloneqq E' - tG$ in terms of the *true excess* $E' \coloneqq \sum \hat{\boldsymbol{C}} + fH - \sum \boldsymbol{C} - sH$
- a **signature** $\sigma$ under $E$ on the empty message $\varepsilon$
- 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 one-time keys $P_i$ from the inputs, shifted by the stealth offset $y$

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

- a **signature** $\chi$ under $X$ on the message $E$

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

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

### 2.3 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:

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

(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 $\log P = kb'$.

To create a transaction that spends transaction outputs $\mathsf{out}_i$ 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 $\sum \hat{\boldsymbol{v}} + f = \sum \boldsymbol{v} + s$, first compute all values $q_i$ and $p_i := \log P_i$, for $i \in [n]$, as above and proceed as follows:

- for each input index $i \in [n]$, compute the signature on the output being spent

$$\psi_i \leftarrow \mathsf{Sch.Sign}(p_i, \mathsf{out}_i)$$

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

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

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

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

$$\hat{P}_i \coloneqq \hat{k}_i B_i \tag{5}$$

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

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

- 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), and, from that, a signature under the excess $\sigma \leftarrow \mathsf{Sch.Sign}(e, \varepsilon)$
- sample uniformly at random $y \leftarrow_{\$} \mathbb{Z}_p$ and compute

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

with $X$ as in (2), and, from that, a signature under the stealth excess $\chi \leftarrow \mathsf{Sch.Sign}(x, E)$

The final transaction is

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

## 2.4 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 it turns out this is insecure (see Section 3.3). Given transactions $\mathsf{tx}_0, \mathsf{tx}_1$, the aggregate transaction $\mathsf{tx}$ 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 $\mathsf{tx}_0$ and $\mathsf{tx}_1$ 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 (from the supplies $s_i$ of $\mathsf{tx}_i$), the fee, offset, and stealth offset of $\mathsf{tx}$ as follows:

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

- concatenate the lists of excesses, together with the associated signatures $\sigma$, and the lists of stealth excesses, together with the associated signatures $\chi$ and sort each list lexicographically

The aggregate transaction is of the form

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

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 signatures $\boldsymbol{\sigma}$ and $\boldsymbol{\chi}$.

## 2.5 Transaction verification

**Simple transactions.** A transaction

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

referencing outputs $\big(\mathsf{out}_i = (C_i, \pi_i, R_i, \rho_i, P_i)\big)_{i\in[n]}$ is valid if all of the following hold:

(i) all input signatures are valid: for all $i \in [n]$ : $\mathsf{Sch.Vf}(P_i, \mathsf{out}_i, \psi_i) = \mathbf{true}$
(ii) all range proofs are valid: for all $i \in [\hat{n}]$ : $\mathsf{Zkp.Vf}(\hat{C}_i, \hat{\pi}) = \mathbf{true}$
(iii) all output signatures are valid: for all $i \in [\hat{n}]$ : $\mathsf{Sch.Vf}(\hat{R}_i, \hat{C}_i \| \hat{\pi}_i \| \hat{P}_i, \hat{\rho}_i) = \mathbf{true}$ (where the message could also contain a time stamp, which is checked for legitimacy)
(iv) the excess signature is valid: $\mathsf{Sch.Vf}(E, \varepsilon, \sigma) = \mathbf{true}$, for

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

(v) the stealth excess signature is valid: $\mathsf{Sch.Vf}(X, E, \chi) = \mathbf{true}$, for

$$X := \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{P} - yG$$

**Aggregate transactions.** An aggregate transaction of the form (8) referencing outputs $\big(\mathsf{out}_i = (C_i, \pi_i, R_i, \rho_i, P_i)\big)_{i\in[n]}$ is verified by checking that inputs are sorted lexicographically, verifying (i)–(iii) as for simple transactions, and checking the following variants of (iv) and (v):

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

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

$$\textstyle\sum \boldsymbol{X} = \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{P} - yG \tag{10}$$

### 2.6 Output verification

We finally 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 $(k, q) := \mathrm{H}(a\hat{R}, (aG, B), \hat{R})$, and accept the payment if

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

## 3 Fallacies in the initial proposal

We list below the main attacks that were found during the review of Yu's scheme [Yu20] and which motivated most of the design decisions for our scheme in Section 2.

### 3.1 Correctness

In Yu's protocol [Yu20, §2.2.2], there are no stealth excesses $X$ (nor the corresponding signatures $\chi$). A valid transaction must satisfy

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

instead of Item (v) in Section 2.5. This 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 signature $\chi$ accounts for the "excess" in stealth addresses. We also introduce a *stealth offset*, so privacy of aggregated transactions is preserved.

### 3.2 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:

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

instead of "multiplicatively", as in our scheme in Equation (5). It turns out that computing the one-time key additively leads to an attack that affects the *transaction binding* property. This

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

vulnerability was originally found by `@south_lagoon77`, alias `kurt`.[4] Using the notation of the preceding sections, Alice, an honest user with address $(aG, B)$, creates two transactions $\mathsf{tx}_0$ and $\mathsf{tx}_1$, both with one input and one output, transferring respectively $v_0$ to Bob and $v_1 > v_0$ to Charlie:

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

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

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:

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

copying the input of $\mathsf{tx}_1$ and the output and the excesses from $\mathsf{tx}_0$. 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 $\rho^*$. (The miner can also create the corresponding coin $C^*$, so that Equation (9) is satisfied for $\mathsf{tx}^*$, 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* $\mathsf{tx}_0$ *and* $\mathsf{tx}_1$ *to Alice in the first place.*) Finally, the miner computes $y^*$ so that Equation (10) is also satisfied for $\mathsf{tx}^*$, that is

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

By validity of $\mathsf{tx}_0$, 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 $\mathsf{tx}_0$ and $\mathsf{tx}_1$), 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).

To prevent these forms of attacks, in our scheme the one-time keys are defined as $P_i = k_iB$ in Equation (5). The term $P_0 - P_1$ then becomes $(k_0 - k_1)B$. In Section 4.4, we show that as long as the adversary cannot find values $k_0$ and $k_1$, output by the random oracle and with $k_0 - k_1 = 0$ (and, more generally, values $k_{0,1}, \ldots, k_{0,n_0}, k_{1,1}, \ldots, k_{1,n_1}$ so that $\sum \boldsymbol{k}_0 - \sum \boldsymbol{k}_1 = 0$), our scheme satisfies transaction binding. This precisely is guaranteed by the assumption we introduce (Section 4.1). Note that all values $k_{i,j}$ must be distinct, as every transaction must have a unique $P$ value. Moreover, note that the number of elements $(n_0 + n_1)$ is upper-bounded by the number of inputs in pending transactions of the attacked user.

## 3.3 Security: on transaction cut-through

Suppose that, in an aggregate transaction, an output $(C, \pi, R, \rho, P)$ of one transaction is referenced as input $(P, \psi)$ 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 "RmP" defined as the difference of the sum of all removed $\hat{R}$ values and all removed $\hat{P}$ values to the aggregated transaction. The check in Equation (10) would

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

be replaced by $\sum \hat{\boldsymbol{R}} - \sum \boldsymbol{P} + \mathrm{RmP} = \sum \boldsymbol{X} + yG$, where the sums are only over the indices that have not been removed in the outputs ($\sum \hat{\boldsymbol{R}}$) and the inputs ($\sum \boldsymbol{P}$).

However, since "RmP" 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 RmP as $\sum \boldsymbol{X} + yG + \sum \boldsymbol{P}$ 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. $\textcircled{3}$], 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 $\textcircled{3}$ and $\textcircled{4}$ incorrect.

**Example.** Consider two 1-input/1-output transactions $\mathsf{tx}_1$ and $\mathsf{tx}_2$ (assume that all supplies and fees are 0).

$$(\dots C_0) \quad \leftarrow \quad \boxed{P_0, \psi_0 \mid C_1, \pi_1, R_1, \rho_1, P_1}_{\,t_1, E_1, \sigma_1, y_1, X_1, \chi_1} \qquad (\mathsf{tx}_1)$$

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

where $\mathsf{tx}_1$ spends some coin $C_0$ creating one output, which is then spent by $\mathsf{tx}_2$ (that is, we have $\mathsf{Sch.Vf}(P_1, C_1, \psi_1) = \mathbf{true}$ ). If $\mathsf{tx}_1$ and $\mathsf{tx}_2$ could be merged as

$$(\dots C_0) \quad \leftarrow \quad \boxed{P_0, \psi_0 \mid C_2, \pi_2, R_2, \rho_2, P_2}_{\,t_1+t_2, (E_1, E_2), (\sigma_1, \sigma_2), y_1+y_2, (X_1, X_2), (\chi_1, \chi_2), \mathrm{RmP}}$$

which is valid if $\psi_0$ and $\rho_2$ (and $\pi_2$), as well as $\sigma_1$, $\sigma_2$, $\chi_1$ and $\chi_2$, are valid on their respective messages, and the following holds:

$$C_2 - C_0 = E_1 + E_2 + (t_1 + t_2)G$$
$$R_2 - P_0 + \mathrm{RmP} = X_1 + X_2 + (y_1 + y_2)G$$

Then a miner could simply choose $r^*, p^*$, set $R_2^* := r^*G$, $P_2^* := p^*G$, create $\rho_2^*$ honestly, define $\mathrm{RmP}^* := X_1 + X_2 + (y_1 + y_2)G - R_2^* + P_0$ and create the following (valid!) transaction:

$$(\dots C_0) \quad \leftarrow \quad \boxed{P_0, \psi_0 \mid C_2, \pi_2, R_2^*, \rho_2, P_2^*}_{\,t_1+t_2, (E_1, E_2), (\sigma_1, \sigma_2), y_1+y_2, (X_1, X_2), (\chi_1, \chi_2), \mathrm{RmP}^*}$$

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

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

$$(\dots C_0) \quad \leftarrow \quad \boxed{P_0, \psi_0 \mid R_1, P_1 \parallel \psi_1 \mid C_2, \pi_2, R_2, \rho_2, P_2}_{\,t_1+t_2, (E_1, E_2), (\sigma_1, \sigma_2), y_1+y_2, (X_1, X_2), (\chi_1, \chi_2)}$$

Intuitively, not having $\rho$ means that $P_1$ is not bound to $R_1$ anymore, which the following attack leverages, where given $\mathsf{tx}_1$ and $\mathsf{tx}_2$, the miner replaces $\mathsf{tx}_2$ 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 $\psi_1^*$ and $\rho_2^*$ honestly and sets $X_1^* := (r_2^* - p_1^* - y_1)G$ and computes $\chi_1^*$ honestly. It also chooses $p_2^*$, sets $P_2^* := p_2^*G$, creates a signature

$\rho_2^*$ on it using $r_2^*$, and computes $X_2^*$ and $\chi_2^*$ honestly. Then the following is a valid transaction for which the miner knows the key to spend the output:

$$(\ldots C_0) \quad \leftarrow \quad \boxed{P_0, \psi_0 \mid R_1, P_1^* \parallel \psi_1 \mid C_2, \pi_2, R_2^*, \rho_2^*, P_2^*}_{\,t_1+t_2,(E_1,E_2),(\sigma_1,\sigma_2),y_1+y_2,(X_1^*,X_2^*),(\chi_1^*,\chi_2^*)}$$

Finally, we show that **removing $\psi$ but keeping $\rho$** 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:

$$(\ldots C_0) \quad \leftarrow \quad \boxed{P_0, \psi_0 \mid C_1, \pi_1, R_1, \rho_1, P_1 \parallel C_2, \pi_2, R_2, \rho_2, P_2}_{\,t_1+t_2,(E_1,E_2),(\sigma_1,\sigma_2),y_1+y_2,(X_1,X_2),(\chi_1,\chi_2)}$$

(Note that $C_1$ and $\pi_1$ need to be present for $\rho_1$ to be verifiable.) Consider an honest transaction $\mathsf{tx}_1$

$$(\ldots C_0) \leftarrow \boxed{P_0, \psi_0 \mid C_1, \pi_1, R_1, \rho_1, P_1}_{\,t_1,E_1,\sigma_1,y_1,X_1,\chi_1}$$

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 $P_1^* := R_1^* + R_2 - P_0 - X_1 - X_2 - t_1 G$. It signs $P_1^*$ (as well as $C_1$ and $\pi_1$) under $R_1^*$ as $\rho_1^*$ and produces $\chi_2$ under $X_2$. It then publishes the following transaction:

$$(\ldots C_0) \quad \leftarrow \quad \boxed{P_0, \psi_0 \mid C_1, \pi_1, R_1^*, \rho_1, P_1^* \parallel C_2, \pi_2, R_2, \rho_2, P_2}_{\,t_1+t_2,(E_1,E_2),(\sigma_1,\sigma_2),y_1,(X_1,X_2),(\chi_1,\chi_2)}$$

Note that the transaction is valid, since we have

$$R_1^* + R_2 - P_0 - P_1^* = X_1 + X_2 + t_1 G$$

and the miner knows the key to spend the output.

### 3.4 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 $\mathsf{out}$, 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 ($\psi$, 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 $\mathsf{out}$ then it can later simply change the time stamp (so the user accepts it), recompute $\rho$, and send it to the user again. If the signature contained in the user's spendings did not sign the time stamp (or $\rho$), then the previous spend would be valid again and the replay attack can still be mounted. This is why in MW-NIT we define $\psi$ as a signature on the *entire* output.

## 4 Security analysis

### 4.1 Assumptions

In our security analysis, we assume that the range proofs are proofs of 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 $(v = 0, q = \log C)$.) We thus assume that there exists an extractor that from (an adversary outputting) a range proof can extract the values $v$ and $q$.

We assume that (key-prefixed) Schnorr signatures are a strongly simulation-sound zero-knowledge proof of knowledge of the secret key. In Section 5, we show that this is the case in an idealized model, namely the combination of the random-oracle model and the algebraic group model [FKL18]. We also 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. We furthermore make the following "subset-sum"-type hardness assumption:

**Assumption 1.** Let $n_{\max} \in \mathbb{N}$. Consider a (polynomial-time) adversary that has access to a random oracle returning values in $\mathbb{Z}_p$ (where $p$ is the order of the discrete-log hard group $\mathbb{G}$). Then it is infeasible for the adversary to find vectors of values $\boldsymbol{x}$ and $\boldsymbol{y}$ output by the oracle so that:

- $\sum \boldsymbol{x} = \sum \boldsymbol{y} \mod p$
- no value appears twice in $\boldsymbol{x} \| \boldsymbol{y}$
- $|\boldsymbol{x}| + |\boldsymbol{y}| \leq n_{\max}$

It is possible to frame this problem as a generalized birthday problem, which is solvable by $k$-list tree [Wag02] in time $O(n_{\max} \cdot 2^{\lceil \log p \rceil / (1 + \lfloor \log n_{\max} \rfloor)})$. The $k$-list algorithm, when provided access to $k$ random oracles $\mathrm{H}_1, \ldots, \mathrm{H}_k$, allows to find different values $a_1, \ldots, a_k$ such that

$$\mathrm{H}_1(a_1) + \mathrm{H}_2(a_2) + \cdots + \mathrm{H}_k(a_k) = 0 \pmod{p}. \tag{13}$$

Consider $k = n_{\max}$, and let $\mathrm{H}_1, \ldots, \mathrm{H}_{|\boldsymbol{x}|}$ be random oracles that forward any query to the random oracle $\mathrm{H}$ using the same input, and return whatever $\mathrm{H}$ replies, and oracles $\mathrm{H}_{|\boldsymbol{x}|}, \ldots, \mathrm{H}_{n_{\max}-1}$ that forward the query to $\mathrm{H}$, but swap the sign of the response before returning. Then, the $k$-list algorithm finds elements $(a_1, \ldots, a_{n_{\max}})$ such that $(\mathrm{H}_1(a_1), \ldots, \mathrm{H}_{n_{\max}}(a_{n_{\max}}))$ satisfies (13), which is also a valid solution also for Assumption 1.

### 4.2 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 $\mathsf{tx}^*$ (accepted by the ledger) that spends an amount that exceeds the supply of the ledger (plus its own supply).
>
> In addition to the amount $\hat{\boldsymbol{v}}$ of the outputs of $\mathsf{tx}^*$, for MW-NIT the adversary must also return view keys $(a_i, B_i)$, which have to accept the outputs $\mathsf{out}_i$ of $\mathsf{tx}^*$. Letting $s$ denote the ledger supply, $s^*$ the supply and $f^*$ the fee of $\mathsf{tx}^*$, the adversary wins if
>
> $$s < \sum \hat{\boldsymbol{v}} + f^* - s^* \ . \tag{14}$$

Below we argue the following:

> *If the used range proof system is extractable and if the discrete-logarithm assumption holds in the underlying group, then MW-NIT satisfies inflation resistance in the ROM.*

**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

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

and a transaction

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

with all $P'_i$ contained in the ledger, that is, $\boldsymbol{P'} = (P_i)_{i\in I}$ for some $I \subseteq \mathbb{N}$, together with output values $(\hat{v}_i)_{i\in[\hat{n}]}$ and view keys $(a_i, B_i)_{i\in[\hat{n}]}$. 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 $\pi_i$; thus $C_i = \mathsf{Cmt}(v_i, q_i)$. Since the ledger is valid, from (9) we have (note that there are no inputs in a ledger):

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

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

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

The pair $\big(\sum \boldsymbol{v} + f - s, \sum \boldsymbol{q} - t\big)$ is an opening of the right-hand side of (15). Thus if (16) did not hold then the pair would be an opening to a *non-zero* value of $\sum \boldsymbol{E}$. Together with the signatures $\boldsymbol{\sigma}$ under keys $\boldsymbol{E}$, this precisely represents a break of the security notion *EUF-NZO*, defined in [FOS19, Def. 5].

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

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

where $I$ is the set of indices of the outputs in the ledger that are spent by $\mathsf{tx}^*$. The same reasoning as for the ledger shows that $\mathsf{tx}^*$ must be balanced, that is

$$s^* + \textstyle\sum_{i\in I} v_i = \sum \hat{\boldsymbol{v}} + f^* \ , \tag{17}$$

otherwise $(\sum \hat{\boldsymbol{v}} - \sum_{i\in I} v_i + f^* - s^*, \sum \hat{\boldsymbol{q}} - \sum_{i\in I} q_i - t)$, together with $\boldsymbol{E}^*$ and $\boldsymbol{\sigma}^*$ would represent a break of EUF-NZO. Together this yields

$$s \overset{(16)}{=} \textstyle\sum \boldsymbol{v} + f \geq \sum_{i\in I} v_i \overset{(17)}{=} \sum \hat{\boldsymbol{v}} + f^* - s^* \ ,$$

which contradicts the condition (14) required for the adversary to win.

Thus, to break inflation-resistance, the adversary must break either knowledge-soundness of the range proofs or the notion EUF-NZO. In [FOS19, Lemma 16], it is shown that Pedersen commitments and Schnorr signatures together satisfy EUF-NZO (via a tight security reduction to DL in the ROM).

## 4.3 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. The notion 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) and the user never accepts a coin it 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 is a lot simpler than the one from [FOS19], which had to take care of the interactive spending protocol. In particular, this involved 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 Schnorr signatures are simulation-sound proofs of knowledge of their secret keys (cf. Section 5) and the the discrete-logarithm assumption holds in the underlying group, then MW-NIT satisfies theft-resistance.*

**Security argument.** Consider a user with stealth key $(A = aG, B)$ owning an output $\mathsf{out} = (\hat{C}, \hat{\pi}, \hat{R}, \hat{\rho}, \hat{P})$ in the ledger. Then $\hat{P}$ is a temporary address derived from the user's stealth address $(A, B)$ and $\hat{R}$ as $\hat{P} := \hat{k}B$ (see (3) and (5)). (If this was not the case, the user would not have accepted the coin.) Spending this coin requires a signature under $\hat{P}$ on the output, but an honest user, unless she spends that output, never signs the output with a key related to her spending key $B$.

We show that a theft can be used to break the DL assumption assuming simulation-extractability of Schnorr signatures. 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 her, the reduction computes the transaction as specified, except that it does not know the logarithm of $P'$ (since it does not know the logarithm of $B^*$). The reduction thus queries its simulation oracle for a signature $\psi$ on $\mathsf{out}'$ under $P'$.

When an output $\mathsf{out} = (\hat{C}, \hat{\pi}, \hat{R}, \hat{\rho}, \hat{P})$ belonging to the user is spent by the adversary, then the transaction contains in its input a signature $\psi^*$ on $\mathsf{out}$ under $\hat{P}$. By the definition of the security game, the honest user has never spent $\mathsf{out}$ before. This means that the reduction has never queried a simulated signature on $\mathsf{out}$ under $\hat{P}$. Therefore, by simulation-extractability, the reduction can extract $\hat{p} = \log \hat{P}$, and since $\hat{P} = \hat{k}B^*$, it can thus compute the solution $\hat{p}\hat{k}^{-1}$ to its DL challenge (by collision resistance of the hash function used to derive $\hat{k}$, the probability that $\hat{k} = 0$ is negligible).

[5] Note that this is unavoidable for non-interactive transactions: knowing the (sum of) the receivers' keys is necessary to compute the kernel signature $\sigma$.

### 4.4 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* (that has not been committed to the ledger (otherwise → theft-resistance)), 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, 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 Schnorr signatures are strongly simulation-sound proofs of knowledge of their secret keys (cf. Section 5), the discrete-logarithm assumption holds in the underlying group, and Assumption 1 (cf. Section 4.1) holds for $n_{max}$ defined as the maximal number of inputs of pending transactions by any user, 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

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

spending $\mathsf{out}$ to Bob. Moreover, assume this is the only transaction Alice ever makes. The adversary's goal is now to replace the output in $\mathsf{tx}$ by a different output $(C^*, \pi^*, R^*, \rho^*, P^*)$, yielding a "forged" transaction $\mathsf{tx}^*$, which we assume to be a *simple* (i.e., non-aggregate) transaction. We consider three different cases:

(I) $R^* = \hat{R}$. Since the output in $\mathsf{tx}^*$ is different, either the message $C^* \| \pi^* \| P^*$ (possibly including the time stamp) signed by $\rho^*$, or $\rho^*$ itself is different. This means that $\rho^*$ is a "strong" forgery (since the message could have remained the same).

For consistency with the remaining cases, we reduce (I) to strong simulation-extractability of Schnorr and hardness of DL. Given a DL challenge $\hat{R}$, the reduction embeds it in the output of $\mathsf{tx}$. Although it does not know $\log \hat{R}$, the reduction can still compute $(\hat{k}, \hat{q}) = H(a'\hat{R}, (a'G, B'), \hat{R})$ 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 signatures $\hat{\rho}$ and $\hat{\chi}$, whose secret keys require knowledge of $\log \hat{R}$, the reduction queries its simulation oracle.

If the adversary is successful in case (I), from $\rho^*$ the reduction can extract $\log \hat{R}$ (since $\rho^*$ or the queried message must be different) and thus solve the DL challenge.

(II) $R^* \neq \hat{R}$ and $X^* = X$, that is, the stealth excess of $\mathsf{tx}^*$ is the same as that of $\mathsf{tx}$. From the definitions of the stealth excesses in (2), we get $R^* - P - y^*G = \hat{R} - P - yG$ and thus $\hat{R} = R^* + (y - y^*)G$. Since $\rho^*$ proves knowledge of $\log R^*$, this means the adversary must also know $\log \hat{R}$.

As in case (I), the reduction embeds a DL challenge as $\hat{R}$ and uses its oracle to obtain the signatures $\hat{\rho}$ and $\hat{\chi}$ to compute $\mathsf{tx}$.

If the adversary is successful in case (II), from $\rho^*$ the reduction can extract $r^* = \log R^*$ (since $\rho^*$ is for a key different from that of the queried signature $\hat{\rho}$) and thus compute the DL solution $r^* + y - y^*$.

(III) $R^* \neq \hat{R}$ and $X^* \neq X$, that is, the stealth excess $X^* := R^* - P - y^*G$ of $\mathsf{tx}^*$ is different from the stealth excess of $\mathsf{tx}$. Since $\chi^*$ contained in $\mathsf{tx}^*$ proves knowledge of $x^* := \log X^*$ and $\rho^*$ proves knowledge of $r^* := \log R^*$, this means that the adversary must also know $\log P = r^* - y^* - x^*$.

This can again be used to solve a DL challenge; however, now we cannot set $P$ to be the challenge, since $P$ is computed by the adversary. Instead, the reduction embeds a DL challenge $B$ into Alice's stealth address $(aG, B)$. When the reduction creates $\mathsf{tx}$, it does not know $p := \log P$, so it queries its simulation oracle for signatures $\psi$ and $\chi$ (whose secret keys are or depend on $p$), all other values are computed as prescribed.

If the adversary is successful in case (III), from $\rho^*$ the reduction extracts $r^*$ (no $\rho$ signatures have ever been simulated), and from $\chi^*$ the reduction extracts $x^*$ (the simulated signature $\chi$ was under $X \neq X^*$) and computes $p := r^* - y^* - x^*$ as above. From Alice's view key, the reduction can now compute $k$ with $P = kB$ and returns the DL solution $p \cdot k^{-1} = \log B$.

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 $\mathsf{tx}^*$
- The transaction $\mathsf{tx}^*$ 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 distinguish three types of attacks, which generalize the ones above. Let $\mathsf{tx}^*$ be the adversary's forgery.

(I) $\mathsf{tx}^*$ contains an output whose $\hat{R}$ value is the same as in the output of an honest transaction, but the output is different

(II) there exists a transaction $\mathsf{tx}'$ with stealth excess $X'$ and output values $\boldsymbol{R}'$ by the user, so that $\boldsymbol{X}$ of $\mathsf{tx}^*$ contains $X'$, but $\mathsf{tx}^*$ does not contain *all* values $\boldsymbol{R}'$

(III) $\mathsf{tx}^*$ does not contain all the $\hat{R}$ values of the attacked transaction $\mathsf{tx}'$ and the stealth excess list $\boldsymbol{X}$ of $\mathsf{tx}^*$ does not contain the stealth excess $X'$ of $\mathsf{tx}'$

We start by showing how breaks of Type (I) are reduced to strong simulation-extractability of Schnorr proofs and hardness of computing DL in $\mathbb{G}$. The reduction receives a DL instance $R^* \in \mathbb{G}$ and guesses which output index $\iota$ in which transaction $\mathsf{tx}'$ will have its $R$-value reused, but the message $m_\iota$ signed by $\hat{\rho}_\iota$ under $\hat{R}_\iota$ altered (thus $\hat{C}_\iota$ or $\hat{P}_\iota$ or $\hat{\pi}_\iota$ (or its timestamp)) or $\hat{\rho}_\iota$ modified. When creating this transaction, the reduction sets $\hat{R}_\iota := R^*$ (and thus does not know its discrete logarithm). It completes the transaction as follows: since the receiver of the output is an honest user, the reduction knows its view key $(a_\iota, B_\iota)$ and can thus compute:

$$(\hat{k}_\iota, \hat{q}_\iota) \coloneqq \mathrm{H}(a_\iota R^*, (A_\iota, B_\iota), R^*) \ .$$

From this, it computes $C$, $P$ and $\pi$ as prescribed for this output. Knowing all $q$ values, it also computes $\sigma$. It queries its Schnorr simulation oracle for the signature $\hat{\rho}_\iota$ and the stealth excess signature $\chi$. Assume the adversary's break is of Type (I) and that the reduction guessed correctly which output was modified (namely the one having $R^*$ embedded). Then the returned transaction contains a signature $\rho^*$ under $R^*$ on a message $m^*$ composed of the corresponding values $\hat{C}$, $\hat{\pi}$, $\hat{P}$ (and possibly a time stamp) so that $(m^*, \rho^*) \neq (\hat{m}, \hat{\rho})$ (otherwise the outputs are the same). Since $(R^*, m^*)$ has not been queried to the simulation oracle (recall that $\rho$ and $\chi$ signatures are domain-separated, cf. Section 2.1), by simulation extractability of Schnorr, the reduction can extract $\log R^*$ and thus solve the DL challenge.

The reduction of Type (II) is very similar: the reduction guesses which transaction $\mathsf{tx}'$ will have its stealth excess included in $\mathsf{tx}^*$, and which index $\iota$ will be such that $\hat{R}'_\iota$ in $\mathsf{tx}'$ will not be present in $\mathsf{tx}^*$. The reduction embeds a DL challenge $R^*$ as $\hat{R}'_\iota$ and simulates the transaction (querying signatures $\hat{\rho}'_\iota$ and $\chi'$ to its Schnorr oracle) exactly as in for Type (I) above.

By validity of $\mathsf{tx}'$, it satisfies the "balance equation" (10)

$$X' = \sum \hat{\boldsymbol{R}}' - \sum \boldsymbol{P}' - y'G \ , \tag{18}$$

where $\hat{R}'_\iota = R^*$, and all values $r'_i = \log \hat{R}'_i$, for $i \neq \iota$, and $p'_i = \log P'_i$ are known to the reduction.

Since $\mathsf{tx}^*$ is of Type (II) for the stealth excess list $\boldsymbol{X}$ of $\mathsf{tx}^*$, we have $X_{i^*} = X'$ for some $i^*$. By validity of $\mathsf{tx}^*$ we further have

$$\textstyle\sum_{i \neq i^*} X^*_i + X' = \sum \hat{\boldsymbol{R}} - \sum \hat{\boldsymbol{P}} - yG \tag{19}$$

Plugging (18) into (19) yields

$$\textstyle\sum_{i \neq i^*} X^*_i + \sum_{i \neq \iota} \hat{R}'_i + R^* - \sum \boldsymbol{P}' - y'G = \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{P} - yG \ . \tag{20}$$

From the signatures $\chi_i$ contained in $\mathsf{tx}^*$, the reduction can extract the values $x_i = \log X_i$ (since the only simulated $\chi$ signature was for $X' \neq X_i$ for all $i \neq i^*$). From the signatures $\psi_i$ contained in $\mathsf{tx}^*$, the reduction can extract the values $p_i = \log P_i$ (since no $\psi$ signatures are simulated). From the signatures $\rho_i$ contained in $\mathsf{tx}^*$, the reduction can extract the values $r_i = \log \hat{R}_i$ (since by assumption,

$R'_\iota$ (for which a $\rho$ signature was simulated) is not among $\hat{\boldsymbol{R}}$). Taking the logarithm to base $G$ of (20) and substituting all logarithms of group elements known to the reduction thus yields:

$$\log R^* = -\textstyle\sum_{i \neq i^*} x_i^* - \sum_{i \neq \iota} \boldsymbol{r}'_i + \sum \boldsymbol{p}' + y' + \sum \boldsymbol{r} - \sum \boldsymbol{p} - y \ ,$$

meaning the reduction can solve the DL challenge.

It remains to show how breaks of type (III) can again be reduced to the hardness of DL (assuming simulation-extractability of Schnorr signatures), and additionally making Assumption 1.

The challenger now embeds a DL challenge $B$ into the wallet $(aG, B)$ of an honest user $U$. Whenever the adversary asks for a transaction spending coins from $U$'s wallet, the input signatures $\psi$ under the keys $P$, as well as the stealth excess signature $\chi$, are simulated. At the end of the execution, the adversary returns a forged transaction $\mathsf{tx}^*$ of the form

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

where $\boldsymbol{P}$ contains an input $P'$ from an honest transaction $\mathsf{tx}'$, the vector $\hat{\boldsymbol{R}}$ does not contain some value $R_\iota$ from the outputs of $\mathsf{tx}'$ and $X'$ is not contained in $\boldsymbol{X}$. In $\mathsf{tx}^*$ we distinguish two types of inputs and two types of stealth excesses:

- let $\bar{P}_i$ denote the inputs that appear in outputs of transactions created by $U$, and $P_i^*$ be those that do not
- let $\bar{X}_i$ denote the stealth excesses that appear in transactions created by $U$ and $X_i^*$ be those that do not

For all $\bar{X}_i$ appearing in a transaction $\mathsf{tx}_i$ by $U$, let $P_{i,j}$ and $\hat{R}_{i,j}$ denote the values contained in the inputs and outputs of $\mathsf{tx}_i$ and let $y_i$ be its stealth offset. By validity of $\mathsf{tx}_i$ we have:

$$\bar{X}_i = \textstyle\sum_j \hat{R}_{i,j} - \sum_j P_{i,j} - y_i G \ . \tag{21}$$

For these values $P_{i,j}$ and the values $\bar{P}_i$ contained in $\mathsf{tx}^*$ (which all belong to $U$), we have, by construction, $P_{i,j} = k_{i,j} \cdot B$ and $\bar{P}_i = k_i \cdot B$ for values $k_{i,j}$ and $k_i$ known to the reduction. (They are the respective first elements of $\mathrm{H}(aR', (A, B), R')$ from (5) for some values $R'$ chosen by the sender of that transaction to $U$.) Moreover, the reduction knows the values $r_{i,j} := \log \hat{R}_{i,j}$, as they were chosen by $U$.

From the signatures $\psi_i$ corresponding to the values $P_i^*$ and $\chi_i$ corresponding to $X_i^*$ contained in $\mathsf{tx}^*$, we can extract the following (since, by definition, $P_i^*$ and $X_i^*$ are values that have not been used by $U$ and thus no signatures have been simulated for them):

- $p_i^* := \log P_i^*$
- $x_i^* := \log X_i^*$

Moreover, from the signatures $\rho_i$ contained in $\mathsf{tx}^*$ we can extract (as no $\rho$ signatures are simulated)

- $r_i := \log \hat{R}_i$

Since $\mathsf{tx}^*$ is valid, it satisfies the "balance equation" (10), that is:

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

where $P' = \bar{P}_i$ for some $i$. Substituting $\bar{X}_i$ via (21), taking the logarithm to base $G$, and using the values known to the reduction, this yields:

$$\textstyle\sum \boldsymbol{x}_i^* + \underbrace{\sum\sum r_{i,j} - \sum\sum k_{i,j} \cdot \log B - \sum y_i}_{= \log \sum \bar{\boldsymbol{X}}} \ = \ \sum r_i - \sum p_i^* - \underbrace{(\sum k_i) \log B}_{= \log \sum \bar{\boldsymbol{P}}} - y^* \ . \tag{22}$$

From this equation, the reduction can compute $\log B$ if

$$\sum\sum k_{i,j} \neq \sum k_i \ , \tag{23}$$

that is, if the sum of $P$ values in the inputs of $U$'s transactions (those whose stealth excess was included in $\mathsf{tx}^*$) is different from the sum of $P$ values in the inputs of $\mathsf{tx}^*$ (those that are contained in inputs of $U$'s transactions).

We show that an adversary achieving a transaction-binding break of Type (III) by having equality in Equation (23) has solved the problem in Assumption 1. In particular, from the values in (23), which are random-oracle outputs, we will construct two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, with no repeating values such that $\sum \boldsymbol{x} = \sum \boldsymbol{y} \pmod p$. Since the reduction can solve the DL challenge if the inequality in (23) holds, this completes the proof.

We first show that there are no repetitions on the LHS and RHS of (23). There are no repetitions among the $k_{i,j}$ values: this follows from all $P$ values in outputs in the ledger being distinct and $U$ not committing any double-spendings. Moreover, there are no repetitions among the $k_i$ values either, as otherwise $\mathsf{tx}^*$ would not be a valid transaction.

Next, there is at least one value $k_i$ which does not appear among the $k_{i,j}$, namely the value $k'$ corresponding to $P'$ from the attacked honest transaction $\mathsf{tx}'$. Indeed, for Type (III) $k'$ is among the $k_i$ (since $P'$ is among the inputs of $\mathsf{tx}^*$); on the other hand, $\mathsf{tx}'$, which is the only transaction of $U$'s that has $P'$ among its inputs, does not appear on the right hand side of (22) (since $X'$ is not contained in $\boldsymbol{X}$).

Removing all values that appear on both sides in Equation (23) yields thus a solution to the problem in Assumption 1.

### 4.5 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* 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 when given the following inputs:
> - transaction outputs $\mathsf{out}_1, \ldots, \mathsf{out}_n$
> - the number of transactions $\ell$ that were aggregated, and the total number of outputs $\hat{n}$
> - the supply and the fee

so that no adversary, which is given an "address oracle" creating and returning stealth addresses $(A_j, B_j)$, can distinguish whether a second oracle, on inputs

- the number of component transactions $\ell$
- for every component transaction $i$:
  - a vector transaction outputs $\mathbf{out}_i$ together with associated values $\boldsymbol{v}_i$ and spending keys $(\boldsymbol{a}_i, \boldsymbol{b}_i)$, so that each view key $(a_{i,j}, b_{i,j}G)$ accepts the pair $(\mathsf{out}_{i,j}, v_{i,j})$
  - output values $\hat{\boldsymbol{v}}_i$ and associated addresses returned from the first oracle $(\boldsymbol{A}_i, \boldsymbol{B}_i)$
  - the supply $s_i$ and the fee $f_i$

either (1) returns the aggregation of the transactions $\mathsf{tx}_i$ created for the respective values, or (2) runs the simulator on input the list $\mathbf{out}_1 \| \ldots \| \mathbf{out}_\ell$, ordered lexicographically, the numbers $\ell$ 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 Schnorr signatures, interpreted as proofs of knowledge of secret keys, and the range proofs 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 $\mathbf{out} = (C_i, \pi_i, R_i, \rho_i, P_i)_{i=1}^{n}$, $\ell$, $\hat{n}$, $s$ and $f$, it does the following:

- pick random values $\hat{C}_1, \ldots, \hat{C}_{\hat{n}}, \hat{R}_1, \ldots, \hat{R}_{\hat{n}}, \hat{P}_1, \ldots, \hat{P}_{\hat{n}} \leftarrow_{\$} \mathbb{G}$
- 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 $X_1 := \sum \hat{\boldsymbol{R}} - \sum \boldsymbol{P} - yG - \sum_{i=2}^{\ell} X_i$ (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 signatures: for all $i \in [n]$: $\psi_i$ on message $\mathsf{out}_i$ under key $P_i$; for $i \in [\hat{n}]$: $\rho_i$ on $\hat{C}_i \| \hat{\pi}_i \| \hat{P}_i$ under $\hat{R}_i$; for $i \in [\ell]$: $\sigma_i$ on $\varepsilon$ under $E_i$ and $\chi_i$ on $E_i$ under $X_i$.

We now show that assuming indistinguishability of simulated range proofs and simulated Schnorr signatures 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.

0. In the "real" game, the component transactions are created as described (in Section 2.3) using the spending keys; they are then aggregated (Section 2.4).
1. In the first hybrid, all range proofs and Schnorr signatures (cf. Section 5) 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.
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 $D_{i,j} \leftarrow_{\$} \mathbb{G}$. This is indistinguishable by the DDH assumption, noting that $\log \hat{R}_{i,j}$ and $\log A_{i,j}$ are never used in the simulation of the game (the former because $\rho_{i,j}$ and $\chi_i$ are simulated; $A_{i,j}$ is created by the challenger but $\log A_{i,j}$ is not used anywhere)
3. The game aborts if the adversary at some point queries $(D_{i,j}, (A_{i,j}, B_{i,j}), R_{i,j})$ for some $i, j$ to the random oracle. Since the adversary has no information on $D_{i,j}$, the probability of aborting is negligible. Note that in Hybrid 3, the values $k_{i,j}$ and $q_{i,j}$ are uniformly random and independent of the $A_{i,j}$ and $R_{i,j}$.

We now argue that a transaction generated in Hybrid 3 is distributed equivalently to a transaction computed by the simulator, which concludes the proof.

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

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

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

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

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

In a transaction produced in Hybrid 3, since the coin openings $\hat{q}$ are uniformly random, 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 $((\hat{\boldsymbol{C}}_i)_{i=1}^\ell, (E_i)_{i=1}^\ell, \sum t_i)$ is uniformly random conditioned on

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

But that is exactly how the simulator produces these values. Analogously, since the values $\hat{k}$ are uniformly random, and by the definition of $X_1, \ldots, X_\ell$, the tuple $((\hat{\boldsymbol{R}}_i)_{i=1}^\ell, (X_i)_{i=1}^\ell, \sum y_i)$ is uniformly random conditioned on $\sum X_i = \sum\sum \hat{R}_{i,j} - \sum\sum P_{i,j} - (\sum y_i)G$. Again, this is how the simulator generates this tuple. Finally, in a transaction produced in Hybrid 3, all range proofs and signatures are simulated, as in outputs of the simulator.

## 5 Strong simulation-extractability of Schnorr Signatures

Key-prefixed Schnorr signatures, formally defined in Figure 2, can be reinterpreted as zero-knowledge proofs of knowledge of the secret key, that is, proofs for the NP language defined by the following relation:

$$\big\{\big((X, m), x\big) \ : \ X = xG\big\} \ .$$

To prove a statement $(X, m)$ with witness $x$, return $\pi \leftarrow \mathsf{Sch.Sign}(x, m)$; to verify a proof $\pi$ for statement $(X, m)$, run $\mathsf{Sch.Ver}(X, m, \pi)$.

We show that this proof system 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).

Strong simulation extractability for the above language means that from any adversary that returns a proof $\pi^*$ for a statement $(X^*, m^*)$, the witness $x^* = \log X^*$ can be extracted; and this is the case even if the adversary gets access to an oracle that on inputs $(X_i, m_i)$ returns simulated proofs $\pi_i$ for these statements. The only restriction is that the pair $((X^*, m^*), \pi^*)$ must be different from all query/response pairs $((X_i, m_i), \pi_i)$ (note that forging a fresh proof $\pi^*$ on a queried statement is considered a break of *strong* simulation-extractability if the extractor fails to extract a witness from $\pi^*$).

(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.)

**Claim 1** *The above proof system is strongly simulation-extractable in the algebraic group model and the random oracle model.*

In the algebraic group model [FKL18], adversaries are assumed to return *representations* of any group elements that they return. This means that, after being given input group elements $Y_1, \ldots,$ whenever the adversary returns a group element $Z$, it must also return coefficients $\upsilon_1, \ldots$ with $Z = \sum \upsilon_i Y_i$.

Note however that the group-element part of the *statement* (which the adversary need not have produced by itself) need *not* be accompanied by a representation. That is, in the proof $X^*$ does not have a representation, and moreover, representations of other elements output by the adversary can depend on $X^*$ (in addition to elements the adversary received as input).

*Proof sketch of Claim 1.* In the ROM, a Schnorr signature under any key $X$ on $m$ can be simulated without knowing a witness (secret key): choose uniform $c$ and $s$ from $\mathbb{Z}_p$, set $R := sG - cX$ and program the random oracle so that $\mathrm{H}(X, R, m) = c$ (Since $R$ is uniform and independent, the probability that the hash value has already been defined is negligible). Thus, our reduction simulates proofs for statements $(X, m)$ this way.

Consider an adversary that has made queries $(X_i, m_i)$, which where answered with $(R_i, s_i)$ with

$$R_i = s_i G - c_i X \quad \text{with} \quad c_i = \mathrm{H}(X_i, R_i, m_i)\ . \tag{24}$$

When the (algebraic) adversary returns a valid statement/proof pair $((X^*, m^*), (R^*, s^*))$, so that $(X^*, m^*, R^*, s^*) \neq (X_i, m_i, R_i, s_i)$ for all $i$, the extractor extracts the witness, that is, the discrete logarithm of $X^*$, as follows. A valid proof satisfies

$$R^* = s^* G - c^* X^* \quad \text{with} \quad c^* = \mathrm{H}(X^*, R^*, m^*)\ . \tag{25}$$

Now consider the point when $H(X^*, R^*, m^*)$ gets defined. This must be during a random-oracle query by the adversary, since a successful adversary cannot have made a simulation query for $(X^*, m^*)$ which returned $R^*$. (As there is only one valid value $s^*$, this would mean that the adversary returned the oracle's response.)

If $R^*$ was created by the adversary then it must have come with a representation $(\gamma^*, \xi^*, \boldsymbol{\rho^*})$ with

$$R^* \;=\; \gamma^* G + \xi^* X^* + \sum_{i=1}^{q_\mathrm{s}} \rho_i^* R_i \;=\; \gamma^* G + \xi^* X^* + \sum_{i=1}^{q_\mathrm{s}} \rho_i^* s_i G - \sum_{i=1}^{q_\mathrm{s}} \rho_i^* c_i X^*\ ,$$

where the equality follows from (24). If $R^*$ was defined during a simulation query, that is $R^* = R_i$ for some $i$, then set $\gamma^* := s_i$, $\xi^* := -c_i$ and $\boldsymbol{\rho}^* := \mathbf{0}$. Together with (25), this yields

$$\big(c^* + \xi^* - \textstyle\sum_{i=1}^{q_\mathrm{s}} \rho_i^* c_i\big) X^* \;=\; \big(s^* - \gamma^* - \textstyle\sum_{i=1}^{q_\mathrm{s}} \rho_i^* s_i\big) G\ .$$

Since $c^*$ was chosen at random *after* the adversary (or the experiment) chose $\gamma^*, \xi^*, \rho_1^*, \ldots$ and $\rho_{q_\mathrm{s}}^*$, the probability that $c^* + \xi^* - \sum_{i=1}^{q_\mathrm{s}} \rho_i^* c_i \not\equiv_p 0$ is overwhelming, in which case the extractor can efficiently compute the discrete logarithm of $X^*$ from the above equation.

## References

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

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. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

FOS19. Georg Fuchsbauer, Michele Orrù, and Yannick Seurin. Aggregate cash systems: A cryptographic investigation of Mimblewimble. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 657–689. Springer, Heidelberg, May 2019.

FPS20. Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind schnorr signatures and signed ElGamal encryption in the algebraic group model. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 63–95. Springer, Heidelberg, May 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.

Seu12. Yannick Seurin. On the exact security of Schnorr-type signatures in the random oracle model. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 554–571. Springer, Heidelberg, April 2012.

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

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

Wag02. David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, Heidelberg, August 2002.

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