Nopenena Untraceable Payments: Defeating Graph Analysis with Small Decoy Sets

Jayamine Alupotha
University of Bern

Mathieu Gestin
Inria - IRISA - CNRS - Université de Rennes

Christian Cachin
University of Bern

ABSTRACT

Decentralized payments have evolved from using pseudonymous identifiers to much more elaborate mechanisms to ensure privacy. They can shield the amounts in payments and achieve untraceability, e.g., decoy-based untraceable payments use decoys to obfuscate the actual asset sender or asset receiver. There are two types of decoy-based payments: *full decoy set payments* that use all other available users as decoys, e.g., Zerocoin, Zerocash, and ZCash, and *user-defined decoy set payments* where the users select small decoy sets from available users, e.g., Monero, Zether, and QuisQuis.

Existing decoy-based payments face at least two of the following problems: (1) *degrading untraceability* due to the possibility of payment-graph analysis in user-defined decoy payments, (2) *trusted setup*, (3) *availability issues* due to *expiring transactions* in full decoy sets and epochs, and (4) an *ever-growing set of unspent outputs* since transactions keep generating outputs without saying which ones are spent. QuisQuis is the first one to solve all these problems; however, QuisQuis requires large cryptographic proofs for validity.

We introduce *Nopenena* (means "cannot see"): account-based, confidential, and user-defined decoy set payment protocol, that has short proofs and also avoids these four issues. Additionally, Nopenena can be integrated with *zero-knowledge contracts* like Zether's $\Sigma-$Bullets and Confidential Integer Processing (CIP) to build decentralized applications. Nopenena payments are about 80% smaller than QuisQuis payments due to Nopenena's novel cryptographic protocol. Therefore, decentralized systems benefit from Nopenena's untraceability and efficiency.

1 INTRODUCTION

Decentralized direct payments like Bitcoin [44] or Ethereum [58] use pseudonymous identifiers like public keys to state the ownership. Still, they reveal potentially harmful information about users due to the readable monetary values and traces to previous owners [3, 20, 28, 43, 47, 53]. The simplest example is *insider tracing*, i.e., a user with known pseudonymous identifiers (possibly because the user received assets from them) can trace these identifiers' payments to see how assets were transferred. Therefore, many decentralized payments are equipped with:

(1) *untraceability* - hiding or obfuscating asset senders' and/or asset receivers' identifiers from blockchain validators,
(2) *special sender anonymity* - hiding or obfuscating senders' identifiers from the *receivers* of the same payment, and
(3) *confidentiality* - hiding transferred amounts from validators.

Decoy-based Untraceable Payments A decoy of a payment is an existing asset in the payment system, but its owner does not actively participate in creating the payment. Decoy-based untraceable payments use decoys to obfuscate the real senders and real receivers. Decoy-based payment systems can be categorized according to their asset type: (1) *unspent transaction outputs* or (2) *accounts*. Unspent output-based payments only reveal that n out of $N(> n)$ outputs actually sent their coins to a new unspent output(s) without revealing which n outputs, e.g., Monero [45, 46], Zerocoin [42], and ZCash [29]. Account-based payments, like Zether [9] and QuisQuis [19], are different from output-based payments since they reveal that n accounts out of N accounts exchanged coins, without revealing which n accounts. Fascinatingly, these payments' cryptographic protocols do not need $(N - n)$ decoy accounts' or decoy outputs' owners to actively participate in creating the cryptographic proofs, e.g., decoys' secret keys are not required.

Advantages of Account-based Untraceable Payments For security, output-based payments must ensure (1) *theft-resistance*, i.e., actual sending outputs' owners agreed to the payment, (2) *balance proofs*, i.e., the sending coin amount is equal to the receiving coin amount, (3) *non-negative coin amounts* in outputs, and (4) *no-double spending*, i.e., outputs were only spent once. An advantage of account-based payments is that they only ensure theft-resistance, balance proofs, and non-negative account balances since preventing double-spending is not applicable for accounts.

Account-based untraceable payments moreover solve the ever-growing output set problem in output-based payments. For example, Monero's and ZCash's payments add unspent outputs to the ledger but do not remove any outputs since they do not reveal which outputs are actually spent. Hence, the output set keeps growing monotonically with the number of transactions, and users and validators must store this large output set to create and verify payments. However, users and validators of account-based payments only need to store the most recent account state set to create and verify payments, which does not grow linearly with transactions.

Another advantage of account-based payments is that they are more compatible with smart contracts than output-based payments due to the long-term ownership of accounts. For example, account-based payments can be integrated with zero-knowledge contracts that hide balances or other contract variables but prove that actual sending and receiving accounts satisfy the contract conditions.

Due to these advantages, decentralized systems benefit from account-based untraceable payments greatly, and it is important to explore novel cryptographic protocols to improve their efficiency.

Types of Decoy-based Payments Decoy-based payments can also be categorized according to how they select the decoys. *Full decoy set payments*, like Zerocoin and ZCash, use all available outputs/accounts as decoys; thus, $(N-n)$, i.e., decoys of a payment, is at its maximum. *User-defined decoy set* payments allow users to select a small set of $(N - n)$ decoys from the available outputs/accounts. While full decoy set payments benefit from the maximal untraceability, they suffer from expiring transactions since adding transactions

Protocol	Untraceable	Confidential	Expiring Probability	No Trusted Setup	DoS Attack Resistance	Graph Analysis Resistance	Non-monotonic Set of Assets	Contract Support
Zerocoin [42]	Maximal	○	High	○	○	●	●	○
ZCash [29]	Maximal	●	High	○	○	●	○	○
Lelantus [31]	Maximal	●	High	●	○	●	○	○
Mimblewimble [30]	No	●	Zero	●	-	-	●	○
Monero [46], [34, 60]	Degrading	●	Zero	●	●	○	○	○
Ring CT v.2 [54]	Degrading	●	Zero	○	●	○	○	○
Zether [9, 15]	Degrading (epoch)	●	High	●	○	◑	●	●
QuisQuis [19]	Non-degrading	●	Low	●	○	●	●	○
PriDe CT [26]	Degrading (epoch)	●	High	●	○	◑	●	●
PriFHEte [39]	Maximal	●	High	●	○	●	●	○
Nopenena (this paper)	Non-degrading	●	Low	●	●	●	●	●

Table 1: A Comparison of Related Work. Here, expiring probability means the probability of a transaction expiring due to epochs or updated assets. We use ◑ to denote DM-decomposition limited to epochs.

changes the asset set and expires the awaiting transactions, e.g., ZCash transactions expire only after 50 mins. Also, current full decoy set payments either require ceremonial-type trusted setup (ZCash) or large cryptographic proofs due to the large decoy set (Zerocoin). On the other hand, user-defined decoy payments mitigate (Monero and Zether) or prevent (QuisQuis) tracing and achieve higher availability, e.g., Monero and QuisQuis.

Degrading Untraceability Ring Confidential Transactions (Ring CTs) like Monero [45, 54, 60] and Zether [9, 15] mitigate tracing real senders and receivers but do not prevent tracing [13, 17]. The reason is that these systems are vulnerable to graph analysis: the *Dulmage-Mendelsohn (DM) decomposition* [16] may deanonymize spent outputs or sending and receiving accounts by solving maximal matching problems on the payment graph. Ring CTs are vulnerable to DM decomposition since their unspent outputs can be only spent *once*, i.e., no-double spending property (see [13, 17] for more details). Zether is also vulnerable to DM decomposition since an account can be only used *once* during an epoch (see Appendix F).

Ring CTs and Zether have *degrading untraceability* since this probability of deanonymization increases per payment. [13, 17] show that deanonymizing probability is negligible if the decoy set is large enough; about $N = 128$ unspent outputs would be required for two spending outputs ($n = 2$). Thus, Monero's N being 16 [46] is not sufficient to achieve untraceability in the payment graph.

QuisQuis[19] The first untraceable payment protocol to solve degrading untraceability with user-defined decoys is QuisQuis, i.e., QuisQuis payments obtain non-degrading untraceability even when N=16 and any n<N. However, QuisQuis does not support contracts.

Also, QuisQuis payments shuffle all accounts (uses a verifiable shuffle algorithm), including decoys, so that updated account states are unlinkable to their previous states. This shuffling creates a searching problem for decoys and receivers since the only way to find the recent account state is to try the secret key and the balance for all possible accounts, which is not ideal for a payment system. Nevertheless, this cryptographic shuffling is essential in QuisQuis to obtain a proof system for non-negative balances with untraceability.

Moreover, QuisQuis transactions are impractically large, i.e., 24KB for 16 accounts, due to this verifiable shuffling and its update proofs that prove the soundness of updated account states.

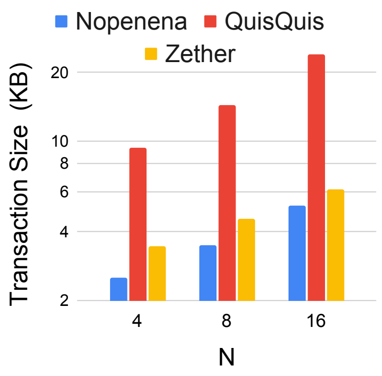


Figure 1: Sizes of untraceable account-based payments for $n = 2$.

Our research question is "can we build a more efficient account-based payment system with non-degrading untraceability?".

Our Contribution

We introduce *Nopenena*[1], *account-based confidential payments* with a novel cryptographic protocol. Nopenena provides the followings:

(1) *Untraceability* of senders/receivers with sender anonymity,
(2) *Non-monotonic set of accounts*,
(3) *Nondegrading untraceability* even with small decoy sets,
(4) *No trusted setup*, and
(5) *High availability* with DoS-resistance and no epochs.

Moreover, payments in Nopenena are smaller than Anonymous Zether [15] and QuisQuis [19], e.g., Nopenena payments are ~ 80% smaller than QuisQuis's, as shown in Figure 1.

Overview Nopenena payments work as follows. First, users register their accounts in the ledger with a zero balance. Each account has a public key pk, a secret key k and a confidential asset $asset(pk, r, v)$ that hides the coin balance v with a blinding key r. These assets are **verifiably rerandomizable**, meaning that *anyone* can update the asset's blinding key into $asset(pk, r + r', v)$ with some r' and can create an update proof σ for the correctness of the update *without knowing* (k, r). The account's balance is also modifiable if k is known, i.e., the asset can be rerandomized into $asset(pk, r + r'', v \pm v')$ with some r'' and create an update proof σ' if they know k; still, r is not needed. This rerandomization provides:

[1] *Nopenena* means "cannot see" in a secret language.

(1) **theft-resistance**: Only owners who know account secret keys can modify balances and create update proofs.
(2) **indistinguishability**: Given these accounts, updated assets, and proofs: $(pk_0, asset(pk_0, r_0, v_0), asset(pk_0, r_0{+}r', v_0), \sigma_0)$ and $(pk_1, asset(pk_1, r_1, v_1), asset(pk_1, r_1+r', v_1 \pm v'), \sigma'_1)$, no one can distinguish which account updated the balance if r' is chosen at random.

To create a Nopenena payment, the sender selects N accounts: $[pk_i, asset_i]_{i=0}^{N}$ such that $n < N$ accounts with indexes $[j_l]_{l=0}^{n}$ are the sending and receiving accounts. Here, each j_l is in $[0, N)$. First, the sender chooses a random r', rerandomizes decoy accounts with r', and creates their update proofs. Then, the sender rerandomizes the sending and receiving accounts with new balances using r'. After that, the sender creates update proofs for sending accounts since the sender knows their secret keys. However, the sender does not know the secret keys of the receiving accounts to create update proofs and cannot share r' with the receivers since the receivers can identify sending accounts from decoys using r'. Therefore, we provide a *multi-party update proof protocol* where the sender and receiver create an update proof for a receiving account. However, the sender does not share r', and the receiver does not share the secret key. Still, these multi-party update proofs are indistinguishable from normal update proofs to hide that they are from receiving accounts. Finally, the sender has rerandomized all accounts, and the receiver does not learn which accounts belong to the sender.

Let $[asset'_j, \sigma_i]_{i=0}^{N}$ be the rerandomized assets and update proofs. We provide a special balance proof protocol for:

(3) **balance proof**: A balance proof $\pi_{balance}$ can be only created if the total coins $\sum_{i=0}^{N} v_i$ in $[asset_i]_{i=0}^{N}$ and $\sum_{i=0}^{N} v'_i$ in $[asset'_i]_{i=0}^{N}$ are $\sum_{i=0}^{N} v_i + f = \sum_{i=0}^{N} v'_i + f'$ for a transaction reward f and transaction fee f'.

We still need one more property, i.e., non-negative account balances. Typically, we could use zero-knowledge range proofs [10] to prove that hidden account balances are non-negative without revealing the account balance. However, these range proofs need the account's secret key, and the sender cannot ask decoys to create range proofs for decoy accounts. As a solution, we propose a novel cryptographic primitive called **anonymous forced openings**.

The forced opening protocol works as follows. First, the sender and receivers create Pedersen commitments $[C_l{=}commit(\alpha_l, v'_{j_l})]_{l=0}^{n}$ with random blinding keys $[\alpha_l]_{l=0}^{n}$ to hide new account balances $[v'_{j_l}]_{l=0}^{n}$. Then, the sender and receivers run a multi-party forced opening protocol and create a proof π_{forced} to prove that $[C_l]_{l=0}^{n}$ commit all updated account balances, i.e., all $v'_i \neq v_i$ for i in $[0, N)$. The protocol is (1) "anonymous" since the proof does not reveal which accounts have changed the balances, and (2) "forced openings" since all accounts that modified the balance must include a commitment in $[C_l]_{l=0}^{n}$. Then, the sender and receivers create range proofs for these commitments $[C_l]_{l=0}^{n}$ to prove that all updated balances are non-negative. Due to our anonymous forced opening protocol, malicious provers cannot hide negative account balances, and honest provers obtain untraceability without asking decoys to create range proofs. Therefore, we obtain:

(4) **non-negative account balances**: Any new account balances after the rerandomization must be non-negative.

Due to the rerandomized accounts and anonymous forced openings, our payments achieve the followings:

(5) **untraceability**: Payments do not reveal sending accounts and receiving accounts to blockchain validators, and
(6) **sender anonymity**: Receivers do not learn which accounts belong to the sender; hence, insider-tracing can be prevented.

We compare Nopenena and related work in Table 1. More details of related work are given in Section 10.

2 MODEL

This section presents the model. First, it is assumed that a trusted append-only, secure, and totally ordered ledger Λ exists, and it maintains two lists: `Accounts`$\in\Lambda$ and `Withheld`$\in\Lambda$ for accounts and contracts, respectively. The ledger supports two operations: (1) Append for transactions and (2) Read to read the current ledger.

Also, Nopenena provides two functions: CreateTx and VerifyTx, which are described in the rest of the paper. $\mathsf{CreateTx}(\Lambda, \cdot) : tx$ is a multi-party protocol of a (coin-)sender and (coin-)receivers [2] that creates *a valid transaction tx* for Λ such that $\mathsf{VerifyTx}(\Lambda, \mathsf{tx}) = 1$. Here, the sender acts as the leader or the combiner of the protocol. We assume that they can communicate and create transactions or proofs together through a point-to-point, authenticated, and tamper-resistant link. However, it is important to note that *neither the sender nor the receiver is assumed to be honest.* Once a transaction is created, the sender submits it to the ledger. We assume that the ledger's operations align with Nopenena's functions as follows:

(1) *Security*: The ledger only appends valid transactions, and Read operation only outputs a ledger of valid transactions.
(2) *Liveness*: Append operation always appends valid transactions, and Read always outputs all appended transactions.
(3) *Append-only*: No transaction can be removed from the ledger.
(4) *Total-order*: Read operations are consistent, i.e., a reader obtains a prefix of the ledger that all other readers also obtain.

The way transactions are appended and ordered in the ledger depends on its implementation and is out of the scope of this paper.

3 PRELIMINARIES

Notation. We use ":=" and "=:" for assignments, e.g., $a =: b$ means that b was assigned from a. $A \setminus B$ is the set minus of A by B. ϕ is an empty set. $\Leftrightarrow$ indicates double implication. For a cyclic group $\mathbb{G}$, we reserve (g, h, μ) to denote Nothing-Up-My-Sleeve (NUMS) generators of $\mathbb{G}$ (Definition 3.1). $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ is a ring of modular integers in $[0, q-1]$ for modulus q. We use a_i to denote ith element of an array. $s \overset{\$}{\leftarrow} \mathcal{S}$ denotes that s is drawn uniformly at random from a set $\mathcal{S}$. We use λ for the security level and $\mathcal{A}$ for a probabilistic polynomial-time (p.p.t.) adversary of λ. We use $\mathcal{A}(s)$ to imply that $\mathcal{A}$ is given s. Also, $\epsilon(\lambda){=}1/o(\lambda^c)$ is a negligible function $\forall c \in \mathbb{N}$. Sometimes, we casually use "negligble" to denote $\epsilon(\lambda)$. $\mathsf{Hash}{:}\{0,1\}^* {\to} \mathbb{Z}_q$ denotes a collision-resistant hash function family. We use "non-interactive" to mean that there are no interactions between the prover(s) and the verifiers; however, they could be multi-party protocols of provers.

[2] These coin senders and coin receivers should not be confused with message senders or receivers in broadcasts since coin-receivers can also be message senders in payment systems. We casually use "sender" and "receiver" to denote coin senders and coin receivers, respectively.

Definition 3.1 (Nothing-Up-My-Sleeve (NUMS) Generators). A NUMS generator is generator of $\mathbb{G}$ chosen uniformly at random.

Definition 3.2. [Discrete Logarithmic (DL) Problem] The advantage $\mathsf{Adv}^{DL}_{\mathbb{G}}$ of $\mathcal{A}$ is $Pr[Y \overset{?}{=} g^x \mid Y \overset{\$}{\leftarrow} \mathbb{G}, x \overset{\$}{\leftarrow} \mathcal{A}(g; Y)]$ for any NUMS generator g. The DL problem is (τ, e)-hard if $\mathcal{A}(\tau, e)$ runs it at most τ times and $\mathsf{Adv}^{DL}_{\mathbb{G},\mathcal{A}} \leq \epsilon(\lambda)$.

Definition 3.3. [Decisional Diffie-Hellman (DDH) Problem] $\mathcal{A}$'s advantage $\mathsf{Adv}^{DDH}_{\mathbb{G}}$ is $Pr[b \overset{?}{=} b' \mid (x, y, c) \overset{\$}{\leftarrow} \mathbb{Z}_q, Y_0 := g^c, Y_1 := g^{xy}, b \overset{\$}{\leftarrow} [0, 1], b' \overset{\$}{\leftarrow} \mathcal{A}(g; g^x, g^y, Y_b)]$ for any NUMS g. The DDH problem is (τ, e)-hard if $\mathcal{A}(\tau, e)$ runs it at most τ times and $\mathsf{Adv}^{DDH}_{\mathbb{G},\mathcal{A}} \leq 1/2 + \epsilon(\lambda)$.

3.1 Pedersen Commitments

We use $\mathsf{Commit}_{h,\mu}(\alpha, v) = h^\alpha \mu^v \in \mathbb{G}$ to denote a Pedersen commitment of blinding key $\alpha \overset{\$}{\leftarrow} \mathbb{Z}_q$ and value $v \in \mathbb{V} \subseteq \mathbb{Z}_q$.

Theorem 3.4. *Pedersen commitments provide the followings:*

- *(Hiding) Upon receiving $(v_0, v_1) \in \mathbb{V}$ from $\mathcal{A}$, the challenger shares $C = \mathsf{Commit}_{h,\mu}(\alpha \overset{\$}{\leftarrow} \mathbb{Z}_q, v_b)$ from a random choice of $b \overset{\$}{\leftarrow} [0, 1]$. The probability of $\mathcal{A}$ finding b is $\leq 1/2 + \epsilon(\lambda)$.*
- *(Computational binding) The probability of $\mathcal{A}$ finding two different, $(\alpha_0, v_0), (\alpha_1, v_1) \in (\mathbb{Z}_q, \mathbb{V})$ such that $\mathsf{Commit}_{h,\mu}(\alpha_0, v_0) = \mathsf{Commit}_{h,\mu}(\alpha_1, v_1)$ and $(\alpha_0, v_0) \neq (\alpha_1, v_1)$ is negligible.*

3.2 Zero-Knowledge Argument (ZKA)

Let there be a polynomial time decidable relation $\mathcal{R}$ and three p.p.t. entities; a Common Reference String (CRS) generator Set, a prover $\mathcal{P}$, and verifier $\mathcal{V}$. Here, $\mathcal{P}$ wants to prove some relation $\mathcal{R}$ of a witness w for a statement u without revealing anything else about w. We denote relation $\mathcal{R}$ for the generated CRS pp as $(pp, u, w) \in \mathcal{R}$. We call all statements that have witness(s) for pp a CRS-dependent language $\mathcal{L}_{pp} = \{x \mid \exists w : (pp, x, w) \in \mathcal{R}\}$. A proving algorithm may take multiple interactions between $\mathcal{P}$ and $\mathcal{V}$. We denote an interaction's transcript, $tr \leftarrow \langle \mathcal{P}(pp, u, w), \mathcal{V}(pp, u) \rangle$. If $\mathcal{V}$ accepts tr, $\mathcal{V}(tr) = 1$. We define ZKA similar to [10, 23, 38] except that we define an adversary $\mathcal{A}$ with some insider-knowledge ζ such that $(w \setminus \zeta) \neq \phi$ since some of our protocols are multi-party protocols of senders and receivers, and $\mathcal{A}$ may control all or some receivers.

Definition 3.5. (Zero-Knowledge Argument) $(\text{Set}, \mathcal{P}, \mathcal{V})$ is a ZKA if they satisfy the following properties:

(**Completeness**) Let $\mathcal{A}^{\mathcal{G}}$ be a p.p.t. adversary who generates the witness w and statement u. $(\text{Set}, \mathcal{P}, \mathcal{V})$ is complete if:

$$Pr\left[(pp, u, w) \overset{?}{\notin} \mathcal{R} \vee \mathcal{V}(\langle \mathcal{P}(pp, u, w), \mathcal{V}(pp, u) \rangle) \overset{?}{=} 1 \mid (u, w) \leftarrow \mathcal{A}^{\mathcal{G}}(pp)\right] = 1$$

(**Computational Witness-Extended Emulation (WEE)**) Let there be two p.p.t. adversaries: WEE adversary $\mathcal{A}$ with insider-knowledge ζ and $\mathcal{A}^{\mathcal{G}}$ who generates the initial witness s and statement u. Also, $\mathcal{P}^*$ is a deterministic p.p.t. prover, and $\mathcal{E}^{\mathcal{W}}$ is a p.p.t. emulator that generates an emulated transcript $(tr, w) \leftarrow \mathcal{E}^{\mathcal{W}}(u)$ with a witness w such that $\mathcal{V}(tr) = 1$. $(\text{Set}, \mathcal{P}, \mathcal{V})$ has WEE if:

$$\left| Pr\left[\mathcal{A}(tr; \zeta) \overset{?}{=} 1 \middle| \begin{matrix} pp := \mathtt{Set}(\lambda), (u, s) \leftarrow A^{\mathcal{G}}(pp) \\ tr \leftarrow \langle \mathcal{P}^*(pp, u, s), \mathcal{V}(pp, u) \rangle \end{matrix}\right] - Pr\left[\begin{matrix} \mathcal{A}(tr; \zeta) \overset{?}{=} 1 \wedge \\ (\mathcal{V}(tr) \overset{?}{\Rightarrow} \\ (pp, u, w) \in \mathcal{R}) \end{matrix} \middle| \begin{matrix} pp := \mathtt{Set}(\lambda), (u, s) \leftarrow A^{\mathcal{G}}(pp) \\ (tr, w) \leftarrow \mathcal{E}^{\mathcal{W}(\langle \mathcal{P}^*(pp,u,s), \mathcal{V}(pp,u) \rangle)}(pp, u) \end{matrix}\right] \right| \leq \epsilon(\lambda)$$

If $\mathcal{A}$ cannot identify emulated transcripts over genuine transcripts, i.e., $\mathcal{A}$ accepts emulated transcripts $\mathcal{A}(tr; \zeta) = 1$ even with knowledge ζ, it implies that $\mathcal{A}$ learns nothing about $(w \setminus \zeta)$, except $\mathcal{R}$.

(**Knowledge Soundness**) Let $\mathcal{P}^*$ be a rewindable prover and $\mathcal{W}$ be a p.p.t. extractor which extracts witnesses by rewinding $\mathcal{P}^*$ to a certain iteration of witness s and resuming with fresh verifier randomness, i.e., $w \leftarrow \mathcal{W}(\langle \mathcal{P}^*(pp, u, s), \mathcal{V}(pp, u) \rangle)$. The previous negligible function of WEE also provides knowledge soundness if $\mathcal{P}^*$ cannot generate valid transcripts that do not align with the relation, i.e., $(\mathcal{V}(tr) \overset{?}{=} 1 \wedge (pp, u, w) \overset{?}{\notin} \mathcal{R})$.

Definition 3.6 (Special Honest Verifier Zero-Knowledge Argument). Let $\mathcal{V}$'s challenges be chosen from a public coin randomness ρ (created from tossing an unbiased public coin) which is independent of provers' messages. $(\text{Set}, \mathcal{P}, \mathcal{V})$ is a special honest verifier ZKA if it is ZKA in Definition 3.5 for a special honest verifier $\mathcal{V}(pp, u; \rho)$.

Non-Interactive Zero-Knowledge Range Proofs A range proof shows that the hidden value of a commitment is in some specific range without revealing the value. In this paper, we are interested in ranges like $[0, 2^L)$ for $L \in \mathbb{N}$ since coin values should be non-negative. Let $(\mathsf{RangeProve}, \mathsf{RangeVerify})$ be a range proof scheme such that the $\mathsf{RangeProve}(C, v, \alpha, L)$ creates a range proof π and $\mathsf{RangeVerify}(C, \pi, L)$ outputs 1 if $v \in [0, 2^L)$, otherwise 0.

Definition 3.7. Range proofs are zero-knowledge if they are ZKA for verifier $\mathcal{V}(pp, u) = \mathsf{RangeVerify}(C, \pi, L)$, $\zeta = \phi$, and a relation:

$$(pp, u = (C, \pi), (\alpha, v)) \in \mathcal{R}_{range} \Leftrightarrow \begin{pmatrix} (h, \mu, L) \overset{?}{\in} pp \wedge v \overset{?}{\in} [0, 2^L) \wedge C \overset{?}{=} \mathsf{Commit}_{h,\mu}(\alpha, v) \\ \wedge \pi \overset{?}{=} \mathsf{RangeProve}(C, v, \alpha, L) \end{pmatrix}$$

One-of-Many Proofs for Zero-Value Commitments A zero-value commitment is a commitment to value $0 \in \mathbb{Z}_q$. Assume that $(\mathsf{ZeroComProve}, \mathsf{ZeroComVerify})$ is a one-of-many proof protocol for zero-value commitments. It proves that the jth commitment is a zero-value commitment without revealing j. Formally, $\mathsf{ZeroComProve}([C]_{i=0}^N, j, \alpha)$ creates a proof π which will result 1 from $\mathsf{ZeroComVerify}([C]_{i=0}^N, \pi)$ if $0 \leq j < N$ and $C_j = \mathsf{Commit}_{h,\mu}(\alpha, 0)$. Otherwise, it outputs 0.

Definition 3.8. One-of-many proofs for zero-value commitments provide zero-knowledge if they are ZKA for verifier $\mathcal{V}(pp, u) = \mathsf{ZeroComVerify}([C]_{i=0}^N, \pi)$, $\zeta = \phi$, and relation $\mathcal{R}_{zero}$:

$$(pp, u = ([C_i]_{i=0}^N, \pi, N), (\alpha, j)) \in \mathcal{R}_{zero} \Leftrightarrow \begin{pmatrix} (h, \mu) \overset{?}{\in} pp \wedge 0 \overset{?}{\leq} j \overset{?}{<} N \wedge C_j \overset{?}{=} \mathsf{Commit}_{h,\mu}(\alpha, 0) \\ \wedge \pi \overset{?}{=} \mathsf{ZeroComProve}([C]_{i=0}^N, j, \alpha) \end{pmatrix}$$

Zero-Knowledge Contracts of Pedersen Commitments We define a generic zero-knowledge contract scheme that takes Pedersen commitments as variables. We define a contract protocol that takes inputs from some set $\mathbb{V}$ as follows:

- $\mathsf{Compile}(\mathcal{F}) : (\mathcal{F}_{zk}; blinds) \triangleright$ compiles a function $\mathcal{F} : \mathbb{V}^* \to 1/0$ into a zero-knowledge function $\mathcal{F}_{zk}$ and outputs the secret information $blinds$ required to prove its validity. This $\mathcal{F}_{zk}$ is basically a customized zero-knowledge verification function for $\mathcal{F}$ but hides $\mathcal{F}$'s variables using $blinds$.
- $\mathsf{ContractProve}(\mathcal{F}, \mathcal{F}_{zk}, [C_l, v_l, \alpha_l]^*_{l=0}, blinds) : \pi_{contract} \triangleright$ If $\mathcal{F}([v_l]^*_{l=0})$=1, it generates a zero-knowledge proof π when $[v_l]^*_{l=0} \in \mathbb{V}^*$ is committed in $[C_l]^*_{l=0}$. If $\mathcal{F}([v_l]^*_{l=0}) = 0$, returns error $\perp$. This proving could be a multi-party protocol.
- $\mathcal{F}_{zk}([C_l]^*_{l=0}, \pi_{contract}) : 1/0 \triangleright$ verifies the contract.

Definition 3.9. A contract $\mathcal{F}_{zk}$ provides zero-knowledge if it is ZKA for $\mathcal{V}(pp, u) = \mathcal{F}_{zk}([C_l]^*_{l=0}, \pi)$, ζ=ϕ, and relation:

$$\big(pp, u{=}(\mathcal{F}_{zk}, [C_i]^*_{i=0}, \pi), (\mathcal{F}, [v_l, \alpha_l]^*_{l=0}, blinds)\big) \in \mathcal{R}_{contract} \Leftrightarrow$$

$$\left(\begin{array}{r} (h,\mu)\overset{?}{\in}pp \wedge \mathcal{F}([v_l]^*_{l=0})\overset{?}{=}1\wedge \\ \mathsf{Compile}(\mathcal{F})\overset{?}{=}(\mathcal{F}_{zk}; blinds) \wedge [C_l\overset{?}{=}\mathsf{Commit}_{h,\mu}(\alpha_l, v_l)]^*_{l=0} \end{array}\right)$$

4 NOPENENA OVERVIEW

In this section, we define Nopenena functions for a ledger Λ : (Accounts, Withheld). We assume that the ledger Λ holds a set of accounts Accounts $\in \Lambda$. Moreover, we assume that accounts can release coins into a Pedersen commitment with an attached zero-knowledge contract. If a transaction contains a valid proof for the attached contract, the ledger allows to get these coins from the commitment to the accounts. We use Withheld $\in \Lambda$ to denote a set of these contract-commitment pairs that have not released their coins. This untraceable payment system provides the following functionalities:

(1) $\underline{Setup(\lambda) : pp}$ ▶ creates public parameters. We assume that pp is public and consistent such that all users can access pp.
(2) $\underline{\mathsf{CreateTx}(\Lambda, [acc_i]^N_{i=0}, [j_l, v_l, v'_l, \alpha_l, C_l]^n_{l=0}, [\mathcal{F}_{zk}, C, \pi_{zk}, \alpha_c, c],}$ $\underline{[\mathcal{F}'_{zk}, C', \alpha_{c'}, c'], f, f'; [k_l]^n_{l=0}) : tx{:=}([asset'_i]^N_{i=0}, \pi)}$ ▶ This protocol is conducted by senders and receivers to create a transaction. The transaction contains rerandomized assets $[asset'_i]^N_{i=0}$ of accounts $[acc_i]^N_{i=0}$ when only n accounts in indexes $[j_l]^n_{l=0} \in [0, N)^n$ are the actual sending/receiving accounts. Other accounts are decoys, and decoys do not participate in creating the transaction. The current account balances are $[v_l]^n_{l=0}$, and new balances are $[v'_l]^n_{l=0}$ in sending/receiving accounts. Here, $[C_l]^n_{l=0}$ are the Pedersen commitments of updated balances such that $[C_l = \mathsf{Commit}_{h,\mu}(\alpha_l, v'_l)]^n_{l=0}$.
$[\mathcal{F}_{zk}, C] \in$ Withheld is the contract-commitment pair that will be proven and released by the transaction when π_{zk} is the contract proof such that $\mathcal{F}_{zk}(C \| C' \| [C_l]^*_{l=0}, \pi_{zk}) = 1$. Also, c is the coin value stored in C, and α_c is the blinding key of C. $[\mathcal{F}'_{zk}, C']$ is the new pair that will be added to Withheld when C' is the commitment of $(\alpha_{c'}, c')$.
Also, the coin reward for the transaction is f, and the transaction fee is f', such that $f + c + \sum^n_{l=0} v_l = f' + c' + \sum^n_{l=0} v'_l$.
In this multi-party protocol, the participants, i.e., sender and receivers, do not share their secret keys $[k_l]^n_{l=0}$ with others.
(3) $\underline{\mathsf{VerifyTx}(\Lambda, tx)\ :1/0}$ ▶ This function verifies the transaction.
(4) $\underline{\mathsf{OpenAsset}(k, v, acc)\ :1/0}$ ▶ verifies whether the secret key and account balance are (k, v) or not.

When the ledger appends tx, it removes current assets of $[acc_i]^N_{i=0}$ from Accounts and adds $[asset'_i]^N_{i=0}$ to Accounts. Also, the ledger removes $[\mathcal{F}_{zk}, C]$ from Withheld and adds $[\mathcal{F}'_{zk}, C']$ to Withheld.

5 NOPENENA PAYMENTS' BUILDING BLOCKS

This section explains the novel protocols used in Nopenena: rerandomizable accounts, anonymous forced openings, and balance-proofs.

5.1 Rerandomizable Accounts

We define a rerandomizable account scheme below. Each account of secret key k, blinding key r, and balance v is:

$acc : (\mathsf{pk}, \mathsf{asset}) := (K = g^k, G = g^r, V = g^{kr}\mu^v)$ and has a fixed $\mathsf{pk} = K$ and an updatable $\mathsf{asset} = (G, V)$. When the account is created the asset is empty. i.e., $G = 1_{\mathbb{G}}$ and $V = 1_{\mathbb{G}}$.

```
1: CreateAccounts(k):                    ▷ create a new account of $\mathbb{G}^3$
2: return $acc$=(pk:=($K$=$g^k$), asset=($G$:=$1_{\mathbb{G}}$, $V$:=$1_{\mathbb{G}}$))
```

Anyone can update an asset without the current blinding key.

```
1: UpdateAsset($r'$, $optional = (v, v')$, $acc = (K, G, V)$):
2: If $v$ and $v'$ are not given:
3:     return $(G' = Gg^{r'}, V' = VK^{r'}) \in \mathbb{G}^2$,
4: Otherwise, return $(G' = Gg^{r'}, V' = VK^{r'}\mu^{v'-v}) \in \mathbb{G}^2$
```

Anyone who knows k can verify the account balance as follows.

```
1: OpenAsset($k, v, (G, V)$): return $\mu^v \overset{?}{=} V(G^{-k}) \in \mathbb{G}$
```

First, we explain how to convince a verifier $\mathcal{V}$ that an asset rerandomization was correctly performed if the balance was updated. In addition to the secret key k and r', this function takes a challenge message W and a random key κ. However, note that update proofs do not require the current blinding key.

```
1: UpdateValueProve($r', k, v, v', acc, \mathsf{asset}'; \kappa, W$):
2: $(K, G, V) := acc$ and $(G', V') := \mathsf{asset}'$
3: $T_1 := g^t$, $T_2 := K^t\mu^\tau$, and $T_3 := K^\tau g^\kappa \in \mathbb{G}$ for $(t, \tau) \xleftarrow{\$} \mathbb{Z}_q$
4:     $T_1, T_2, T_3 \to \mathcal{V}$                       ▷ send to the verifier
5:     $x \in \mathbb{Z}_q \leftarrow \mathcal{V}$                      ▷ receive from the verifier
6:     ▷ $x$:=Hash($W, T_1, T_2, T_3, acc, \mathsf{asset}'$) in non-interactive setup
7: $s_1 := t + x(r') \in \mathbb{Z}_q$ and $s_2 := \tau + x(v' - v) \in \mathbb{Z}_q$
8: $s_3 := -\kappa + x(v' - v)k \in \mathbb{Z}_q$     ▷ needs the key $k$ if $(v' - v) \neq 0$
9: return $\sigma := (x, s_1, s_2, s_3)$
```

We do not want k in **UpdateValueProve** when $(v'-v)$=0 since $x(v'-v)k$=0 in Step 8. Hence, we use the following for decoys:

UpdateProve$(r', acc, \mathsf{asset}'; \kappa, w)$:
return **UpdateValueProve**$(r', -, 0, 0, acc, \mathsf{asset}'; \kappa, w)$

In payments, we want to update all accounts with the same r', chosen by the sender. Hence, the sender can update all sending accounts and decoys accounts with r' by himself or herself. However, the sender cannot share r' with the receivers to update the receiving accounts since the receivers can identify sending accounts from decoys if r' is known. Also, receivers cannot share the account secret key with the sender. Hence, we turn **UpdateValueProve** into a multi-party protocol where the sender does not reveal r', and the receiver does not reveal the secret key k and κ, as follows:

```
1: $\textbf{UpdateValueMProve}(r', k, v, v', acc, \texttt{asset}'; \kappa, W)$:
2: $(K, G, V) := acc$ and $(G', V') := \texttt{asset}'$
3: The sender computes: $t \xleftarrow{\$} \mathbb{Z}_q$, $T_1 := g^t$, and $T := K^t \in \mathbb{G}$
4: The receiver gets $T$ from the sender and computes:
5:     $\tau \xleftarrow{\$} \mathbb{Z}_q$, $T_2 := T\mu^\tau$, and $T_3 := K^\tau g^\kappa \in \mathbb{G}$
6: $(T_1, T_2, T_3) \rightarrow \mathcal{V}$ and $x \xleftarrow{\$} \mathcal{V}$
7:         ▷ Non-interactive: $x := \mathsf{Hash}(W, T_1, T_2, T_3, acc, \texttt{asset}') \in \mathbb{Z}_q$
8: The sender computes: $s_1 := t + x(r') \in \mathbb{Z}_q$
9: The owner computes and shares $(s_2, s_3)$ with the sender:
10:     $s_2 := \tau + x(v' - v)$ and $s_3 := -\kappa + x(v' - v)k \in \mathbb{Z}_q$
11: The sender combines: return $\sigma := (x, s_1, s_2, s_3) \in \mathbb{Z}_q^4$
```

We define the verification function for any update proof below:

```
1: $\textbf{VerifyUpdate}(acc, \texttt{asset}', \sigma = (x, s_1, s_2, s_3); W)$:     ▷ $\mathcal{V}$ has
   $T_1, T_2, T_3$ in the interactive version
2: $(K, G, V) := acc$ and $(G', V') := \texttt{asset}'$
3: return $T_1 = g^{s_1}((G')^{-1}G)^x \wedge T_2 = K^{s_1}\mu^{s_2}((V')^{-1}V)^x \wedge T_3 = K^{s_2}g^{-s_3}$
```

In the non-interactive setup, the verification function recomputes (T_1, T_2, T_3) from the given x and checks if x is equal to $\mathsf{Hash}(W, T_1, T_2, T_3, acc, \texttt{asset}')$. Therefore, the non-interactive proof only contains four elements of $\mathbb{Z}_q$, and we do not share (T_1, T_2, T_3) with verifiers.

Security Definitions. Rerandomizable accounts provide ZKA, insider ZKA, and strong theft-resistance as defined below.

Definition 5.1 (ZKA of Rerandomized Accounts). Rerandomized accounts are zero-knowledge if they are ZKA for the verifier $\mathcal{V}(pp, u) = \mathsf{UpdateVerify}(acc, asset', \sigma; W)$, $\zeta = \phi$, and relation:

$$\left(pp, u = \begin{pmatrix} W, acc{=}(K, G, V), \\ asset'{=}(G', V'), \sigma \end{pmatrix}, (r', k, v, v', \kappa)\right) \in \mathcal{R}_{accounts} \Leftrightarrow$$

$$\begin{pmatrix} \begin{array}{r} (g, \mu) \overset{?}{\in} pp \wedge K \overset{?}{=} g^k \wedge \mathsf{OpenAsset}(k, v, (G, V)) \overset{?}{=} 1 \wedge \\ asset' \overset{?}{=} \mathsf{UpdateAsset}(r', v, v', acc) \wedge \mathsf{OpenAsset}(k, v', asset') \overset{?}{=} 1 \wedge \\ \left(\begin{pmatrix} \begin{array}{r} (v' \overset{?}{\neq} v) \wedge (\sigma \overset{?}{=} \mathsf{UpdateValueProve}(r', k, v, v', asset'; \kappa, W) \vee \\ \sigma \overset{?}{=} \mathsf{UpdateValueMProve}(r', k, v, v', asset'; \kappa, W)) \end{array} \end{pmatrix} \\ \vee \left((v' \overset{?}{=} v) \wedge \sigma \overset{?}{=} \mathsf{UpdateUpdate}(r', asset'; \kappa, W)\right)\right) \end{array} \end{pmatrix}$$

This relation $\mathcal{R}_{accounts}$ implies that given accounts and their update proofs, no one learns anything else about (r', k, v, v', κ).

We also want to obtain zero-knowledge for multiple account updates with the same r' when the accounts belong to the sender, receivers, and decoys. In this case, the adversary may controls all or some of the receivers, and can see the transcripts between the sender and these receivers, not only the transcripts between the sender and the verifiers. Therefore, we define *insider ZKA* for the following relation $\mathcal{R}^{insiders}_{accounts}$ against an adversary who controls $R_\mathcal{A}$ receiving accounts in indexes $J_\mathcal{A} = [j_\rho]_{\rho=0}^{R_\mathcal{A}} \subset [j_l]_{l=0}^{n}$.

Definition 5.2 (Insider ZKA of Rerandomized Accounts). Rerandomized accounts are insider zero-knowledge if they provide zero-knowledge argument for $0 \leq R_\mathcal{A} < n < N$,

verifier: $\mathcal{V}(pp, u) = \wedge_{i=0}^{N} \mathsf{UpdateVerify}(acc_i, asset'_i, \sigma_i; W)$,

insider knowledge: $\zeta = (J_\mathcal{A} = [j_\rho]_{\rho=0}^{R_\mathcal{A}}, [k_{j_\rho}, v_{j_\rho}, v'_{j_\rho}, \kappa_{j_\rho}]_{\rho=0}^{R_\mathcal{A}})$,

and $\left(pp, u = \begin{pmatrix} [acc_i, asset'_i, \\ \sigma_i]_{i=0}^{N}, W \end{pmatrix}, w = \begin{pmatrix} r', [j_l]_{l=0}^{n} \\ [k_i, v_i, v'_i, \kappa_i]_{i=0}^{N} \end{pmatrix}\right) \in \mathcal{R}^{insiders}_{accounts} \Leftrightarrow$

$$\begin{bmatrix} \bigwedge_{i=0}^{N} \left((pp, L), \left(W, acc_i, asset'_i, \sigma_i\right), \left(r', k_i, v_i, v'_i, \kappa_i\right)\right) \overset{?}{\in} \mathcal{R}_{accounts} \wedge \\ \bigwedge_{\rho=0}^{R_\mathcal{A}} \sigma_{j_\rho} \overset{?}{=} \mathsf{UpdateValueMProve}(r', k_{j_\rho}, v_{j_\rho}, v'_{j_\rho}, asset'_{j_\rho}; \kappa_{j_\rho}, W) \end{bmatrix}$$

It is important to note that $\mathcal{R}^{insiders}_{accounts}$ is equivalent to $\mathcal{R}_{accounts}$ of N accounts when $R_\mathcal{A} = 0$, i.e., the adversary does not control any insiders and only sees the transcripts of the sender and verifiers.

$\mathcal{R}^{insiders}_{accounts}$ implies that the adversary who controls all or a set of the receivers does not learn anything about the other witnesses, i.e., $(w \setminus \zeta) = (r', [j_l]_{l=0}^{N} \setminus J_\mathcal{A}, [k_i, v_i, v'_i, \kappa_i]_{i=0, i \notin J_\mathcal{A}}^{N})$ even if all accounts are updated from the same r', and the adversary knows insider knowledge ζ. Hence, **indistinguishability** is included in $\mathcal{R}^{insiders}_{accounts}$ such that given an account with a new balance and another account without a balance update, the adversary cannot identify which account has updated the balance. Moreover, $\mathcal{R}^{insiders}_{accounts}$ infers **accounts' sender-anonymity**, i.e., the receivers cannot identify the sending accounts' indexes even if the same r' is used for all account updates. We state indistinguishability in Definition B.1 and sender-anonymity in Definition B.2 (Appendix B) for the readers interested in singular definitions even if they are implied in $\mathcal{R}^{insiders}_{accounts}$.

Rerandomized accounts are theft-resistant if an account balance can be only modified during the rerandomization if and only if the prover knows the account secret key k. However, we need these accounts to provide *strong theft-resistance*, i.e., the theft-resistance property must hold even if the adversary has access to the previous update proofs. This strong theft-resistance property is necessary because an adversary can see correct proofs of previous transactions by observing the ledger.

Definition 5.3 (Theft-Resistance of Rerandomized Accounts). Let there be an account $acc = (\mathsf{pk}, \mathsf{a}_0 := \texttt{asset}_{\mathsf{pk}}(v_0, r_0))$ of key k and an oracle $\mathcal{O}_{k,acc,\mathbf{Q}}(r_i, v_{i-1}, v_i)$ that outputs ith updated asset a_i and its update proof σ_i as defined below.

```
$\underline{\mathcal{O}_{k,acc,\mathbf{Q}}(r, d, W):}$
$i := |\mathbf{Q}|$ and $v_i = v_{i-1} + d$ when $(i-1, v_{i-1}, \mathsf{a}_{i-1}) \in \mathbf{Q}$
    $\mathsf{a}_i := \mathsf{UpdateAsset}(r, k, v_{i-1}, v_i, \mathsf{pk}, \mathsf{a}_{i-1})$ and $\kappa \xleftarrow{\$} \mathbb{Z}_q$
    **if** $d \neq 0$ : $\sigma_i := \mathsf{UpdateValueProve}(r, k, v_{i-1}, v_i, \mathsf{pk}, \mathsf{a}_{i-1}, \mathsf{a}_i; \kappa, W)$
    **else**: $\sigma_i := \mathsf{UpdateProve}(r, \mathsf{pk}, \mathsf{a}_{i-1}, \mathsf{a}_i; \kappa \xleftarrow{\$} \mathbb{Z}_q, W)$
    $\mathbf{Q} := \mathbf{Q} \cup (i, v_i, \mathsf{a}_i)$ and return $(\mathsf{a}_i, \sigma_i)$
```

The oracle saves all of its outputs in $\mathbf{Q}$. We say that the accounts are strong theft resistance if the following probability is negligible.

$$Pr\left[\begin{array}{r|r} (i', v', \mathsf{a}') \notin \mathbf{Q} \wedge \mathsf{OpenAsset}(k, v', \mathsf{a}') \wedge & \mathcal{A}^{\mathcal{O}(\cdot)}(v_0, pk, \mathsf{a}_0) \\ \mathsf{UpdateVerify}(\mathsf{pk}, \mathsf{a}_{i'-1}, \mathsf{a}', \sigma) \wedge v_{i'-1} \neq v' & \rightarrow (i', v', \mathsf{a}', \sigma) \end{array}\right]$$

THEOREM 5.4. *Rerandomized accounts provide ZKA, insider ZKA, and strong theft resistance if the DL and DDH problems are hard, and Pedersen commitments are hiding and binding.*

Proof: We prove that Theorem 5.4 is true from Lemma C.1-C.4.

5.2 Anonymous Forced Openings

As we explained before, when an account is updated, the fact that the coin balance was modified or not is hidden. However, we must also verify that all updated accounts have non-negative balances. Therefore, we introduce a novel primitive called anonymous forced opening protocol, which forces to anonymously create commitments of *all* sending and receiving accounts' new account balances.

Our protocol works as follows. First, the participants create W including commitments of all new balances and other metadata according to **ForcedOutCreate**. Then, they use W as the challenge message in account updates (see **UpdateValueProve**). After that, they anonymously prove that they have opened all updated values' commitment by creating $\pi_{forced} \leftarrow$ **ForcedOutProve**. Later, verifiers check π_{forced} by running the verification **ForcedOutVerify**.

First, we explain **ForcedOutCreate**, which creates the commitments $[C_l]_{l=0}^{n}$ of new balances of $[v'_l]_{l=0}^{n}$. Here, $[j_l]_{l=0}^{n}$ are the sorted indexes of sending and receiving accounts in ascending order.

```
1:  **ForcedOutCreate**($[\mathsf{asset}'_i{=}(G'_i, V'_i)]_{i=0}^{N}, [j_l, v_l, v'_l, \alpha_l, C_l]_{l=0}^{n}$)
2:  if $v_l \notin [0, 2^L)$ or $v'_l \notin [0, 2^L)$: return $\perp$
3:  Each real participant $l \in [0, n)$ with $k_l$ of $\mathsf{asset}'_{j_l}$ creates:
4:      $(a_l, a'_l, b_l) \xleftarrow{\$} \mathbb{Z}_q$ ▷ Here, $C_l := \mathsf{Commit}_{h,\mu}(\alpha_l, v'_l)$ for key $\alpha_l$
5:      $A_l := C_l^{a_l} \in \mathbb{G}, A'_l := h^{a'_l} \in \mathbb{G}, B_l := (G'_{j_l})^{b_l} \in \mathbb{G}$
6:                                          ▷ shares $(A_l, A'_l, B_l)$ with the sender
7:                      ▷ The sender shares $A' := \prod_{l=0}^{n} A'_l \in \mathbb{G}$ with others
8:      $([G'_i, V'_i)]_{i=0}^{N}, C_l, A_l, A', B_l)) \rightarrow \mathcal{V}$ and $y_{a,l} \xleftarrow{\$} \mathcal{V}$
9:          ▷ Non-interactive: $y_{a,l}{:=}\mathsf{Hash}([G'_i, V'_i)]_{i=0}^{N}, C_l, A_l, A', B_l)$
10:     When $k_l$ is the secret key of $\mathsf{asset}'_{j_l}$'s account:
11:         $f_l := k_l(v'_l - v_l) - y_{a,l}a_l \in \mathbb{Z}_q$
12:         $f_l \rightarrow \mathcal{V}$
13:         $y_{b,l} \xleftarrow{\$} \mathcal{V}$      ▷ Non-interactive: $y_{b,l} := \mathsf{Hash}(y_{a,l}, f_l) \in \mathbb{Z}_q$
14:         $z_l := k_l^2(v'_l - v_l) - y_{b,l}b_l \in \mathbb{Z}_q$
15:     $D_{j_l} := (V'_{j_l})^{\kappa_{j_l}}$ for $\kappa_{j_l} \xleftarrow{\$} \mathbb{Z}_q$              ▷ keeps $\kappa_{j_l}$ a *secret*
16: The sender computes for $i \in [0, N), i \notin [j_l]_{l=0}^{n}$:
17:     $D_i := (V'_i)^{\kappa_i}$ for $\kappa_i \xleftarrow{\$} \mathbb{Z}_q$
18: The sender aggregates: $D := \prod_{i=0}^{N} D_i \in \mathbb{G}$
19: return $W := ([C_l, \pi_l, A_l, B_l, f_l, z_l]_{l=0}^{n}, D, A'); secrets : [\kappa_i]_{i=0}^{N}$
```

Once all accounts have been updated with W and $[\kappa_i]_{i=0}^{N}$, the participants prove that all updated balances are committed in commitments. Here, $[\sigma_i]_{i=0}^{N}$ are the set of the update proofs of accounts.

```
1:  **ForcedOutProve**($[(G'_i, V'_i), \sigma_i]_{i=0}^{N}, [j_l, v_l, v'_l, \alpha_l]_{l=0}^{n}, W$):
2:  $([C_l, \pi_l, A_l, A'_l, B_l, f_l, z_l]_{l=0}^{n}, D) := W$        ▷ Challenge message
3:  $[(x_i, s_{1,i}, s_{2,i}, s_{3,i}) := \sigma_i]_{i=0}^{N}$                          ▷ Update proofs
4:  sender: $\bar{V} := D \prod_{i=0}^{N} (V'_i)^{s_{3,i}}$          ▷ $\bar{V} = \prod_{l=0}^{n} (V'_{j_l})^{x_{j_l} k_l (v'_l - v_l)}$
5:      $([s_{3,i}, x_i]_{i=0}^{N}, \bar{V}) \rightarrow \mathcal{V}$
6:      $y \xleftarrow{\$} \mathcal{V}$        ▷ Non-interactive:$y := \mathsf{Hash}(W, [s_{3,i}, x_i]_{i=0}^{N}, \bar{V})$
7:      $\bar{D} := \bar{V}(A')^y \in \mathbb{G}$
8:      $[\bar{C}_l := B_l^{y_{b,l}} C_l^{f_l} A_l^{y_{a,l}}]_{l=0}^{n} \in \mathbb{G}^n$
9:  Each participant $j_l$ computes: $\xi_l := \alpha_l x_{j_l} k_l (v'_l - v_l) + y a'_l$ and
    shares $\xi_l$ with the sender.
10: The sender creates ascending-order $M$ index sets of all $n$ combi-
    nations of $N$ when $M{=}\frac{N!}{(n)!(N-n)!}$. Let $[[i_{m,l}]_{l=0}^{n}]_{m=0}^{M}$ be these
    $M$ combinations, and combination $j$ be the sending/receiving
    accounts' index set such that $[i_{j,l}]_{l=0}^{n} = [j_l]_{l=0}^{n}$.
11: for $m{\in}[0, M) : H_m{:=}\bar{D} \prod_{l=0}^{n} \left((G'_{i_{m,l}})^{z_l} \bar{C}_l\right)^{-x_{i_{m,l}}} \in \mathbb{G}$
12:         ▷ $H_j{=}\mu^0 h^{-\sum_{l=0}^{n} \xi_l}$ for combination $j$ of senders/receivers'
13: The sender computes the proof when $j$ is the group index
14: return $\pi_{forced} := \mathsf{ZeroComProve}([H_m]_{m=0}^{M}, j, \xi := \sum_{l=0}^{n} \xi_l)$
```

The verification function checks if all new balances were opened in commitments as follows.

```
1:  **ForcedOutVerify**($[\mathsf{asset}'_i{=}(G'_i, V'_i), \sigma_i]_{i=0}^{N}, W, \pi_{forced}$):
2:  $([C_l, \pi_l, A_l, A'_l, B_l, f_l, z_l]_{l=0}^{n}, D) := W$        ▷ Challenge message
3:  $[(x_i, s_{1,i}, s_{2,i}, s_{3,i}) := \sigma_i]_{i=0}^{N}$                          ▷ Update proofs
4:      ▷ Non-interactive: For $l \in [0, n)$: $y_{a,l} := \mathsf{Hash}([G'_i, V'_i)]_{i=0}^{N}$,
5:                          ▷ $C_l, A_l, A', B_l), y_{b,l} := \mathsf{Hash}(y_{a,l}, f_l)$
6:  $\bar{V} := D \prod_{i=0}^{N} (V'_i)^{s_{3,i}}$
7:      ▷ Non-interactive version: $y := \mathsf{Hash}(W, [s_{3,i}, x_i]_{i=0}^{N}, \bar{V})$
8:  $\bar{D} := \bar{V}(A')^y \in \mathbb{G}$
9:  $[\bar{C}_l := B_l^{y_{b,l}} C_l^{f_l} A_l^{y_{a,l}}]_{l=0}^{n} \in \mathbb{G}^n$
10: Let $[[i_{m,l}]_{l=0}^{n}]_{m=0}^{M}$ be all $n$ combinations of $N$ indexes.
11: for $m \in [0, M) : H_m := \bar{D} \prod_{l=0}^{n} \left((G'_{i_{m,l}})^{z_l} \bar{C}_l\right)^{-x_{i_{m,l}}} \in \mathbb{G}$
12: return $\mathsf{ZeroComVerify}([H_m]_{m=0}^{M}, \pi_{forced})$
```

Security Definitions. We define ZKA of anonymous forced openings for relation $\mathcal{R}_{forced}$ when the adversary controls $R_{\mathcal{A}} \geq 0$ number of receiving accounts in indexes $J_{\mathcal{A}} = [j_\rho]_{\rho=0}^{R_{\mathcal{A}}}$ as follows.

Definition 5.5 (Secure Anonymous Forced Openings). Anonymous Forced Openings are zero-knowledge if they provide ZKA for

verifier: $\mathcal{V}(pp, u) = \mathsf{ForcedOutVerify}([\mathsf{asset}'_i, \sigma_i]_{i=0}^{N}, W, \pi_{forced})$,

insider knowledge: $\zeta = (J_{\mathcal{A}}{=}[j_\rho]_{\rho=0}^{R_{\mathcal{A}}}, [k_{j_\rho}, v_{j_\rho}, v'_{j_\rho}, \kappa_{j_\rho}]_{\rho=0}^{R_{\mathcal{A}}}$,

$[v''_l, \alpha_l]_{l=0, j_l \in J_{\mathcal{A}}}^{n}$) and the relation $\mathcal{R}_{forced}$:

$$\left(pp, u{=}\begin{pmatrix} [acc_i, \mathsf{asset}'_i, \sigma_i]_{i=0}^{N} \\ W, \pi_{forced} \end{pmatrix}, w{=}\begin{pmatrix} j, r', \xi, [j_l]_{l=0}^{n} \\ [k_i, v_i, v'_i, \kappa_i]_{i=0}^{N} \\ [v''_l, \alpha_l]_{l=0}^{n} \end{pmatrix}\right) \in \mathcal{R}_{forced} \Leftrightarrow$$

$$\left(\begin{array}{r} (g, h, \mu) \overset{?}{\in} pp \wedge [j_l]_{l=0}^{n} \overset{?}{\in} [0, N)^n \wedge j \overset{?}{\in} [0, M) \wedge \\ \left(\bigwedge_{l=0}^{n} v''_l \overset{?}{=} v'_{j_l} \wedge \bigwedge_{i=0, i \notin [j_l]_{l=0}^{n}}^{N} v_i \overset{?}{=} v'_i\right) \wedge [C_l = \mathsf{Commit}_{h,\mu}(\alpha_l, v''_l)]_{l=0}^{n} \overset{?}{\in} W \\ \wedge \left(W; [\kappa_i]_{i=0}^{N}\right) \overset{?}{=} \mathsf{ForcedOutCreate}\begin{pmatrix} [acc_i, \mathsf{asset}'_i]_{i=0}^{N}, \\ [j_l, v_{j_l}, v'_{j_l}, \alpha_l, C_l]_{l=0}^{n}] \end{pmatrix} \wedge \\ \left(pp, \begin{pmatrix} [acc_i, asset'_i, \sigma_i]_{i=0}^{N} \\ W \end{pmatrix}, \begin{pmatrix} r', [j_l]_{l=0}^{n} \\ [k_i, v_i, v'_i, \kappa_i]_{i=0}^{N} \end{pmatrix}\right) \overset{?}{\in} \mathcal{R}_{accounts}^{insiders} \wedge \\ \pi_{forced} \overset{?}{=} \mathsf{ForcedOutProve}\begin{pmatrix} [\mathsf{asset}'_i, \sigma_i]_{i=0}^{N}, [j_l, v_{j_l}, v'_{j_l}, \alpha_l]_{l=0}^{n}, \\ W; [\kappa_i]_{i=0}^{N}, [k_{j_l}]_{l=0}^{n}] \end{pmatrix} \wedge \\ M \overset{?}{=} \frac{N!}{(n)!(N-n)!} \wedge (pp, ([H_m]_{m=0}^{M}, \pi_{forced}, N), (\xi, j)) \overset{?}{\in} \mathcal{R}_{zero} \end{array}\right)$$

when $0 \leq R_{\mathcal{A}} < n < N$ and $[H_m]_{m=0}^{M}$ is created according to Step 11 of **ForcedOutCreate** and Step 11 of **ForcedOutVerify** such that

$$\left[H_m = D \prod_{i=0}^{N} (V_i')^{s_{3,i}} (A')^{y_{a'}} \prod_{l=0}^{n} \left((G_{j_l}')^{z_l} B_l^{y_{b,l}} C_l^{f_l} A_l^{y_{a,l}} \right)^{-x_{j_l}} \right]_{m=0}^{M} .$$

Here, ZKA implies that the adversary learns nothing about $(w \setminus \zeta) = (j, r', [j_l]_{l=0}^{N} \setminus J_{\mathcal{A}}, [k_i, v_i, v_i', \kappa_i]_{i=0, i \notin J_{\mathcal{A}}}^{N}, [v_l'', \alpha_l]_{l=0, j_l \notin J_{\mathcal{A}}}^{n})$ except $\mathcal{R}_{forced}$ even with insider knowledge ζ.

Theorem 5.6. *The anonymous forced opening protocol is ZKA for relation $\mathcal{R}_{forced}$ under the assumptions that Pedersen commitments are hiding and binding, rerandomized accounts provide ZKA for $\mathcal{R}_{account}^{insiders}$, and one-of-many proofs are ZKA for $\mathcal{R}_{zero}$.*

Proof: We claim the validity of Theorem 5.6 from Lemma D.1 and Lemma D.2 (see Appendix D).

5.3 Balance Proofs

A decentralized transaction may contain rewards and transaction fee as an incentive to add the transaction to a block. To prevent illegal coin generation, we ensure that the total input balance, releasing withheld coins, and rewards are equal to the total output balance, new withheld coins, and transaction fee. We propose the following balance proof protocol for rerandomizable accounts. The sender who updated accounts by some r' creates proofs as follows.

```
1:  BalanceProve($f, f', C, c, \alpha_c, C', c', \alpha_{c'}, [acc_i, \mathtt{asset}_i']_{i=0}^{N}; r'$):
2:  if $f \notin [0, 2^L) \lor f' \notin [0, 2^L)$: return $\perp$
3:  $(R_i, K_i, G_i, V_i) := acc_i$ and $(G_i', V_i') := \mathtt{asset}_i'$
4:  $E := C'C^{-1} \prod_{i=0}^{N} V_i' \times V_i^{-1} \in \mathbb{G}$
5:  $U = h^{u'} (\prod_{i=0}^{N} K_i)^u \in \mathbb{G}$ for $(u, u') \xleftarrow{\$} \mathbb{Z}_q$
6:  $(f, f', C, C', U, E) \longrightarrow \mathcal{V}$                  ▷ to the verifier
7:  $y \xleftarrow{\$} \mathcal{V}$      ▷ Non-interactive: $y := \mathsf{Hash}(f, f', C, C', U, E) \in \mathbb{Z}_q$
8:  $\sigma_{balance} := (U, \bar{s} := r' - yu, \bar{s}' := (\alpha_{c'} - \alpha_c) - yu')$
9:  BalanceVerify($f, f', C, C', [acc_i, \mathtt{asset}_i']_{i=0}^{N}, \sigma_{balance}$):
10: $E := C'C^{-1} \prod_{i=0}^{N} V_i' \times V_i^{-1} \in \mathbb{G}$
11:            ▷ Non-interactive: $y := \mathsf{Hash}(f, f', C, C', U, E) \in \mathbb{Z}_q$
12: return $(f, f') \in [0, 2^L)^2 \land U^y h^{\bar{s}'} (\prod_{i=0}^{N} K_i)_i^{\bar{s}} = \mu^{f' - f} E \in \mathbb{G}$
```

We define zero-knowledge of balance proofs below.

Definition 5.7 (Zero-Knowledge Balance Proofs). Balance proofs are zero-knowledge if they are ZKA for $\mathcal{V}(pp, u) = \text{BalanceVerify}(f, f', C, C', [acc_i, \mathtt{asset}_i']_{i=0}^{N}, \sigma_{balance})$, $\zeta = \phi$, and relation $\mathcal{R}_{balance}$:

$$\left(pp, u = \begin{pmatrix} [acc_i, \mathtt{asset}_i, \sigma_i]_{i=0}^{N}, \\ f, f', C, C', \sigma_{balance} \end{pmatrix}, w = \begin{pmatrix} [v_i, v_i', k_i, \kappa_i]_{i=0}^{N} \\ \alpha_c, c, \alpha_{c'}, c', r' \end{pmatrix} \right) \in \mathcal{R}_{balance} \Leftrightarrow$$

$$\left(\begin{array}{r} (g, h, \mu, L) \overset{?}{\in} pp) \land n \overset{?}{\leq} N \land [j_l]_{l=0}^{n} \overset{?}{\in} [0, N)^n \land \\ f + c + \sum_{l=0}^{n} v_l \overset{?}{=} f' + c' + \sum_{l=0}^{n} v_l' \land \\ \land_{i=0}^{N} \left((pp, L), \left(W, acc_i, asset_i', \sigma_i \right), \left(r', k_i, v_i, v_i', \kappa_i \right) \right) \overset{?}{\in} \mathcal{R}_{accounts} \land \\ C \overset{?}{=} \text{Commit}_{h,\mu}(\alpha_c, c) \land C' \overset{?}{=} \text{Commit}_{h,\mu}(\alpha_{c'}, c') \land \\ \sigma \overset{?}{=} \text{BalanceProve}(f, f', C, c, \alpha_c, C', c', \alpha_{c'}, [acc_i, \mathtt{asset}_i']_{i=0}^{N}; r') \end{array} \right)$$

Theorem 5.8. *Nopenena balance proof protocol is ZKA for $\mathcal{R}_{balance}$ if the DL problem is hard, Pedersen commitments are binding, and rerandomized accounts are ZKA for $\mathcal{R}_{accounts}$.*

Proof: We claim the validity of Theorem 5.8 from Lemma E.1-E.2.

6 NOPENENA TRANSACTIONS

We present a Nopenena payment protocol in this section.

We explain the Nopenena protocol by taking a simple example, where Alice sends coins from account acc_1 to Bob's account acc_3 by taking Charles' account acc_2 as a decoy. We illustrate the example in Figure 2. First, the sender, i.e., Alice, rerandomizes all accounts with new coins balances using the function UpdateAsset. Then, Alice creates a commitment C_1 for her new account balance, and Bob creates a commitment C_2 for his new account balance. Then, they run ForcedOutCreate to create the metadata of forced outputs W. Alice and Bob create range proofs for C_1 and C_2, respectively, to show the validity of new balances. After that, Alice proves the validity of her account's update by running UpdateValueProve and the validity of the decoy account's update by running UpdateProve. Alice and Bob also prove his account's update by running the multi-party UpdateValueMProve. All these update proofs take W as the challenge message. Once updates are proven, Alice and Bob prove the validity of the commitments (C_1, C_2) from ForcedOutProve. Finally, Alice creates a balance proof to show that the input coin amount is equal to the output coin amount. In this example, (1) the verifiers only learn that two accounts exchanged coins out of (acc_1, acc_2, acc_3) but not exactly which two accounts, i.e., untraceability, and (2), Bob only learns that either acc_1 or acc_2 belong to Alice, but not exactly which account, i.e., sender anonymity.

We explain the protocol below. The steps related to contracts are in blue-colored text; they can be ignored if contracts are not used.

```
1:  CreateTx($\Lambda, [acc_i]_{i=0}^{N}, [j_l, v_l, v_l', \alpha_l, C_l]_{l=0}^{n}, [\mathcal{F}_{zk}, C, \pi_{zk}, \alpha_c, c]$,
2:      $[\mathcal{F}_{zk}', C', \alpha_{c'}, c'], f, f'; [k_l]_{l=0}^{n})$ $:tx := ([acc_i']_{i=0}^{N}, \pi)$
3:  if $f + c + \sum_{l=0}^{n} v_l \neq f' + c' + \sum_{l=0}^{n} v_l'$ : return $\perp$
4:  if $[acc_i]_{i=0}^{N} \notin \mathtt{Accounts} \in \Lambda$: return 0        ▷ the accounts exist
5:  if $[\mathcal{F}_{zk}, C] \notin \mathtt{Withheld} \in \Lambda$: return 0             ▷ the coins exist
6:  checks the correctness of the given contract data
7:  if $\mathcal{F}_{zk}(C \| C' \| [C_l]_{l=0}^{n}, \pi_{zk}) = 0$: return $\perp$
8:  The sender gets $r' \xleftarrow{\$} \mathbb{Z}_q$ and updates all accounts.
9:      $\forall l \in [0, n)$: $\mathtt{asset}_{j_l}' :=$ UpdateAsset($r', v_l, v_l', acc_{j_l}$)
10:     $\forall i \in [0, N), i \notin [j_l]_{l=0}^{n}$: $\mathtt{asset}_i' :=$ UpdateAsset($r', acc_i$)
11: All participants run a multi-party protocol to open all value-
    changed accounts, i.e., their balance has been changed.
12: $(W; [\kappa_i]_{i=0}^{N}) :=$ ForcedOutCreate($[\mathtt{asset}_i']_{i=0}^{N}, [j_l, v_l, v_l', \alpha_l, C_l]_{l=0}^{n}$)
13: Each participant in $[j_l]_{l=0}^{n}$ proves that the new balance $\in [0, 2^L)$:
14:     $\pi_{range,l} :=$ RangeProve($C_l, v_l', \alpha_l, L$)
15: The sender creates range proofs for new withheld coins
16: $\pi_{C'} :=$ RangeProve($C', c', \alpha_{c'}, L$)
17: Each participant creates the challenge message from $W$
18: $\hat{W} = (W, [\mathcal{F}_{zk}, C], [\mathcal{F}_{zk}', C'])$               ▷ challenge message
19: The sender in $[j_l]_{l=0}^{n}$ proves the correctness of the update:
20: $\sigma_{j_l} :=$ UpdateValueProve($r', k_l, v_l, v_l', acc_{j_l}, \mathtt{asset}_{j_l}'; \kappa_{j_l}, \hat{W}$)
21: The sender and each receiver in $[j_l]_{l=0}^{n}$ proves the update:
22: $\sigma_{j_l} :=$ UpdateValueMProve($r', k_l, v_l, v_l', acc_{j_l}, \mathtt{asset}_{j_l}'; \kappa_{j_l}, \hat{W}$)
23: The sender proves the updates of decoy accounts.
24:     for $i \in [0, N), i \notin [j_l]_{l=0}^{n}$:
25:         $\sigma_i :=$ UpdateProve($r', acc_i, \mathtt{asset}_i'; \kappa_i, \hat{W}$)
```

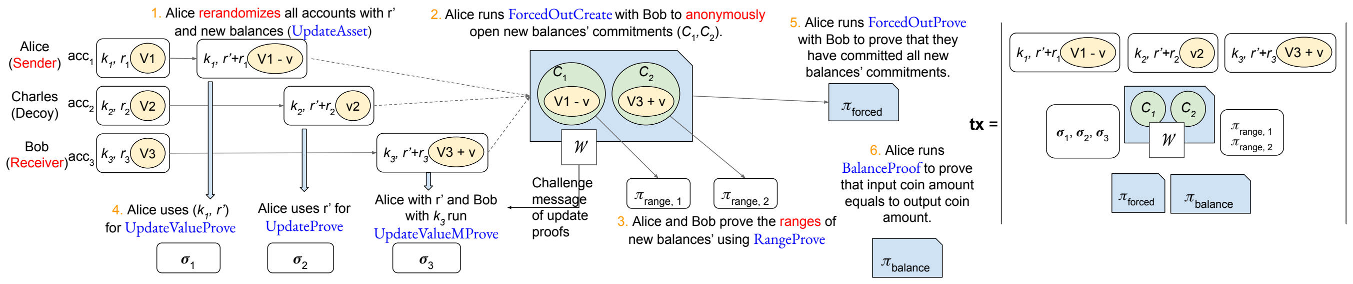

Alice (Sender)
Charles (Decoy)
Bob (Receiver)
acc1
acc2
acc3
k1, r1 V1
k2, r2 V2
k3, r3 V3
1. Alice rerandomizes all accounts with r' and new balances (UpdateAsset)
k1, r'+r1 V1 - v
k2, r'+r2 v2
k3, r'+r3 V3 + v
2. Alice runs ForcedOutCreate with Bob to anonymously open new balances' commitments (C1,C2).
C1 V1 - v
C2 V3 + v
W
Challenge message of update proofs
5. Alice runs ForcedOutProve with Bob to prove that they have committed all new balances' commitments.
π forced
π range, 1
π range, 2
3. Alice and Bob prove the ranges of new balances' using RangeProve
6. Alice runs BalanceProof to prove that input coin amount equals to output coin amount.
π balance
4. Alice uses (k1, r') for UpdateValueProve
Alice uses r' for UpdateProve
Alice with r' and Bob with k3 run UpdateValueMProve
σ1
σ2
σ3
tx =
σ1, σ2, σ3
π range, 1 π range, 2
π forced
π balance

Figure 2: An examplary protocol flow of Nopenena. Here, Alice sends v coins to Bob without revealing Alice's account to Bob, i.e., Bob only learns that either acc_1 or acc_2 belong to Alice. Also, Alice and Bob hide which accounts are transferring coins from verifiers, i.e., verifiers only see that two accounts from (acc_1, acc_2, acc_3) exchange coins. Note that Alice *does not* reveal her (k_1, r') to Bob, and Bob *does not* reveal his k_3 to Alice.

```
26: Participants prove that all changed balances were committed:
27: $\pi_{forced}$:=ForcedOutProve($[\mathsf{asset}'_i, \sigma_i]_{i=0}^N, [j_l, v_l, v'_l, \alpha_l]_{l=0}^n, W$)
28: The sender proves that no new coins were generated:
29: $\sigma_{balance}$:=BalanceProve($f, f', C, c, \alpha_c, C', c', \alpha_{c'}$,
30:         $[acc_i, \mathsf{asset}'_i]_{i=0}^N; r'$)
31: The sender combines all proofs and creates the transaction:
32: return $tx := (f, f', [acc_i, \mathsf{asset}'_i, \sigma_i]_{i=0}^N, (W, \pi_{forced},$
33:     $\sigma_{balance}, \pi_{contract}, [\pi_{range,l}]_{l=0}^n), [\mathcal{F}_{zk}, C], [\mathcal{F}'_{zk}, C', \pi_{C'}])$
```

```
1: VerifyTx($tx$):
2: $tx =: (f, f', [acc_i, \mathsf{asset}'_i, \sigma_i]_{i=0}^N, (W, \pi_{forced},$
3:     $\sigma_{balance}, \pi_{contract}, [\pi_{range,l}]_{l=0}^n), [\mathcal{F}_{zk}, C], [\mathcal{F}'_{zk}, C', \pi_{C'}])$
4: if $[acc_i]_{i=0}^N \notin \mathsf{Accounts} \in \Lambda$: return 0    ▷ the accounts exist
5: if $[\mathcal{F}_{zk}, C] \notin \mathsf{Withheld} \in \Lambda$: return 0    ▷ the coins exist
6: $\hat{W} = (W, [\mathcal{F}_{zk}, C], [\mathcal{F}'_{zk}, C'])$    ▷ challenge message
7: for $i \in [0, N)$:    ▷ verify updates
8:     if ¬UpdateVerify($acc_i, \mathsf{asset}'_i, \sigma_i, \hat{W}$): return 0
9: if ¬ForcedOutVerify($[\mathsf{asset}'_i, \sigma_i]_{i=0}^N, W, \pi_{forced}$): return 0
10: for $l \in [0, n)$: if ¬RangeVerify($C_l, \pi_{range,l}$) : return 0
11: if $\mathcal{F}_{zk}(C \| C' \| [C_l]_{l=0}^n, \pi_{zk}) = 0$: return 0
12: if ¬RangeVerify($C', \pi_{C'}$): return 0    ▷ valid withheld coins
13: return BalanceVerify($f, f', C, C', [acc_i, \mathsf{asset}'_i]_{i=0}^N, \sigma_{balance}$)
```

Once the transaction is accepted into the ledger, the accounts' current assets $[\mathsf{asset}_i]_{i=0}^N$ will be replaced with new $[\mathsf{asset}'_i]_{i=0}^N$. Moreover, the withheld coins, i.e., $[\mathcal{F}_{zk}, C]$ will be removed, and new withheld coins $[\mathcal{F}'_{zk}, c']$ will be added to Withheld. Therefore, the sizes of Accounts and Withheld do not grow monotonically.

Nopenena Unexpiring Transactions Decentralized payments' latency depends on many external factors like transaction fees, and users can add time-sensitive transactions faster by paying a satisfactory fee compared to others. However, it is better to have a recovery mechanism like unexpiring transactions as a last resort against targeted asset updates that prohibit an account owner from conducting his/her transactions. Therefore, we introduce a protocol for *unexpiring transactions* at the cost of untraceability.

Let there be n number of accounts who want to exchange coins where $n - |open|$ accounts want to remain anonymous while $|open|$ accounts want to ensure that their exchange will not be affected by other transactions' intentional or unintentional asset updates. For example, Bob, in *open* accounts, wants to receive coins as soon as possible, but Alice, who is in unopened $n - |open|$ accounts **does not** want to send coins revealing her account.

First, *open* accounts send an *unexpiring conditional transaction*, instructing to temporarily lock their accounts because they want to exchange coins with accounts $[\mathsf{pk}_i]_{i=0}^N$ when the first $|open|$ public keys are theirs. In the conditional transaction, they agree to a condition that they will only send coins to $[\mathsf{pk}_i]_{i=0}^N$ and pay a transaction fee f in the second transaction. After stopping further asset updates, they send the second transaction with f to complete the transaction and unlock open accounts. However, a locked account can be unlocked before the second transaction by paying an unlock fee f_u. Therefore, even if the second transaction does not happen, users can unlock the accounts, and the miners receive compensation for adding the conditional transaction. Also, these additional fees prevent users from locking accounts all the time and leaving a small unlocked account set for decoys. First, we explain the conditional transaction protocol.

Here, we assume that each account pk_i has been locked $lock_i$ times in the past, which can be identified from looking at ledger Λ.

```
1: CreateConditionalTx($\Lambda, f, f_u, [\mathsf{pk}_i]_{i=0}^N; [k_i]_{i=0}^{|open|}$):    ▷ locks
2: $\hat{W} := ([\mathsf{pk}_i]_{i=0}^N, |open|, f, f_u)$    ▷ the condition and challenge
3: Each $i \in [0, |open|)$ proves the knowledge of $k_i$ via value-updates
4:     $\hat{W}_i := \{\hat{W} \| lock_i \in \Lambda\}$    ▷ personalize the challenge
5:     $acc_i := (\mathsf{pk}_i, \mathsf{asset}_{\mathsf{pk}_i}(0,0))$ and $(r_i, \kappa_i) \xleftarrow{\$} \mathbb{Z}_q^2$
6:     $\sigma_i$:=UpdateValueProve($r'_i, k_i, 0, 1, acc_i, \mathsf{asset}_{\mathsf{pk}_i}(1,0), \kappa_i, \hat{W}_i$)
7: return $ctx := (|open|, [\mathsf{pk}_i]_{i=0}^N, [\sigma_i]_{i=0}^{|open|}, f, f_u)$
```

Verifiers check the conditional transaction as follows.

```
1: VerifyConditionalTx($ctx$=($|open|, [\mathsf{pk}_i]_{i=0}^N, [\sigma_i]_{i=0}^{|open|}, f, f_u$))
2: $\hat{W} := ([\mathsf{pk}_i]_{i=0}^N, |open|, f, f_u)$ ▷ the condition and the challenge
3: $\forall i \in [0, |open|)$: $\hat{W}_i := \{\hat{W} \| lock_i \in \Lambda\}$    ▷ updated challenge
4:     if ¬UpdateVerify($\mathsf{pk}_i, \mathsf{asset}_{\mathsf{pk}_i}(0,0), \mathsf{asset}_{\mathsf{pk}_i}(1,0), \bar{\sigma}_i, \hat{W}_i$):
5:         return 0
```

These conditional transactions *do not expire* if someone else updates their accounts' assets. Once the conditional transaction is added to the ledger, (1) the ledger **marks** *open* accounts as "locked"; hence, others cannot use locked accounts anymore as decoys, and (2) the ledger updates the number of times that account i was locked, i.e., $lock_i = lock_i + 1$. We use $lock_i$ to prevent replaying transactions since previous $lock_i$ is no longer valid.

After that, they collect last account states, $[acc_i]_{i=0}^N$ from the ledger and send the completing transaction. Here, $[j_l]_{l=0}^{|open|}$ =

$[l]_{l=0}^{|open|}$ are the open accounts' indexes. Other real participants' indexes are in $[j_l]_{l=|open|}^{n}$ that could be any index in $[|open|, N)$.

```
1: $\textbf{CreateCompletingTx}(f, [acc_i]_{i=0}^N, [j_l, v_l, v'_l]_{l=0}^n; r')$:
2: return $tx := \textbf{CreateTx}([acc_i]_{i=0}^N, [j_l, v_l, v'_l]_{l=0}^n, c, f)$
```

The verifiers check the condition and the transaction.

```
1: $\textbf{VerifyCompletingTx}(tx, ctx)$:
2: $(|open|, [\bar{\mathsf{pk}}_i]_{i=0}^N, [\sigma_i]_{i=0}^n, \bar{f}, f_u) := ctx$
3: $(c, f, [acc_i{:=}(\mathsf{pk}_i, \mathsf{asset}_i), \mathsf{asset}'_i, \sigma_i]_{i=0}^N, \mathsf{txh}){:=}tx$
4: return $[\mathsf{pk}]_{i=0}^N = [\bar{\mathsf{pk}}]_{i=0}^N$ and $f = \bar{f}$ and $\textbf{VerifyTx}(tx)$
```

If an account *acc* decides to unlock before the second transaction, the next transaction with *acc* must pay f_u in addition to the transaction fee, which will compensate the miners for locking *acc*.

Security Definitions. Nopenena payments provide the following security properties: ZKA and strong theft-resistance.

We define the zero-knowledge argument of payments for a relation $\mathcal{R}_{Nopenena}$ as follows when the adversary controls $R_{\mathcal{A}}$ $(0 \leq R_{\mathcal{A}} < n)$ receiving accounts out of n sending and receiving accounts.

Definition 6.1. Nopenena payments are zero-knowledge if they provide ZKA for the followings:

verifier: $\mathcal{V}(pp, tx) = \mathsf{VerifyTx}(\Lambda, tx)$,

insider knowledge: $\zeta = (J_{\mathcal{A}}{=}[j_\rho]_{\rho=0}^{R_{\mathcal{A}}}, [k_{j_\rho}, v_{j_\rho}, v'_{j_\rho}, \kappa_{j_\rho}]_{\rho=0}^{R_{\mathcal{A}}}, [\alpha_l]_{l=0, j_l \in J_{\mathcal{A}}}^{n})$, and relation $\mathcal{R}_{Nopenena}$ when $0{\leq}R_{\mathcal{A}}{<}n < N$:

$$\left(pp, tx, w = \begin{pmatrix} j, r', \xi, [j_l]_{l=0}^n, [k_i, v_i, v'_i, \kappa_i]_{i=0}^N \\ [\alpha_l]_{l=0,}^n, \alpha_c, c, \alpha_{c'}, c', blinds \end{pmatrix}\right) \in \mathcal{R}_{Nopenena} \Leftrightarrow$$

$$\left(\begin{array}{r} (g, h, \mu, L) \overset{?}{\in} pp \wedge [j_l]_{l=0}^n \overset{?}{\in} [0, N)^n \wedge \\ tx \overset{?}{=} \begin{pmatrix} f, f', [acc_i, \mathsf{asset}'_i, \sigma_i]_{i=0}^N, W, \pi_{forced}, \sigma_{balance}, \\ \pi_{contract}, [\pi_{range,l}]_{l=0}^n, [\mathcal{F}_{zk}, C], [\mathcal{F}'_{zk}, C', \pi_{C'}] \end{pmatrix} \wedge \\ \left(pp, \begin{pmatrix} [acc_i, asset'_i, \sigma_i]_{i=0}^N \\ W \end{pmatrix}, \begin{pmatrix} r', [j_l]_{l=0}^n \\ [k_i, v_i, v'_i, \kappa_i]_{i=0}^N \end{pmatrix}\right) \overset{?}{\in} \mathcal{R}_{accounts}^{insiders} \wedge \\ \left(pp, \begin{pmatrix} [acc_i, \mathsf{asset}'_i, \sigma_i]_{i=0}^N \\ W, \pi_{forced} \end{pmatrix}, \begin{pmatrix} j, r', \xi, [j_l]_{l=0}^n \\ [k_i, v_i, v'_i, \kappa_i]_{i=0}^N \\ [v''_l, \alpha_l]_{l=0}^n \end{pmatrix}\right) \overset{?}{\in} \mathcal{R}_{forced} \\ \wedge \left[((pp, L), (C_l, \pi_{range,l}), (\alpha_l, v'_{j_l})) \overset{?}{\in} \mathcal{R}_{range}\right]_{l=0}^n \wedge \\ \left(pp, \begin{pmatrix} [acc_i, \mathsf{asset}_i]_{i=0}^N, \\ C, C', \sigma_{balance} \end{pmatrix}, \begin{pmatrix} [j_l, v_l, v'_l, \alpha_l, k_l]_{l=0}^n \\ \alpha_c, c, \alpha_{c'}, c' \end{pmatrix}\right) \overset{?}{\in} \mathcal{R}_{balance} \wedge \\ (pp(\mathcal{F}_{zk}, [C_i]_{i=0}^*, \pi)(\mathcal{F}, [v_l, \alpha_l]_{l=0}^*, blinds)) \overset{?}{\in} \mathcal{R}_{contract} \end{array}\right)$$

when $(w \backslash \zeta){=}(j, r', \xi, [j_l]_{l=0}^N \backslash J_{\mathcal{A}}, [k_i, v_i, v'_i, \kappa_i]_{i=0, i \notin J_{\mathcal{A}}}^N, [\alpha_l]_{l=0, j_l \notin J_{\mathcal{A}}}^n, \alpha_c, c, \alpha_{c'}, c', blinds)$.

$\mathcal{R}_{Nopenena}$ implies **untraceability** since an adversary who only sees transcripts between the sender and verifiers, i.e., $R_{\mathcal{A}} = 0$, does not learn anything about the secret witnesses, which include the indexes of sending and receiving accounts $[j_l]_{l=0}^0$. Also, $\mathcal{R}_{Nopenena}$ implicates **sender-anonymity** since an adversary who controls all receiving accounts, only learns that the sending accounts are in $[0, N) \setminus J_{\mathcal{A}}$. We define untraceability in Definition B.3 and sender-anonymity in Definition B.3 inferred by $\mathcal{R}_{Nopenena}$ for the interested readers.

Theft-resistance defines that an adversary who does not know the secret key of an honest account cannot change account balances. Here, "strong" theft resistance denotes that a p.p.t. adversary can query transactions from the honest account owners yet cannot create a **fresh** transaction that changes the account balance. This is because, in decentralized payment systems, the adversary can see previous payments of the honest owner in the ledger. Here, we give an oracle O_k to the adversary to create transactions on behalf of the honest owner. The transactions created by O_k are stored in **TX**. Transactions are strongly theft-resistant if the adversary cannot create a fresh transaction that is not in **TX** and changes the account balance of the honest account.

Definition 6.2 (Strong Theft-resistance). Assume that the adversary does not know the secret key k of $acc : (\mathsf{pk}(k), \mathsf{asset}_{\mathsf{pk}}(*, v)) \in$ `Accounts`. The transactions are strongly theft-resistant if

$$Pr\left[\begin{array}{r|r} \mathsf{VerifyTx}(tx) \overset{?}{=} 1 \wedge acc \overset{?}{\in} tx \wedge & tx \leftarrow \mathcal{A}^{O_k(\cdot)}(\Lambda, \\ tx \overset{?}{\notin} \textbf{TX} \wedge \mathsf{OpenAsset}(k, v, acc') \overset{?}{=} 0 & acc, v) \end{array}\right] \leq \epsilon(\lambda)$$

when (1) oracle $O_k(\cdot)$ creates transactions behalf of *acc* for the adversary's queries without revealing k, and (2) **TX** are the transactions created by $O_k(\cdot)$. Here, acc' is the updated account of *acc*.

Theorem 6.3. *Nopenena transactions provide zero-knowledge argument and strong theft-resistance if the rerandomizable accounts are ZKA, insider-ZKA, and theft-resistant, and anonymous forced-opening, balance proofs, range proofs, and contracts are zero-knowledge.*

Proof: Proving the security of Nopenena transactions is straightforward since they directly integrate rerandomizable accounts, anonymous forced openings, balance proofs, zero-knowledge contracts, and zero-knowledge range proofs as shown in Definition 6.1. We conclude that Nopenena payments provide ZKA for relation $\mathcal{R}_{Nopenena}$ if subprotocols hold ZKA for relations: $\mathcal{R}_{accounts}$, $\mathcal{R}_{accounts}^{insider}$, $\mathcal{R}_{forced}$, $\mathcal{R}_{range}$, $\mathcal{R}_{balance}$, and $\mathcal{R}_{contracts}$. Also, we directly claim that Nopenena payments are theft resistant if rerandomized accounts are theft-resistant. Therefore, we conclude that Theorem 6.3 is true.

7 NOPENENA FOR ANONYMOUS CONTRACTS

This section explains the workflow of the anonymous contracts using an exemplary escrow and split transactions, i.e., splitting a payment into sending and receiving transactions. Also, we emphasize that Nopenena payments can be used with *any* Pedersen commitment-based zero-knowledge contracts like Zethers' Σ-Bullets [9] or CIP.

7.1 Escrow for Anonymous Shopping

Assume that Alice buys pizza from Bob. However, after Alice sends the money, Bob may not send the pizza or may delay the pizza until it's cold. To resolve disputes like this in online shopping, many hire a trusted third party, e.g., UberEats, eBay or Amazon. Similarly, escrows are essential for decentralized payments, which are usually constructed as a smart contract where the escrow takes a smaller commission to resolve conflicts. First, we explain the unblinded contract $\mathcal{F}$ of the escrow contract below.

```
contract escrow(inputs: c0, c1):
    variable e; # prenagotiated escrow fee
    variable cS; # seller's coin amount
    variable cB; # buyer's refund
    if e == c1: # correct escrow fee
        if  cS == c0: # seller gets coins
            return 1
        if  cB == c0: # buyer gets a refund
            return 1
    return 0
```

Listing 1: Unblinded escrow contract

However, at this stage, this contract is insecure because any transaction with these coin values can receive the coins. To make it theft-resistant and, moreover, confidential, we compile this contract to $\mathcal{F}_{zk}$ where each variable is committed into a Pedersen commitment, and the blinding keys of the commitment are only known to the expected owners of the coins. In that way, unless all needed parties agree, i.e., escrow with the seller or the buyer, the zero-knowledge proof for the contract cannot be computed. This compiling works as follows:

- The escrow creates a commitment E for e with key α_e.
- The seller computes a commitment CS for cS with key α_{cS}.
- The buyer computes CB for cB with key α_{cB}.
- The buyer collects CS and E and compiles the contract into the following zero-knowledge function $\mathcal{F}_{zk}$.

```
contract escrow(inputs: C0, C1):
    commitment E,  CS, CB;
    if zkEqual(E, C1):
        if  zkEqual(CS, C0): return 1
        if  zkEqual(CB, C0): return 1
    return 0
```

Listing 2: Open-logic zero-knowledge escrow $\mathcal{F}_{zk}$

Then, the buyer sends `c = cB + e=cS + e` coins to the withheld with $\mathcal{F}_{zk}$. Once the escrow decides to forward or refund the coins, the escrow creates a transaction to get the withheld coins back with the buyer or the seller. Assume that the escrow sends coins to the seller. Then, the escrow and the sender create a transaction to obtain these withheld coins back to some accounts of zero coins. They first open commitments `[C0, C1]` $= (C_0, C_1) \in W$ of new account balances anonymously such that escrows' commitment is C_1 (see Step 4 of **ForceOutCreate**). Then, they generate the proof as follows:

(1) The escrow computes a single-party CIP equal proof $\pi_{eqE} = \mathsf{EqProve}(\mathsf{E}, C_1, \alpha_e, \alpha_1)$ for `zkEqual(E, C1)`.
(2) The seller computes a single-party CIP equal proof $\pi_{eqE} = \mathsf{EqProve}(\mathsf{CS}, C_0, \alpha_{cS}, \alpha_0)$ for `zkEqual(CS, C0)`.
(3) The contract proof $\pi_{contract} := (\pi_{eqE}, \pi_{eqS})$

This contract and its proof satisfy the following relation:

$$(pp, (\mathsf{E}, \mathsf{CS}, \mathsf{CB}, \pi_{eqE}, \pi_{eqS}), (\alpha_e, \alpha_{cS}, \alpha_{cB}, \mathsf{cB}, \mathsf{cS}, \mathsf{e}, \mathsf{c}))\mathcal{R}_{escrow} \Leftrightarrow$$
$$\begin{bmatrix} \mathsf{E} \times C_1^{-1} \overset{?}{=} \mathsf{Commit}_{h,\mu}(\alpha_e - \alpha_1, 0) \\ \left(C_0^{-1}\mathsf{CS} \overset{?}{=} \mathsf{Commit}_{h,\mu}(\alpha_{cS} - \alpha_0, 0) \vee C_0^{-1}\mathsf{CB} \overset{?}{=} \mathsf{Commit}_{h,\mu}(\alpha_{cB} - \alpha_0, 0)\right) \end{bmatrix}$$

The validators check if $\mathcal{F}_{zk}([C_0, C_1], \pi_{contract})$ outputs 1 by verifying `zkEqual(E, C1)` from $\mathsf{EqVerify}(\mathsf{E}, C_1, \pi_{eqE}) \overset{?}{=} 1$ and verifying `zkEqual(CS, C0)` from $\mathsf{EqVerify}(\mathsf{CS}, C_0, \pi_{eqS}) \overset{?}{=} 1$ (see Appendix A).

Similarly, the escrow can refund the buyer by proving `zkEqual(CB, C0)`. However, only the seller or the buyer can obtain cS coins or cB coins since one of $(\alpha_{cS}, \alpha_{cB})$ is required. Therefore, the contract is secure even if the escrow is opportunistically malicious, i.e., given an opportunity steals coins.

This contract is open-logic, i.e., the logic of the contract is visible to verifiers. Although, the logic can be obfuscated by adding other random steps to the contract, these steps typically add a computational cost to the contract.

Note: The decision criteria of escrow, i.e., how escrow decides to whom to transfer coins, is out of scope of this paper, An interested reader may refer to [1, 18] for more details.

7.2 Split Payments

Nopenena payments described in Section 6 are sender-receiver transactions since not only the sender, but also the receiver actively participates in creating the transaction. However, Nopenena also facilitates sender-only transactions where the sender transfers coins to a commitment in `Withheld` and shares the blinding key of the commitment with the receiver. Later, the receiver claims these coins to his/her account by submitting a receiver-only transaction. Therefore, the receiver does not have to actively participate in the sender's transaction. We call these payments, *split payments*.

Split transactions work as follows: First, the sender creates a transaction $tx_{sending}$ to withhold some c coins in commitment $C{=}\mathsf{Commit}(\alpha_c, c)$ with a blinding key α_c as explained in **CreateTx**. Also, these coins are attached to a contract that always outputs 1:

```
contract receiver(inputs: C0): return 1
```

Then, the sender submits this sender-only transaction $tx_{sending}$ to the ledger, and the sender shares α_c with the receiver. Upon receiving α_c, the receiver creates a receiver-only transaction $tx_{receiving}$ using α_c to receive the withheld coins in C. These two transactions $(tx_{sending}, tx_{receiving})$ can be sent at the *same time* by putting them together, or the receiver can claim coins later. Anyway, the payment concludes when the receiver transfers the withheld c coins to his/her account. Here, security comes from the blinding key α_c since only the sender and the receiver know α_c.

Spilt payments moreover benefit from faster transaction verification. As explained in Section 5.2, the verification time of forced opening is $\tilde{\mathcal{O}}(M)$ when $M{=}\frac{N!}{(n)!(N-n)!}$. We get the smallest M for any $N \in \mathbb{N}$ when $n = 1$, i.e., $M = N$. Hence, even verifying two split transactions are faster than verifying the unsplit sender-receiver transaction when $N > 8$ (see Figure 5). Moreover, this splitting could be cost-effective if the expected transaction fees are linear to the computation cost like Gas in Ethereum [58].

8 IMPLEMENTATION

We implement Nopenena as a C library from Libsecp256k1 C library [22] such that elements of $\mathbb{G}$ and $\mathbb{Z}_q$ are 33 bytes and 32 bytes, respectively. We use SHA256 for the `Hash` function and aggregate Bulletproofs [10] (used Mimblewimble Bulletproof construction [22]) for range proofs when $L = 64$, e.g., balances in $[0, 2^{64})$ are valid.

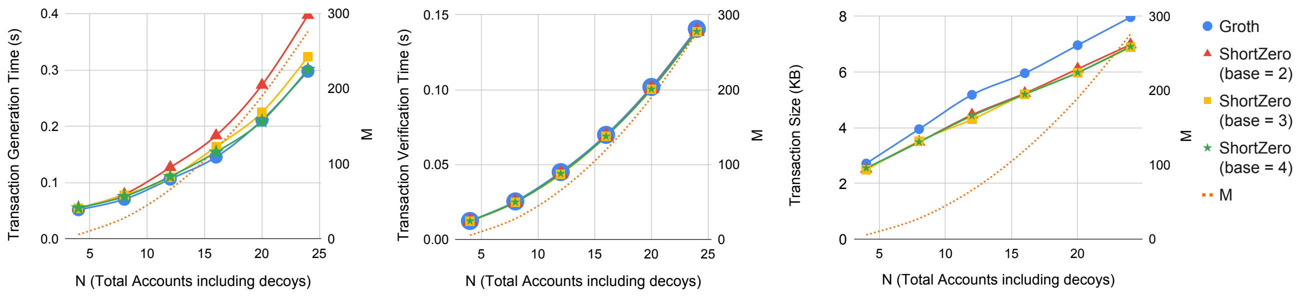


Figure 3: Transaction size and verification time vs. N. Here, “ShortZero” is used for [7] and base (d) is selected as $M = \lceil \text{base}^m \rceil$ for some m. M values are shown on the right vertical axis.

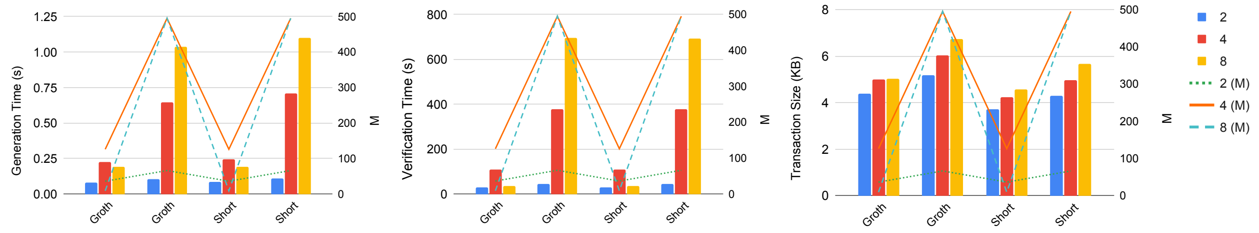


Figure 4: Transaction size and verification time vs. $n \in [2, 4, 8]$. Here, “Short” is used for [7] when the base is $d = 3$. M values (lines) for $n \in [2, 4, 8]$ are shown on the right vertical axis.

Also, due to the aggregation of Bulletproof range proofs, a range proof is 739 bytes and 803 bytes for 2 and 4 outputs, respectively.

We provide two constructions of Nopenena payments with two different one-of-many proofs for zero-value commitments: (1) Groth-Kohlweis protocol [24] and (2) short one-of-many protocol [7]. We implemented these two one-of-many protocols from Libsecp256k1 C library. [24] and [7] generate proofs of $\mathcal{O}(\log_2 m)$ and $\mathcal{O}(\log_d m)$, respectively, when $M = \lceil 2^m \rceil$ [24] or $M = \lceil d^m \rceil$ [7] for some $d \in \mathbb{N}$ and $m \in \mathbb{N}$. Also, their verification times are $\mathcal{O}(M)$. Our Nopenena library makes it possible to customize these d and m. Moreover, we implemented functionalities for the proposed escrow contract.

9 PERFORMANCE ANALYSIS

We analyze the following questions in this section.

(1) What is the correlation between (N, n) and transaction sizes, generation times, and verification times?
(2) What is the performance difference between transactions with and without escrow contracts?
(3) What is the performance difference between transactions with and without splitting?

Workload and Benchmarks We implement benchmarks to measure transaction sizes and verification times. Also, we implement micro-benchmarks for escrow contracts and forced openings. All the presented verification times are measured on a single-thread of 12th generation i7-1260P x 16 unless otherwise mentioned. We selected N to be in $[2, 24]$, which was inspired by Monero of $N = 16$ since our target is to build small decoy set payments that are resistant to the DM-decomposition. Also, n is chosen from $[1, 8]$ since statistics show that only 2-3 accounts participate in a payment [5].
Nopenena Transactions vs. (N, n) We measure transaction sizes, verification times, and generation times for various (N, n) as shown in Figure 3 and Figure 4.

We observe that Nopenena transactions’ generation time and verification time is proportional to $M = \frac{N!}{(n)!(N-n)!}$ when n is constant, and N increases from 4 to 24. That is due to the computation of $[H_m]_{m=0}^{M}$ in Step 11 of ForceOut. However, as shown in Figure 3, the transaction sizes are proportional to N, not to M because the forced openings’ size complexities are logarithmic in M, i.e., $\mathcal{O}(\log_2 M)$ [24] or $\mathcal{O}(\log_d M)$ [7].

Similarly, when $n \in [2, 4, 8]$ as shown in Figure 4, we observe that transaction verification and generation times are proportional to M instead of n. Moreover, we identify that the transaction size increases with n. Till, the size increment is minimal (see Figure 4) since (1) outputs’ commitments are only 32 bytes, (2) outputs’ range proofs are computed from aggregate Bulletproofs, which are also logarithmic-sized, and (3) forced-openings are logarithmic-sized.
Nopenena Transactions with Escrow Contracts We measure transactions with and without escrow contracts. We measured the escrow transactions that (2) withhold coins with a new compiled escrow contract and (1) obtain withheld coins from proving that the transaction satisfies an existing escrow contract. We observe that additional time required for escrows and withholding is 3.34 ms, and 293 bytes are required for escrow proofs (130 bytes), withheld coins (33 bytes for each), and the compiled contract.
Performance of Nopenena Split Payments As discussed in Section 7.2, a payment can be split into two transactions for sending and receiving. We analyze the sizes and verification times for normal payments ($n = 2$) and split payments submitted together (two transactions of $n = 1$) for $N \in [4, 24]$. Our results are shown in Figure 5. According to the results, split transactions’ verification time is proportional to N since $M = N$, and are more efficient when $N > 8$. However, transaction splitting increases the size because two lists of new rerandomized assets must be provided.

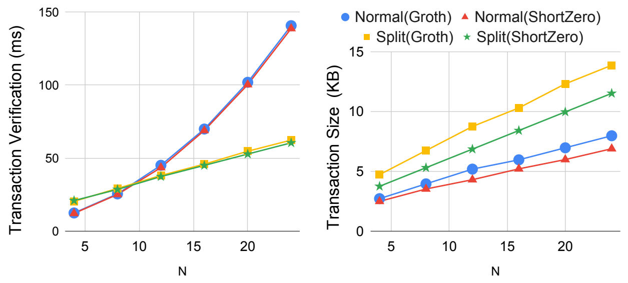


Figure 5: Total transaction size and verification time vs. splitting. Here, a split transaction is an aggregation of two $n = 1$ transactions, and "ShortZero" is used for [7] with base of $d = 3$.

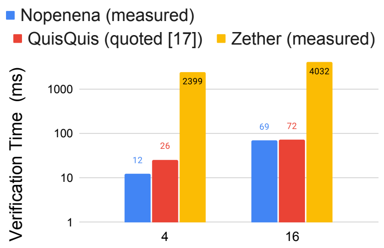


Figure 6: Verification times vs. N in account-based untraceable payments for $n = 2$. Here, we tested Zether [15] in our local machine (12th Gen i7-1260P x 16) and measured the time for "verifyTransfer". As for QuisQuis, we quoted data included in QuisQuis [19] since the implementation is not public. We emphasize that Zether's verification times may be inflated since it is written in JavaScript/Solidity.

10 RELATED WORK

Untraceable payments are either decoy-based or tumblers. While decoy-based payments use decoys to obfuscate the real sender/receiver, tumblers mix and aggregate irrelevant payments into one to hide which output/account is used in which payment. These tumblers are either centralized, [4, 6, 27] or decentralized [12, 14, 41, 50, 51, 56]. Decoy-based payments do not need a (semi-)trusted mixer like in centralized tumblers or do not have high latency in transaction generation similar to decentralized tumblers, since decoys do not participate in transaction generation actively.

Full decoy set untraceable payments obtain the maximum anonymity that the ledger can offer. However, they either suffer from large-sized transactions or need to rely on trusted setups. For example, Lelantus [31, 32] that modified the idea of Zerocoin [42] with Maxwell's CT [40] and logarithmic balance proofs [42] still produces large-sized transactions due to the large number of assets, that could be millions in a stable ledger. Some full decoy transactions like Zerocash [49, 52], ZCash [29], and BlockMaze [25] reduced the transaction size using zk-SNARK (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge) [48] but require trusted ceremonies to generate the public parameters and rely on relatively new knowledge of exponent assumption. More importantly, full decoy set payments suffer from *availability issues* since transactions frequently expire when other transactions update the ledger, e.g., ZCash has enforced epochs of 50 mins. We gear up Nopenena with unexpiring transactions for high availability even though expiring is not common in Nopenena (even in QuisQuis) compared to full decoy set payments like ZCash.

Ring CTs [46, 54, 60] originated with CryptoNote [57] and became popular with Monero [46]. Later, Ring CT v.2 [54] and Ring CT v.3 [60] introduced more efficient protocols, e.g., Ring CT v.3 provides $\sim 98\%$ size reduction for large-sized rings. They allow users to select a smaller decoy set and do not expire like full decoy set payments. However, they suffer from ever-growing output problem and DM-decomposition [13, 17], leading to degrading untraceability.

Zether [9] introduced the account-based untraceable payment modules to solve the ever-growing UTXO problem. Zether offers contracts built from Σ-bullets of Inner-product argument [8]. Anonymous zether [15] and PriDe CT [26] improve Zether's idea of untraceability with improved cryptographic protocols. However, Zether and its variants use epoch-based one-time keys to prevent front-running and replay attacks on contracts. These one-time keys trigger DM decomposition for each epoch. By reducing the epoch size, the impact of the attack can be reduced, but it increases the probability of transaction expiration.

QuisQuis proposed the first untraceable payments that prevent the graph analysis with *smaller* decoy sets. However, the senders of QuisQuis transactions *must shuffle* the transactions' accounts to verify the non-negativity of account balances while protecting the untraceability. This shuffling algorithm creates a time-consuming account searching problem for decoys and receivers. Moreover, QuisQuis transactions are extremely inefficient due to this shuffling algorithm. Instead of a shuffling-based solution, Nopenena uses a novel cryptographic primitive, anonymous forced openings, to obtain untraceability.

Regulated currencies. Apart from these cryptocurrencies, Central Bank Digital Currencies (CBDC) [33, 36, 55, 59] and auditable currencies [11, 35, 37] also use some untraceability techniques. However, they do not provide untraceability from *all* validators due to unavoidable regulations like revealing the real sender to the regulators or the senders' banks [33, 59].

We compare related work in Table 1. Also, existing account-based untraceable payments are compared in Figure 1 and Figure 6, according to the measured data of [15] and reported data[3] of [19]. We observe that Nopenena provides shorter and faster transactions compared to others as shown in Figure 1 and Figure 6.

11 CONCLUSION

This paper presented Nopenena, a new untraceable decentralized payment protocol with a non-monotone-sized ledger. Nopenena proposed novel update proofs and a forced opening protocol to obtain theft resistance and untraceability, i.e., hiding the sender and receiver of a transaction among a set of decoy accounts. This paper presented a formal definition of Nopenena along with a concrete instantiation that is proven secure under the Discrete logarithm and the Decisional Diffie-Helman assumptions. Finally, Nopenena was implemented and showed efficiency compared to the state of the art. For example, Nopenena has reduced the payment size by 80% compared to the previous small-decoy set payments with graph-analysis resistance.

[3]QuisQuis C library is not public and implementing it is out of the scope.

REFERENCES

[1] Ahamed Ali et al. 2022. Decentralised Escrow Protocol that Facilitates Secure Transactions between Trustless Parties. In *Proceedings of the International Conference on Innovative Computing & Communication (ICICC)*.

[2] Jayamine Alupotha and Xavier Boyen. 2022. Practical UC-Secure Zero-Knowledge Smart Contracts. *IACR Cryptol. ePrint Arch.* 2022 (2022), 670. https://api.semanticscholar.org/CorpusID:249751945

[3] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. 2013. Evaluating user privacy in bitcoin. In *International Conference on Financial Cryptography and Data Security*. Springer, 34–51.

[4] George Bissias, A Pinar Ozisik, Brian N Levine, and Marc Liberatore. 2014. Sybil-resistant mixing for bitcoin. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. 149–158.

[5] blockchain.com. 2023. Blockchain Charts. (2023). https://www.blockchain.com/explorer/charts.

[6] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A. Kroll, and Edward W. Felten. 2014. Mixcoin: Anonymity for Bitcoin with Accountable Mixes. In *Financial Cryptography and Data Security*, Nicolas Christin and Reihaneh Safavi-Naini (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 486–504.

[7] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, Jens Groth, and Christophe Petit. 2015. Short accountable ring signatures based on DDH. In *European Symposium on Research in Computer Security*. Springer, Springer, 243–265.

[8] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. 2016. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. Springer, 327–357.

[9] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. 2020. Zether: Towards Privacy in a Smart Contract World. In *Financial Cryptography and Data Security*, Joseph Bonneau and Nadia Heninger (Eds.). Springer International Publishing, Cham, 423–443.

[10] Benedikt Bunz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More. In *IEEE Symposium on Security and Privacy*. 315–334. https://doi.org/10.1109/SP.2018.00020

[11] Panagiotis Chatzigiannis and Foteini Baldimtsi. 2021. Miniledger: compact-sized anonymous and auditable distributed payments. In *European Symposium on Research in Computer Security*. Springer, 407–429.

[12] Alexander Chepurnoy and Amitabh Saxena. 2020. Zerojoin: Combining zerocoin and coinjoin. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology: ESORICS 2020 International Workshops, DPM 2020 and CBT 2020, Guildford, UK, September 17–18, 2020, Revised Selected Papers 15*. Springer, 421–436.

[13] Sherman SM Chow, Christoph Egger, Russell WF Lai, Viktoria Ronge, and Ivy KY Woo. 2023. On sustainable ring-based anonymous systems. *Cryptology ePrint Archive* (2023).

[14] Dominic Deuber and Dominique Schröder. 2021. CoinJoin in the Wild: An Empirical Analysis in Dash. In *Computer Security–ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part II 26*. Springer, 461–480.

[15] Benjamin E Diamond. 2021. Many-out-of-many proofs and applications to anonymous zether. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1800–1817.

[16] Andrew L Dulmage and Nathan S Mendelsohn. 1958. Coverings of bipartite graphs. *Canadian Journal of Mathematics* 10 (1958), 517–534.

[17] Christoph Egger, Russell WF Lai, Viktoria Ronge, Ivy KY Woo, and Hoover HF Yin. 2022. On Defeating Graph Analysis of Anonymous Transactions. *Proceedings on Privacy Enhancing Technologies* 3 (2022), 538–557.

[18] Anjaneyulu Endurthi and Akhil Khare. 2021. Cheat Proof Escrow System for Blockchain. In *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, 294–298.

[19] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. 2019. Quisquis: A new design for anonymous cryptocurrencies. In *Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part I 25*. Springer, 649–678.

[20] Michael Fleder, Michael S Kester, and Sudeep Pillai. 2015. Bitcoin transaction graph analysis. *arXiv preprint arXiv:1502.01657* (2015).

[21] Oliver Giel and Ingo Wegener. 2003. Evolutionary algorithms and the maximum matching problem. In *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 415–426.

[22] Grin. 2017. Fork of secp256k1-zkp for the Grin/MimbleWimble project. (2017). https://github.com/mimblewimble/secp256k1-zkp.

[23] Jens Groth and Yuval Ishai. 2008. Sub-linear zero-knowledge argument for correctness of a shuffle. In *Advances in Cryptology–EUROCRYPT 2008: 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings 27*. Springer, 379–396.

[24] Jens Groth and Markulf Kohlweiss. 2015. One-out-of-many proofs: Or how to leak a secret and spend a coin. In *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*. Springer, 253–280.

[25] Zhangshuang Guan, Zhiguo Wan, Yang Yang, Yan Zhou, and Butian Huang. 2020. BlockMaze: An efficient privacy-preserving account-model blockchain based on zk-SNARKs. *IEEE Transactions on Dependable and Secure Computing* 19, 3 (2020), 1446–1463.

[26] Yue Guo, Harish Karthikeyan, Antigoni Polychroniadou, and Chaddy Huussin. 2023. PriDe CT: Towards Public Consensus, Private Transactions, and Forward Secrecy in Decentralized Payments. *Cryptology ePrint Archive* (2023).

[27] Ethan Heilman, Leen AlShenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. 2017. TumbleBit: an untrusted Bitcoin-compatible anonymous payment hub. (2017). https://open.bu.edu/handle/2144/29224

[28] Jordi Herrera-Joancomartí. 2014. Research and challenges on bitcoin anonymity. In *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*. Springer, 3–16.

[29] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2016. Zcash protocol specification. *Tech. rep. 2016–1.10. Zerocoin Electric Coin Company, Tech. Rep.* (2016).

[30] Tom Elvis Jedusor. 2016. Mimblewimble. (2016). https://docs.beam.mw/Mimblewimble.pdf.

[31] Aram Jivanyan. 2019. Lelantus: A new design for anonymous and confidential cryptocurrencies. *Cryptology ePrint Archive* (2019).

[32] Aram Jivanyan and Aaron Feickert. 2022. Lelantus Spark: Secure and flexible private transactions. In *International Conference on Financial Cryptography and Data Security*. Springer, 409–447.

[33] Aggelos Kiayias, Markulf Kohlweiss, and Amirreza Sarencheh. 2022. Peredi: Privacy-enhanced, regulated and distributed central bank digital currencies. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 1739–1752.

[34] Russell WF Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. 2019. Omniring: Scaling private payments without trusted setup. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 31–48.

[35] Chao Lin, Debiao He, Xinyi Huang, Muhammad Khurram Khan, and Kim-Kwang Raymond Choo. 2020. DCAP: A secure and efficient decentralized conditional anonymous payment system based on blockchain. *IEEE Transactions on Information Forensics and Security* 15 (2020), 2440–2452.

[36] Chao Lin, Debiao He, Xinyi Huang, Xiang Xie, and Kim-Kwang Raymond Choo. 2020. Ppchain: A privacy-preserving permissioned blockchain architecture for cryptocurrency and other regulated applications. *IEEE Systems Journal* 15, 3 (2020), 4367–4378.

[37] Chao Lin, Xinyi Huang, Jianting Ning, and Debiao He. 2022. Aca: Anonymous, confidential and auditable transaction systems for blockchain. *IEEE Transactions on Dependable and Secure Computing* (2022).

[38] Lindell. 2003. Parallel coin-tossing and constant-round secure two-party computation. *Journal of Cryptology* 16 (2003), 143–184.

[39] Varun Madathil and Alessandra Scafuro. 2023. PriFHEte: Achieving Full-Privacy in Account-based Cryptocurrencies is Possible. *Cryptology ePrint Archive* (2023).

[40] Greg Maxwell. 2015. Confidential transactions. (2015). https://people.xiph.org/~{}greg/confidential_values.txt(Accessed09/01/2021).

[41] Sarah Meiklejohn and Rebekah Mercer. 2018. Möbius: Trustless tumbling for transaction privacy. (2018).

[42] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. 2013. Zerocoin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 397–411.

[43] Liam Morris. 2015. Anonymity analysis of cryptocurrencies. Rochester Institute of Technology.

[44] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[45] Shen Noether, Adam Mackenzie, et al. 2016. Ring Confidential Transactions. *Ledger* 1 (2016), 1–18.

[46] Shen Noether and Sarang Noether. 2014. Monero is not that mysterious. *Technical report* (2014). Online available at: https://web.getmonero.org/ru/resources/research-lab/pubs/MRL-0003.pdf.

[47] Fergal Reid and Martin Harrigan. 2013. An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks*. Springer, 197–223.

[48] Christian Reitwiessner. 2016. zkSNARKs in a nutshell. *Ethereum blog* 6 (2016), 1–15.

[49] Antoine Rondelet and Michal Zajac. 2019. Zeth: On integrating zerocash on ethereum. *arXiv preprint arXiv:1904.00905* (2019).

[50] Tim Ruffing and Pedro Moreno-Sanchez. 2017. Valueshuffle: Mixing confidential transactions for comprehensive transaction privacy in bitcoin. In *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*. Springer, 133–154.

[51] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. 2014. Coinshuffle: Practical decentralized coin mixing for bitcoin. In *Computer Security-ESORICS 2014: 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II 19*. Springer, 345–364.

[52] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 459–474.

[53] Michele Spagnuolo, Federico Maggi, and Stefano Zanero. 2014. Bitiodine: Extracting intelligence from the bitcoin network. In *International Conference on Financial Cryptography and Data Security*. Springer, 457–468.

[54] Shi-Feng Sun, Man Ho Au, Joseph K Liu, and Tsz Hon Yuen. 2017. RingCT 2.0: A Compact Accumulator-based (Linkable Ring Signature) protocol for blockchain cryptocurrency Monero. In *European Symposium on Research in Computer Security*. Springer, 456–474.

[55] Alin Tomescu, Adithya Bhat, Benny Applebaum, Ittai Abraham, Guy Gueta, Benny Pinkas, and Avishay Yanai. 2022. Utt: Decentralized ecash with accountable privacy. *Cryptology ePrint Archive* (2022).

[56] Luke Valenta and Brendan Rowan. 2015. Blindcoin: Blinded, accountable mixes for bitcoin. In *Financial Cryptography and Data Security: FC 2015 International Workshops, BITCOIN, WAHC, and Wearable, San Juan, Puerto Rico, January 30, 2015, Revised Selected Papers*. Springer, 112–126.

[57] Nicolas Van Saberhagen. 2013. CryptoNote v 2.0. (2013).

[58] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151 (2014), 1–32.

[59] Karl Wüst, Kari Kostiainen, Noah Delius, and Srdjan Capkun. 2022. Platypus: A central bank digital currency with unlinkable transactions and privacy-preserving regulation. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2947–2960.

[60] Tsz Hon Yuen, Shi-feng Sun, Joseph K Liu, Man Ho Au, Muhammed F Esgin, Qingzhao Zhang, and Dawu Gu. 2020. RingCT 3.0 for blockchain confidential transaction: Shorter size and stronger security. In *International Conference on Financial Cryptography and Data Security*. Springer, 464–483.

A CONTRACT PROOFS FOR EQUALITY

Let there be $C_1 = g^{k_1}\mu^v \in \mathbb{G}$ and $C_2 = g^{k_2}\mu^v \in \mathbb{G}$. We want to prove that they commit to the same value without revealing v. Let $(\mathsf{EqProve}, \mathsf{EqVerify})$ be the equal proof protocol that works as follows:

$$\underline{\mathsf{EqProve}(C_1, C_2, k_1, k_2):}$$

$$y \xleftarrow{\$} \mathbb{Z}_q; R := g^y \in \mathbb{G}, x := \mathsf{Hash}(C_1, C_2, R)$$
$$s = y + (k_1 - k_2)x \in \mathbb{Z}_q \text{ return } \pi_{eq} := (s, R)$$

$$\underline{\mathsf{EqVerify}(C_1, C_2, \pi_{eq} : (s, R)):}$$

$$\text{return } g^s \overset{?}{=} R(C_1 C_2^{-1})^{\mathsf{Hash}(C_1, C_2, R)} \in \mathbb{G}$$

Theorem A.1. $(\mathsf{EqProve}, \mathsf{EqVerify})$ *provide ZKA for a statement "hidden values are equal" when the DL problem is hard [2].*

B DEFINITIONS IMPLIED BY ZERO-KNOWLEDGE RELATIONS

Definition B.1 (Indistinguishability of Rerandomized Accounts). The asset rerandomization is indistinguishable if

$$Pr\left[b' \overset{?}{=} b \middle| \begin{array}{r} (k_0, k_1, r_0, r_1, \kappa_0, \kappa_1, r') \xleftarrow{\$} \mathbb{Z}_q, v' \xleftarrow{\$} [0, 2^L), \mathcal{A} \longrightarrow (v_0, v_1, W) \\ acc_0 := (g^{k_0}, g^{r_0}, g^{k_0 r_0}\mu^{v_0}), acc_1 := (g^{k_1}, g^{r_1}, g^{k_1 r_1}\mu^{v_1}) \\ b \xleftarrow{\$} [0,1], \mathsf{asset}_b := \mathsf{UpdateAsset}(r', v_b, v', acc_b) \\ \hat{b} := ([0,1] \setminus b) \in [0,1], \mathsf{asset}_{\hat{b}} := \mathsf{UpdateAsset}(r', acc_{\hat{b}}) \\ \sigma_{\hat{b}} := \mathsf{UpdateProve}(r', acc_{\hat{b}}, \mathsf{asset}_{\hat{b}}; \kappa_{\hat{b}}, W) \\ \sigma_b := \mathsf{UpdateValueProve}(r', k_b, v_b, v', acc_b, \mathsf{asset}_b; \kappa_b, W) \\ \mathcal{A}(acc_0, acc_1, \mathsf{asset}_0, \mathsf{asset}_1, \sigma_0, \sigma_1) \longrightarrow b' \end{array} \right]$$

Definition B.2 (Sender-Anonymity of Rerandomized Accounts). The asset rerandomization is sender-anonymous if

$$Pr\left[b' \overset{?}{=} b \middle| \begin{array}{r} (k_0, k_1, r_0, r_1, \kappa_0, \kappa_1, r') \xleftarrow{\$} \mathbb{Z}_q, v' \xleftarrow{\$} [0, 2^L), \mathcal{A} \longrightarrow (v_0, v_1, W) \\ acc_0 := (g^{k_0}, g^{r_0}, g^{k_0 r_0}\mu^{v_0}), acc_1 := (g^{k_1}, g^{r_1}, g^{k_1 r_1}\mu^{v_1}) \\ b \xleftarrow{\$} [0,1], \mathsf{asset}_b := \mathsf{UpdateAsset}(r', v_b, v', acc_b) \\ \hat{b} := ([0,1] \setminus b) \in [0,1], \mathsf{asset}_{\hat{b}} := \mathsf{UpdateAsset}(r', acc_{\hat{b}}) \\ \sigma_{\hat{b}} := \mathsf{UpdateProve}(r', acc_{\hat{b}}, \mathsf{asset}_{\hat{b}}; \kappa_{\hat{b}}, W) \\ \sigma_b := \mathsf{UpdateValueProve}(r', k_b, v_b, v', acc_b, \mathsf{asset}_b; \kappa_b, W) \\ \mathcal{A}^{O_{\mathsf{UpdateValueMProve}(r',*)}}([acc_i, \mathsf{asset}_i, \sigma_i]_{i-0}^{1}) \longrightarrow b' \end{array} \right]$$

when $O_{\mathsf{UpdateValueMProve}(r',*)}$ is an oracle which runs the multi-party $\mathsf{UpdateValueMProve}$ with the adversary (simulates a receiver) for the same r'.

Untraceability means that the verifiers cannot identify which accounts are actual senders/receivers. We define untraceability such that the adversary (simulates verifiers) selects two sets of sending/receiving accounts' indexes, $[j_{0,l}]_{l=0}^n$ and $[j_{1,l}]_{l=0}^n$, yet the adversary cannot win the following game of identifying which set was used for the transaction with more than $\frac{1}{2} + \epsilon(\lambda)$ probability.

Definition B.3 (Untraceability). The payments are untraceable if

$$Pr\left[b \overset{?}{=} b' \middle| \begin{array}{r} n < N, ([j_{0,l}]_{l=0}^n, [j_{1,l}]_{l=0}^n) \leftarrow \mathcal{A}(N, n) \\ \text{s.t.} [j_{0,l}]_{l=0}^n \neq [j_{1,l}]_{l=0}^n; \;\; b \xleftarrow{\$} [0,1] \\ tx_b := \mathsf{CreateTx}\begin{pmatrix} \Lambda, [acc_i]_{i=0}^N, [j_{b,l}, v_l, v'_l, \alpha_l, \\ C_l]_{l=0}^n, [\mathcal{F}_{zk}, C, \pi_{zk}, \alpha_c, c] \\ [\mathcal{F}'_{zk}, C', \alpha_{c'}, c'], f, f'; [k_l]_{l=0}^n \end{pmatrix} \\ b' \leftarrow \mathcal{A}(tx_b) \end{array} \right] \leq \frac{1}{2} + \epsilon(\lambda).$$

Sender anonymity means that receivers cannot identify which account is sending coins. We define the following game such that the adversary (simulates receivers) tries to identify the sending account's index (could be $j_{0,0}$ or $j_{1,0}$). The payment system provides sender anonymity if the adversary cannot win the following game with more than $\frac{1}{2} + \epsilon(\lambda)$ probability.

Definition B.4 (Sender Anonymity). The payments provide sender anonymity if

$$Pr\left[b\overset{?}{=}b' \;\middle|\; \begin{array}{r} n < N, (j_{0,0}, j_{1,0}, [j_l, k_{j_l}]_{l=1}^{n}) \leftarrow \mathcal{A}(N, n) \text{ s.t.} \\ j_{0,l} \neq j_{1,l};\; b \xleftarrow{\$} [0,1]; [j'_l]_{l=0}^{n} := j_{b,0} \cup [j_l]_{l=1}^{n} \\ tx_b := \text{CreateTx}\left(\begin{array}{r} \Lambda, [acc_i]_{i=0}^{N}, [j'_l, v_l, v'_l, \alpha_l, \\ C_l]_{l=0}^{n}, [\mathcal{F}_{zk}, C, \pi_{zk}, \alpha_c, c] \\ [\mathcal{F}'_{zk}, C', \alpha_{c'}, c'], f, f'; [k_l]_{l=0}^{n} \end{array} \right) \\ b' \leftarrow \mathcal{A}(tx_b) \end{array} \right] \leq \frac{1}{2} + \epsilon(\lambda).$$

C SECURITY PROOFS OF NOPENENA ACCOUNTS

We prove Theorem 5.4 in this section.

C.1 Zero-knowledge argument

We unfold the left hand sides of (T_1, T_2, T_3) to show the correctness of update proofs as stated below.

$$T_1 = g^{s_1}((G')^{-1}G)^{x} = g^{t+x(r')}g^{-r'x} = g^{t}$$
$$T_2 = K^{s_1}\mu^{s_2}((V')^{-1}V)^{x}$$
$$= K^{t+x(r')}\mu^{\tau+x(v'-v)}(K^{r'x}\mu^{(v'-v)x})^{-1} = K^{t}\mu^{\tau}$$
$$T_3 = K^{s_2}g^{-s_3} = K^{\tau+x(v'-v)}g^{\kappa-x(v'-v)k}$$
$$= K^{\tau}(g^{k})^{x(v'-v)}g^{\kappa-x(v'-v)k} = K^{\tau}g^{\kappa}$$

C.2 Strong theft-resistance

Lemma C.1. *Nopenena rerandomized accounts are theft-resistant if the DL problem is hard, and Pedersen commitments are binding.*

Proof: We prove theft-resistance by showing that if there is an adversary $\mathcal{A}$ who breaks the theft resistance then we can use $\mathcal{A}$ to break the DL problem in Definition 3.2.

```
1: **input** : $(g, Y)$ from DL challenger (Definition 3.2)
2: **reduction:** $acc = (K = Y, g^r, K^r\mu^{v_0})$ for $r \xleftarrow{\$} \mathbb{Z}_q$
3:     $T_1, T_2, T_3 \leftarrow \mathcal{A}(v_0, acc)$    ▷ get the preinputs
4:     $(i, v', (G', V'), \sigma'=(x, s_1, s_2, s_3)) \leftarrow \mathcal{A}(x \xleftarrow{\$} \mathbb{Z}_q)$    ▷ from $\mathcal{A}$ in Definition 5.3
5:     $(i, v', (G', V'), \sigma''=(x', s'_1, s'_2, s'_3)) \leftarrow \mathcal{A}(x' \xleftarrow{\$} \mathbb{Z}_q)$    ▷ rewinds $\mathcal{A}$ with a fresh verifier randomness
6:     $k := (s'_3 - s_3)/((x - x')(v' - v_0))$ ▷ computes the secret key
7: **output** : $k$ to DL challenger (Definition 3.2)
```

Here, we assume that Pedersen commitments are binding such that the adversary cannot find s such that $K^s = K^r\mu^{v'-v_0} = (V')^{-1}V$ when $v' - v_0$ is not zero. Therefore, we conclude that Lemma C.1 is true, i.e., Nopenena accounts has strong theft resistance if solving the DL problem is hard, and Pedersen commitments are binding.

Lemma C.2 (Witness-extended emulation). *Rerandomizable accounts provide witness-extended emulation for $\mathcal{R}_{account}$ if Pedersen commitments are hiding, and the DDH problem is hard.*

Proof: We assume that there exists an adversary $\mathcal{A}_{WEE}$ who breaks the witness-extended emulation game with more than negligible probability. Then, we reduce $\mathcal{A}_{WEE}$ to break Pedersen commitments' hiding property and the DDH problem as follows.

```
1: **premessage**: $mode \xleftarrow{\$} [\text{PC, DDH}]$
2:   if $mode$='PC': send $(v_0, v_1) \in [0, 2^L)$ to Theorem 3.4's challenger.
3: **input**:
4:   if $mode$='PC': get $(g, \mu, C)$ from the challenger in Theorem 3.4.
5:   if $mode$='DDH': get $(g, X, Y, C)$ from Definition 3.3's challenger.
6: **reduction**: creates a *looks-like* account and proof for a *precomputed* $(k, r, r', t, \tau, \kappa) \xleftarrow{\$} \mathbb{Z}_q$ and $x$ is taken from $\mathcal{V}$
7:   if $mode$ = 'PC':
8:     $acc = (K = g, G = C^{1/k}, V = C) \in \mathbb{G}^3$
9:     $\hat{g} = g^{1/k} \in \mathbb{G}$
10:    $G' := G\hat{g}^{r'}, V' := K^{r'}\mu^{v_0}$
11:    $s_1 = t + x(r'), s_2 = \tau + xv_0, s_3 = -\kappa + xv_0k$    ▷ in $\mathbb{Z}_q$
12:    $T_1 = \hat{g}^tC^{x/k}, T_2 = K^t\mu^\tau C^x, T_3 = K^\tau\hat{g}^\kappa$    ▷ in $\mathbb{G}$
13:    $tr = \big((\hat{g}, \mu), u = (acc, asset' = (G', V'),$
14:                    $\sigma = (x, s_1, s_2, s_3), T_1, T_2, T_3)\big)$
15:    $b \leftarrow \mathcal{A}_{WEE}(tr; \zeta := \phi)$
16:  if $mode$ = 'DDH':
17:    $acc = (K = X, G = \hat{g}^r, V = K^r\mu^{v_0}) \in \mathbb{G}^3$
18:    $\hat{g} = X^{1/k} \in \mathbb{G}$
19:    $G' := GY^{1/k}, V' := VC$
20:    $s_1 = t + x(r'), s_2 = \tau - xv_0, s_3 = -\kappa - xv_0k$    ▷ in $\mathbb{Z}_q$
21:    $T_1 = \hat{g}^tY^{x/k}\hat{g}^{xr'}, T_2 = K^t\mu^\tau C^x, T_3 = K^\tau\hat{g}^\kappa$    ▷ in $\mathbb{G}$
22:    $tr = \big((\hat{g}, \mu), u = (acc, asset' = (G', V')$
23:                    $\sigma = (x, s_1, s_2, s_3), T_1, T_2, T_3)\big)$
24:    $b \leftarrow \mathcal{A}_{WEE}(tr; \zeta := \phi)$
25: **output**:
26:   $mode$ = 'PC': $[0, 1] \setminus b$ to the challenger of Theorem 3.4.
27:   $mode$ = 'DDH': $[0, 1] \setminus b$ to the challenger of Definition 3.3.
```

mode = **'PC'** . Here, if C is a commitment to v_0, then the transcript tr is a genuinely created. Hence, $\mathcal{A}_{WEE}$ accepts tr, i.e., $\mathcal{A}_{WEE}(tr; \zeta) = 1$ with more than $1/2 + \epsilon(\lambda)$ probability. However, if C is a commitment to v_1, then the transcript tr is emulated, i.e, looks-like a valid transcript but actually commits an invalid value. Hence, $\mathcal{A}_{WEE}$ rejects, i.e., $\mathcal{A}_{WEE}(tr; \zeta) = 0$ with more than $1/2 + \epsilon(\lambda)$ probability. Therefore, if $\mathcal{A}_{WEE}$ distinguishes genuine transcripts over emulated transcripts with more than $1/2 + \epsilon(\lambda)$ probability, the hiding game of Definition 3.4 can be solved with more than $1/2 + \epsilon(\lambda)$ probability.

mode = **'DDH'** . From the DDH challenger, we get $C = g^{xy}$ or $C = g^c$ for some unknown (x, y, c) with $X = g^x, Y = g^y$. Let the exponent of Y be the randomness for the new asset. If $C = g^{xy}$, the transcript tr is genuinely created, and $\mathcal{A}_{WEE}$ outputs 1 with more than $1/2 + \epsilon(\lambda)$ probability. If $C = g^c$ for some $c \neq xy$, then the transcript tr is emulated, and $\mathcal{A}_{WEE}$ outputs 0 with more than $1/2 + \epsilon(\lambda)$ probability. Hence, DDH problem is solvable if $\mathcal{A}_{WEE}$ distinguishes genuine transcripts over emulated transcripts.

Therefore, we claim that Lemma C.2 is true.

Lemma C.3 (Knowledge Soundness). *Rerandomizable accounts provide knowledge soundness of $\mathcal{R}_{account}$ if Pedersen commitments are binding.*

Proof: First, we prove that there exists a p.p.t. witness extractor $\mathcal{W}$ given access to a rewindable prover $\mathcal{P}^*_{KS}$, i.e., the prover outputs (s_1, s_2, s_3) for the first round, (s'_1, s'_2, s'_3) and the rewinded round for the same (k, r') and the verifier challenges (x, x'). Then, the extractor can extract $k = (s'_3 - s_3)/(x' - x)(v - v')$. Likewise, the extractor can extract all the witnesses. We assume that there exists a rewindable prover $\mathcal{P}^*_{KS}$ who wins the knowledge soundness game in Definition 3.5 with more than negligible probability for $\mathcal{V}(tr) = 1 \land (pp, u, w) \notin \mathcal{R}_{accounts}$.

Then, we reduce $\mathcal{P}^*_{KS}$ to break Pedersen commitments' binding property as follows.

```
1: **input**:
2:     get $(g, \mu)$ from the challenger in Theorem 3.4.
3: **reduction**:
4:     initial witness $s = r$ and $pp = (g, \mu)$
5:     extract $(tr, w) \leftarrow \mathcal{E}^{\mathcal{W}(\langle \mathcal{P}^*_{KS}(pp,acc,s), \mathcal{V}(pp,acc)\rangle)}$
6:         such that $tr =: (\sigma = (x, s_1, s_2, s_3), G', V', T_1, T_2, T_3)$ :
7:             $\big(acc = (K = g^k, G = g^{rk}, V = G\mu^v) \in \mathbb{G}^3 \land$
8:             $T_1 = g^{s_1}((G')^{-1}G)^x \land T_2 = K^{s_1}\mu^{s_2}((V')^{-1}V)^x \land$
9:             $T_3{=}K^{s_2}g^{-s_3} \land G'{=}Gg^{r'} \land V'{=}VK^{r'}\mu^{v'-v}\big){=}1$   ▷ $\mathcal{V}(tr){=}1$
10:        and $w =: (r', v', t, \tau, \kappa)$:   ▷ since $(pp, u, w) \notin \mathcal{R}_{account}$
11:            $\Big(s_1 \overset{?}{\neq} t + x(r') \lor s_2 \overset{?}{\neq} \tau{+}x(v'{-}v) \lor s_3 \overset{?}{\neq} {-}\kappa{+}x(v'{-}v)k\Big) = 1$
12:    if $s_1 \overset{?}{\neq} t + x(r')$:
13:        $out = \big((r', 0), ((s_1 - t)x^{-1}, 0)\big)$   ▷ for commitment $G'G^{-1}$
14:    if $s_2 \overset{?}{\neq} \tau{+}x(v'{-}v)$:
15:        $out = \big((r'k, v'{-}v), (k(s_1{-}t)x^{-1}, (s_2{-}\tau)x^{-1})\big)$   ▷ $V'V^{-1}$
16:    if $s_3 \overset{?}{\neq} {-}\kappa{+}x(v'{-}v)k$):
17:        $out = \big((k, 0), ((s_3 + \kappa)(x(v' - v))^{-1}, 0)\big)$   ▷ for $K$
18: **output**:
19:     $out$ to the challenger of Theorem 3.4.
```

Here, $\mathcal{W}$ extracts witnesses from rewindable $\mathcal{P}^*_{KS}$ such that relation in Step 11 is 1. We can get at least one-pair of different openings that create the same commitment, e.g., if $s_1 \overset{?}{\neq} t + x(r')$ then $g^{r'} = g^{(s_1 - t)x^{-1}}$ but $r' \neq ((s_1 - t)x^{-1})$. In other words, we can break the binding property of Pedersen commitment if $\mathcal{P}^*_{KS}$ exists.

Therefore, we claim that Lemma C.3 is true.

Lemma C.4. *Nopenena rerandomized accounts hold zero-knowledge argument for $\mathcal{R}_{accounts}$ if the DDH problem is hard, and Pedersen commitments are hiding and binding.*

Proof: We claim that Lemma C.4 is true due to the completeness, Lemma C.2, and Lemma C.3.

C.3 Insider Zero-Knowledge Argument

Lemma C.5. *Rerandomized accounts provide the insider ZKA if the DDH problem is hard, and Pedersen commitments are hiding and binding, and Rerandomized accounts are ZKA for $\mathcal{R}_{accounts}$.*

Proof: Let there be an adversary $\mathcal{A}_{WEE}$ with insider knowledge $\zeta = (J_{\mathcal{A}}{=}[j_\rho]_{\rho=0}^{R_{\mathcal{A}}}, [k_{j_\rho}, v_{j_\rho}, v'_{j_\rho}, \kappa_{j_\rho}]_{\rho=0}^{R_{\mathcal{A}}})$ who wins the witness-extended emulation game in Definition 3.5 for relation $\mathcal{R}^{insider}_{accounts}$. We reduce $\mathcal{A}_{WEE}$ to break the hiding property of Pedersen commitments as follows. Let that $\mathcal{A}_{WEE}$ controls all receivers, except the sender and one decoy, i.e., $R_{\mathcal{A}} + 1 = n$ and $R_{\mathcal{A}} + 2 = N$.

```
1: **premessage**:
2:     get $(v_0, v_1)$ from $\mathcal{A}_{WEE}$
3:     send $(v_0, v_1){\in}[0, 2^L)$ to Theorem 3.4's challenger.
4: **input**:
5:     get $(g, \mu, C)$ from the challenger in Theorem 3.4.
6: **reduction**: creates a *looks-like* account and proof for $(k_0, k_1, \kappa_0,$
   $\kappa_1, r', t_0, t_1, \tau_0, \tau_1) \xleftarrow{\$} \mathbb{Z}_q$ and $(x_0, x_1)$ from the verifier
7:     $W, [acc_i]_{i=0, i\in J_{\mathcal{A}_{WEE}}}^N \leftarrow \mathcal{A}_{WEE}$
8:     For $i \in J_{\mathcal{A}_{WEE}}$:
9:         $asset'_i :=$ UpdateAsset$(r', v'_i, v_i, acc_i)$
10:        $\sigma_i :=$ UpdateValueMProve$(r', k_i, v'_i, v_i, acc_i; \kappa_i, W)$   ▷ a
   multi party protocol between the reducing algorithm (without
   sharing $r'$) and $\mathcal{A}_{WEE}$ with $(k_i, \kappa_i)$
11:    Let $[j_0, j_1]$ be $[0, N) \setminus J_{\mathcal{A}_{WEE}}$
12:    $\hat{g} = g^{1/k_0}$
13:    $acc_{j_0} = (K_0 = g, G_0 = C^{1/k_0}, V_0 = C) \in \mathbb{G}^3$
14:    $asset'_{j_0} = (G'_0 := G_0\hat{g}^{r'}, V'_0 := K_0^{r'}\mu^{v_1}) \in \mathbb{G}^2$
15:    $\sigma_{j_0} = (x_0, t_0 + x_0(r'), \tau_0 + x_0 v_1, -\kappa_0 + x_0 v_1 k_0; T_1{=}\hat{g}^{t_0}C^{x_0/k_0},$
16:            $T_2{=}K_0^{t_0}\mu^{\tau_0}C^{x_0}, T_3{=}K_0^{\tau_0}\hat{g}^{\kappa_0}) \in (\mathbb{Z}_q^4, \mathbb{G}^3)$
17:    $acc_{j_1} = (K_1 = g^{k_1}, G_1 = C^{k_1/k_0}, V_1 = C^{k_1}) \in \mathbb{G}^3$
18:    $asset'_{j_1} = (G'_1 := G_1\hat{g}^{r'}, V'_1 := K_1^{r'}\mu^{v_0}) \in \mathbb{G}^2$
19:    $\sigma_{j_1} = (x_1, t_1{+}x_1(r'), \tau_1{+}x_1 v_0, -\kappa_1{+}x_1 v_0 k_0/k_1; T_1{=}\hat{g}^{t_1}C^{x_1 k_1/k_0},$
20:            $T_2{=}K_1^{t_1}\mu^{\tau_1}C^{x_1 k_1}, T_3{=}K_1^{\tau_1}\hat{g}^{\kappa_1}) \in (\mathbb{Z}_q^4, \mathbb{G}^3)$
21:    $b \leftarrow \mathcal{A}_{WEE}(((g, \mu), [acc_i, asset'_i, \sigma_i]_{i=0}^N); \zeta)$
22: **output**: send $[0, 1] \setminus b$ to Theorem 3.4's challenger.
```

If C has committed v_0 then $asset'_0$ changes the balance from v_0 to v_1, and $\mathcal{A}_{WEE}$ outputs 1 with more than $1/2 + \epsilon(\lambda)$ probability. Similarly, if C has committed v_1 then $asset'_1$ changes the balance from v_1 to v_0, and $\mathcal{A}_{WEE}$ outputs 0 with more than $1/2 + \epsilon(\lambda)$ probability. Therefore, we can win the hiding game of Pedersen commitments if $\mathcal{A}_{WEE}$ exists. If this reduction works for $R_{\mathcal{A}}{+}2 = N$, then it also can be extended for any $0 \le R_{\mathcal{A}} < n < N$ by increasing the number of decoys and sending accounts.

Also, we can see that rerandomized accounts provide knowledge soundness for $\mathcal{R}^{insider}_{accounts}$ if rerandomized accounts has ZKA for $\mathcal{R}_{accounts}$. Thus, rerandomized accounts provides knowledge soundness for $\mathcal{R}^{insider}_{accounts}$ if the DDH problem is hard and Pedersen commitments are hiding and binding as shown in Lemma C.3.

Therefore, we conclude that Lemma C.5 is valid, i.e., Nopenena accounts provide insider ZKA if Pedersen commitments are hiding and binding, the DDH problem is hard, rerandomized accounts are zero-knowledge for $\mathcal{R}_{accounts}$.

D SECURITY PROOFS OF ANONYMOUS FORCED OPENINGS

We prove Theorem 5.6 in this section.

Completeness. We prove the completeness by showing that H_j (Step 11 of **ForcedOutProve**) is a zero-value commitment.

$$
\begin{aligned}
H_j &= \bar{D}\prod_{l=0}^{n}\left((G'_{i_{m,l}})^{z_l}\bar{C}_l\right)^{-x_{i_{m,l}}} \\
&= D\prod_{i=0}^{N}(V'_i)^{s_{3,i}}(A')^{y_{a'}}\prod_{l=0}^{n}\left((G'_{j_l})^{z_l}B_l^{y_{b,l}}C_l^{f_l}A_l^{y_{a,l}}\right)^{-x_{j_l}} \\
&= \prod_{l=0}^{n} h^{y_{a'}a'_l}(G'_{j_l})^{x_{j_l}k_l^2(v'_l-v_l)}\mu^{v'x_{j_l}k_l(v'_l-v_l)} \\
&\times\left((G'_{j_l})^{k_l^2(v'_l-v_l)}C^{k_l(v'_l-v_l)}\right)^{-x_{j_l}} \\
&= \prod_{l=0}^{n} h^{y_{a'}a'_l}\mu^{v'x_{j_l}k_l(v'_l-v_l)}h^{-x_{j_l}\alpha_l k_l(v'_l-v_l)}\mu^{-v'_l x_{j_l}k_l(v'_l-v_l)} \\
&= \mu^0\prod_{l=0}^{n} h^{\sum_{l=0}^{n}\xi_l}
\end{aligned}
$$

Therefore, if one-of-many proofs for zero-value commitments are complete, we conclude that the protocol is complete.

Lemma D.1. *Anonymous forced opening protocol provides witness-extended emulation for relation $\mathcal{R}_{forced}$ if Pedersen commitments are hiding, one-of-many proofs for zero-value commitments holds ZKA for relation $\mathcal{R}_{zero}$, and rerandomized accounts provides (insider-)ZKA for relation $\mathcal{R}^{insider}_{accounts}$.*

Proof: Assume that there exists an adversary $\mathcal{A}_{WEE}$ that wins the witness-extended emulation game in Definition 3.5 for relation $\mathcal{R}_{forced}$, and $\mathcal{A}_{WEE}$ controls $R_{\mathcal{A}_{WEE}}$ accounts such that $R_{\mathcal{A}_{WEE}} < n$ and $n < N$. Therefore, the adversary has access to knowledge $\zeta = (J_{\mathcal{A}_{WEE}}{=}[j_\rho]_{\rho=0}^{R_{\mathcal{A}_{WEE}}}, [k_{j_\rho}, v_{j_\rho}, v'_{j_\rho}, \kappa_{j_\rho}]_{\rho=0}^{R_{\mathcal{A}_{WEE}}}, [v''_l, \alpha_l]_{l=0, j_l\in J_{\mathcal{A}_{WEE}}}^{n})$.

When the ZKA of $\mathcal{R}_{zero}$ and $\mathcal{R}^{insider}_{accounts}$ holds, we the reduce $\mathcal{A}_{WEE}$ to break the Pedersen commitments' hiding property as follows:

```
1:  premessage: send $(\hat{v}_0, \hat{v}_1){\in}[0, 2^L)$ to Theorem 3.4's challenger.
2:  input: get $(h, \mu, C)$ from the challenger in Theorem 3.4.
3:  reduction: create accounts, update proofs, and looks-like/genuine
    forced openings for $([k_i, r_i, v_i, v'_i, \kappa_i]_{i=0}^{N}, [\alpha_l, a_l, a'_l, b_l]_{l=0}^{n}, r') \xleftarrow{\$} \mathbb{Z}_q^*$
4:      $v_{i'} = v'_{i'} = \hat{v}_0$ when $i' \in [0, N)$ and $j_{l'} = i'$ for $l' \in [0, n)$ and
    $i' \notin J_{\mathcal{A}_{WEE}}$        ▷ the sending account's index
5:      get indexes $[j_l]_{l=0}^{n}$ such that $[v'_{j_l}{\neq}v_{j_l}]_{l=0}^{n} \wedge [v'_i = v_i]_{i=0, i\notin[j_l]_{l=0}^{n}}^{N}$
6:      $W, [acc_i]_{i=0, i\in J_{\mathcal{A}_{WEE}}}^{N}, [C_l]_{l=0, j_l\in J_{\mathcal{A}_{WEE}}}^{n} \leftarrow \mathcal{A}_{WEE}$
7:      For $i \in J_{\mathcal{A}_{WEE}}$:
8:          $asset'_i$ := UpdateAsset$(r', v'_i, v_i, acc_i)$
9:          $\sigma_i$ := UpdateValueMProve$(r', k_i, v'_i, v_i, acc_i; \kappa_i, W)$        ▷ a
    multi party protocol between the reducing algorithm (without
    sharing $r'$) and $\mathcal{A}_{WEE}$ without sharing $(k_i, \kappa_i)$
10:     $[acc_i := (K_i = g^{k_i}, G_i = g^{r_i}, V_i = G_i^{k_i}\mu^{v_i})]_{i=0, i\notin J_{\mathcal{A}_{WEE}}}^{N}$
11:     $[asset'_i$ := UpdateAsset$(r', v_i, v'_i, acc_i)]_{i=0}^{N}$
12:     $[\sigma_i$ := UpdateValueProve$(r', k_i, v_i, v'_i, acc_i, asset'_i; \kappa_i, W)]_{i=0, i\notin J_{\mathcal{A}_{WEE}}}^{N}$
13:     $[C_l := h^{\alpha_l}\mu^{v'_{j_l}}]_{l=0, l\neq l', l\notin J_{\mathcal{A}_{WEE}}}^{n}$ and $C_{l'} = C$
14:     get $([y_{a,l}, y_{b,l}]_{l=0}^{n}, y) \in \mathbb{Z}_q^{2n+1}$ from the verifier
15:     for each $l \in [0, n)$ : $A_l := C_l^{a_l}, A'_l := h^{a'_l}, B_l := (G'_{j_l})^{b_l}$
16:             $f_l := k_{j_l}(v'_{j_l} - v_{j_l}) - y_{a,l}a_l \in \mathbb{Z}_q$
17:             $z_l := k_{j_l}(v'_{j_l} - v_{j_l}) - y_{b,l}b_l \in \mathbb{Z}_q$
18:     for $l = l'$: $A_l := C^{-f_l}h^{(\alpha_l k_{j_l}(v'_{j_l}-v_{j_l}))/y_{a,l}}\mu^{(v'_{j_l}k_{j_l}(v'_{j_l}-v_{j_l}))/y_{a,l}}$
19:     $W = ([C_l, \pi_l, A_l, B_l, f_l, z_l]_{l=0}^{n}, D = \prod_{i=0}^{N}(V'_i)^{\kappa_i}, A' = \prod_{l=0}^{n}A'_l)$
20:     $\pi_{forced}$ := ForcedOutProve$([asset'_i, \sigma_i]_{i=0}^{N}, [j_l, v_l, v'_l, \alpha_l], W)$
21:         $tr{=}((g, h, \mu), [acc_i, asset'_i, \sigma_i]_{i=0}^{N}, W, \pi_{forced})$
22:         $b \leftarrow \mathcal{A}_{WEE}(tr; \zeta)$
23: output:
24:     $[0, 1] \setminus b$ to the challenger of Theorem 3.4
```

Here, if C has committed $\hat{v}_0$, tr is a genuine a transcript, and $\mathcal{A}_{WEE}$ outputs 1 with more than $1/2 + \epsilon(\lambda)$ probability. Similarly, $\mathcal{A}_{WEE}$ outputs 0 with more than $1/2{+}\epsilon(\lambda)$ probability if C has committed $\hat{v}_1$ since tr is an emulated transcript created for $\hat{v}_0$. Therefore, we win the game of Pedersen commitments' binding property if $\mathcal{A}_{WEE}$ exists. Hence, we conclude that Lemma D.1 is true.

Lemma D.2. *Anonymous forced opening protocol provides knowledge soundness if rerandomized accounts provide ZKA for $\mathcal{R}_{accounts}$, one-of-many proofs provide ZKA for $\mathcal{R}_{zero}$, and Pedersen commitments are binding.*

Proof: Let there be an extractor $\mathcal{W}$ and rewindable prover $\mathcal{P}^*_{KS}$ which wins the knowledge soundness in Definition 3.5 of for relation $\mathcal{R}_{forced}$. We prove that we can reduce $\mathcal{P}^*_{KS}$ to solve the binding game of Pedersen commitment as follows, when rerandomized accounts and one-of-many proofs provide ZKA for $\mathcal{R}_{accounts}$ and $\mathcal{R}_{zero}$, respectively.

Here, we prove the binding property for three generator commitments, i.e., $C = \text{Commit}_{g,h,\mu}(\tau, \tau', \tau'') = g^{\tau}h^{\tau'}\mu^{\tau''} \in \mathbb{G}$

```
1:  premessage: send $(\hat{v}_0, \hat{v}_1){\in}[0, 2^L)$ to Theorem 3.4's challenger.
2:  input: get $(g, \mu, h)$ from the challenger in Theorem 3.4
3:  reduction: $([k_i, r_i, v_i]_{i=0}^{N} \xleftarrow{\$} \mathbb{Z}_q^{3N}$
4:      public parameters $pp = (g, h, \mu)$
5:      extract $(tr, w) \leftarrow \mathcal{E}^{\mathcal{W}(\langle\mathcal{P}^*_{KS}(pp,u,s), \mathcal{V}(pp,u)\rangle)}$ for
6:          initial witness $s = ([k_i, r_i, v_i]_{i=0}^{N}, r')$
7:          such that $tr = (n, [acc_i, asset'_i, \sigma_i]_{i=0}^{N}, W, \pi_{forced})$ :
8:          and $w =: ([j_l, \alpha_l, a_l, a'_l, b_l]_{l=0}^{n}, [v'_l, \kappa_i]_{i=0}^{N}, j, \xi)$
9:              $(W{=:}([C_l, \pi_l, A_l, B_l, f_l, z_l]_{l=0}^{n}, D, A'))$ when
10:             $\Big([asset'_i$ := UpdateAsset$(r', v_i, v'_i, acc_i)]_{i=0}^{N}\wedge$
11:                 $[\sigma_i$ := UpdateVerify$(acc_i, asset'_i, \sigma; W)]_{i=0}^{N}\wedge$
12:                 $[$OpenAsset$(k_i, v'_i, \text{asset}'_i){=}1]_{l=0}^{n}\wedge$
13:                 ForcedOutVerify$([asset'_i, \sigma_i]_{i=0}^{N}, W, \pi_{forced}) = 1\wedge$
14:                 $[C_l := h^{\alpha_l}\mu^{v''_l}]_{l=0}^{n}\Big) = 1$        ▷ since $\mathcal{V}(tr) = 1$
15:             ▷ since $(pp, ([asset'_i, \sigma_i]_{i=0}^{N}, W, \pi_{forced}), w){\notin}\mathcal{R}_{forced}$ :
16:             $\Big(\bigvee_{l=0}^{n} v''_l \overset{?}{\neq} v'_{j_l} \vee \bigvee_{i=0, i\notin[j_l]_{l=0}^{n}}^{N} v_i \overset{?}{\neq} v'_i\Big){=}1$
17:         $\tau := (\sum_{i=0}^{N} x_i k_i^2(v'_i - v_i)r_i - \sum_{l=0}^{n} x_{j_l}r_{j_l}k_{j_l}(z_l + y_{b,l}b_l)){\in}\mathbb{Z}_q$
18:         $\tau' := \sum_{l=0}^{n} x_{j_l}(\alpha_l f_l + \alpha_l y_{a,l}a_l){\in}\mathbb{Z}_q$
19:         $\tau'' := (\sum_{i=0}^{N} x_i k_i v'_i(v'_i{-}v_i){-}\sum_{l=0}^{n}(x_{j_l}v''_{j_l}f_l{+}x_{j_l}v''_{j_l}y_{a,l}a_l)){\in}\mathbb{Z}_q$
20:         $out = ((0, 0, \xi), (\tau, \tau', \tau''))$
21: output: $out$ to the challenger of Theorem 3.4
```

When $(\bigvee_{l=0}^{n}(v''_l \overset{?}{\neq} v'_{j_l}) \vee \bigvee_{i=0, i\notin[j_l]_{l=0}^{n}}^{N} (v_i \overset{?}{\neq} v'_i))=1$, in Step 11,

$$H_j = g^0 h^{\xi} \mu^0 = D \prod_{i=0}^{N} (V'_i)^{s_{3,i}} (A')^{y_{a'}} \prod_{l=0}^{n} \Big((G'_{j_l})^{z_l} B_l^{y_{b,l}} C_l^{f_l} A_l^{y_{a,l}} \Big)^{-x_{j_l}}$$

since $\mathcal{V}(tr) = 1$. However, their exponents (τ, τ', τ'') are different from $(0, 0, \xi)$ since $\mathcal{R}_{forced}$ does not hold but ZKA of $\mathcal{R}_{account}$ and $\mathcal{R}_{zero}$ valid. This reduction wins the game of binding property if $\mathcal{P}^*_{KS}$ exists. Hence, we claim the validity of Lemma D.2.

E SECURITY PROOFS OF BALANCE PROOFS

We prove the security of balance proofs in this section.

We show the completeness of balance proofs by proving the correctness of Step 12. If $\sum_{i=0}^{N} v_i + c + f = \sum_{i=0}^{N} v'_i + c' + f'$ then Step 12 is valid as follows:

$$\mu^{f'-f} = E^{-1} U^y h^{\bar{s}'} (\prod_{i=0}^{N} K')_i^{\bar{s}} \in \mathbb{G} \tag{1}$$

$$= \Big(C' C^{-1} \prod_{i=0}^{N} V'_i \times V_i^{-1} \Big)^{-1} U^y h^{\bar{s}'} (\prod_{i=0}^{N} K')_i^{\bar{s}} \tag{2}$$

$$= \Big(h^{\alpha_{c'}} \mu^{c'} h^{-\alpha_c} \mu^{-c} \prod_{i=0}^{N} K_i^{r'} \mu^{v'_i - v_i} \Big)^{-1} \Big(h^{u'} (\prod_{i=0}^{N} K_i)^u \Big)^y \tag{3}$$

$$\times h^{(\alpha_{c'} - \alpha_c) - yu'} (\prod_{i=0}^{N} K')_i^{r' - yu} \tag{4}$$

$$= \mu^{\sum_{i=0}^{N} (v'_i - v_i) + c' - c} \tag{5}$$

LEMMA E.1. *Nopenena balance proof protocol has witness-extended emulation for relation $\mathcal{R}_{balance}$ if Pedersen commitments are hiding, and rerandomized accounts provide ZKA for $\mathcal{R}_{account}$.*

Proof: Let $\mathcal{A}_{WEE}$ be an adversary who wins the game of witness-extended emulation in Definition 3.5 for $\mathcal{R}_{balance}$. We reduce $\mathcal{A}_{WEE}$ to break Pedersen commitments' hiding property as follows:

```
1:  premessage: send $(c_0, c_1) \in [0, 2^L)$ to Theorem 3.4's challenger.
2:  input: get $(h, \mu, \hat{C})$ from the challenger in Theorem 3.4.
3:  reduction: create accounts, update proofs, and emulated and
    genuine proofs for $([k_i, r_i, \kappa_i]_{i=0}^{N}, \alpha_{c'}, \alpha_c, c', c, r', u, u') \xleftarrow{\$} \mathbb{Z}_q^*$ and
    $f + c_0 + \sum_{i=0}^{N} v_i = f' + c' + \sum_{i=0}^{N} v'_i$
4:      $[acc_i := (K_i = g^{k_i}, G_i = g^{r_i}, V_i = G_i^{k_i} \mu^{v_i})]_{i=0}^{i=N}$
5:      $[asset'_i : (G'_i, V'_i) := \mathsf{UpdateAsset}(r', v_i, v'_i, acc_i)]_{i=0}^{N}$
6:      $[\sigma_i := \mathsf{UpdateValueProve}(r', k_i, v_i, v'_i, acc_i, asset'_i; \kappa_i, W)]_{i=0}^{N}$
7:      $y \in \mathbb{Z}_q$ from the verifier
8:      $C = \hat{C} h^{\alpha_c}$ and $C' = h^{\alpha_{c'}} \mu^{c'}$
9:      $E = C' C^{-1} \prod_{i=0}^{N} V'_i \times V_i^{-1} \in \mathbb{G}$
10:     $U = U(C \mu^{c_0} h^{\alpha_c})^{-1/y}$
11:     $\bar{s} = (r' - yu)$ and $\bar{s}' = (\alpha_{c'} - \alpha_c) - yu \in \mathbb{Z}_q$
12:     $u = ([acc_i, asset'_i, \sigma_i]_{i=0}^{N}, W, f, f', C, C', \pi_{balance} = (U, \bar{s}, \bar{s}'))$
13:     $b \leftarrow \mathcal{A}_{WEE}(((g, h, \mu), u); \zeta := \phi)$
14: output:
15:     $[0, 1] \setminus b$ to the challenger of Theorem 3.4
```

If $\hat{C}$ is a commitment of c_0, then $\mathcal{A}_{WEE}$ outputs 1 with non-neglible probability since balance proof is created for the correct balance. However, $\mathcal{A}_{WEE}$ outputs 0 if $\hat{C}$ is a commitment of c_1 since balance proof is created for $f + c_1 + \sum_{i=0}^{N} v_i \neq f' + c' + \sum_{i=0}^{N} v'_i$. Therefore, we win the hiding game of Pedersen commitments if $\mathcal{A}_{WEE}$ exists even when rerandomized accounts provide ZKA for $\mathcal{R}_{account}$. Thus, we conclude that Lemma E.1 is true.

LEMMA E.2. *Nopenena balance proof protocol provides knowledge soundness for relation $\mathcal{R}_{balance}$ if Pedersen commitments are binding, and rerandomized accounts provide ZKA for $\mathcal{R}_{account}$.*

Proof: We assume that there exist an extractor $\mathcal{W}$ and a rewindable prover $\mathcal{P}^*_{KS}$ who breaks the knowledge soundness of $\mathcal{R}_{balance}$. We reduce $\mathcal{P}^*_{KS}$ to break the binding property of three generator Pedersen commitments such that

$$\mathsf{Commit}_{g,h,\mu}(\tau, \tau', \tau'') = g^{\tau} h^{\tau'} \mu^{\tau''} \in \mathbb{G}.$$

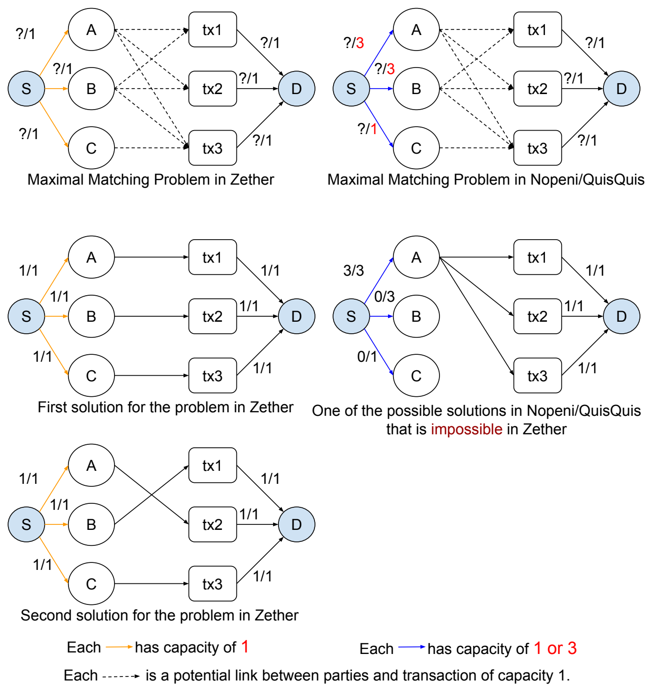
```
1:  input:
2:      get $(g, h, \mu)$ from the challenger in Theorem 3.4.
3:  reduction:
4:      $pp = (g, \mu)$ and initial witness $s = \phi$
5:      extract $(tr, w) \leftarrow \mathcal{E}^{\mathcal{W}(\langle \mathcal{P}^*_{KS}(pp, acc, s), \mathcal{V}(pp, acc) \rangle)}$ when
6:          $tr =: ([acc_i, asset'_i, \sigma_i]_{i=0}^{N}, W, f, f', C, C', \pi_{balance} = (U, \bar{s}, \bar{s}'), y)$
7:          and $w =: ([k_i, r_i, v_i, v'_i, \kappa_i]_{i=0}^{N}, r', u, u', \alpha_{c'}, \alpha_c, c', c)$
8:              such that $\big([acc_i := (K_i = g^{k_i}, G_i = g^{r_i}, V_i = G_i^{k_i} \mu^{v_i})]_{i=0}^{i=N}$
9:              $[asset'_i : (G'_i, V'_i) := \mathsf{UpdateAsset}(r', v_i, v'_i, acc_i)]_{i=0}^{N}$
10:             $[\sigma_i = \mathsf{UpdateValueProve}(r', k_i, v_i, v'_i, acc_i, asset'_i; \kappa_i, W)]_{i=0}^{N}$
11:             $T_3 = K^{s_2} g^{-s_3} \wedge G' = G g^{r'} \wedge V' = V K^{r'} \mu^{v' - v}) = 1$  ▷ $\mathcal{V}(tr) = 1$
12:             and            ▷ since $(pp, u, w) \notin \mathcal{R}_{account}$ in Equation 1
13:             $\Big( (f' - f \overset{?}{\neq} \sum_{i=0}^{N} v_i - \sum_{i=0}^{N} v'_i + c - c') \vee$
14:             $(0 \overset{?}{\neq} \alpha_c - \alpha_{c'} - u'y - \bar{s}') \vee (0 \overset{?}{\neq} (r' - yu) - \bar{s}) \Big) = 1$
15:         $out = \Big( (\bar{s}, \bar{s}', f' - f), \big(r' - yu, \alpha_c - \alpha_{c'} - u'y, \sum_{i=0}^{N} v_i$
16:                 $- \sum_{i=0}^{N} v'_i + c - c'\big) \Big)$
17: output:
18:     $out$ to the challenger of Theorem 3.4.
```

Here, $\mathsf{Commit}_{g,h,\mu}(\bar{s}, \bar{s}', f' - f) = \mathsf{Commit}_{g,h,\mu}(r' - yu, \alpha_c - \alpha_{c'} - u'y, \sum_{i=0}^{N} v_i - \sum_{i=0}^{N} v'_i + c - c')$ but $\Big((\bar{s}, \bar{s}', f' - f)$ is not equal to $(r' - yu, \alpha_c - \alpha_{c'} - u'y, \sum_{i=0}^{N} v_i - \sum_{i=0}^{N} v'_i + c - c')$ if $\Big((f' - f \overset{?}{\neq} \sum_{i=0}^{N} v_i - \sum_{i=0}^{N} v'_i + c - c')$. Therefore, the following reduction breaks the binding property of Pedersen commitments even when rerandomized accounts provide ZKA for $\mathcal{R}_{account}$. Hence, we conclude that Lemma E.2 is correct.

F EXAMPLE OF GRAPH ANALYSIS IN ZETHER VS. NOPENENA/QUISQUIS

We use the following example to elaborate more on how to apply graph analysis on Zether to discover more unintentional knowledge and why the graph analysis does not work on Nopenena and QuisQuis. We leave a more formal analysis for future work.

Assume that verifiers see three transactions (tx_1, tx_2, tx_3) that take the decoy sets: (A, B), (A, B), and (A, B, C), respectively, and denote that there is only one sender in each transaction, e.g., sending coins to a contract/withheld list. In Zether, we assume that all



?/1
A
tx1
?/1
?/1
S
B
tx2
?/1
D
?/1
C
tx3
?/1
Maximal Matching Problem in Zether
?/3
?/3
?/1
Maximal Matching Problem in Nopeni/QuisQuis
1/1
First solution for the problem in Zether
3/3
0/3
0/1
One of the possible solutions in Nopeni/QuisQuis that is impossible in Zether
Second solution for the problem in Zether
Each has capacity of 1
Each has capacity of 1 or 3
Each is a potential link between parties and transaction of capacity 1.

Figure 7: Maximal matching problem in Zether vs. Nopenena/QuisQuis. Here, "S" is the source, and "D" is the drain. A solution(s) to this problem is sending maximum flow units (3 units in this example) from the source to the drain when each arrow (pipeline) has a maximum capacity.

three transactions happen in the *same epoch*. We can draw the maximal matching problem [21] for Zether, QuisQuis, and Nopenena as shown in Figure 7. In Zether, the "yellow-colored arrows" have the capacity of 1 since an account can be spent only once in each epoch. However, in Nopenena and QuisQuis, the "blue-colored arrows" have a maximum capacity of 1 for C or 3 for A, B, i.e., the number of times each account was used as decoys.

There are two solutions to Zether's maximal matching problem, as shown in Figure 7. In both solutions, C must be connected to tx_3. Thus, we can deanonymize the sender of tx_3 as C. However, in Nopenena and QuisQuis, C is not connected to tx_3 in all solutions. For example, A may have spent coins three times as shown in Figure 7 since there is no limit on the number of times an account can be used. Thus, graph analysis does not reveal any unintentional knowledge in Nopenena and QuisQuis.