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
### Table 1: A Comparison of Related Work

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Zerocoin [42]</td>
<td>Maximal</td>
<td>☐</td>
<td>High</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>ZCash [29]</td>
<td>Maximal</td>
<td>☐</td>
<td>High</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Lelantus [31]</td>
<td>Maximal</td>
<td>☐</td>
<td>High</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Mimblewimble [30]</td>
<td>Non</td>
<td>☐</td>
<td>Zero</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Monero [46], [34, 60]</td>
<td>Degrading</td>
<td>☐</td>
<td>Zero</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Ring CT v.2 [54]</td>
<td>Degrading</td>
<td>☐</td>
<td>High</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Zether [9, 15]</td>
<td>Degrading (epoch)</td>
<td>☐</td>
<td>High</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>QuisQuis [19]</td>
<td>Non-degrading</td>
<td>☐</td>
<td>Low</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>PriDe CT [26]</td>
<td>Degrading (epoch)</td>
<td>☐</td>
<td>High</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>PriFHEte [39]</td>
<td>Maximal</td>
<td>☐</td>
<td>High</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Nopenena (this paper)</td>
<td>Non-degrading</td>
<td>☐</td>
<td>Low</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

Our research question is "can we build a more efficient account-based payment system with non-degrading untraceability?".
(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: \((pk_0, asset(pk_0, r_0, v_0), asset(pk_0, r_0+r', v_0), s_0)\) and \((pk_1, asset(pk_1, r_1, v_1), asset(pk_1, r_1+r', v_1+\varepsilon'), s'_0)\), 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 \([j_i]_{i=0}^n\) are the sending and receiving accounts. Here, each \(j_i\) 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 \([asset_j, s_j]_{j=0}^n\) 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 \(\sum_{i=0}^N s_i + f = \sum_{i=0}^N s'_i + f'\) 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 \([C_i = commit(a_i, v'_i)]_{i=0}^n\) with random blinding keys \([\sigma_i]_{i=0}^n\) to hide new account balances \([\sigma'_i]_{i=0}^n\). Then, the sender and receivers run a multi-party forced opening protocol and create a proof \(\pi_{forced}\) to prove that \([C_i]_{i=0}^n\) 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 \([C_i]_{i=0}^n\). Then, the sender and receivers create range proofs for these commitments \([C_i]_{i=0}^n\) 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 \(\Lambda\) 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. CreateTx\((\Lambda, \cdot) : tx\) is a multi-party protocol of a (coin-)sender and (coin-)receivers that creates a valid transaction \(tx\) for \(\Lambda\) such that VerifyTx\((\Lambda, tx) = 1\). 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 \(\approx:\) and \(\approx:\) for assignments, e.g., \(a \approx b\) means that \(b\) was assigned from \(a\). \(A \setminus B\) is the set minus of \(A\) by \(B\). \(\phi\) is an empty set. \(\approx\) indicates double implication. For a cyclic group \(G\), we reserve \((g, h, \mu)\) to denote Nothing-Up-My-Sleeve (NUMS) generators of \(G\) (Definition 3.1). \(Z_q = \mathbb{Z}/q\mathbb{Z}\) is a ring of modular integers in \([0, q - 1]\) for modulus \(q\). We use \(a_i\) to denote \(i\)th element of an array. \(s \in \mathcal{S}\) denotes that \(s\) is drawn uniformly at random from a set \(\mathcal{S}\). We use \(\lambda\) for the security level and \(\mathcal{A}\) for a probabilistic polynomial-time (p.p.t.) adversary of \(\lambda\). We use \(\mathcal{A}(s)\) to imply that \(\mathcal{A}\) is given \(s\). Also, \(\epsilon(\lambda) = 1/\lambda(\lambda^c)\) is a negligible function \(\forall \epsilon \in \mathbb{N}\). Sometimes, we casually use “negligible” to denote \(\epsilon(\lambda)\). Hash: \(\{0, 1\}^* \rightarrow Z_q\) denotes a collision-resistant hash function family. We use “non-active” to mean that there are no interactions between the prover(s) and the verifiers; however, they could be multi-party protocols of provers.

---

2These 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 $\text{Adv}_{\mathbb{G}}^{\text{DL}}$ of $A$ is $Pr[Y^2=g^X | Y \leftarrow A(g; Y)]$ for any NUMS generator $g$. The DL problem is $(\tau, \epsilon)$-hard if $A(\tau, \epsilon)$ runs it at most $\tau$ times and $\text{Adv}_{\mathbb{G}}^{\text{DL}} \leq \epsilon(\lambda)$.

Definition 3.3. [Decisional Diffie-Hellman (DDH) Problem] $A$'s advantage $\text{Adv}_{\mathbb{G}}^{\text{DDH}}$ is $Pr[b=1' | (x, y, e) \leftarrow A(g, g^x, g^y, Y_0)]$ for any NUMS $g$. The DDH problem is $(\tau, \epsilon)$-hard if $A(\tau, \epsilon)$ runs it at most $\tau$ times and $\text{Adv}_{\mathbb{G}}^{\text{DDH}} \leq \epsilon(\lambda)$.

3.1 Pedersen Commitments

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

Theorem 3.4. Pedersen commitments provide the followings:

- (Hiding) Upon receiving $(\alpha_0, \alpha_1) \in \mathbb{V}$ from $A$, the challenger shares $C = \text{Commit}_{h,\mu}(\alpha \in \mathbb{Z}_q, v_0)$ from a random choice of $b \leftarrow \{0, 1\}$. The probability of $A$ finding $b$ is $\leq 1/2 + \epsilon(\lambda)$.
- (Computational binding) The probability of $A$ finding two different, $(\alpha_0, \alpha_1), (\alpha_1, \alpha_0) \in (\mathbb{Z}_q \times \mathbb{V})$ such that $\text{Commit}_{h,\mu}(\alpha_1, v_0)$ = $\text{Commit}_{h,\mu}(\alpha_1, v_0)$ and $(\alpha_0, \alpha_1) \neq (\alpha_1, \alpha_0)$ 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 $L_{pp} = \{x \in \mathbb{W} : \exists (pp, x, w) \in \mathcal{R}\}$. A proving algorithm may take multiple interactions between $\mathcal{P}$ and $\mathcal{V}$. We denote an interaction’s transcript, $tr \leftarrow (\mathcal{P}^{\star}(pp, u, w), \mathcal{V}(pp, u))$. If $\mathcal{V}$ accepts $tr$, $\mathcal{V}(tr) = 1$. We define ZKA similar to [10, 23, 38] except that we define an adversary $A$ with some insider-knowledge $\xi$ such that $(w \setminus \xi) \neq A$ 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) (Set, $\mathcal{P}$, $\mathcal{V}$) is a ZKA if they satisfy the following properties:

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

  $Pr[(pp, u, w) \in \mathcal{R} \land \mathcal{V}(\mathcal{P}(pp, u, w), \mathcal{V}(pp, u)) \leftarrow 0 | (u, w) \in A^{\mathbb{G}}(pp)] = 1$

- (Computational Witness-Extended Emulation (WEE)) Let there be two p.p.t. adversaries: WEE adversary $A$ with insider-knowledge $\xi$ and $A^{\mathbb{G}}$ who generates the initial witness $s$ and statement $u$. Also, $\mathcal{P}^{\star}$ 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 $\mathcal{V}(tr) = 1$. (Set, $\mathcal{P}$, $\mathcal{V}$) has WEE if:

  $Pr[A(tr; \xi) | pp = \text{Set}(\lambda), (u, s) \leftarrow A^{\mathbb{G}}(pp)] < 1/2$.

$Pr[A(tr; \xi) | pp = \text{Set}(\lambda), (u, s) \leftarrow A^{\mathbb{G}}(pp)] \leftarrow \mathcal{V}(pp, \mathcal{V}(\mathcal{P}(pp, u, w), \mathcal{V}(pp, u)) | (pp, u, w) \in \mathcal{R}) \leq \epsilon(\lambda)$.

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

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

Definition 3.6 (Special Honest Verifier Zero-Knowledge Argument). Let $\mathcal{V}$’s challenges be chosen from a public coin randomness $\rho$ (created from tossing an unbiased public coin) which is independent of provers’ messages. (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 $\{0, 2^{\ell}\}$ for $\ell \in \mathbb{N}$ since coin values should be non-negative. Let $\text{RangeProve, RangeVerify}$ be a range proof scheme such that the $\text{RangeProve}(C, o, A, L)$ creates a range proof $\pi$ and $\text{RangeVerify}(\pi, L, O)$ outputs 1 if $o \in \{0, 2^{\ell}\}$, otherwise 0.

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

$\text{RangeVerify}(pp, \pi, L) \Rightarrow \text{RangeProve}(C, o, L)$

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

Definition 3.8. One-of-many proofs for zero-value commitments provide zero-knowledge if they are ZKA for verifier $\mathcal{V}(pp, u) = \text{ZeroComVerify}(\{C\}_{i=0}^{N-1}, \pi) \leftarrow j \leq N$ and $\xi \leftarrow \phi$, and relation $\mathcal{R}_{\text{zero}}$:

$\text{RangeVerify}(pp, \pi, L) \Rightarrow \text{RangeProve}(\{C\}_{i=0}^{N-1}, j, \alpha)$

$\text{ZeroComVerify}(\{C\}_{i=0}^{N-1}, \pi) \leftarrow j \leq N$ and $C_j = \text{Commit}_{h,\mu}(\alpha, 0)$. Otherwise, it outputs 0.
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 $V$ as follows:

- **Compile** ($F$) : $[F_{zk} : \text{blind}]:$ compiles a function $F : V^* \rightarrow 1$ into a zero-knowledge function $F_{zk}$ and outputs the secret information blind required to prove its validity. This $F_{zk}$ is basically a customized zero-knowledge verification function for $F$ but hides $F$'s variables using blind.

- **ContractProve** ($F, F_{zk} : C_i, y_i, a_i | i = 0 \ldots n$, blind) : $\pi_{\text{contract}} \triangleright$ If $F([y_i]_{i=0}^n) = 1$, it generates a zero-knowledge proof $\pi$ when $[y_i]_{i=0}^n \in V^*$ is committed in $[C_i]_{i=0}^n$. If $F([a_i]_{i=0}^n) = 0$, it returns error $\perp$. This proving could be a multi-party protocol.

- **$F_{zk}([C_i]_{i=0}^n, \pi_{\text{contract}}): 1/0 \triangleright$ verifies the contract.

**Definition 3.9.** A contract $F_{zk}$ provides zero-knowledge if it is ZKA for $V((p, u) = F_{zk}([C_i]_{i=0}^n), \zeta = \phi)$, and relation:

$$(p, u) = (F_{zk}([C_i]_{i=0}^n), \pi) \in R_{\text{contract}} \Rightarrow$$

$$\left( (h, \mu) = \pi \land F_{zk}([C_i]_{i=0}^n) = 1 \right)$$


4 NOPENENA OVERVIEW

In this section, we define Nopenena functions for a ledger $A : (\text{Accounts, Withheld})$. We assume that the ledger $A$ holds a set of accounts $\text{Accounts} \in A$. 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. We use Withheld $\in A$ to denote a set of these contract-commitment pairs that have not released their coins. This untraceable payment system provides the following functionalities:

1. **Setup** ($\lambda$) : $pp \triangleright$ creates public parameters. We assume that $pp$ is public and consistent such that all users can access $pp$.

2. **CreateTx** ($A, [acc_i]_{i=0}^n, [u_i, v_i, a_i]_{i=0}^n, [F_{zk}, C, \pi_{zk}, \alpha_c, \gamma_c], [F^*_c, C^*, \alpha_c^*, \gamma_c^*], f, f'; [k^*_i]_{i=0}^n) : tx := ([asset']_{i=0}^n, \pi) \triangleright$ This protocol is conducted by senders and receivers to create a transaction. The transaction contains rerandomized assets $[asset']_{i=0}^n$ of accounts $[acc_i]_{i=0}^n$ when only $n$ accounts in indexes $[j]_{j=0}^n \in [0, N]^n$ are the actual sending/receiving accounts. Other accounts are decoys, and decoys do not participate in creating the transaction. The current account balances are $[v_i]_{i=0}^n$ and new balances are $[v_i']_{i=0}^n$ in sending/receiving accounts. Here, $[C_i]_{i=0}^n$ are the Pedersen commitments of updated balances such that $[C_i \leftarrow \text{Commit}_{\pi_{zk}}(a, v_i')]_{i=0}^n$. $[F_{zk}, C] \in \text{Withheld}$ is the contract-commitment pair that will be proven and released by the transaction when $\pi_{zk}$ is the contract proof such that $F_{zk}(C[C^*']([C_i]_{i=0}^n, \pi_{zk}) = 1$.

3. **Verify** ($F, F_{zk} : C_i, y_i, a_i | i = 0 \ldots n$, blind) : $\pi_{\text{contract}} \triangleright$ This function verifies the transaction.

4. **OpenContract** ($k, v, acc, \pi) : 1/0 \triangleright$ verifies whether the secret key and account balance are $(k, v)$ or not.

When the ledger appends $tx$, it removes current assets $[acc_i]_{i=0}^n$ from Accounts and adds $[asset']_{i=0}^n$ to Accounts. Also, the ledger removes $[F_{zk}, C]$ from Withheld and adds $[F^*_c, C^*]$ 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 $\gamma$, and balance $v$ is:

$$acc := (pk, asset) := ([k = g^\gamma], G = g^\gamma, V = g^\gamma \mu^\gamma \rho^\gamma)$$

We do not want anyone who knows $k, \gamma, v$ to see the asset.

1. **CreateAccounts** ($k, v, (G, V)$): $\triangleright$ create a new account of $\mathbb{G}$

2. **UpdateAsset** ($r, v' = v + r, k, \gamma, \rho, \mu$): $\triangleright$ update the asset

3. **Receive** ($\gamma' = \gamma + \rho, \nu' = V \nu'$): $\triangleright$ receive the asset

4. **Send** ($r, \gamma' = \gamma + \rho, \nu' = V \nu'$): $\triangleright$ send the asset

5. **Balance** ($\gamma, \nu'$): $\triangleright$ balance

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

$$\text{UpdateValueProve}(r', \rho', \nu', k, \gamma, \rho, \mu, (G, V))$$

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
decays 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 $k$, as follows:

1. UpdateValueProve$(r', k, a', v', acc, asset', x, W)$:
2. $(K, G, V) := acc$ and $(G', V') := asset'$
3. The sender computes: $t := \text{Round}(T, T_{1} := g^{t}$, and $T := K'$
4. The receiver gets $T$ from the sender and computes: $r := t + x(r') \in \mathbb{Z}$
5. $x \in \mathbb{R}$
6. $(T_{1}, T_{2}, T_{3}) \rightarrow V$ and $x \in \mathbb{R}$
7. Non-interactive: $x := \text{Hash}(W, T_{1}, T_{2}, T_{3}, acc, asset') \in \mathbb{Z}$
8. The sender computes: $x := t + x(r') \in \mathbb{Z}$
9. The owner computes and shares $(s_{2}, s_{3})$ with the sender:
10. $s_{2} := x + x'(r' - v) \in \mathbb{Z}$
11. The owner combines: return $\sigma := (x, s_{1}, s_{2}, s_{3}) \in \mathbb{Z}$

We define the verification function for any update proof below:

1. VerifyUpdate$(acc, asset', \sigma = (x, s_{1}, s_{2}, s_{3}); W)$: $V$ has $T_{1}, T_{2}, T_{3}$ in the interactive version:
2. $(K, G, V) := acc$ and $(G', V') := asset'$
3. return $T_{1} = g^{t}$ and $T_{2} = T_{3}$
4. 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 $\text{Hash}(W, T_{1}, T_{2}, T_{3}, acc, asset')$. Therefore, the non-interactive only contains four elements of $\mathbb{Z}$, and we do not share $(T_{1}, T_{2}, T_{3})$ with verifiers.

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

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

$$pp, u = (W, acc = (K, G, V), asset' = (G', V'), \sigma), (r', k, a', v', k) \in R_{\text{accounts}} \Leftrightarrow$$

$$\left( g, \mu \right)^{pp} \land \Delta_{k} = g^{k} \land \text{OpenAsset}(k, a, (G, V)) = 1 \land$$

$$\text{asset'} = \text{UpdateAsset}(r', a', v', acc) \land \text{OpenAsset}(k, a', asset') = 1 \land$$

$$\left( (a' \neq 0) \land (\sigma = \text{UpdateValueProve}(r', k, a', v', asset', x; W)) \lor \right.$$  

$$\left( \sigma = \text{UpdateValueMProve}(r', k, a', v', asset', x; W)) \right) \lor \right.$$  

$$\left( (a' \neq 0) \land \sigma = \text{UpdateUpdate}(r', asset', k, W)) \right).$$

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

We also want to obtain zero-knowledge for multiple account updates with the same $r'$ when the accounts belong to the sender, receivers, and decays. In this case, the adversary may control 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 $\text{R}_{\text{accounts}}$ against an adversary who controls $R_{A}$ receiving accounts in indexes $J_{A} = [j_{p}^{R_{A}}]^{n}_{p=0}$.

**Definition 5.2 (Insider ZKA of Randomized Accounts).** Randomized accounts are insider zero-knowledge if they provide zero-knowledge argument for $0 \leq R_{A} < n < N$, verifier: $V(pp, u) = \bigwedge_{i=0}^{n_{i}} \text{UpdateVerify}(acc, asset', \sigma; W)$, insider knowledge: $\zeta = \bigcup_{i=0}^{n_{i}} \text{UpdateValueProve}\left( j_{i}^{R_{A}} \right)$, and

$$(pp, u) = \left( [acc, asset', \sigma]_{i=0}^{N} W \right) \left( [j_{i}^{R_{A}}]^{n_{i}} \right) \in \text{R}_{\text{insiders}} \Leftrightarrow$$

$$\bigwedge_{i=0}^{N} \left( (pp, L), (W, acc, asset, \sigma), (r', k, a', v', k) \in R_{\text{accounts}} \land \right.$$  

$$\bigwedge_{i=0}^{R_{A}} \sigma_{j_{i}^{R_{A}}} \in \text{UpdateValueMProve}(r', k, a', v', asset', \sigma; W) \right).$$

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

**Definition 5.3 (Theft-Resistance of Randomized Accounts).** Let there be an account $acc = \left( pk, a_{0}, \text{asset}_{pk}(a_{0}, r_{0}) \right)$ of key $k$ and an oracle $O_{\text{acc}, Q}(i, v_{i-1}, v_{i})$ that outputs ith updated asset $a_{i}$ and its update proof $\sigma_{i}$ as defined below.

$$O_{\text{acc}, Q}(r, d, W) :$$

$$i := Q \lor v_{i-1} = + d \land Q \land (i - 1, v_{i-1}, a_{i-1}) \in Q \land$$

$$a_{i} := \text{UpdateAsset}(r, v_{i-1}, a_{i-1}, a_{i}, k, \text{asset}_{pk}(a_{0}, r_{0})) \land \Delta_{k} = Z_{Q}$$

$$\text{else}: \sigma_{i} := \text{UpdateProve}(r, pk, a_{i-1}, a_{i}, k, W) \land$$

$$Q := Q \lor (i, v_{i-1}, a_{i}) \land \text{return} \left( a_{i}, \sigma_{i} \right).$$

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

$$P_{R} \left( (i', v', a') \in Q \land \text{OpenAsset}(k, a', \text{asset}_{pk}(a', \text{asset}_{pk}(a', r_{0})) \land \text{UpdateVerify}(pk, a_{i-1}, a', \sigma) \land v_{i-1} = a' \right) \rightarrow (i', v', a', \sigma)$$
As we explained before, when an account is updated, the fact that participants prove that all updated balances are committed in comments as the challenge message in account updates (see \[5.2\] Anonymous Forced Openings).

Let \(\mathcal{D}_16: \gamma_1: \alpha \quad \gamma_2: \beta \quad \gamma_3: \alpha\) be these commitments, and combination } j be the sending/receiving accounts’ index set such that \([i,j]_n^0 = [i,j]_{n-1}^0 \in \mathbb{G}\).

For \(m \in \{0, M\} : \mathcal{H}_{\gamma_1} = \prod_{i=0}^n \left( (\mathcal{G}_i')^2 \mathcal{C}_i \right)^{2m} \in \mathbb{G}
\)

Once all accounts have been updated with \(\mathcal{D}_16: \gamma_1: \alpha \quad \gamma_2: \beta \quad \gamma_3: \alpha\), the sender computes the proof when \(j \) is the group index.

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

### 5.5. Security Definitions

We define ZKA of anonymous forced openings for relation \(\mathcal{R}_{\text{forced}}\) when the adversary controls \(R_{\text{ad}} \geq 0\) number of receiving accounts in indexes \(I_{\text{ad}} = [j_0]_{R_{\text{ad}}^0}\) as follows.

Once all accounts have been updated with \(W\) and \([k_j]_{I_{\text{ad}}^{0}}\), the participants prove that all updated balances are committed in commitments. Here, \([k_j]_{I_{\text{ad}}^{0}}\) are the set of proofs of accounts.

For \(\mathcal{O}_{\text{forced}}\) to be the sending/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 \(\text{ForcedOutCreate}\). Then, they use \(W\) as the challenge message in account updates (see \(\text{UpdateValueProof}\)). After that, they anonymously prove that they have opened all updated values’ commitment by creating \(\pi_{\text{forced}} \leftarrow \text{ForcedOutCreate}\). Later, verifiers check \(\pi_{\text{forced}}\) by running the verification \(\text{ForcedOutVerify}\).
A decentralized transaction may contain rewards and transaction fees as an incentive to add the transaction to a block. To prevent illegal coin generation, we ensure that the total input balance, re-leased withheld coins, and rewards are equal to the total output balance, new withheld coins, and transaction fee. We propose the following balance protocol for re-randomizable accounts. The sender who updated accounts by some \( r' \) creates proofs as follows.

1. **BalanceProof**(\( f, f', C, c, a_c, C', c', a_c', [acci, asset]^N_{i=0}, r' \)):
   1. \( f \notin [0, 2^l] \lor f' \notin [0, 2^l] \): return \( \bot \)
   2. \((k_i, K_i, G_i, V_i) := \text{acci} \) and \((G_i', V_i') := \text{asset}^i \)
   3. \( E := C'C^{-1} \prod_{i=0}^N V_i' \times V_i \in \mathbb{G} \)
   4. \( U = h^r \prod_{i=0}^N K_i^u \in \mathbb{G} \) for \( (u, u) \rightarrow z_q \)
   5. \( y \sim V \rightarrow \text{Non-active:} y := \text{Hash}(f, f', C, C', U, E) \in z_q \)
   6. \( \sigma_{balance} := (\text{g}, s := \text{r}' - y, u, s' := (a_c' - a_c) - y'u) \)
   7. **BalanceVerify**(\( f, f', C, C', [acci, asset]^N_{i=0}, \sigma_{balance} \)):
   8. \( E := C'C^{-1} \prod_{i=0}^N V_i' \times V_i \in \mathbb{G} \)
   9. \( y \sim V \rightarrow \text{Non-active:} y := \text{Hash}(f, f', C, C', U, E) \in z_q \)
   10. \( \text{return } f, f' \in [0, 2^l] \land u \in \mathbb{G} \rightarrow E' = E \)

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 \( \forall (pp, u) = \text{BalanceVerify}(f, f', C, C', [acci, asset]^N_{i=0}, \sigma_{balance}) \), \( \xi = \phi \), and relation \( \mathcal{R}_{balance} \):

\[
\left( \text{pp}, u, \left( \text{acci}, \text{asset}^i \right)_{i=0}^N, \sigma_{balance} \right) \rightarrow \left( \text{f}, \text{f}', \text{C}, \text{C}' \right), \text{balance}, \left( \text{g}, s', \text{r}' \right) \in \mathcal{R}_{balance} \Leftrightarrow (g, h, \mu, L) \in \text{pp} \land s', s'_{i=0}^N \in \mathbb{G} \land \left( \text{f}, \text{f}' \right) \in [0, 2^l] \land
\begin{align*}
& f + c + \sum_{i=0}^n y_i \cdot f' + c' + \sum_{i=0}^n y_i' \cdot f' \\
& \land \forall i \in \mathbb{G} \left( \text{L}, \left( \text{W}, \text{acci}, \text{asset}_i \right), \sigma_i \right), \left( r'_i, v_i, v'_i, k_i \right) \rightarrow \mathcal{R}_{accounts} \land \text{C} \rightarrow \text{Commit}_{h, p}(\text{g}, c) \land \text{C} \rightarrow \text{Commit}_{h, p}(\text{g}, c') \land
& \text{\#BalanceProof}(f, f', C, C', c, c', a_c, [acci, asset]^N_{i=0}, r')
\end{align*}
\]

**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 \( acc_2 \) by taking Charles' account \( acc_3 \) 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 \( \text{UpdateAsset} \). 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 \( \text{ForcedOutCreate} \) 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 \( \text{UpdateValueProof} \) and the validity of the decoy account's update by running \( \text{UpdateProof} \). Alice and Bob also prove his account's update by running the multi-party \( \text{UpdateValueMProof} \). 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 \( \text{ForcedOutProof} \). 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**(\( \Lambda, \text{acci}, \text{asset}^i \)\( [j_i, v_j, v_i', j_i, C]^N_{i=0}, \text{[F}_{\text{zk}}, C, \pi_{\text{zk}}, a_c, c] \))
2. \( \text{F}_{\text{zk}}(C, c, \pi_{\text{zk}}, a_c, c_i) \)
3. \( \text{F}_{\text{zk}}(C, C', \pi_{\text{zk}}, a_c, c_i) \)
4. \( \text{UpdateAsset}(r, v_j, v_i', j_i, C, \pi_{\text{zk}}, a_c, c_i) \)
5. \( \text{UpdateValueProof}(r', k_i, v_i, v_i', j_i, C, \pi_{\text{zk}}, a_c, c_i) \)
6. \( \text{UpdateValueMProof}(r', k_i, v_i, v_i', j_i, C, \pi_{\text{zk}}, a_c, c_i) \)
7. \( \text{UpdateProof}(r, \text{acci}, \text{asset}^i, j_i, W) \)
8. \( \text{RangeProof}(C, v_i, j_i, L) \)
9. \( \text{RangeProof}^C(C', c', \pi_{\text{zk}}, a_c, L) \)
10. \( \text{UpdateProof}(C', c', \pi_{\text{zk}}, a_c, L) \)
Figure 2: An exemplary protocol flow of Nopena. Here, Alice sends \( s \) coins to Bob without revealing Alice’s account to Bob, i.e., Bob only learns that either acc1 or acc2 belong to Alice. Also, Alice and Bob hide which accounts are transferring coins from verifiers, i.e., verifiers only see that two accounts from (acc1, acc2, acc3) exchange coins. Note that Alice does not reveal her \((k_1, r')\) to Bob, and Bob does not reveal his \( k_3 \) to Alice.

First, open accounts send an unexpiring conditional transaction, instructing to temporarily lock their accounts because they want to exchange coins with accounts \([pk_i]_{i=0}^N\) when the first [open] public keys are theirs. In the conditional transaction, they agree to a condition that they will only send coins to \([pk_j]_{j=0}^N\) 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 \( \Lambda \).

1. CreateConditionalTx(\( \Lambda, f, f_u, [pk_i]_{i=0}^N; [k_i]_{i=0}^{[open]} \)) \( \triangleright \) locks
2. \( \hat{W} := ([pk_j]_{j=0}^N; [open], f, f_u) \) \( \triangleright \) the condition and challenge
3. Each \( i \in [0, [open]) \) proves the knowledge of \( k_i \) via value-updates
4. \( \hat{W}_i := (W[k_i] \in \Lambda) \) \( \triangleright \) personalizes the challenge
5. \( acc_i := (pk_i, \text{asset}_{pk_i}(0, 0)) \) and \( (r_i, k_i) \in \hat{Z}_Q \)
6. \( \sigma_i := \text{UpdateValueProof}(r_i, k_i, 0, 1, acc_i, \text{asset}_{pk_i}(1, 0), \hat{W}_i) \)
7. return \( ctx := ([open], [pk_i]_{i=0}^N; \sigma_i |_{[open]} f, f_u) \)

Verifiers check the conditional transaction as follows.

1. VerifyConditionalTx(\( ctx=(\text{open}, [pk_i]_{i=0}^N; \sigma_i |_{[open]} f, f_u) \))
2. \( \hat{W} := ([pk_j]_{j=0}^N; [open], f, f_u) \) \( \triangleright \) the condition and challenge
3. \( \forall i \in [0, [open]) \) : \( \hat{W}_i := (W[k_i] \in \Lambda) \) \( \triangleright \) updated challenge
4. if \( \neg \text{UpdateVerify}(pk_i, \text{asset}_{pk_i}(0, 0), \text{asset}_{pk_i}(1, 0), \sigma_i, \hat{W}_i) \)
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, \( acc_i |_{[open]} \) from the ledger and send the completing transaction. Here, \( [ji]_{[i]}^{[open]} = \)
[\{\text{open}\}] are the open accounts’ indexes. Other real participants’ indexes are in \([{\text{open}}]_{\text{N}}\) that could be any index in \([0, N]\).

1. **CreateCompletingTx**(\(f_i, [\text{acc}i_{\text{N}}]_{\text{E=0}} [j_i, v_i, v_i^\prime]_{\text{N}} r^\prime\));
2. return \(tx := \text{CreateTx}(\{\text{acc}i_{\text{N}}]_{\text{E=0}} [j_i, v_i, v_i^\prime]_{\text{N}} c, f)\)

The verifiers check the condition and the transaction.

1. **VerifyCompletingTx**(\(tx, ctx\));
2. \((\{\text{open}\}, [pk]_{\text{N}} [\sigma_i]_{\text{E=0}} f, f^\prime) := ctx\)
3. \((c, f, \{\text{acc} := (pk_i, \text{asset}_i), \text{asset}^i, \sigma_i\} = tx : x) := tx\)
4. return \([pk]_{\text{N}} = [pk]_{\text{N}} \) and \(f = f^\prime \) and **VerifyTx**(\(tx\))

If an account \(acc\) decides to unlock before the second transaction, the next transaction with \(acc\) must pay \(f\) 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 \(R_{\text{Nopenena}}\) as follows when the adversary controls \(R_A\) (0 ≤ \(R_A < n\)) 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:** \(V(ppp, tx) := \text{VerifyTx}(\Lambda, tx)\),
- **insider knowledge:** \(\zeta := (\Lambda = \sum_{j\in A} \sum_{i\in J_A} \sum_k [k_i, v_i, v_i^\prime, k_i, k_{j\cdot p_a}] = \pi_{\text{contract}}, \text{\pi_{range}}, \pi_{\text{balance}})\), and relation \(R_{\text{Nopenena}}\) when 0 ≤ \(R_A < n < N\):

\[
\begin{align*}
\text{ppp}, tx, w = (j, r^\prime, \xi, [j]_{\text{N}} = [k_i, v_i, v_i^\prime, k_i]_{\text{N}}) &\in R_{\text{Nopenena}} \Leftrightarrow \\
\text{tx = } (\sum_{i\in J_A} [\sum_{j\in A} \sum_{k_{j\cdot p_a}} [k_i, v_i, v_i^\prime, k_i, k_{j\cdot p_a} = \pi_{\text{contract}}, \pi_{\text{range}}, \pi_{\text{balance}}]) &\land \\
\text{ppp = } ([\sum_{i\in J_A} \sum_{j\in A} \sum_{k_{j\cdot p_a}} [k_i, v_i, v_i^\prime, k_i]) &\in R_{\text{insiders accounts}} \land \\
\text{w = } ([k_i, v_i, v_i^\prime, k_i]) &\in R_{\text{forced}} \land \\
\text{w, \pi_{\text{forced}} = } ([k_i, v_i, v_i^\prime, k_i]) &\in R_{\text{forced}} \land \\
\text{w, \pi_{\text{balance}} = } ([k_i, v_i, v_i^\prime, k_i]) &\in R_{\text{balance}} \land \\
\text{ppp = } (\sum_{i\in J_A} \sum_{j\in A} \sum_{k_{j\cdot p_a}} [k_i, v_i, v_i^\prime, k_i]) &\in R_{\text{contract}}
\end{align*}
\]

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

Thief-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 \(acc : \ (pk(k), \text{asset}_k, v)\) ∈ Accounts. The transactions are strongly theft-resistant if

\[
\Pr \left[ \text{VerifyTx}(tx) \vdash \exists \text{acc}tx \mid tx \leftrightarrow A^{\text{O}_k}\right] (\Lambda, acc, v) \leq \epsilon(\lambda)
\]

when (1) oracle \(O_k\) creates transactions behalf of \(acc\) for the adversary’s queries without revealing \(k\), and (2) TX are the accounts created by \(O_k\). Here, \(acc^\prime\) 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 relations \(R_{\text{Nopenena}}\) if subprotocols hold ZKA for relations: \(R_{\text{accounts}}, R_{\text{insiders accounts}}, R_{\text{forced}}, R_{\text{range}}, R_{\text{balance}}, R_{\text{contracts}}\). 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 \( F \) of the escrow contract below.

```plaintext
contract escrow(inputs: C0, C1):
  variable e; # prenegotiated escrow fee
  variable C5; # seller's coin amount
  variable CB; # buyer's refund
  if e == c1: # correct escrow fee
    if C5 == 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 \( F_{zk} \) where each variable is committed into a Pedersen commitment, and the binding 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 \( a_e \).
- The seller computes a commitment \( C5 \) for \( C5 \) with key \( a_{C5} \).
- The buyer computes \( CB \) for \( CB \) with key \( a_{CB} \).
- The buyer collects \( CS \) and \( E \) and compiles the contract into the following zero-knowledge function \( F_{zk} \):

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

Listing 2: Open-logic zero-knowledge escrow \( F_{zk} \)

Then, the buyer sends \( c = CB + e = CS + e \) coins to the withheld with \( F_{zk} \). 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 \( [C0, C1] = (C0, C1) \in W \) of new account balances anonymously such that escrows’ commitment is \( C1 \) (see Step 4 of \textit{ForceOutCreate}). Then, they generate the proof as follows:

1. The escrow computes a single-party CIP equal proof \( \pi_{eqE} = EqProve(E, C1, a_e, a_1) \) for \( zkEqual(E, C1) \).
2. The seller computes a single-party CIP equal proof \( \pi_{eqC5} = EqProve(C5, C0, a_{C5}, a_0) \) for \( zkEqual(C5, C0) \).
3. The contract proof \( \pi_{contract} = (\pi_{eqE}, \pi_{eqC5}) \)

This contract and its proof satisfy the following relation:

\[
\left( \mathbb{C}^{-1} \sum_{0}^{C5} \mathbb{C}^0 \sum_{0}^{CB} \mathbb{C} \sum_{0}^{e, c, E} \right) \pi_{escrow} \implies \hat{E} \times C1 \sum_{0}^{C5} \mathbb{C}^0 \sum_{0}^{CB} \mathbb{C} \sum_{0}^{e, c, E} \left( \mathbb{C}^{-1} C5 = \text{Commit}_\mu(a_{CS} - a_{E}, 0) \lor \mathbb{C}^{-1} CB = \text{Commit}_\mu(a_{CB} - a_{E}, 0) \right)
\]

The validators check if \( F_{zk}([C0, C1], \pi_{contract}) \) outputs 1 by verifying \( zkEqual(E, C1) \) from \( EqVerify(E, C1, \pi_{eqE})=1 \) and verifying \( zkEqual(C5, C0) \) from \( EqVerify(C5, C0, \pi_{eqC5})=1 \) (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 \( (a_{CS}, a_{CB}) \) 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.

\textbf{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 \( W \) and shares the binding 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 coins in commitment \( C = \text{Commit}(a_{C}, c) \) with a blinding key \( a_{c} \) as explained in CreateTx. Also, these coins are attached to a contract that always outputs 1:

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

Then, the sender submits this sender-only transaction \( tx_{sending} \) to the ledger, and the sender shares \( a_c \) with the receiver. Upon receiving \( a_{C} \), the receiver creates a receiver-only transaction \( tx_{receiving} \) using \( a_{C} \) 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 coins to his/her account. Here, security comes from the blinding key \( a_{c} \) since only the sender and the receiver know \( a_{c} \).

Split payments moreover benefit from faster transaction verification. As explained in Section 5.2, the verification time of forced opening is \( O(M) \) 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 \( \mathbb{C} \) and \( \mathbb{D} \) 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.
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 x16 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 \( M = \lceil \frac{N}{m!} \rceil \) when n is constant, and N increases from 4 to 24. That is due to the computation of \([H_w]_{m=0}^M\) 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., \( O(\log_2 M) \) [24] or \( O(\log_d M) \) [7].

Similarly, when n is 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.

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 \( O(\log_2 m) \) and \( O(\log_d m) \), 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 \( O(M) \). 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 x16 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.
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 ~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 $\Sigma$-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 data3 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.

---

3QuisQuis C library is not public and implementing it is out of the scope.
The asset rerandomization is sender-anonymous if

\[ \Pr \left[ b \overset{\$}{\leftarrow} b' \right] = \left( \frac{1}{2} \right)^N \]

and the adversary (simulates receivers) tries to identify the sending account is sending coins. We define the following game such that the adversary cannot win the following game of identifying which set of payments from bitcoin. In [23],

\[ \mathcal{A}^{\text{UpdateValueMProve}}(r') \left( \{ \text{acc}_{0}, \text{acc}_{1}, \text{acc}_{2}, \text{acc}_{3} \} \right) \rightarrow b' \]

with more than \( \frac{1}{2} + \epsilon(\lambda) \) probability.

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

\[ \Pr \left[ b \overset{\$}{\leftarrow} b' \right] \leq \frac{1}{2} + \epsilon(\lambda). \]

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.
We prove Theorem 5.4 in this section.

We unfold the left hand sides of statements’ hiding property and the DDH problem as follows.

\[ a_n^\lambda \text{ if the DL problem is hard, and Pedersen commitments are binding.} \]

\[ \text{C.2 Strong theft-resistance} \]

\[ \text{Lemma C.1. Nopenenena rerandomized accounts are theft-resistant if the DL problem is hard, and Pedersen commitments are binding.} \]

\[ \text{Proof:} \] 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**: \(\text{acc} = (K = g, G = C^{1/k}, V = C) \in \mathbb{G}^3\)

3. \( \mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3 \leftarrow \mathcal{A}(\text{acc})\) \(\Rightarrow\) get the preinputs

4. \((i', \sigma', (G', V')) = (x, s_1, s_2, s_3) \Rightarrow \mathcal{A}(x) \leftarrow \mathcal{A}(\text{acc})\) \(\Leftarrow\) from \(\mathcal{A}\) in Definition 3.3

5. \((i', \sigma', (G', V')), \sigma'' = (x', s'_1, s'_2, s'_3) \Rightarrow \mathcal{A}(x') \leftarrow \mathcal{A}(\text{acc})\) \(\Leftarrow\) rewinds \(\mathcal{A}\) with a fresh verifier randomness

6. \(k := (s'_1 - s_1)/(x - x')(\sigma' - v_0)\) \(\Rightarrow\) computes the secret key

7. **output**: \(k \rightarrow \) to DL challenger (Definition 3.2)

Here, we assume that Pedersen commitments are binding such that the adversary cannot find \( s \) such that \( K^s = K'^s \mu^{x_0} = (V')^{-1}V \) when \( \sigma' - v_0 \) is not zero. Therefore, we conclude that Lemma C.1 is true, i.e., Nopenenena accounts have 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 \(\mathcal{R}_{\text{account}}\) if Pedersen commitments are hiding, and the DDH problem is hard.**

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

1. **premessage: mode \(\text{PC} \leftarrow [\mathcal{PC}, \text{DDH}]\)**

2. if mode = ‘\(\text{PC}\)’; send \((v_0, v_1) \in [0, 2^k]\) to Theorem 3.4’s challenger.

3. **input:**

4. if mode = ‘\(\text{PC}\)’; get \((g, \mu, C)\) from the challenger in Theorem 3.4.

5. if mode = ‘\(\text{DDH}\)’; get \((G, X, Y, C)\) from Definition 3.3’s challenger.

6. **reduction:** creates a looks-like account and proof for a precomputed \((k, r, r', t, r, k) \leftarrow \mathbb{Z}_q\) and \(x\) is taken from \(\mathbb{V}\)

7. if mode = ‘\(\text{PC}\)’:

8. \(\text{acc} = (K = g, G = C^{1/k}, V = C) \in \mathbb{G}^3\)

9. \(\hat{g} = g^{1/k} \in \mathbb{G}\)

10. \(\sigma' := \hat{g} \cdot V' := C\mu/\mu^0\)

11. \(s_1 = t + x(r'), s_2 = r + xv_0, s_3 = -k + xv_0k \Rightarrow \text{in } \mathbb{Z}_q\)

12. \(T_1 = \hat{g}^{xv_0k}, T_2 = K\mu/\mu^0, T_3 = K\hat{g}^{xv_0k} \Rightarrow \text{in } \mathbb{G}\)

13. \(\sigma = (x, s_1, s_2, s_3, T_1, T_2, T_3)\)

14. \(b \leftarrow \mathcal{A}_{\text{WEE}}(tr; \xi := \hat{\phi})\)

15. if mode = ‘\(\text{DDH}\)’:

16. \(\text{acc} = (K = X, G = \hat{g}, V = C\mu/\mu^0) \in \mathbb{G}^3\)

17. \(\hat{g} = X^{1/k} \in \mathbb{G}\)

18. \(\sigma' := \hat{g} \cdot V' := V\)

19. \(s_1 = t + x(r'), s_2 = r - xv_0, s_3 = -k - xv_0k \Rightarrow \text{in } \mathbb{Z}_q\)

20. \(T_1 = \hat{g}^{xv_0k}, T_2 = K\mu/\mu^0, T_3 = K\hat{g}^{xv_0k} \Rightarrow \text{in } \mathbb{G}\)

21. \(\sigma = (x, s_1, s_2, s_3, T_1, T_2, T_3)\)

22. \(b \leftarrow \mathcal{A}_{\text{WEE}}(tr; \xi := \phi)\)

23. **output:**

24. **mode** = ‘\(\text{PC}\)’; \([0, 1] \leftarrow b\) to the challenger of Theorem 3.4.

25. **mode** = ‘\(\text{DDH}\)’; \([0, 1] \leftarrow b\) to the challenger of Definition 3.3.

26. **mode** = ‘\(\text{PC}\)’. Here, if \(C\) is a commitment to \(v_0\) then the transcript \(tr\) is genuinely created. Hence, \(\mathcal{A}_{\text{WEE}}\) accepts \(tr\), i.e., \(\mathcal{A}_{\text{WEE}}(tr; \xi := 1)\) 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, \(\mathcal{A}_{\text{WEE}}\) rejects, i.e., \(\mathcal{A}_{\text{WEE}}(tr; \xi := 0)\) with more than \(1/2 + \epsilon(\lambda)\) probability. Therefore, if \(\mathcal{A}_{\text{WEE}}\) 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.

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

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

**Lemma C.3 (Knowledge Soundness). Rerandomizable accounts provide knowledge soundness of \(\mathcal{R}_{\text{account}}\) if Pedersen commitments are binding.**
We reduce, and Rerandomized accounts are ZKA for exists a rewindable prover (Proof:

Lemma C.4.

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.
\[ H_j = D \sum_{l=0}^{n} \left( \left( G_{m_l} \right)^{2i} \mathcal{C}_l \right) \ldots \mathcal{F}_m \]

\[ = D \prod_{i=0}^{N} \left( V_i \right)^{x_{i,j}} \left( \mathcal{A}^j \right)^{y_{i,j}} \prod_{i=0}^{n} \left( \left( G_i \right)^{z_{i,j}} B_{l,i}^r \mathcal{C}_i^{y_{i,j}} A_{l,i}^n \right)^{y_{i,j}} \]

\[ \times \left( \left( G_i \right) \mathcal{C}_i \right)^{y_{i,j}} \left( \mathcal{A}^j \right)^{y_{i,j}} \prod_{i=0}^{n} \left( h_{l,i} \right) \mu \left( x_{i,j} a_i k_i \left( z_{i,j} \right) y_{i,j} \right) \]

\[ \times \left( h_{l,i} \right) \mu \left( x_{i,j} a_i k_i \left( z_{i,j} \right) y_{i,j} \right) \prod_{i=0}^{n} \left( h_{l,i} \right) \mu \left( x_{i,j} a_i k_i \left( z_{i,j} \right) y_{i,j} \right) \]

\[ = \mu^0 \prod_{l=0}^{n} \left( h_{l,i} \right) \mu^0 \]

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-extracted emulation for relation \( R_{\text{forced}} \) if Pedersen commitments are hiding, one-of-many proofs for zero-value commitments holds ZKA for relation \( R_{\text{zero}} \), and randomized accounts provides (insider-)ZKA for relation \( R_{\text{accounts}} \).

**Proof:** Assume that there exists an adversary \( \mathcal{A}_{\text{WEE}} \) that win the witness-extracted emulation game in Definition 3.5 for relation \( R_{\text{forced}} \) and \( \mathcal{A}_{\text{WEE}} \) controls \( \mathcal{R}_{\text{WEE}} \) accounts such that \( \mathcal{R}_{\text{WEE}} < n \) and \( n < N \). Therefore, the adversary has knowledge \( \xi = (J, \mathcal{R}_{\text{WEE}}, \mathcal{R}_{\text{WEE}}, \mathcal{R}_{\text{accounts}}, \mathcal{R}_{\text{WEE}}) \). When the ZKA of \( R_{\text{WEE}} \) and \( R_{\text{accounts}} \), we reduce \( \mathcal{A}_{\text{WEE}} \) to break the Pedersen commitments’ hiding property as follows:

1. **premessage:** send \((\tilde{0}, \tilde{1}) \in \{0, 2^l\} \) to Theorem 3.4’s challenger.
2. **input:** get \((g, h, C)\) from the challenger in Theorem 3.4.
3. **reduction:** create accounts, proofs, and looks-like/genuine forced openings for \((k_1, r_1, \alpha_1, k_1) \in \{0, 1\}^{0, 1, 0} \times \mathbb{Z}_q \times h_{\text{WEE}} \)
4. \( \alpha_0 = \alpha_1 \) if \( i \in [0, N) \) and \( j = i \) for \( i \in [0, n) \) and \( i' \notin J_{\text{WEE}} \)
5. get indexes \( [j]_{l=0}^{n} \) such that \( [\alpha_0, \alpha_1] \) \( \equiv \) \( n_{l=0}^{n} \) \( \equiv \) \( W_{\text{WEE}} \times \) \( [\mathcal{C}_i]_{l=0}^{n} \)
6. For \( i \in J_{\text{WEE}} \):
   a. \( \mathcal{C}_i = \text{UpdateAsset}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \pi_{\text{WEE}}, \mathcal{C}_i) \)
   b. \( \mathcal{I}_i = \text{UpdateValueMProve}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
7. For \( i \in J_{\text{WEE}} \):
   a. \( \mathcal{I}_i = \text{UpdateAsset}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
   b. \( \mathcal{I}_i = \text{UpdateValueMProve}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
8. \( \mathcal{C}_j = \mathcal{C}_j \times \mathcal{C}_j \times \mathcal{C}_j \times \mathcal{C}_j \)
9. get \((\alpha_0, \alpha_1) \)
10. if \( i \in J_{\text{WEE}} \):
    a. \( \mathcal{C}_i = \text{UpdateAsset}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
    b. \( \mathcal{I}_i = \text{UpdateValueMProve}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
11. \( \mathcal{C}_j = \mathcal{C}_j \times \mathcal{C}_j \times \mathcal{C}_j \times \mathcal{C}_j \)
12. \( \mathcal{I}_i = \text{UpdateAsset}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
13. \( \mathcal{I}_j = \text{UpdateValueMProve}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
14. get \((\alpha_0, \alpha_1) \)
15. if \( i \in \{0, n\} \):
    a. \( \mathcal{C}_i = \mathcal{C}_i \times \mathcal{C}_i \times \mathcal{C}_i \times \mathcal{C}_i \)
    b. \( \mathcal{I}_i = \text{UpdateAsset}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
16. \( \mathcal{I}_i = \text{UpdateValueMProve}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
17. \( \mathcal{I}_i = \text{UpdateAsset}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
18. \( \mathcal{I}_i = \text{UpdateValueMProve}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
19. \( \mathcal{I}_i = \text{UpdateAsset}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
20. \( \mathcal{I}_i = \text{UpdateValueMProve}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
21. \( \mathcal{I}_i = \text{UpdateAsset}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
22. \( \mathcal{I}_i = \text{UpdateValueMProve}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
23. \( \mathcal{I}_i = \text{UpdateAsset}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
24. \( \mathcal{I}_i = \text{UpdateValueMProve}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
25. \( \mathcal{I}_i = \text{UpdateAsset}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)
26. \( \mathcal{I}_i = \text{UpdateValueMProve}(\mathcal{I}_i, \mathcal{I}_j, \mathcal{I}_k, W, \mathcal{I}_i, \mathcal{C}_i, W) \)

Hence, we conclude that Lemma D.1 is true.
We prove the security of balance proofs in this section. When probability since balance proof is created for the correct balance.

extended emulation in Definition 3.5 for emulation for relation

Step 12 is valid as follows:

\[ (\mathcal{V}_3: \mathcal{E}_6: \exists \mathcal{Q}_z) = 1 \]

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:

\[
\mu' - f = E^{-1} U^{\theta} = \left( \prod_{i=0}^{N} K'_i \right)^{-1} \in \mathbb{G}
\]

(1)

\[
= \left( C\delta \prod_{i=0}^{N} V_i \times V'_i \right)^{-1} U^{\theta} \left( \prod_{i=0}^{N} K'_i \right)^{-1} \in \mathbb{G}
\]

(2)

\[
= \left( h^{\delta} \prod_{i=0}^{N} v_i \right)^{-1} \left( \prod_{i=0}^{N} K'_i \right)^{-1} \in \mathbb{G}
\]

(3)

\[
\times h^{\delta} \prod_{i=0}^{N} v'_i \left( \prod_{i=0}^{N} K'_i \right)^{-1} \in \mathbb{G}
\]

(4)

\[
= \mu \sum_{i=0}^{N} v'_i + c - c' \in \mathbb{G}
\]

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

1. **premessage:** send \( (c_0, c_1) \in \{0, 1\} \) to Theorem 3.4’s challenger.
2. **input:** get \( (g, h, \mu) \) from the challenger in Theorem 3.4.
3. **reduction:** create accounts, update proofs, and emulated and genuine proofs for \( (k_0, k_1) \in \{0, 1\} \).
4. \( [ac_1 := (K_j = g^{\theta}, G_1 = g^r, V_j = G_1^{k_j} \mu)]_{i=0}^{N} \)
5. \( [asset_1 := (G'_i, V'_i) := \text{UpdateAsset}(r, u'_i, asset_1)]_{i=0}^{N} \)
6. \( [\sigma_1 := \text{UpdateTimeValueProve}(r', k_0, v'_0, w)]_{i=0}^{N} \)
7. \( y \in \mathbb{Z}_q \) from the verifier
8. \( C' = \hat{C}h^{\delta}c' \)
9. \( E = C C^{-1} \prod_{i=0}^{N} v'_i \times V'^{-1} \in \mathbb{G} \)
10. \( U = U(C^{\theta} h^{\delta}c')^{-1} y \)
11. \( s = (r' - yu) \) and \( s' = (ac_1 - ac_1 - uy) \in \mathbb{Z}_q \)
12. \( w = (acc, asset_1, \sigma_1)_{i=0}^{N} (w, f', c', \pi_{balance} = (U, s, s')) \)
13. \( b = A_{WEE}(((g, h, \mu), w); \zeta = \phi) \)
14. **output:**
15. \( [0, 1] \) b to the challenger of Theorem 3.4

However, \( A_{WEE} \) 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 \( A_{WEE} \) 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 \( \mathcal{V} \) 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

\[
\text{Commit}_{g,h,\mu}(r, t', t'') = g^{\theta} h^{\delta} c' \in \mathbb{G}
\]

1. **input:**
2. get \( (g, h, \mu) \) from the challenger in Theorem 3.4.
3. **reduction:**
4. \( pp = (g, h, \mu) \) and initial witness \( s = \phi \)
5. extract \((r, u, w) \leftarrow \mathcal{E}((\mathcal{P}_{KS}(pp, acc), v), (pp, acc))\) when \( tr = ((acc, asset_1, \sigma)_{i=0}^{N}, w, f', c', \pi_{balance} = (U, s, s'), y) \)
6. \( w = ((k_0, r, v_0, k_1, \zeta, \tau_1), r', u', ac_1, ac_2, c', c) \)
7. such that \((acc := (k_i = g^{\theta}, G_i = g^r, V_i = G_1^{k_i} \mu))_{i=0}^{N} \)
8. \( [asset_1 := (G'_i, V'_i) := \text{UpdateAsset}(r', v'_0, asset_1)]_{i=0}^{N} \)
9. \( [\sigma_1 := \text{UpdateTimeValueProve}(r', k_0, v'_0, w)]_{i=0}^{N} \)
10. \( T_3 = K^{\theta} g^{-\delta} \land G = g^r \land V = VK'^{-1} \mu^{-1} = 1 \Rightarrow \mathcal{V}(r') = 1 \)
12. and \( \neq \) since \((pp, u, w) \notin \mathcal{R}_{account} \) in Equation 1
13. \( (f' - f)^{\Delta} \sum_{i=0}^{N} v_i - \sum_{i=0}^{N} v'_i + c - c' \)
14. \((0 \neq ac_1 - ac_1 - u' y - s') \land (0 \neq (r' - yu) - s') \Rightarrow 1 \)
15. **output:**
16. output: **out**
17. **out**
18. **out** to the challenger of Theorem 3.4.

Here, \( Commit_{g,h,\mu}(r, t', t'') = f = Commit_{g,h,\mu}(r', u', ac_1 - ac_1 - u' y, \sum_{i=0}^{N} v_i - \sum_{i=0}^{N} v'_i + c - c' \)

\( (\bar{s}, s', f' - f) \) is not equal to \( (\bar{s}, s', f' - f) \) if \( (f' - f) \neq \sum_{i=0}^{N} v_i - \sum_{i=0}^{N} v'_i + c - c' \). 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/QUISQUIZ**

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: \((A, B), (A, B), (A, B, C), \) 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
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 tx3. Thus, we can deanonymize the sender of tx3 as C. However, in Nopenena and QuisQuis, C is not connected to tx3 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.