Reusable, Instant and Private Payment Guarantees for Cryptocurrencies

Akash Madhusudan[1], Mahdi Sedaghat[1], Samarth Tiwari[2], Kelong Cong[1], and Bart Preneel[1]

[1] imec-COSIC, KU Leuven, Leuven, Belgium
{akash.madhusudan,ssedagha,kelong.cong,bart.preneel}@esat.kuleuven.be
[2] Centrum Wiskunde & Informatica, Amsterdam, The Netherlands
samarth.tiwari@cwi.nl

Abstract. Despite offering numerous advantages, public decentralized cryptocurrencies such as Bitcoin suffer from scalability issues such as high transaction latency and low throughput. The vast array of so-called Layer-2 solutions tackling the scalability problem focus on throughput, and consider latency as a secondary objective. However, in the context of retail payments, instant finality of transactions is arguably a more pressing concern, besides the overarching concern for privacy.
In this paper, we provide an overlay network that allows privacy-friendly low latency payments in a retail market. Our approach follows that of a recent work called Snappy, which achieved low latency but exposed identities of customers and their transaction histories. Our construction ensures this data is kept private, while providing merchants with protection against double-spending attacks. Although our system is still based upon customers registering with a collateral, crucially this collateral is reusable over time.
The technical novelty of our work comes from randomness-reusable threshold encryption (RRTE), a cryptographic primitive we designed specifically for the following features: our construction provably guarantees payments to merchants, preserves the secret identity of honest customers and prevents their transactions from being linked. We also present an implementation of our construction, showing its capacity for fast global payments in a retail setting with a delay of less than 1 second.

1 Introduction

Public decentralized cryptocurrencies such as Bitcoin and Ethereum offer increased transparency and avoid trust in a central party. However, these advantages come at the cost of performance, rendering them unfit for high throughput, real-time applications. The throughput of cryptocurrencies is orders of magnitude lower than that of traditional payment service providers such as Visa. Further, transactions require time before being considered final: the convention has been to wait for 6 confirmations or approximately an hour. For this reason, securely improving throughput and transaction confirmation latency (hereon referred to as latency) of public cryptocurrencies is a major area of research.

Layer-2 solutions tackle these constraints by offloading some elements of transactions off-chain [41]. However, these solutions focus on maximizing throughput and not on minimizing latency, which is dealt with as a secondary objective. Rollups, an increasingly popular solution in both industry and academia alike, increase the transaction throughput of Ethereum up to 100 times compared to its current throughput [18,32,9]. Yet, their latency matches that of the underlying blockchain. Payment Channel Networks (PCNs), another popular scaling solution, allow for arbitrarily many transactions on their network at the cost of a constant number of on-chain transactions, thereby drastically increasing throughput and reducing transaction fees. Yet, there are various factors negatively impacting the latency of PCN transactions, such as liveness of intermediate nodes [28], and

the route discovery mechanisms [57]. Retail payments is a scenario where instant finality (hereon referred to as fast payments) is of the utmost importance; waiting an hour for a coffee is not an option. Additionally, retail payments are usually unilateral, i.e., from customers to merchants. An acknowledged problem with PCNs is channel depletion i.e., repeated use of channels in the same direction results in depleted channels, prohibiting further payments in the same direction [5]. Unilateral retail transactions only aggravate this issue. Similarly, by updating its layer-1, Ethereum 2.0 has increased its transaction throughput significantly. However, latency still takes $\sim$ 14 minutes [33]. This suggests a decoupling of two performance measures, namely throughput and latency, which need to be tackled separately.

Hence, *collateral reusability* and *fast payments* become interesting properties for solutions that perform retail payments with cryptocurrencies. Snappy is a fast on-chain payment system designed for a retail environment, where payers can assure payees of their ability to pay for a certain commodity [51]. Payees are protected from double-spending at rates much faster than the underlying blockchain while allowing collaterals to be reused. In order to work, Snappy depends on a set of *statekeepers* responsible for proactively detecting double-spending attempts in the system. These statekeepers have access to all transactions made by the customer, either by simply querying the blockchain or locally logging them when they receive it. Thus, their system suffers from privacy issues and all transactions involving the same payer can be linked together by third parties, such as, the statekeepers. The general demand for greater privacy becomes even more relevant in the retail context. For instance, the retail giant Target was able to deduce the pregnancy of a teenager even before her own parents found out [46]. Such unfortunate leaks can be prevented, if merchants are unable to link customers' various purchases. Hence, there is a need for a solution that has the following properties:

P1. Instant payments with double-spending protection that is much faster than the underlying blockchain (*fast payments*).
P2. Prevents third parties in the system from being able to link different transactions involving the same honest payer (*transaction unlinkability*).
P3. Privacy for honest customers is provided efficiently without incurring latency constraints of payments (*efficient privacy*).
P4. Honest payers do not need to replenish their collaterals (*reusability*).

At ICDCS'21, Ng et al. [54] proposed LDSP that adds privacy to Snappy. LDSP achieves **P1**, **P2** and **P3**; however, at the cost of reusable collaterals (**P4**). Thus, previous works either trade-off privacy in order to achieve fast payments with collateral reusability or vice-versa.

Our Contributions. In this paper, we provide a new overlay network that fulfills all aforementioned properties. To the best of our knowledge, we are the first to simultaneously achieve the combination of properties (**P1-4**) in a payment system utilised in a retail context. Our contributions may be split into three as follows:

Private Low Latency Double-Spending Protection. We provide an overlay network capable of providing payees protection from double-spent transactions much faster than the underlying blockchain. No party in the system is able to link different transactions involving the same honest payer, nor do these privacy guarantees adversely affect latency of transactions. Customer collaterals are also *reusable*, so that customers can repeatedly guarantee payments over time.

Formal Security Analysis. We formally prove that our construction preserves the secret identity of honest customers (anonymity) and disables anyone from linking their transactions (unlinkability), yet at the same time guarantees payment to the merchants (payment certainty).

Implementation and evaluation. We implement our construction and show that it allows for fast global retail payments with a delay of less than 1 second.

Outline. The rest of this paper is organized as follows: Section 2 gives an overview of our construction, its participants and how they interact. Section 3 formally describes our construction, its threat model and proves how our construction satisfies its security requirements. Section 4 depicts an efficient instantiation of our construction and lists the cryptographic primitives used in detail. In Section 5 we evaluate the performance of this instantiation while focusing on latency. In Section 6 we discuss our limitations and point out the differences our construction has when compared to similar state-of-the-art research and finally in Section 7 we conclude our paper.

2 Our construction

2.1 Design Motivation

Snappy In this section, we only give a high-level description of a transaction in Snappy in order to point out its privacy shortcomings. For a detailed understanding of their construction, interested readers are referred to the Snappy paper [51].

In Snappy, a customer initiates a payment intent (INT_c) for a merchant of their choice. INT_c can be treated as a *payment guarantee* and is a normal blockchain transaction which is not yet signed by the customer. Hence, it includes typical transaction details such as the pseudonyms of payer and payee, transaction amount, etc. Snappy considers payment privacy to be orthogonal to their work, and claim that it is inherited from the underlying blockchain; however, by sharing INT_c with the statekeepers, payment privacy is also not provided on a protocol-level. In Figure 1a, all interactions impacting protocol-level privacy have been highlighted with a red dotted rectangle. By protocol-level privacy, we mean the privacy for payer and payee interactions from all third parties within the protocol. Even honest-but-curious statekeepers could potentially track all payments a customer makes without the need of eavesdropping, as this data readily becomes available to them.

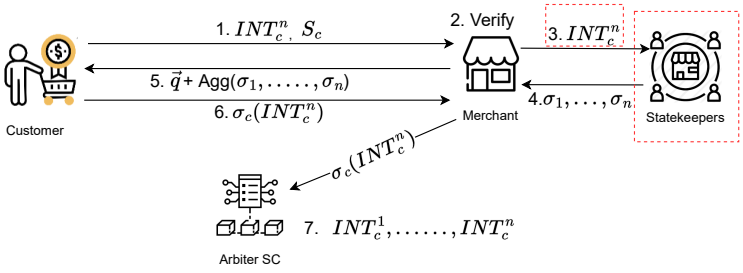
2.2 Strawman Construction

Here, let us see why a naive application of ZKPs to Snappy does not fulfil our desired properties.

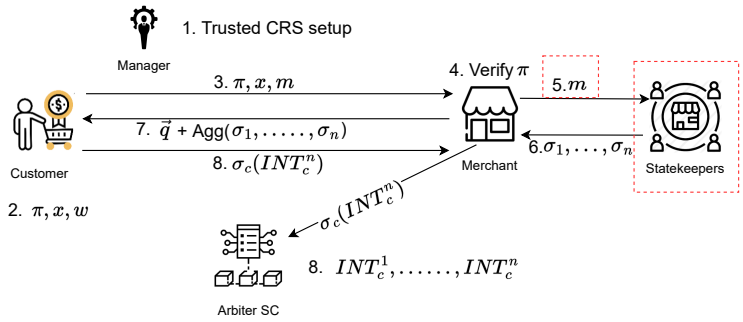
As envisioned by Mavroudis et al. [51], one way to provide on-chain privacy in Snappy is to utilize privacy-preserving blockchains combined with private smart contracts. However, as we mention before, in addition to on-chain privacy, Snappy does not offer any protocol-level privacy as well. Keeping on-chain privacy orthogonal, an obvious solution to provide protocol-level privacy in Snappy would be to replace all sensitive and public information in the transactions on Snappy with zk-SNARKs. Note that, due to the choice of signature scheme in Snappy, i.e., ECDSA [47] and BLS [14], zk-SNARKs are the only option in terms of ZK proofs since less expressive proof systems, such as, Groth-Sahai proofs cannot work in combination with non-algebraic constraints, such as, hash functions.

Such a construction is shown in Figure 1b, which works as follows:

1. A trusted manager performs the one-time trusted setup required to use zk-SNARKs. Note that although alternatives to avoid this trusted setup exist [17,49], their performance is not practical for use in real-time applications.
2. The customer generates a proof π of sufficient funds to pay the merchant, in addition to a statement x and witness $\hat{\mathsf{w}}$.



(a) Snappy overview.



(b) Snappy + zk-SNARKS.

Fig. 1: Snappy's construction and a naive attempt to provide protocol-level privacy.

3. The customer keeps $\hat{w}$ private, and forwards π and x to the merchant, along with a transaction counter m for their new payment intention. Note that m is similar to the index of INT_c in the original Snappy construction.
4. Upon receiving π, x and m, the merchant verifies π and is ensured that the customer has sufficient funds to pay for the services/product.
5. The merchant forwards m to the statekeepers in order for them to check whether they have seen it before or not (proactive double-spending protection).

The rest of the protocol proceeds similar to the original snappy protocol.

Observe that m is an index of the payment intention INT_c; hence, the payments from two different customers can have an identical m. As a result, statekeepers cannot verify whether or not a payment intention is a double-spend without also having access to the customer's identity.

Despite utilizing zk-SNARKs, the customer's identity *cannot* be hidden from statekeepers in Snappy, whilst using their current approach for proactive double-spending. Clearly, there is a need for a new way of (proactively) tracking double-spending attempts in order to preserve protocol-level privacy (in combination with ZKP). Additionally, assuming this new technique was to be included in Snappy's construction, there would still be a big constraint of inefficient ZK proof generation for customers. Since, the time required to generate a proof using zk-SNARKs is linear in the order of circuit size, the customer would spend substantial time in order to create a zk-SNARK for each INT_c.

2.3 Our solution

The strawman solution shown in Section 2.2 depicts the difficulty of a straightforward solution in achieving **P1-4**.

While designing a solution that fulfills all aforementioned properties, we faced two challenges. First, our construction must utilize a proactive double-spending protection mechanism that is able to *selectively* reveal information of dishonest parties while still keeping all information about honest parties private. Second, since the payment latency needs to be low, we must enable payers to prove vital information privately yet very efficiently. We address the first challenge by proposing a novel variant of threshold encryptions, which we hereon refer to as randomness-reusable threshold encryptions (RRTE) that has the unique property of revealing the plaintext if two ciphertexts are encrypted using the same randomness. We use RRTE in combination with Pseudo-Random Functions (PRF) such that the statekeepers are able to proactively protect from double-spend attempts without having knowledge of any transaction details; yet, are still able to reveal vital information in the case of double-spent transactions. The second challenge is addressed by a combination of Threshold Structure-Preserving Signatures (TSPS) [24] and Non-Interactive Zero-Knowledge Proofs (NIZKs). The compatibility of TSPS with efficient NIZKs enables a payer to prove vital information to payees in our system in zero-knowledge, without impacting the latency of transactions.

Participants. First, in order to explain the interplay of these cryptographic primitives that solve the aforementioned challenges and fulfill **P1-4**, we introduce the participants of our construction. Similar to Snappy, our participants include: (1) customers willing to purchase products/services using their cryptocurrencies while expecting low latency; (2) an established consortium of merchants willing to accept cryptocurrency payments and (3) statekeepers who are selected from the merchant consortium. Additionally, we also include a group of authorities *trusted* for registration of users and verification of collaterals. Although authorities have greater power during the setup of our network, they *do not* have any access to future transactions between a customer and a merchant. For more discussion on these trust assumptions see Section 6.

Interaction of Participants. In our construction, the participants function as follows; each *customer* in our systems owns collateral(s), which is uniquely linked to a secret PRF key(s). This secret PRF key is signed by the group of *authorities* by utilizing TSPS once they verify its existence on the arbiter smart contract. The PRF is used in combination with our RRTE to set up our double-spending protection. By using this signed secret PRF key, each customer generates a randomness, which they then use in combination with the public key of *target merchant* to encrypt their secret identity. For encryption we use RRTE, that reveals the secret identity of a malicious customer (plaintext) when they double-spend. More precisely, double-spending in our system amounts to certifying multiple transactions with the same collateral, by which RRTE reveals the dishonest customer's identity. Thus, our construction fulfills **P1** by enabling any *statekeeper* who receives two ciphertexts that are encrypted using the same randomness to catch double-spending and reveal the identity of the perpetrator. However, honest customers who only certify one transaction per collateral remain unaffected; hence, also fulfilling **P2**.

Next, the customer needs to convince the target merchant about the existence of their collateral. However, they must do this without revealing any identifying details. To achieve this, we utilize TSPS and NIZKs. The payment guarantee is an indirect proof of collateral by proving knowledge of the TSPS provided to them by the authorities. In case of a double-spend attempt, this payment guarantee is sufficient for the merchant to be reimbursed. This can be done efficiently by efficient proof systems fulfilling **P3**.

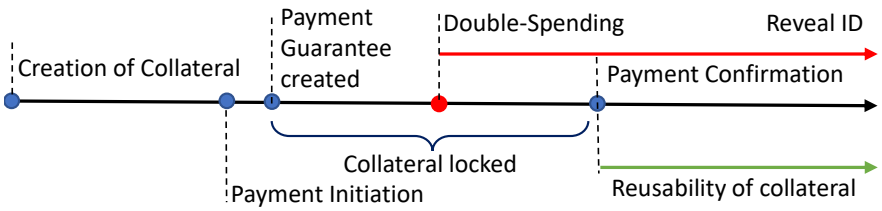


Fig. 2: Timeline of a transaction.

Finally, our construction also allows a customer to *reuse* their collaterals after a certain time interval in order to achieve **P4**. As illustrated in Figure 2, a collateral is considered *locked* for the time period between the customer sending a payment guarantee to the merchant and the actual confirmation of their blockchain transaction. However, after the confirmation of a blockchain transaction, the customer's collateral can be reused. Note that this is similar to the execution of a multi-hop payment on PCNs, that rely on Hashed-Time-Locked-Contracts, locking up the collateral for a similar time period [57,3]. Thus, using a collateral twice in quick succession is treated as a double-spend and can be detected, but it may be reused once the locking period of the collateral has elapsed.

Construction Overview. Figure 3 illustrates our construction. In order to explain our construction, we assume an established set of authorities and a fixed merchant consortium. The payment protocol between a customer and a merchant has off-chain and on-chain components that have been depicted in Figure 3 with dotted and solid arrows respectively. The protocol proceeds as follows:

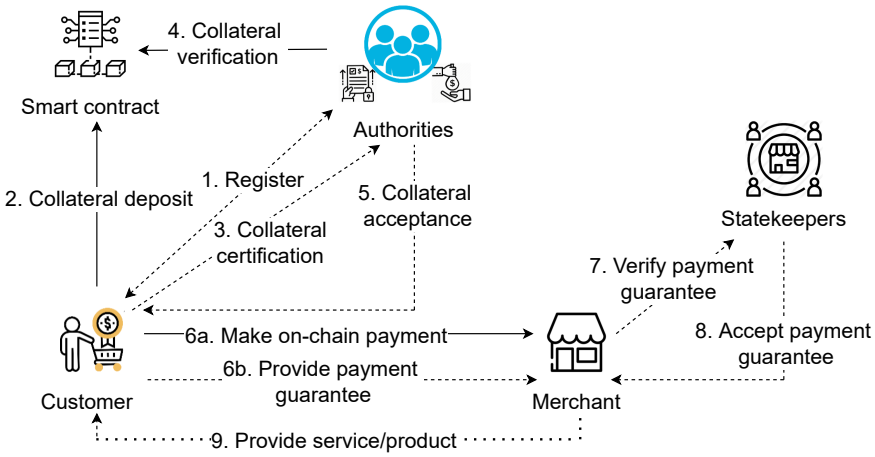


Fig. 3: Our construction.

1) The customer begins by registering themselves with a set of authorities. **2)** The customer deposits a collateral in the smart contract, controlled by this set of authorities. **3)** Once the deposit is confirmed on the blockchain, the customer requests a collateral certification, which is necessary to generate payment guarantees for merchants. **4)** Upon receiving the certification request of the customer, the authorities check the smart contract to confirm if the customer deposited a collateral. **5)** The authorities provide the customer with a signed collateral certification. **6a)** During

a transaction in the retail market, the customer makes a payment to the target merchant using the underlying cryptocurrency. This is done by sending a payment to the merchant through the smart contract. **6b)** The customer simultaneously generates an encrypted *payment guarantee* by using the collateral certification and sends this to the target merchant along with the transaction identifier of their cryptocurrency payment. **7)** The target merchant forwards this encrypted *payment guarantee* to the statekeepers who individually confirm that it is not a double-spend attempt. The statekeepers compare the received guarantee with each guarantee they received within a fixed time period. If none of these comparisons reveal the plaintext, i.e, secret identity of the customer, they are guaranteed about its uniqueness. Note that this verification can be done by utilizing our proposed RRTE and does not require knowing a customer's identity. **8)** Each statekeeper returns a signed payment guarantee to the merchant. **9)** If a majority of the statekeepers return a confirmation, the merchant aggregates these signatures and accepts the payment guarantee and provides the customer with necessary services/products.

Double-Spending. Double-spending in our construction means a scenario where the customer is malicious and wants to double-spend their payment guarantee, i.e., use the same collateral to promise payments to two distinct merchants. In this case, when the statekeepers receive these guarantees from two distinct merchants, they are able to combine them and reveal the secret identity of the cheating customer. Along with this identity, a secret to the customer's collateral is also revealed. This secret is used by the victim merchant for remuneration. This is depicted in Figure 4

More details about how RRTE enables statekeepers to reveal the secret identity of a malicious customer are given in its formal definition in Section 4.

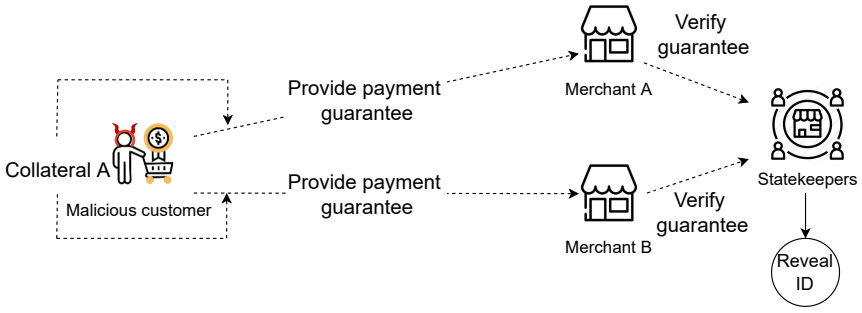


Fig. 4: Proactive double-spending detection.

Limitations. Although we address the privacy shortcomings of Snappy's design, our construction still inherits some of its other limitations, such as, the large collateral requirement for statekeepers and inadequately defined user incentives. To be specific, we also require the merchants to deposit collaterals in order to play the role of statekeepers fairly. We also assume that the amount of a customer's deposit is *fixed*. The assumption of a fixed set of merchants is also an additional concern, since it does not allow for any merchant churn in the protocol. However, in this paper we do not attempt to address these concerns. A more thorough discussion about these shortcomings is given in Section 6.

3 Preliminaries and Formal Construction

Throughout this paper, we let the security parameter of the scheme to be λ with unary representation of 1^λ, and $\mathsf{negl}(\lambda)$ denotes a negligible function. We use $x \leftarrow\!\!\$\, X$ to denote that x is sampled uniformly from the set X. $[n]$ denotes the set of integers in the range of 1 to n. For clarity, the secret values in our construction are represented with a *hat* operator (e.g., $\hat{\mathsf{sk}}$) and masked values are represented with the notation x' for the value x.

Threat Model. Our construction is designed to resist active adversaries that can corrupt a set of customers, merchants (which are also statekeepers) and authorities. To be precise, an adversary can only corrupt a minority of authorities during the initialization phase. During the processing of transactions, the adversary can corrupt all but one customers, and a minority of merchants.

Network Assumptions. We assume the existence of secure and reliable communication channels so that parties receive messages sent by honest parties eventually. The honest merchants/statekeepers are also assumed to be live and responsive, for the verification of transactions. We do not expect the customers to be online, unless they need to transact with a merchant. The underlying blockchain is assumed to be persistent and live and the adversary cannot influence the consensus mechanism.

Beyond the threat model. Let us mention a few considerations that are outside the scope of our model. Firstly, we consider protocol-level privacy, and not the privacy of the underlying blockchain. We do not consider side-channel attacks, such as metadata-level attacks on communication channels. Finally, the selection of authorities is also orthogonal to our work. One possibility is to choose them from the set of reputable merchants in the consortium, and have them shuffled periodically to ensure a majority of authorities always remain honest.

We would also like to note that weaker assumptions of trust or liveness are possible without compromising security. However, we select a simple set of assumptions, following those of Snappy, in order to focus on the privacy aspect instead of system-level optimizations.

Formal Construction. Our construction builds on Pseudo-Random Function (PRF), Non-Interactive Zero-Knowledge (NIZK) arguments, Digital Signatures (DS), Threshold Structure-Preserving Signatures (TSPS), Commitments (CO), and a novel randomness-reusable Threshold Encryption (RRTE). We list the formal definition and the security properties in Appendix A and outline the scheme in Algorithm 1 for a relation $\mathbf{R_L}$ over three main NP-languages $\mathbf{L} := (\mathbf{L}_1, \mathbf{L}_2, \mathbf{L}_3)$. The list of master public keys, mpk, is considered as an implicit input for all algorithms except the parameter generation algorithm (PGen). Additionally, all algorithms are PPT unless otherwise specified. We now formalise the functions in the Algorithm 1. All functions are split based on when they happen in our construction and are formalized as follows:

Bootstrapping Phase as depicted in Figure 3:

- $\mathsf{PGen}(1^\lambda, \mathbf{R_L})$ takes λ in its unary representation and relation $\mathbf{R_L}$ as inputs and returns the master public key mpk.
- $\mathsf{AuKeyGen}(\mathcal{AU})$ is executed by $\mathcal{AU}$ that returns $(\hat{\mathsf{sgk}}_{ai}, \mathsf{vk}_{ai})$ for $i \in [n]$ along with a global verification key vk_a. $\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}, \mathsf{vk}_{ai}, \mathsf{vk}_a]_{i=1}^n$ represents the list of credentials for $\mathcal{AU}$.
- $\mathsf{MKeyGen}(\mathsf{M}_m)$ is executed by merchant $\mathsf{M}_m \in \mathcal{M}$ in order to join the network. It initially generates a pair of signing/verification keys $(\hat{\mathsf{sgk}}_{bm}, \mathsf{vk}_{bm})$ and returns a tuple $(\hat{\mathsf{sgk}}_{bm}, \mathsf{vk}_{bm}, \mathsf{pk}_{bm} = \perp)$. The list of keys belonging to the group of merchants is recorded in $\mathcal{M}[\hat{\mathsf{sgk}}_{bi}, \mathsf{vk}_{bi}]_{i=1}^\ell$.
- $\mathsf{MRegister}(\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]_{i=1}^t, \mathcal{M}[\mathsf{vk}_{bi}]_{i=1}^\ell)$ is executed by any subset of $\mathcal{AU}$ of size at least t to register the merchants who deposit a collateral to join the merchant consortium and assign them a public key pk_{bm}. For each merchant $\mathsf{M}_m \in \mathcal{M}$, it takes the secret signing key of the authorities $\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]$

for $1 \leq i \leq t$, and returns public key pk_{bm} as output. After this phase, the list of parameters for the m^{th} merchant can be updated as $\mathcal{M}[\hat{\mathsf{sgk}}_{bm}, \mathsf{vk}_{bm}, \mathsf{pk}_{bm}]$.

- $\mathsf{CuKeyGen}(\mathsf{C}_n)$ is executed by the customers, that for each customer $\mathsf{C}_n \in \mathcal{C}$, a Pseudo-ID generator function (PID) generates an initial secret key $\hat{\mathsf{sk}}_{cn}$ and its corresponding public key pk_{cn}. It returns a tuple of $(\mathsf{pk}_{cn}, \hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn} = \perp)$ with a NIZK proof π_1 to prove that the relation $\mathbf{R}_{\mathbf{L}_1}$ fulfills. The list of customers' keys are kept in $\mathcal{C}[\hat{\mathsf{sk}}_{ci}, \mathsf{pk}_{ci}, \hat{\mathsf{cert}}_{ci} = \perp]_{i=1}^{k}$.

Protocol as depicted in Figure 3:

- $\mathsf{CuRegister}(\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]_{i=1}^{t}, \mathcal{C}[\mathsf{pk}_{cn}], \pi_1)$ is depicted in step 1 of Figure 3. It is executed by any subset of $\mathcal{AU}$ of size at least t to certify the public key pk_{cn} of a customer C_n corresponds to some secret value $\hat{\mathsf{sk}}_{cn}$ under the relation $\mathbf{R}_{\mathbf{L}_1}$. Once the customer C_n is registered by the authorities, it receives a certificate $\hat{\mathsf{cert}}_{cn}$ and it updates $\mathcal{C}[\mathsf{pk}_{cn}, \hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn}]$.
- $\mathsf{CuCreate}(\mathcal{C}[\hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn}])$ is depicted in step 2-3 of Figure 3. It is executed by the customer to request for certification of their collateral. A successfully registered customer C_n, with certificate $\hat{\mathsf{cert}}_{cn} \neq \perp$, can deposit collaterals in the smart contract. For each deposit j in the smart contract, the customer samples a random value k_j from a uniform distribution $\mathcal{K}_{\mathsf{PRF}}$ in a way that the deposit is not directly linkable to the customer. Then it returns a tuple $\mathcal{CL}[\hat{k}_j, k'_j, \perp]$ as an uncertified collateral along with a proof π_2 depicting the fact that the relation $\mathbf{R}_{\mathbf{L}_2}$ fulfills.
- $\mathsf{AuCreate}(\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]_{i=1}^{t}, \mathcal{C}[\mathsf{cert}'_{cn}], \mathcal{CL}[\hat{k}_j, k'_j], \pi_2)$ is depicted in step 4-5 of Figure 3. It is executed by a group of authorities $\mathcal{AU}$ of size at least t. It takes the authorities' secret signing keys ($\hat{\mathsf{sgk}}_{ai}$), an indexed DH message space of PRF key and a NIZK proof π_2 as inputs. To create a certified collateral, it checks the validity of the proof π_2 and whether this collateral exists in the smart contract, and returns certificate $\hat{\mathsf{cert}}_j$ as output. The list of parameters for each collateral is kept by $\mathcal{CL}[\hat{k}_j, k'_j, \hat{\mathsf{cert}}_j]$.
- $\mathsf{Spend}(\mathcal{C}[\hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn}], \mathcal{CL}[\hat{k}_j, \hat{\mathsf{cert}}_j], \mathcal{M}[\mathsf{pk}_{bm}], \mathsf{t})$ is depicted in step 6a and 6b of Figure 3. It is executed by a customer $\mathsf{C}_n \in \mathcal{C}$ who performs a payment to the merchant $\mathsf{M}_m \in \mathcal{M}$ using the underlying cryptocurrency of their choice at time t. The registered customer uses a certified collateral $\mathcal{CL}[\hat{k}_j, \hat{\mathsf{cert}}_j]$ to provide a payment guarantee to the merchant M_m. The payment made by the customer is always bounded by a publicly known collateral amount. It returns the transaction details as a list of parameters $\mathcal{T}[\mathsf{x}_3, \pi_3, Tx_{ID}]$, which contains a pair of instance and proof (x_3, π_3), along with a set of auxiliary data R_{t}. This function is executed in parallel with an on-chain payment. In particular, the customer must first sign and broadcast an on-chain transaction Tx, and then include its identifier (Tx_{ID}) in the payment guarantee of Spend. The Tx_{ID} could have different formats depending on the underlying blockchain.[3]
- $\mathsf{Vf}(\mathcal{M}[\mathsf{pk}_{bm}], \mathcal{T}[\pi_m, \mathsf{x}_m, Tx_{ID}], \mathsf{t})$ is depicted in step 7-9 of Figure 3. The merchant $\mathsf{M}_m \in \mathcal{M}$ executes it to check the validity of a received payment guarantee. Once the proof is verified successfully by merchant M_m along with the majority of statekeepers, StK, confirmation that they have not seen a similar payment guarantee in the current epoch (by providing their signatures), the merchant verifies their individual signatures and aggregates them. Once the aggregation is complete, and if Tx_{ID} specifies the merchant's address as the receiver of funds, the merchant provides the items/services to the customer without waiting for the transaction confirmation of

[3] For a public blockchain such as Bitcoin or Ethereum, Tx_{ID} could be the hash of a transaction (H[Tx]); for an anonymous blockchain like Zcash, it could be the viewing key of the transaction that enables the merchant to check if he is the receiver of the shielded transaction [29]

Algorithm 1: OUR CONSTRUCTION.

Function $\mathtt{PGen}(1^\lambda, \mathbf{R_L})$**:**
- $(\vec{\mathsf{crs}}, \vec{\mathsf{ts}}, \vec{\mathsf{te}}) \leftarrow \mathcal{ZK}.\mathsf{K}_{\vec{\mathsf{crs}}}(1^\lambda, \mathbf{R_L})$
- $(\mathsf{pp}) \leftarrow \mathcal{TSPS}.\mathsf{Setup}(1^\lambda)$
- **return** $\mathsf{mpk} := (\mathsf{pp}, \vec{\mathsf{crs}})$

Function $\mathtt{AuKeyGen}(\mathcal{AU})$**:**
- $(\vec{\hat{\mathsf{sgk}}}_a, \vec{\mathsf{vk}}_a, \mathsf{vk}_a) \leftarrow \mathcal{TSPS}.\mathsf{KGen}(\mathsf{mpk}, n, t)$
- **return** $(\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}, \mathsf{vk}_{ai}, \mathsf{vk}_a]_{i=1}^n)$

Function $\mathtt{MKeyGen}(\mathsf{M}_m)$**:**
- $(\hat{\mathsf{sgk}}_{bm}, \mathsf{vk}_{bm}) \leftarrow \mathcal{DS}.\mathsf{KGen}(\mathsf{pp})$
- **return** $(\mathcal{M}[\hat{\mathsf{sgk}}_{bm}, \mathsf{vk}_{bm}])$

Function $\mathtt{MRegister}(\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]_{i=1}^t, \mathcal{M}[\mathsf{vk}_{bi}]_{i=1}^\ell)$**:**
- **for** $j \in \mathrm{range}(\ell)$ **do**
  - $(\mathsf{pk}_{bj}, \mathsf{pk}_b) \leftarrow \mathcal{RRTE}.\mathsf{KGen}(\mathsf{mpk}, \ell, t, 2)$
- **return** $(\mathcal{M}[\mathsf{pk}_{bi}]_{i=1}^\ell, \mathsf{pk}_b)$

Function $\mathtt{CuKeyGen}(\mathsf{C}_n)$**:**
- $\hat{\mathsf{sk}}_{cn} := \mathsf{PID}(\mathsf{C}_n) \in \mathbb{Z}_\mathsf{p}^*$
- $\mathsf{sk}'_{cn} \leftarrow \mathcal{CO}.\mathsf{Com}(\mathsf{pp}, \hat{\mathsf{sk}}_{cn})$
- $\mathsf{pk}_{cn} := (\mathsf{sk}'_{cn}, M_1, M_2) \leftarrow \mathsf{iDH}^{\mathsf{H}}(\mathsf{sk}'_{cn}, \hat{\mathsf{sk}}_{cn})$
- $\mathsf{x}_1 = (\mathsf{pk}_{cn})$
- $\hat{\mathsf{w}}_1 = (\hat{\mathsf{sk}}_{cn})$
- $\pi_1 \leftarrow \mathcal{ZK}.\mathsf{P}(\mathbf{R}_{\mathbf{L}_1}, \vec{\mathsf{crs}}, \mathsf{x}_1, \hat{\mathsf{w}}_1)$
- **return** $(\mathcal{C}[\mathsf{pk}_{cn}, \hat{\mathsf{sk}}_{cn}], \pi_1)$

Function $\mathtt{CuRegister}(\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]_{i=1}^t, \mathcal{C}[\mathsf{pk}_{cn}], \pi_1)$**:**
- **if** $\mathcal{ZK}.\mathsf{Vf}(\mathbf{R}_{\mathbf{L}_1}, \vec{\mathsf{crs}}, \mathsf{x}_1, \pi_1) = 1$ **then**
  - $(\hat{\mathsf{cert}}_{cn}) \leftarrow \mathcal{TSPS}.\mathsf{Sign}(\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]_{i=1}^t, \mathsf{pk}_{cn})$
- **return** $(\mathcal{C}[\hat{\mathsf{cert}}_{cn}])$

Function $\mathtt{CuCreate}(\mathcal{C}[\hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn}])$**:**
- $\hat{k}_j \leftarrow \mathcal{PRF}.\mathsf{KGen}(\mathsf{pp})$
- $k'_j \leftarrow \mathcal{CO}.\mathsf{Com}(\mathsf{pp}, \hat{k}_j)$
- $M_j := (k'_j, M_1, M_2) \leftarrow \mathsf{iDH}^{\mathsf{H}}(k'_j, k_j)$
- $\mathsf{x}_2 = (k'_j)$
- $\hat{\mathsf{w}}_2 = (\hat{k}_j, \hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn})$
- $\pi_2 \leftarrow \mathcal{ZK}.\mathsf{P}(\mathbf{R}_{\mathbf{L}_2}, \vec{\mathsf{crs}}, \mathsf{x}_2, \hat{\mathsf{w}}_2)$
- **return** $(\mathcal{CL}[\hat{k}_j, M_j], \mathcal{C}[\mathsf{cert}'_{cn}], \pi_2)$

Function $\mathtt{AuCreate}(\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}, \mathsf{vk}_a]_{i=1}^t,$
$\mathcal{C}[\mathsf{cert}'_{cn}], \mathcal{CL}[\hat{k}_j, k'_j], \pi_2)$**:**
- **if** $\mathcal{ZK}.\mathsf{Vf}(\mathbf{R}_{\mathbf{L}_2}, \vec{\mathsf{crs}}, \mathsf{x}_2, \pi_2) = 1$ **then**
  - $(\hat{\mathsf{cert}}_j) \leftarrow \mathcal{TSPS}.\mathsf{ParSign}(\mathcal{AU}[\hat{\mathsf{sgk}}_{ai}]_{i=1}^t, M_j)$
- **return** $(\mathcal{CL}[\hat{\mathsf{cert}}_j])$

Function $\mathtt{Spend}(\mathcal{C}[\hat{\mathsf{cert}}_{cn}, \hat{\mathsf{sk}}_{cn}],$
$\mathcal{CL}[\hat{k}_j, \hat{\mathsf{cert}}_j], \mathcal{M}[\mathsf{pk}_{bm}], \mathsf{t})$**:**
- $(r_\mathsf{t}) \leftarrow \mathsf{PRF}_{\hat{k}_j}(\mathsf{t})$
- $R_\mathsf{t} := e(r_\mathsf{t}, h)$ ▷ h is the generator of $\mathbb{G}_2$.
- $\mathtt{Ct}_m \leftarrow \mathcal{RRTE}.\mathsf{Enc}(\mathsf{pk}_{bm}, \hat{\mathsf{sk}}_{cn}; r_\mathsf{t})$
- $\hat{\mathsf{w}}_3 = (\hat{\mathsf{cert}}_j, \hat{k}_j, r_\mathsf{t}, \hat{\mathsf{sk}}_{cn})$
- $\mathsf{x}_3 = (R_\mathsf{t}, \mathtt{Ct}_m, \mathsf{t})$
- $\pi_3 \leftarrow \mathcal{ZK}.\mathsf{P}(\mathbf{R}_{\mathbf{L}_3}, \vec{\mathsf{crs}}, \mathsf{x}_3, \hat{\mathsf{w}}_3)$
- **return** $(\mathcal{T}[\pi_3, \mathsf{x}_3, Tx_{ID}])$

Function $\mathtt{Vf}(\mathcal{M}[\mathsf{pk}_{bm}], \mathcal{T}[\pi_3, \mathsf{x}_3, Tx_{ID}], \mathsf{t})$**:**
- **if** $\mathcal{ZK}.\mathsf{Vf}(\mathbf{R}_{\mathbf{L}_3}, \vec{\mathsf{crs}}, \mathsf{x}_3, \pi_3) = 1$ **then**
  - **for** $i \in \mathsf{StK}$ **do**
    - **if** $R_\mathsf{t} \notin \mathcal{L}_i$ **then**
      - $(\sigma_{R_\mathsf{t},i}) \leftarrow \mathcal{DS}.\mathsf{Sign}(\hat{\mathsf{sgk}}_{bi}, R_\mathsf{t})$
  - **if** $\mathcal{DS}.\mathsf{Vf}(\mathsf{vk}_{bi}, \sigma_{R_\mathsf{t},i}) = 1 \ \wedge \ |\sigma_{R_\mathsf{t}}| \geq (|\mathsf{StK}|/2) + 1$ **then**
    - **return** 1

Function $\mathtt{RevealID}(\mathtt{Ct}_m, \mathtt{Ct}_{m'}, v)$**:**
- $(\hat{\mathsf{sk}}_{cn}) \leftarrow \mathcal{RRTE}.\mathsf{Dec}(\mathtt{Ct}_m, \mathtt{Ct}_{m'}, v)$
- **return** (sk_{cn})

the customer's original payment on the blockchain. If the proof verification fails, or the majority of statekeepers do not confirm the guarantee, or the on-chain transaction specifies the wrong receiver address, the merchant rejects the payment guarantee.

Double-spend detection as depicted in Figure 4:

- $\mathsf{RevealID}(\mathtt{Ct}_m, \mathtt{Ct}_{m'}, v)$ is a deterministic algorithm that takes two ciphertexts $\mathtt{Ct}_m$ and $\mathtt{Ct}_{m'}$ generated under the public key of two distinct merchants M_m and $\mathsf{M}_{m'}$ and returns the plaintext, i.e., the identity of the customer and the secret to their collateral to redeem it. This ID, sk_{cn}, is no longer hidden and can be used by $\mathcal{AU}$ to blacklist the cheating customer and its collateral(s) can be used to remunerate the victim merchant.

3.1 NIZK Languages

In the proposed generic construction in Algorithm 1 we rely on three languages for the NIZK systems, described below.

– Language $\mathbf{L}_1$: Used to prove the correct formation of the customers' public key pk_{cn}, based on the knowledge of secret key $\hat{\mathsf{sk}}_{cn}$. We depict this language formally below, which is used during $\mathsf{CuKeyGen}(\mathsf{C}_n)$.

$$\mathbf{L}_1 = \mathsf{NIZK}\big\{(\hat{\mathsf{sk}}_{cn}) \mid \mathsf{sk}'_{cn} := \mathcal{CO}.\mathsf{Com}(\hat{\mathsf{sk}}_{cn})\big\}$$

– Language $\mathbf{L}_2$: Used to prove eligibility to request a collateral by deriving certificate fulfillment. This language is used during $\mathsf{CuCreate}(\mathcal{C}[\hat{\mathsf{cert}}_{cn}])$.

$$\mathbf{L}_2 = \mathsf{NIZK}\big\{(\hat{k}_j, \hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn}) \mid k'_j := \mathcal{CO}.\mathsf{Com}(\hat{k}_j), \hat{k}_j \in \mathcal{K}_{\mathsf{PRF}}, \mathcal{TSPS}.\mathsf{Vf}(\mathsf{pk}_{cn}, \hat{\mathsf{cert}}_{cn}) = 1\big\}$$

– Language $\mathbf{L}_3$: Used to prove the possession of a valid collateral, correctness of PRF evaluation algorithm and RRTE's ciphertext. This language is used during $\mathsf{Spend}(\mathcal{C}[\hat{\mathsf{sk}}_{cn}, \hat{\mathsf{cert}}_{cn}], \mathcal{CL}[\hat{k}_j, \hat{\mathsf{cert}}_j], \mathcal{M}[\mathsf{pk}_{bm}], \mathsf{t})$.

$$\begin{aligned}\mathbf{L}_3 =& \mathsf{NIZK}\big\{(\hat{\mathsf{cert}}_j, \hat{k}_j, r_\mathsf{t}, \hat{\mathsf{sk}}_{cn}) \mid r_\mathsf{t} \leftarrow \mathsf{PRF}_{\hat{k}_j}(\mathsf{t}), R_t := e(r_t, h),\\ & \mathtt{Ct}_m := \mathcal{RRTE}.\mathsf{Enc}(\mathsf{pk}_{bm}, \hat{\mathsf{sk}}_{cn}; r_\mathsf{t}), \mathcal{TSPS}.\mathsf{Vf}(M_j, \hat{\mathsf{cert}}_j) = 1\big\}\end{aligned}$$

3.2 Security Analysis

Next we formally define the two main security requirements for our construction, namely (1) Anonymity of honest customers and Unlinkability of payment guarantees, and (2) Payment certainty for honest merchants. Note that the $\mathsf{AllGen}(.)$ algorithm (see Figure 5) generates all system setup parameters at once. In the described definitions, it is implicitly assumed that there exists a PPT adversary $\mathcal{A}$ who has access to the following oracles provided by the challenger $\mathcal{B}$:

– **Oracle** $\mathcal{O}_{\mathsf{AuCorrupt}}(Au_i)$: By calling this oracle under the input Au_i, $\mathcal{A}$ can corrupt Au_i and receive its internal states. The set of corrupted authorities is denoted by $\mathcal{AU}'$ and we have $|\mathcal{AU}'| < t$.
– **Oracle** $\mathcal{O}_{\mathsf{CuCorrupt}}(.)$: Adversary $\mathcal{A}$ can corrupt any customer $\mathsf{C}_n \in \mathcal{C}$ by querying this oracle, and receive its uncertified secret key $\hat{\mathsf{sk}}_{cn}$.

- **Oracle** $\mathcal{O}_{\mathsf{ColCorrupt}}(.)$: $\mathcal{A}$ can corrupt at most q_D collaterals $CL_j \in \mathcal{CL}$ to receive their uncertified secret value $\hat{k}_j$. The list of corrupted collaterals is represented by $\mathcal{CL}'$.
- **Oracle** $\mathcal{O}_{\mathsf{MCorrupt}}(.)$: Adversary $\mathcal{A}$ can corrupt a minority set of merchants (statekeepers) like $\mathsf{M}_m \in \mathcal{M}$ and receives its pair of public key pk_{bm} and secret signing key $\hat{\mathsf{sgk}}_{bm}$. The list of corrupted merchants is denoted by $\mathcal{M}'$ s.t. we have, $|\mathcal{M}'| < |\mathsf{stk}|/2$.
- **Oracle** $\mathcal{O}_{\mathsf{Revoke}}(.)$: Adversary $\mathcal{A}$ can revoke at most q_R certified collaterals $CL_j \in \mathcal{CL}$ and redeem the deposited money.
- **Oracle** $\mathcal{O}_{\mathsf{Spend}}(.)$: $\mathcal{A}$ can make at most q_S payment guarantees created by any arbitrary non-corrupted customer to any non-corrupted merchant.

Definition 1 (Payment Unlinkability and Anonymity). *This construction preserves the anonymity of honest customers and provides unlinkability of payment guarantees, if no PPT adversary $\mathcal{A}$ by getting access to $\mathcal{O}_{\mathsf{AuCorrupt}}, \mathcal{O}_{\mathsf{CuCorrupt}}, \mathcal{O}_{\mathsf{MCorrupt}}, \mathcal{O}_{\mathsf{Spend}}$ oracles, $\mathcal{O}_{\mathsf{ANON}}$ in short, and with advantage of $Adv^{\mathrm{ANON}}_{\mathcal{A}}(\lambda, \beta) = 2\left((\mathrm{Exp}^{\mathrm{ANON}}_{\mathcal{A}}(1^\lambda, \beta) = 1) - 1/2\right)$, has a non-negligible chance of winning the experiment described in Figure 5, i.e. we have, $\left|Adv^{\mathrm{ANON}}_{\mathcal{A}}(\lambda, \beta = 0) - Adv^{\mathrm{ANON}}_{\mathcal{A}}(\lambda, \beta = 1)\right| \leq \mathsf{negl}(\lambda)$.*

$\mathrm{Exp}^{\mathrm{ANON}}_{\mathcal{A}}(1^\lambda, \beta)$

1 : $(\mathsf{mpk}, \mathsf{H}, \mathcal{AU}[\hat{\mathsf{sgk}}_{ai}, \mathsf{vk}_{ai}, \mathsf{vk}_a]_{i=1}^n, \mathcal{M}[\hat{\mathsf{sgk}}_{bi}, \mathsf{vk}_{bi}, \mathsf{pk}_{bi}]_{i=1}^\ell) \leftarrow \mathsf{AllGen}(1^\lambda, \mathbf{R_L})$
2 : $(\mathsf{M}_m, \hat{\mathsf{sk}}_0^*, \hat{\mathsf{sk}}_1^*, \hat{k}_0^*, \hat{k}_1^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{ANON}}}(\mathsf{mpk})$
3 : $\beta \leftarrow\$ \{0, 1\}$
4 : $(\pi_\beta, \mathsf{x}_\beta, R_{\mathsf{t},\beta}) \leftarrow \mathsf{Spend}\big(\mathcal{C}[\hat{\mathsf{cert}}_{c\beta}^*, \hat{\mathsf{sk}}_\beta^*], \mathcal{CL}[\hat{k}_\beta^*, \hat{\mathsf{cert}}_\beta^*], \mathcal{M}[\mathsf{pk}_{bm}], \mathsf{t}\big)$
5 : $\beta' \leftarrow\$ \mathcal{A}^{\mathcal{O}_{\mathsf{ANON}}}(\mathcal{T}[\pi_\beta, \mathsf{x}_\beta, R_{\mathsf{t},\beta}])$
6 : **return** $\beta' == \beta$

$\mathrm{Exp}^{\mathrm{PC}}_{\mathcal{A}}(1^\lambda)$

1 : $(\mathsf{mpk}, \mathsf{H}, \mathcal{AU}[\hat{\mathsf{sgk}}_{ai}, \mathsf{vk}_{ai}, \mathsf{vk}_a]_{i=1}^n, \mathcal{M}[\hat{\mathsf{sgk}}_{bi}, \mathsf{vk}_{bi}, \mathsf{pk}_{bi}]_{i=1}^\ell) \leftarrow \mathsf{AllGen}(1^\lambda, \mathbf{R_L})$
2 : $(\pi^*, \mathsf{x}^*, R_{\mathsf{t}}^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathrm{PC}}}(\mathsf{mpk})$
3 : **if** $\mathcal{ZK}.\mathsf{Vf}(\mathbf{R_L}, \vec{\mathsf{crs}}, \pi^*, \mathsf{x}^*) == 1 \ \wedge \ q_D < q_S + q_R$: **return** 1
4 : **else return** 0

Fig. 5: Security Games.

Definition 2 (Payment Certainty). *This construction provides payment certainty (PC) if no transaction τ is approved with a non-negligible advantage s.t. $q_S + q_R + \tau > q_D$. No PPT adversary $\mathcal{A}$ with access to $\mathcal{O}_{\mathsf{AuCorrupt}}, \mathcal{O}_{\mathsf{CuCorrupt}}, \mathcal{O}_{\mathsf{ColCorrupt}}, \mathcal{O}_{\mathsf{Revoke}}, \mathcal{O}_{\mathsf{Spend}}$ oracles, $\mathcal{O}_{\mathsf{PC}}$ in short, can win the experiment described in Figure 5 with a non-negligible advantage in λ and we can write, $Adv^{\mathrm{PC}}_{\mathcal{A}}(\lambda) := \Pr[\mathrm{Exp}^{\mathrm{PC}}_{\mathcal{A}}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda)$.*

As a consequence of payment unlinkability and anonymity, no PPT adversary can expose any information about the transaction such as the identity of honest customers or be able to link it to

any other transaction made by the customer. As a consequence of payment certainty, no entity, not even after colluding with a group of participants, can transfer and/or revoke more money than the amount deposited. Finally, any system satisfying both these definitions simultaneously reveals the identity of a malicious customer attempting to use one collateral to pay multiple merchants at the same time.

Theorem 1. *The proposed generic construction in Algorithm 1 satisfies the unlinkability and anonymity of payment guarantees as defined in Definition 1.*

Proof. For each payment request, the customer should transfer a tuple $\mathcal{T}[\pi, \mathsf{x}, R_{\mathsf{t}}, H(Tx)]$ where R_{t} is the auxiliary data at time slot t to convince the merchant and the group of statekeepers about the uniqueness of a collateral. Under the existence of a privacy-preserving blockchain Tx_{ID} does not reveal any information beyond the validity of the transaction and it protects the anonymity of the costumers. In this case, to prove that our construction preserves the anonymity of honest customers and provides unlinkability of payments we show that no PPT adversary, $\mathcal{A}$, by providing two pair of challenge secret keys/collateral keys $(\hat{\mathsf{sk}}_0^*, \hat{k}_0^*)$ and $(\hat{\mathsf{sk}}_1^*, \hat{k}_1^*)$, can distinguish between $(\pi_0, \mathsf{x}_0, R_{\mathsf{t},0})$ and $(\pi_1, \mathsf{x}_1, R_{\mathsf{t},1})$ as the output of the spending algorithm. This property is guaranteed because of the following main security properties for the given primitives: Zero-Knowledge property of the given NIZK proof system, computationally hiding property of the given commitment scheme, static-semantically secure property of the given randomness-reusable threshold encryption in bilinear groups and also the weak robustness of the given PRF.

Let the hybrid H^β be the case where the *Anonymity* experiment, $\text{Exp}_{\mathcal{A}}^{\text{ANON}}(\lambda, \beta)$ is run for $\beta = \{0, 1\}$. In this case, we form a sequence of hybrids and show that each of the successive hybrids are computationally indistinguishable from the preceding ones.

- **Hybrid H_1^β:** In this game, we assume the existence of an efficient simulator Sim and then modify the previous hybrid, H^β, by generating the challenge NIZK proof π_β via the simulation algorithm, $\pi'_\beta \leftarrow \mathcal{ZK}.\mathsf{Sim}(\vec{\mathsf{crs}}, \hat{\vec{\mathsf{ts}}}, \mathsf{x}_\beta)$.

The Zero-Knowledge property of NIZK arguments provided in Definition 22 guarantees that this experiment is indistinguishable from the one for H^β and we can write $H_1^\beta \approx_\lambda H^\beta$.

- **Hybrid H_β^2:** In this game, we modify H_1^β s.t. for generating the index id the challenger commits $\hat{\mathsf{sk}}_{1-\beta}^*$ instead of $\hat{\mathsf{sk}}_\beta^*$.

According to the hiding property of the given commitment scheme, this experiment is indistinguishable from H_β^1 and we can write, $H_2^\beta \approx_\lambda H_1^\beta$.

- **Hybrid H:** In this game, we modify H_β^2 by assuming the challenger runs the RRTE encryption algorithm under the message $m_{1-\beta}$ instead of m_β.

According to the Static Semantic Security property of the proposed randomness-reusable Threshold encryption, this experiment is indistinguishable from H_β^2. To be more concrete, $\mathcal{A}$ cannot distinguish between $\mathtt{Ct}_\beta$ and $\mathtt{Ct}_{1-\beta}$ as long as no twin ciphertext is generated even if the proofs are simulated. Thereby we have, $H_0 \approx_\lambda H_0^1 \approx_\lambda H_0^2 \approx_\lambda H \approx_\lambda H_1^1 \approx_\lambda H_1^2 \approx_\lambda H_1$.

To conclude this security property for the proposed construction, based on the weakly robust property of the given PRF, it is straightforward to demonstrate that the output of a PRF under two distinct keys is computationally indistinguishable and no PPT adversary can distinguish $R_{\mathsf{t},0}$ and $R_{\mathsf{t},1}$. □

Theorem 2. *The proposed generic construction in Algorithm 1 satisfies the payment certainty of payment guarantees as defined in Definition 2.*

Proof. We prove this security property by contradiction and for the simplicity we avoid the hat notion for the secret parameters. Let there is a PPT adversary $\mathcal{A}$ that can break the payment certainty of the scheme and pass the verification phase without meeting at least of the following cases.

- **Case 1.** The adversary $\mathcal{A}$ can forge a valid payment guarantee, $\mathcal{T}$.
- **Case 2.** The adversary $\mathcal{A}$ can forge a valid aggregated signature σ_{R_t} s.t. $|\sigma_{R_\mathsf{t}}| \geq (|\mathsf{StK}|/2) + 1$.

By relying on the existence of a weakly-robust PRF, a *Knowledge Sound* NIZK argument, an existentially unforgeable TSPS construction we show that the success probablity of adversary in "**Case 1**" is negligible. Thus having played a sequence of indistinguishable games between $\mathcal{B}^{\text{PRF}}_{\text{WR}}(1^\lambda), \mathcal{B}^{\text{TSPS}}_{\text{EUF-CIMA}}(1^\lambda), \mathcal{B}^{\text{NIZK}}_{\text{KS}}(1^\lambda)$ and a PPT adversary $\mathcal{A}$, we gradually turn the payment certainty security game into the security features of the underlying primitives.

- **Game** G_0**:** In the first security game, let $\mathcal{A}$ forms a challenge transaction τ^* such that $\sum \mathcal{L}_c + \tau^* > \mathsf{col}_\mathcal{A}$ return a valid pair (π^*, x^*) with a non-negligible advantage ϵ. By contradiction, we assume $\mathcal{A}$ can win this game with a non-negligible advantage ϵ and we can write, $Adv^{\mathsf{PC}}_\mathcal{A}(\lambda) = \Pr[\mathcal{A} \text{ Wins } G_0] \geq \epsilon$.
- **Game** G_1**:** In this game, we modify G_0 such that we assume the existence of an efficient extractor $\mathsf{Ext}(.)$. In this case, there exists an extractor that takes the extraction trapdoor $\vec{\mathsf{te}}$ and the received challenge tuple $(\pi^*, \pi^*_j, \mathsf{x}^*)$ as inputs, and returns the corresponding witness $(\mathsf{w}^*) \leftarrow \mathsf{Ext}(\vec{\mathsf{te}}, \mathsf{x}^*, \pi^*)$ s.t. $\mathsf{w}^* = \Big(\mathsf{cert}^*, \mu^*, \mathsf{sk}^*_c, r^*_\mathsf{t}, (M^*_j, k^*)\Big)$. To be more precise, the extractor first extracts the indexed DH message $M^*_j := (id^*_j, M^*_{j1}, M^*_{j2})$, and then can extracts the secret PRF key k^* from the proof (π^*_j) as a proof to show the well-formedness of the index id^*_j. The indistinguishability of G_0 and G_1 can be proven via the *Knowledge Extraction* property of NIZK arguments, defined in Definition 23. This property guarantees the existence of the defined extractor under non-falsifiable assumptions and we can write, $Adv^{\mathsf{PC}}_\mathcal{A}(\lambda) = \Pr[\mathcal{A} \text{ Wins } G_0] \approx \Pr[\mathcal{A} \text{ Wins } G_1]$ and this advantage consequently depends on two possible cases,

  $$\Pr[\mathcal{A} \text{ Wins } G_1] = \Pr[\mathcal{A} \text{ Wins } G_1 : (\mathsf{w}^*, \mathsf{x}^*) \in \mathbf{R_L}] + \Pr[\mathcal{A} \text{ Wins } G_1 : (\mathsf{w}^*, \mathsf{x}^*) \notin \mathbf{R_L}] \ .$$

  The probability of an adversary in the latter case can be bounded by the advantage a NIZK's knowledge soundness.

  $$Adv^{\mathsf{PC}}_\mathcal{A}(\lambda) \leq \Pr[\mathcal{A} \text{ Wins } G_1 : (\mathsf{w}^*, \mathsf{x}^*) \in \mathbf{R_L}] + Adv^{\text{NIZK}}_{\mathcal{B}_{\text{KS}}}(\lambda) \, .$$

  Under the assumption that the given NIZK is KS, the adversary $\mathcal{A}$ can win the game when the event of $(\mathsf{w}^*, \mathsf{x}^*) \in \mathbf{R_L}$ occurs.
- **Game** G_2**:** The challenger for the payment certainty security game can modify G_1 to an attacker against the weakly-robust PRF security game. The intended key k^* is either a valid key $k^* \in \mathcal{K}$ s.t. it is not corrupted by the adversary, i.e. $k^* \notin \mathcal{CL}'$ or it is generated under a random key $k^* \notin \mathcal{K}$. The latter case will be bounded by the advantage of $\mathcal{B}^{\text{PRF}}_{\text{WR}}(1^\lambda)$ attacker, then we can write,

  $$\Pr[\mathcal{A} \text{ Wins } G_2] = \Pr[\mathcal{A} \text{ Wins } G_2 : k^* \in \mathcal{K} \ \wedge \ k^* \notin \mathcal{CL}'] +$$
  $$\Pr[\mathcal{A} \text{ Wins } G_2 : k^* \notin \mathcal{K}] \leq \Pr[\mathcal{A} \text{ Wins } G_2 : k^* \in \mathcal{K} \ \wedge \ k^* \notin \mathcal{CL}'] + Adv^{\text{PRF}}_{\mathcal{B}_{\text{WR}}}(\lambda) \ .$$

– **Game G_3:** This is the game G_2, except for a valid pair of witness and statement in $\mathbf{R_L}$ and a fresh and not queried PRF key, one can reduce it to a forgery attack for the underlying TSPS scheme. More specifically, if $k^* \in \mathcal{K}$ and not corrupted before then the challenger can generate its iDH message format M_j^*. Lets the set of authorities indices that are queried before to get a certificate by the adversary is denoted by $\mathcal{S}_{(\star,M_{j2}^*)}$. If $|\mathcal{S}_{(\star,M_{j2}^*)} \cup \mathcal{AU}'| < t$, $\mathcal{B}^{\text{TSPS}}_{\mathsf{EUF\text{-}CiMA}}(1^\lambda)$ returns the pair (M_j^*, cert^*) as a valid forgery for the defined threshold EUF-CiMA security game in Def. [24, 4.3]. Thus, we can write,

$$
\begin{aligned}
&Adv^{\mathsf{PC}}_{\mathcal{A}}(\lambda) \leq Adv^{\text{NIZK}}_{\mathcal{B}_{\text{KS}}}(\lambda) + Adv^{\text{PRF}}_{\mathcal{B}_{\text{WR}}}(\lambda) + \Pr[\mathcal{A} \text{ Wins } G_3 : |\mathcal{S}_{(\star,M_{j2}^*)} \cup \mathcal{AU}'| < t] + \\
&\Pr[\mathcal{A} \text{ Wins } G_3 : |\mathcal{S}_{(\star,M_{j2}^*)} \cup \mathcal{AU}'| \geq t] \leq Adv^{\text{NIZK}}_{\mathcal{B}_{\text{KS}}}(\lambda) + \\
&Adv^{\text{PRF}}_{\mathcal{B}_{\text{WR}}}(\lambda) + Adv^{\text{TSPS}}_{\mathcal{B}_{\mathsf{EUF\text{-}CiMA}}}(\lambda) + \Pr[\mathcal{A} \text{ Wins } G_3 : |\mathcal{S}_{(\star,M_{j2}^*)} \cup \mathcal{AU}'| \geq t] \quad .
\end{aligned}
$$

Since it is assumed that the adversary $\mathcal{A}$ should provide a fresh and not queried collateral, the probability of the event, "$\mathcal{A}$ Wins $G_3 \wedge |\mathcal{S}_{(\star,M_{j2}^*)} \cup \mathcal{AU}'| \geq t$", is equal to zero. Then we can write,

$$
Adv^{\mathsf{PC}}_{\mathcal{A}}(\lambda) \leq Adv^{\text{NIZK}}_{\mathcal{B}_{\text{KS}}}(\lambda) + Adv^{\text{PRF}}_{\mathcal{B}_{\text{WR}}}(\lambda) + Adv^{\text{TSPS}}_{\mathcal{B}_{\mathsf{EUF\text{-}CiMA}}}(\lambda) \quad .
$$

Similarly to demonstrate that the probability of **Case 2** is negligible we rely on the unforgeability of the given aggregatable digital signature. If the adversary $\mathcal{A}$ be able to forge a valid aggregated signature for a majority of the statekeepers then as it is assumed it only can corrupt at most $|\mathcal{M}'| < |\mathsf{StK}|/2$, then we can form an efficient algorithm $\mathcal{B}^{\text{DS}}_{\text{EUF-CMA}}(1^\lambda)$ to break the EUF-CMA property of the underlying DS scheme. Then we can write:

$$
Adv^{\mathsf{PC}}_{\mathcal{A}}(\lambda) \leq Adv^{\text{NIZK}}_{\mathcal{B}_{\text{KS}}}(\lambda) + Adv^{\text{PRF}}_{\mathcal{B}_{\text{WR}}}(\lambda) + Adv^{\text{TSPS}}_{\mathcal{B}_{\mathsf{EUF\text{-}CiMA}}}(\lambda) + Adv^{\text{DS}}_{\mathcal{B}_{\text{EUF-CMA}}}(\lambda) \quad .
$$

Thus, as long as the underlying primitives are knowledge sound, weakly robust and existentially unforgeable then we can conclude the theorem. □

4 An Efficient Instantiation

In this section, we specify the concrete cryptographic primitives used to instantiate our construction. With the exception of RRTE, which is our novel construction, we refer formal definitions of primitives and their security properties to Appendix A. We would like to stress the modularity of our construction. The below tools are used in a black box manner and can be replaced by superior tools that future research will inevitably develop.

Randomness-Reusable Threshold Encryption. (ℓ, t, k)-RRTE is a new observation on threshold encryption (TE) schemes (see Appendix A.5) and enables plaintext confidentiality as long as less than k number of ciphertexts with the same randomness is generated. Once a data owner issues at least k supplementary ciphertexts, it is publicly retrievable and everybody can blind out the encrypted data. We formulate this primitive for compatibility with the rest of our system, but it is worth noting that the underlying idea is similar to offline double spending detection used in e-cash schemes.

Definition 3 (Randomness-Reusable Threshold Encryption). *For a given public parameters* pp *and security parameter* λ*, a* (ℓ, t, k)*-RRTE,* Ψ_{RRTE}*, over the message space* $\mathcal{M}$ *and ciphertext space* $\mathcal{C}$ *consists of three main PPT algorithms defined as follows:*

- $(\vec{\mathsf{pk}}, \mathsf{pk}) \leftarrow \mathcal{RRTE}.\mathsf{KGen}(\mathsf{pp}, \ell, t, k)$*: Key generation is a probabilistic and distributed algorithm that takes* pp *along with three integers* $\ell, t, k \in \mathsf{poly}(\lambda)$ *as inputs. It then returns a vector of public key* $\vec{\mathsf{pk}}$ *of size* ℓ *and a general public key* pk *as outputs.*
- $(\mathtt{Ct}_j, v) \leftarrow \mathcal{RRTE}.\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m, \mathsf{pk}_j)$*: The encryption algorithm as a probabilistic algorithm takes* pp*, global public key* pk*, a message* $m \in \mathcal{M}$ *along with a public key* pk_j *as inputs. It returns ciphertext* $\mathtt{Ct}_j \in \mathcal{C}$ *associated with the recipient* $j \in \mathcal{R}$ *and an auxiliary value* v *as outputs.*
- $(\perp, m) \leftarrow \mathcal{RRTE}.\mathsf{Dec}(\mathsf{pp}, \{\mathtt{Ct}_j\}_{j\in\mathcal{K}}, v)$*: The decryption algorithm takes* pp*, a set of ciphertexts* $\{\mathtt{Ct}_j\}_{j\in\mathcal{K}}$ *along with an auxiliary value* v *as inputs. If* $|\mathcal{K}| \geq k$*, it returns* $m \in \mathcal{M}$*, else it responds by* $\perp$*.*

Note that in Appendix A.5, we elaborate more on the security requirements and then we propose an efficient construction based on threshold ElGamal encryptions.

Pseudo-Random Function. We utilise a weakly-robust PRF proposed by Dodis and Yampolskiy [27] in order to make customers' collaterals reusable. This PRF enables us to define a time of payment, i.e. x in $\mathsf{PRF}_k(x) = g^{1/(k+x)}$ function and prove the validity of operations, efficiently. This ensures that a customer always has to input the time of payment, which is then verified by a receiving merchant. By utilizing this property and its combination with RRTE, as discussed in Section 1, we can block a customer from reusing the same collateral for a pre-defined time period.
Digital Signature Schemes. We require two types of signatures, one for the authorities and another for the statekeepers. These signatures need non-overlapping properties which we detail below.

- Threshold Structure-Preserving Signatures [24]. There are two reasons to use the TSPS scheme proposed by Crites et al. Firstly, like any other digital signature, it provides authentication, such that no entity except the qualified authorities can issue collateral proofs. Secondly, due to its threshold nature, TSPS enables our construction to rely on an honest majority (authorities) instead of a central trusted party.
- BLS Signatures [14]. BLS Signatures are efficiently aggregatable, and thus they are useful in our setting. A statekeeper must validate payment requests from various merchants, and they do so using BLS signatures. For a victim merchant to redeem user collateral from the smart contract, they may first aggregate the signatures allowing for a shorter interaction.

Commitment Scheme. In our construction, we use the Pedersen commitment scheme [56] due to the following reasons; firstly, the TSPS construction is defined over the indexed DH message spaces (Definition 5) and each secret PRF key needs to get an index. Hence, these commitments are used to the secret scalar PRF keys as an index. Secondly, the hiding property of such commitments masks the secret PRF keys used in our construction. In addition, the binding property of Pedersen commitments ensures the unforgeability of these secret PRF keys. Finally, Pedersen commitments are compatible with discrete logarithm-based proofs like original Sigma protocols and enables customers to efficiently prove knowledge of these committed values.

NIZK Proofs. To instantiate the described NP-relations in Section 3.1, we utilize three main proof systems: Sigma protocols [59], range-proofs [17] and GS proof systems [39] (see Appendix A.7). Sigma protocols are an efficient choice as the main proof system in our implementation; we use the Fiat-Shamir heuristic [34] to make then non-interactive. Range-proofs enable us to prove that a hidden value lies in a range interval. GS proof systems are useful as they are secure in the standard model and support a straight-line extraction of the witnesses. Additionally, the instantiation of these proofs does not require any trusted setup and can be batched: this enables an efficient verification [45].

5 Performance Analysis

In this section, we demonstrate the performance of our system. Based on the application, the costs incurred in each phase are divided into two parts, termed "offline phase" and "online phase". The former includes the parameter generation, key generation and registration functions. The latter is solely responsible for spending and verification and is the main focus of this evaluation.

Our experimental setup is similar to the one in Snappy. Namely, we distribute various parties in different regions around the world and measure the end to end latency of transactions.[4]

Specifically, our implementation uses the Charm-Crypto framework [2], a Python library for Pairing-based Cryptography and obtained the benchmarks on four AWS EC2 instances. The scenario we consider is similar to the one in Snappy. Namely, merchants, customers and statekeepers are globally distributed in four different locations and we create 1 000 tps in order to measure the average time it takes for one transaction to complete. Since transactions are distributed to many merchants and the merchants run independently, it is possible to create an equivalent scenario and only consider the work needed for one merchant. Consider the workload from the perspective of a single statekeeper: its workload depends on transactions that are passing through all other merchants. To accurately estimate the workload of a single statekeeper, we injecting artificial verification requests to it. Our scenario is summarized in Figure 6.

All our EC2 instances had the same computational configuration, i.e., an Ubuntu Server 20.04 LTS (HVM) with an Intel (R) Xeon(R) CPU @ 2.50 GHz and 16 GB of memory. We apply the Barreto-Naehrig (BN254) curve (also known as type F groups), $y^2 = x^3 + b$ with embedding curve degree 12 [7]. In this pairing group, the base field order is 256 bits.

Latency. As illustrated in Figure 7, the latency for each transaction grows linearly with the number of statekeepers verifying this transaction (depicted with the orange line). During our evaluations, we noticed that the time required for a customer to generate and send a payment guarantee (depicted with the red line) is mostly constant, i.e., $\sim$ 240 ms. However, in the case of 40 statekeepers, our construction allows a customer and merchant to successfully transact within $\sim$ 550 milliseconds (ms). In contrast, in the case of 200 statekeepers, a transaction takes $\sim$ 1.3 seconds (s). Hence, the time required to guarantee a payment to merchants in our construction is bounded by the set of statekeepers. A direct comparison with Snappy is not possible for two reasons; firstly, the evaluation of Snappy only considers the time taken for payment approval, i.e., it does not consider the time spent on customer-merchant interaction. Secondly, their simulation code is not freely available. A standalone analysis shows that our construction provides privacy against statekeepers without impacting the latency of payments. Greater number of statekeepers provides more robustness and

[4] The open-source implementation can be found in this repository.

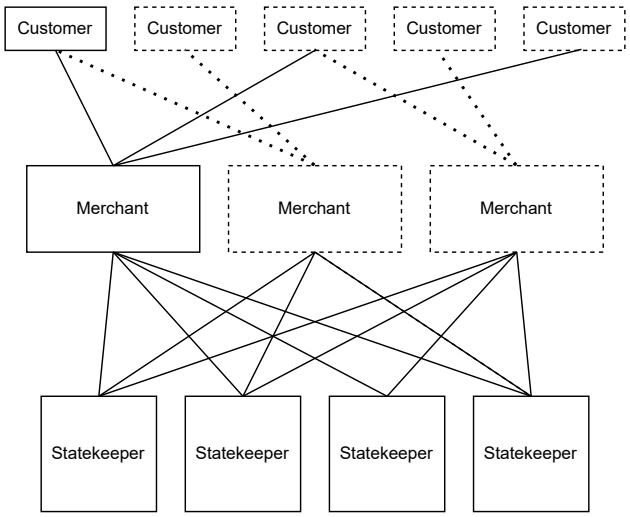


Fig. 6: Experimental setup. Dotted lines and shapes represent parties and workload that do not affect the leftmost customer or the leftmost Merchant. Our experiment only considers the solid lines and shapes. From the perspective of the leftmost client and the leftmost merchant, this is equivalent to running the full system. While statekeepers are the same as merchants, we make these two sets distinct for clarity.

better protection against double-spends, but requires stronger liveness assumptions on top of increasing the latency. We find 120 statekeepers to be an optimal trade-off of these factors, ultimately leading to latency lower than 1 sec.

Smart Contract Cost. The transition between states happens depending on the function calls on the smart contract. For simplicity, we describe here only the functionality of the smart contract focusing on one customer and multiple merchants. We refer to our smart contract as $Auth_{SC}$, an entity is referred to as e_x where $x = c$ or m for customer or merchant respectively. The underlying ledger is referred to as L_{SC} and an entity's account on that ledger is referred to as Acc_L^x where $x = c$ or m respectively. The private ledger of the merchants is referred to as $Bull_m$, since it behaves like a bulletin board. $Auth_{SC}$ has seven states as follows:

init: $Auth_{SC}$ is deployed. e_c can now deposit funds (col_c). If so, then change state to *ready*. Else do not change state.
ready: e_c successfully registers by depositing col_c in $Auth_{SC}$. If col_c is available in $Auth_{SC}$, change state to *pay*. Else do not change state.
pay: If e_c has made payment (pay_{m_i}), change state to $reclaim_m$. Else do not change state.
$reclaim_m$: Check $Bull_m$ for double-spends from e_c. If double-spend present, use secret to reclaim pay_{m_i} and change state to *withdraw*. If no double-spend found until actual payment received, change state to $reclaim_c$.
$reclaim_c$: If 1 day has passed since pay_{m_i}, reclaim pay_{m_i} and add it to col_c. Then, change state to *withdraw*. Else do not change state.
withdraw: If e_c wants to exit the system and the state is *withdraw* or *ready*, send money from $Auth_{SC}$ to Acc_L^c and change state to *exit*. If e_w wants to exit the system and the state is *withdraw*, send money from $Auth_{SC}$ to Acc_L^w and change state to *exit*. Else do not change state.

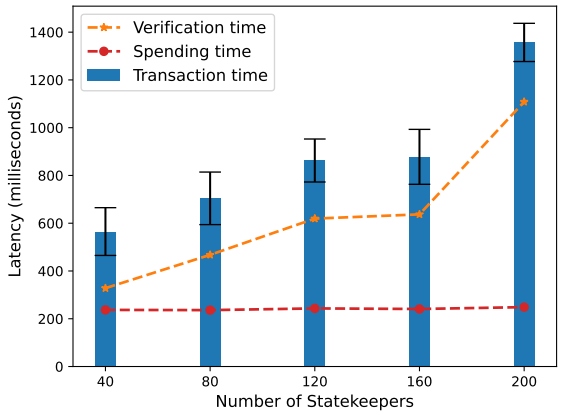


Fig. 7: Latency comparison. Total transaction time for 1000 tx per second vs the number of statekeepers.

exit: Remove e_x from $Auth_{SC}$ and change state to *init*.

Table 1 lists the gas fees of executing various functions of our system. In the first column, we mention the amount of base gas fee, and then we express it as a proportion of minimum gas fee of a transaction. The minimum gas fee is set to 21 000 GWei, which is also the amount of gas that standard on-chain payments require. Note that the table reflects the fees users can expect to pay, although the actual amount also depends on the so-called priority fee which depends on the current traffic in the Ethereum transaction market. More information about the costs incurred due to our SC is given in Section 6.

Table 1: Costs of transactions on our smart contract deployed on Ethereum, and the cost as a proportion of a standard transaction $\Delta = 21\,000$.

Function	Customer registration	Pay	Merchant registration	Reclaim	Withdraw Balance
Gas	107 400	44 055	54 317	34 972	22 352
$\times\Delta$	5.1	2.09	2.59	1.65	1.06

6 Discussion

Collateral reusability. The capital demands of existing layer-2 proposals have been frequently acknowledged [57,53,26,52,43,44,48]. Roughly speaking, the risk the system is willing to take on behalf of a user is proportional to the amount of money the user commits as collateral. This limitation is difficult to overcome in a trustless setting, and our work can be seen as partial progress in this direction.

Payment channel networks have heavy collateral requirements in order to maintain trustlessness. A payment of, say, 10 dollars across 3 channels requires committed 10 dollar's worth of collateral on each of the three channels. These funds are locked up for a fixed amount of time and cannot be used for other payments. In early works, the funds were locked up for a time proportional to the length of the path used, but recent advances have reduced this to an amount depending only the security parameter and block generation time [3]. Despite this recent innovation, we find the collateral requirements prohibitive for an efficient payment scheme.

Payment channels, if used in the same direction, also suffer from collateral depletion, rendering further payments in that direction impossible. This problem of channel depletion has somewhat been addressed with re-balancing techniques [5]. However, rebalancing is generally not possible for a user that only employs their channels for payments and not for getting paid. Hence, rebalancing is incompatible with retail payments which are generally unilateral i.e., from the set of consumers to the set of merchants.

In our system, payment guarantees assure a payee (merchant) that they will receive a payment once the transaction on the underlying blockchain has reached its eventual finality, even if the payer (customer) is malicious. The collateral provided by a payer is *reusable*: this reusability comes from the abstraction of payment guarantees, where the customer makes a payment on the underlying blockchain and assures the payee about this payment by using irrefutable evidence that this payment will be successful. Unlike existing systems like Snappy, where reusability of collaterals comes at the cost of privacy, we utilize RRTE to guarantee privacy for all honest customers. As long as customers don't attempt to double spend, that is, simultaneously issue multiple payment guarantees with the same collateral, their identities are kept secret.

Trust Assumptions. There is a general trade-off between the efficiency and privacy of a financial system and the level of trust assumed between participants. For instance, a trusted central entity can efficiently set up a digital currency system, as evidenced by Chaumian e-cash. Measures of transaction latency and throughput thrive at the high cost of trust in the central authority. On the other end, decentralized blockchains achieve functional but slow financial systems without requiring trust in any single party.

In payment channel networks, a similar trade-off has been observed by Avarikioti et al. [4], who suggest that PCNs are more stable and efficient when centralized structures are present. In an empirical survey, Zabka et al. [62] observe the rising centrality in the Lightning Network as the capacity and capabilities of Lightning grew over time.

To the best of our knowledge, we are the first ones to utilize these building blocks and design an instant finality layer-2 scheme that utilizes *reusable* collaterals, while offering several privacy-preserving properties. In order to offer these properties, we need to rely on a set of trust assumptions. Our architecture is semi-decentralized in the sense that we rely on an honest majority of authorities to initialize our construction. This is similar to the approach of LDSP [54]. However, unlike LDSP, our authorities do not play any role in our payment protocol. The merchants and statekeepers have greater power to punish dishonest customers by confiscating their collateral. Yet, we allow an honest majority of merchants to do so *only* against customers who attempt to double-spend, not honest customers. Moreover, the design is permissionless in that cryptocurrency holders can freely participate as customers. Considering the general trade-off between centrality, trust, performance and efficiency, we consider our setup to lie in a "sweet spot." In our case, this balance was achieved through advances in cryptography, namely RRTE. We call for further cryptographic innovations and welcome research into even more trustless, robust and secure systems.

Privacy. The main idea underlying our private double-spending protection, goes all the way back to the e-cash schemes introduced by Chaum [22]. "On-line" e-cash tackled the problem of double spending by having the issuing bank verify each transaction individually before it was marked successful. Chaum, Fiat and Naor later extended this idea to support "off-line" payments, i.e, a customer could make untraceable payments to the merchant without involving a bank for every transaction [23]. Due to this "off-line" nature of payments, the solution for double-spending combined prevention with tamper-resistant hardware along with detection through successful tracking, i.e., the issuing bank would check the list of all coins spent, and once double-spending was spotted the identity of the perpetrator would be revealed. This approach of realizing offline payments while detecting double-spending, known as the Chaum-Fiat-Naor (CFN) approach was adopted and improved by several following e-cash systems [15,35,20,21,6]. The existing plethora of literature has made several improvements to Chaum's e-cash, however, all work with centrally issued currency and mostly rely on a custodian bank to catch double-spending. Our work is also an application of the CFN approach, with the major difference of building upon decentralized cryptocurrencies for safely reducing latency, instead of building an entire e-cash scheme from the ground up. When compared to double-spend detection techniques in e-cash (for instance the one recently used in [8]), our novel RRTE is more efficient in terms of communication rounds, allows the deposited collaterals to be reused and by default enables anyone to track double-spends.

However, *on-chain privacy* is derived from the underlying blockchain, and is the highest level of privacy that one can hope to achieve at the protocol level. In other words, implementing our overlay on a completely de-anonymized and public blockchain cannot make the payments private, since the underlying blockchain will reveal private data no matter how secure the protocol. Similarly, developing on top of private blockchains such as Monero doesn't directly solve the privacy issues of earlier works that allowed transactions of the same user to be linked.

In this way, the question of blockchain-level privacy is relevant yet orthogonal to our work. While Snappy claims that future improvements such as deployment on privacy-preserving blockchains that support privacy-preserving SC will enable their construction to provide on-chain privacy, that is not true. As explained in Section 2.1, an inherent design flaw in their proactive double-spending detection is overlooked.

Our construction provides this proactive double-spending detection while guaranteeing protocol-level privacy. Hence, deployment on a privacy-preserving blockchain makes our construction fully private, even on-chain. There are possible workarounds such as change addresses and mixing services to provide increased on-chain pseudonymity in case of public blockchains. Achieving a greater synergy between these two levels of a privacy seems to be a promising avenue for future work.

On-chain Transaction Fees. The transaction fees in our system differ from conventional fees since the customer pays the merchant indirectly through a smart contract (SC). This is necessary to prevent an on-chain double-spend by a malicious customer. By on-chain double-spend, we mean to distinguish between a double-spend attempt of customer collateral, and a double-spend on the underlying blockchain itself. Even if a malicious customer can influence miners and induce a blockchain double-spend, the SC-based transaction is able to remunerate the affected merchant. Simply put, the SC can escrow the funds until sufficient confirmations of on-chain payment have been found. We stick to the convention of 6 succeeding blocks after said transaction.

As discussed in Section 5, executing payment through our SC incurs twice the on-chain transaction fees of a standard on-chain Ethereum payment. There is an additional fee incurred by the merchant during withdrawal of payments, but this is far less frequent than the former. Nevertheless,

it is desirable to construct a more cost efficient yet secure system for direct customer to merchant payments.

A potential fix could be to encode specific spend conditions for user collaterals. To be precise, any merchant can move the collateral by providing evidence of a conflicting transaction on the blockchain. This could be implemented via a payment guarantee to said merchant, along with on-chain evidence of a conflicting payment. We leave this implementation, along with other possible optimizations of fees, for future work.

Incentives of Involved Parties. This work deals with the cryptographic challenges of achieving privacy while reducing latency of cryptocurrency payments. Our focus is admittedly myopic, as we overlook practical aspects of incentives. For instance, we refer to authorities that register merchants and customers, but these authorities lack a concrete incentive to fulfil this role honestly. As an initial and arbitrary choice, we selected a subset of involved merchants to play this role of authorities, while requiring an honest majority of authorities.

It is unclear who should be playing this role, and what their incentives should be. Could we perhaps allocate a small fee per merchant to authority? Or automatically grant a fraction of collaterals confiscated from dishonest users? Or even eliminate this issue entirely by building more advanced cryptography so that our overlay can be set up even without their existence?

Similarly, we lack a clear explanation of incentives for statekeepers. A basic solution would be to allocate a certain fraction of each transaction value to the statekeepers; however, this still needs to be properly analyzed in order to confirm if such an incentive is sufficient.

7 Conclusion

In this paper, we present a new overlay network for instant confirmation of cryptocurrency transactions, that also maintains anonymity of users and unlinkability of their transactions. On the one hand, it allows merchants in a retail system to safely accept fast payments without risk of double-spending. On the other, dishonest customers who attempt to double-spend get their identities exposed and their collateral confiscated to reimburse the merchants. Honest customers, however, are able to *reuse* their collaterals.

To this end, we designed a novel randomness-reusable threshold scheme, that enables participants to audit the payments in the network and reveal the identity of malicious customer who perform double-spending. This threshold encryption scheme maintains the privacy of honest customers who do not attempt to double-spend. We provide a formal proof of security with respect to three main features namely *customers' anonymity, unlinkability of transactions* and *payment certainty for merchants*. We motivate our choice of cryptographic primitives and efficiently implement them. Our evaluation shows that our construction allows for fast global payments with a delay of less than 1 seconds.

Acknowledgment. We would like to thank Svetla Nikova, Philipp Jovanovic, Christian Badertscher and Daniel Slamanig for the helpful discussions and the anonymous reviewers for their valuable comments. This work was supported by CyberSecurity Research Flanders with reference number VR20192203. Akash Madhusudan, Mahdi Sedaghat and Bart Preneel were supported in part by the Flemish Government through the FWO SBO project SNIPPET S007619 and the Research Council KU Leuven C1 on Security and Privacy for Cyber-Physical Systems and the Internet of Things with contract number C16/15/058. Samarth Tiwari was supported by

ERC Starting Grant QIP–805241. Kelong Cong was supported by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. FA8750-19-C-0502. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA, the US Government, Cyber Security Research Flanders or the FWO. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

1. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 209–236. Springer, Heidelberg (Aug 2010). https://doi.org/10.1007/978-3-642-14623-7_12
2. Akinyele, J.A., Garman, C., Miers, I., Pagano, M.W., Rushanan, M., Green, M., Rubin, A.D.: Charm: a framework for rapidly prototyping cryptosystems. Journal of Cryptographic Engineering **3**(2), 111–128 (Jun 2013). https://doi.org/10.1007/s13389-013-0057-3
3. Aumayr, L., Abbaszadeh, K., Maffei, M.: Thora: Atomic and privacy-preserving multi-channel updates. IACR Cryptol. ePrint Arch. p. 317 (2022), https://eprint.iacr.org/2022/317
4. Avarikioti, Z., Heimbach, L., Wang, Y., Wattenhofer, R.: Ride the lightning: The game theory of payment channels. In: FC 2020. pp. 264–283. LNCS, Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-51280-4_15
5. Avarikioti, Z., Pietrzak, K., Salem, I., Schmid, S., Tiwari, S., Yeo, M.: HIDE & SEEK: Privacy-preserving rebalancing on payment channel networks. Cryptology ePrint Archive, Report 2021/1401 (2021), https://eprint.iacr.org/2021/1401
6. Baldimtsi, F., Chase, M., Fuchsbauer, G., Kohlweiss, M.: Anonymous transferable E-cash. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 101–124. Springer, Heidelberg (Mar / Apr 2015). https://doi.org/10.1007/978-3-662-46447-2_5
7. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (Aug 2006). https://doi.org/10.1007/11693383_22
8. Bauer, B., Fuchsbauer, G., Qian, C.: Transferable E-cash: A cleaner model and the first practical instantiation. pp. 559–590. LNCS, Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-75248-4_20
9. Benmeleh, Y.: Blockchain firm starkware valued at $2 billion in funding round. https://www.bloomberg.com (2021)
10. Blazy, O., Canard, S., Fuchsbauer, G., Gouget, A., Sibert, H., Traoré, J.: Achieving optimal anonymity in transferable e-cash with a judge. In: Nitaj, A., Pointcheval, D. (eds.) AFRICACRYPT 11. LNCS, vol. 6737, pp. 206–223. Springer, Heidelberg (Jul 2011)
11. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications. In: Proceedings of the twentieth annual ACM symposium on Theory of computing. pp. 103–112. ACM (1988)
12. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 435–464. Springer, Heidelberg (Dec 2018). https://doi.org/10.1007/978-3-030-03329-3_15
13. Boneh, D., Franklin, M.K.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (Aug 2001). https://doi.org/10.1007/3-540-44647-8_13
14. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. Journal of Cryptology **17**(4), 297–319 (Sep 2004). https://doi.org/10.1007/s00145-004-0314-9
15. Brands, S.: Untraceable off-line cash in wallets with observers (extended abstract). In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 302–318. Springer, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48329-2_26
16. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci. **37**(2), 156–189 (1988). https://doi.org/10.1016/0022-0000(88)90005-0, https://doi.org/10.1016/0022-0000(88)90005-0
17. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press (May 2018). https://doi.org/10.1109/SP.2018.00020

18. Buterin, V.: An incomplete guide to rollups. https://vitalik.ca/general/2021/01/05/rollup.html (2021)
19. Camenisch, J., Dubovitskaya, M., Haralambiev, K., Kohlweiss, M.: Composable and modular anonymous credentials: Definitions and practical constructions. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 262–288. Springer, Heidelberg (Nov / Dec 2015). https://doi.org/10.1007/978-3-662-48800-3_11
20. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact e-cash. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 302–321. Springer, Heidelberg (May 2005). https://doi.org/10.1007/11426639_18
21. Canard, S., Gouget, A.: Multiple denominations in e-cash with compact transaction data. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 82–97. Springer, Heidelberg (Jan 2010)
22. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) CRYPTO'82. pp. 199–203. Plenum Press, New York, USA (1982)
23. Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. In: Goldwasser, S. (ed.) CRYPTO'88. LNCS, vol. 403, pp. 319–327. Springer, Heidelberg (Aug 1990). https://doi.org/10.1007/0-387-34799-2_25
24. Crites, E., Kohlweiss, M., Preneel, B., Sedaghat, M., Slamanig, D.: Threshold structure-preserving signatures. Cryptology ePrint Archive, Paper 2022/839 (2022), https://eprint.iacr.org/2022/839
25. Damgård, I., Ganesh, C., Khoshakhlagh, H., Orlandi, C., Siniscalchi, L.: Balancing privacy and accountability in blockchain identity management. pp. 552–576. LNCS, Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-75539-3_23
26. Decker, C., Wattenhofer, R.: A fast and scalable payment network with Bitcoin duplex micropayment channels. In: Symposium on Self-Stabilizing Systems. pp. 3–18. Springer (2015)
27. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 416–431. Springer, Heidelberg (Jan 2005). https://doi.org/10.1007/978-3-540-30580-4_28
28. Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.: Perun: Virtual payment hubs over cryptocurrencies. In: 2019 IEEE Symposium on Security and Privacy. pp. 106–123. IEEE Computer Society Press (May 2019). https://doi.org/10.1109/SP.2019.00020
29. Electric Coin Company: Explaining viewing keys. https://electriccoin.co/blog/explaining-viewing-keys/, accessed: 2023-02-13
30. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO'84. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (Aug 1984)
31. Escala, A., Groth, J.: Fine-tuning Groth-Sahai proofs. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 630–649. Springer, Heidelberg (Mar 2014). https://doi.org/10.1007/978-3-642-54631-0_36
32. Ethereum.org: Layer 2 rollups. Available in this Link (2021)
33. ethos.dev: The beacon chain ethereum 2.0 explainer you need to read first. https://ethos.dev/beacon-chain, accessed: 2023-02-13
34. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7_12
35. Frankel, Y., Tsiounis, Y., Yung, M.: "indirect discourse proof": Achieving efficient fair off-line E-cash. In: Kim, K., Matsumoto, T. (eds.) ASIACRYPT'96. LNCS, vol. 1163, pp. 286–300. Springer, Heidelberg (Nov 1996). https://doi.org/10.1007/BFb0034855
36. Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discrete Applied Mathematics **156**(16), 3113–3121 (2008). https://doi.org/https://doi.org/10.1016/j.dam.2007.12.010, applications of Algebra to Cryptography
37. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions (extended abstract). In: 25th FOCS. pp. 464–479. IEEE Computer Society Press (Oct 1984). https://doi.org/10.1109/SFCS.1984.715949
38. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on computing **18**(1), 186–208 (1989)
39. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 415–432. Springer (2008)
40. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (Apr 2008). https://doi.org/10.1007/978-3-540-78967-3_24
41. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: SoK: Layer-two blockchain protocols. In: FC 2020. pp. 201–226. LNCS, Springer, Heidelberg (2020). https://doi.org/10.1007/978-3-030-51280-4_12
42. Hanzlik, L., Slamanig, D.: With a little help from my friends: Constructing practical anonymous credentials. pp. 2004–2023. ACM Press (2021). https://doi.org/10.1145/3460120.3484582

43. Hearn, M.: Micro-payment channels implementation now in bitcoinj. Bitcointalk.org (2013)
44. Heilman, E., Alshenibr, L., Baldimtsi, F., Scafuro, A., Goldberg, S.: TumbleBit: An untrusted bitcoin-compatible anonymous payment hub. In: NDSS 2017. The Internet Society (Feb / Mar 2017)
45. Herold, G., Hoffmann, M., Klooß, M., Ràfols, C., Rupp, A.: New techniques for structural batch verification in bilinear groups with applications to groth-sahai proofs. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1547–1564. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134068
46. Hill, K.: How target figured out a teen girl was pregnant before her father did. https://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/?sh=53d927356668, accessed: 2022-08-30
47. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ecdsa). Int. J. Inf. Secur. **1**(1), 36–63 (aug 2001). https://doi.org/10.1007/s102070100002, https://doi.org/10.1007/s102070100002
48. Khalil, R., Zamyatin, A., Felley, G., Moreno-Sanchez, P., Gervais, A.: Commit-chains: Secure, scalable off-chain payments. Tech. rep., Cryptology ePrint Archive, Report 2018/642 (2018)
49. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2111–2128. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3339817
50. Maurer, U.M.: Unifying zero-knowledge proofs of knowledge. In: Preneel, B. (ed.) AFRICACRYPT 09. LNCS, vol. 5580, pp. 272–286. Springer, Heidelberg (Jun 2009)
51. Mavroudis, V., Wüst, K., Dhar, A., Kostiainen, K., Capkun, S.: Snappy: Fast on-chain payments with practical collaterals. In: NDSS 2020. The Internet Society (2020)
52. Miller, A., Bentov, I., Bakshi, S., Kumaresan, R., McCorry, P.: Sprites and state channels: Payment networks that go faster than lightning. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 508–526. Springer, Heidelberg (Feb 2019). https://doi.org/10.1007/978-3-030-32101-7_30
53. Network, Raiden: What is the raiden network. https://raiden.network/101.html (2019)
54. Ng, L.K.L., Chow, S.S.M., Wong, D.P.H., Woo, A.P.Y.: Ldsp: Shopping with cryptocurrency privately and quickly under leadership. In: 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS). pp. 261–271 (2021). https://doi.org/10.1109/ICDCS51616.2021.00033
55. Pedersen, T.P.: A threshold cryptosystem without a trusted party (extended abstract) (rump session). In: Davies, D.W. (ed.) EUROCRYPT'91. LNCS, vol. 547, pp. 522–526. Springer, Heidelberg (Apr 1991). https://doi.org/10.1007/3-540-46416-6_47
56. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (Aug 1992). https://doi.org/10.1007/3-540-46766-1_9
57. Poon, J., Dryja, T.: The Bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf (2016)
58. Reyzin, L., Smith, A., Yakoubov, S.: Turning HATE into LOVE: Homomorphic ad hoc threshold encryption for scalable MPC. Cryptology ePrint Archive, Report 2018/997 (2018), https://eprint.iacr.org/2018/997
59. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg (Aug 1990). https://doi.org/10.1007/0-387-34805-0_22
60. Sedaghat, M., Preneel, B.: Cross-domain attribute-based access control encryption. pp. 3–23. LNCS, Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-92548-2_1
61. Shamir, A.: How to share a secret. Communications of the Association for Computing Machinery **22**(11), 612–613 (Nov 1979)
62. Zabka, P., Foerster, K.T., Schmid, S., Decker, C.: A centrality analysis of the lightning network (2022). https://doi.org/10.48550/ARXIV.2201.07746, https://arxiv.org/abs/2201.07746

A Omitted Definitions and Preliminaries

In this section, we recall the needed preliminaries and definitions and then review the required building blocks, which include pseudo-random functions, Shamir secret sharing, aggregatable digital signature, Threshold Structure-Preserving Signatures, commitments and non-interactive zero-knowledge proofs. Finally, we also recall the definition of threshold encryptions as a starting point for the defined RRTE in Section 4.

Definition 4 (Bilinear Groups [13]). *A* Type-III bilinear group generator[5] $\mathcal{BG}(1^\lambda)$ *returns a tuple* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{p}, e, g, h)$*, consisting of cyclic Abelian groups* $\mathbb{G}_1$*,* $\mathbb{G}_2$*, and* $\mathbb{G}_T$ *with the same prime order* p*. For given generators of groups* $\mathbb{G}_1$ *and* $\mathbb{G}_2$ *namely* g *and* h*, an efficient non-degenerate Type-III bilinear pairing is denoted by* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$*, such that,* $\forall\ a, b \in \mathbb{Z}_\mathsf{p} :\ e(g^a, h^b) = e(g^b, h^a) = e(g, h)^{ab}$ *and* $e(g, h) \neq 1_{\mathbb{G}_T}$*.*

Definition 5 (Indexed Diffie-Hellman Message Space [24]). *For a given bilinear group,* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{p}, e, g, h) \leftarrow \mathcal{BG}(1^\lambda)$*, and hash function* H *that is modeled in the random oracle,* $\mathcal{M}^{\mathsf{H}}_{\mathsf{iDH}}$ *is called the indexed DH message space, if we have:*

1. $\forall\ (id, M_1, M_2) \in \mathcal{M}^{\mathsf{H}}_{\mathsf{iDH}}\ \exists\ m \in \mathbb{Z}_\mathsf{p} : \boldsymbol{g} = \mathsf{H}(id), M_1 = \boldsymbol{g}^m, M_2 = h^m$*.*
2. *No two messages use the same index, i.e.,* $\forall\ (id, M_1, M_2) \in \mathcal{M}^{\mathsf{H}}_{\mathsf{iDH}}, (id', M_1', M_2') \in \mathcal{M}^{\mathsf{H}}_{\mathsf{iDH}}$*,* **if** $id = id' \Rightarrow (M_1, M_2) = (M_1', M_2') \in \mathcal{M}_{\mathsf{iDH}}$*.*

$\mathsf{iDH}^{\mathsf{H}}(id, m)$	$\mathsf{H}(id)$
$1:\ \boldsymbol{g} \leftarrow \mathsf{H}(id)$	$1:$ **If** $\mathcal{Q}_\mathsf{H}[id] = \perp$:
$2:\ M_1 = \boldsymbol{g}^m$	$2:\ r \leftarrow\$\ \mathbb{Z}_\mathsf{p}$
$3:\ M_2 = h^m$	$3:\ \mathcal{Q}_\mathsf{H}[id] \leftarrow g^r := \boldsymbol{g}$
$4:$ **return** $(id, M_1, M_2) \in \mathcal{M}^{\mathsf{H}}_{\mathsf{iDH}}$	$4:$ **return** $\mathcal{Q}_\mathsf{H}[id]$

Fig. 8: Indexed Diffie-Hellman Message Space in the ROM.

More formally, we recall the indexed DH message space generator in Figure 8, such that $\mathsf{iDH}^{\mathsf{H}} : \mathbb{I} \times \mathbb{Z}_\mathsf{p} \to \mathbb{I} \times \mathbb{G}_1 \times \mathbb{G}_2$, where $\mathbb{I}$ is the index space.

A.1 Pseudo-Random Function (PRF)

Definition 6 (Pseudo-Random Functions [37]). *Let* $\mathcal{K}$ *and* $\mathcal{I}$ *be the key and input spaces, respectively. We say a family of functions like* $f : \mathcal{K} \times \mathcal{I} \to \mathcal{F}$ *is a pseudo-random function (PRF) family if it is efficiently computable and for all PPT distinguishers* $\mathcal{D}$ *we have,*

$$\left| \Pr_{k \leftarrow\$ \mathcal{K}}[\mathcal{D}^{\mathsf{PRF}_k}\ \textit{accepts}] - \Pr_{g \leftarrow\$ \mathcal{G}}[\mathcal{D}^{g}\ \textit{accepts}] \right| \leq \mathsf{negl}(\lambda)\ ,$$

where $\mathcal{D}^O$ *denotes the output of distinguisher* $\mathcal{D}$ *when given access to oracle* O*. It is assumed that the distinguisher can adaptively choose the inputs and* $\mathcal{G} : \mathcal{I} \to \mathcal{F}$ *is a set of uniformly random functions.*

We recall the weak notion of robustness for a PRF function, defined by Damgård et al. [25], such that no PPT adversary can find a key that produces collisions with a PRF generated by an honest key.

[5] No nontrivial homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ exists [36].

Definition 7 (Weakly-Robust PRF [25]). *A PRF scheme, Ψ_{PRF}, under query set $\mathcal{Q} = (x, y) \in \mathcal{I} \times \mathcal{F}$ is weakly-robust if for all PPT adversaries $\mathcal{A}$ we have:*

$$\Pr\begin{bmatrix} k \leftarrow\$ \mathsf{KGen}(\lambda), (x^*, k^*) \leftarrow\$ \mathcal{A}^{\mathsf{PRF}_k(.)}(1^\lambda) : \exists\ (x, y) \in \mathcal{Q}, \\ \mathsf{PRF}_{k^*}(x^*) = y = \mathsf{PRF}_k(x) \end{bmatrix} \leq \mathsf{negl}(\lambda)\ .$$

In order to make customers' collaterals reusable, we utilise a weakly-robust PRF proposed by Dodis and Yampolskiy [27]. Let a cyclic group $\mathbb{G}$ of prime order p with a generator g. The PRF on input $x \in \mathbb{Z}_\mathsf{p}$ under key space $\mathbb{Z}_\mathsf{p}$ is defined by $\mathsf{PRF}_k(x) = g^{1/(k+x)}$, where $k \leftarrow\$ \mathbb{Z}_\mathsf{p}$ is a secret key.

A.2 Shamir Secret Sharing

A (n, t)-Shamir Secret Sharing (SSS) [61] divides a secret s among n shareholders such that each subset of t shareholders can reconstruct secret s and any smaller subset of them learn nothing about the secret. For this purpose, the dealer who knows the secret s forms a polynomial $f(x)$ of degree $t + 1$ with a randomly chosen coefficients such that $f(0) = s$. Then the dealer securely provides each shareholder with $s_i = f(i), i \in \{1, \ldots, n\}$. Particularly, each subset of $\mathcal{T} \subset \{1, \ldots, n\}$ with size at least t by pooling their shares can reconstruct the secret s using the Lagrange polynomial interpolation as $s = f(0) = \sum_{i \in \mathcal{T}} s_i L_i^{\mathcal{T}}(0)$, where $L_i^{\mathcal{T}}(x) = \prod_{j \in \mathcal{T}, j \neq i} \frac{x-j}{i-j}$.

A.3 Digital Signatures and TSPS

Digital signatures are an electronic analogue of written signatures that ensure data authentication, and the non-repudiation of the sender. Informally, an aggregatable signature over message space $\mathcal{M}$ consists of four main PPT algorithms: setup, key generation, signing and verification. On input of the security parameter, the setup algorithm generates the public parameters pp, the randomized key generation algorithm generates a pair of secret signing/verification key $(\hat{\mathsf{sk}}, \mathsf{vk})$. The signing algorithm generates a signature σ by taking a message $m \in \mathcal{M}$ along with pp and the secret signing key $\hat{\mathsf{sk}}$. The deterministic verification algorithm takes the verification key vk, a message m, and a signature σ as inputs and outputs 1 (accept) if the signature is valid and 0 (reject), otherwise.

Definition 8 (Digital Signature). *For a given security parameter λ, a digital signature consists of the following PPT algorithms defined as follows:*

- $\mathsf{pp} \leftarrow \mathcal{DS}.\mathsf{Setup}(1^\lambda)$*: It takes the security parameter λ in its unary representation as input and outputs the set of public parameters* pp*.*
- $(\hat{\mathsf{sgk}}_i, \mathsf{vk}_i) \leftarrow \mathcal{DS}.\mathsf{KGen}(\mathsf{pp}, i)$*: This algorithm takes the global public parameters* pp *and an identification value i as inputs. It then returns the pair of signing/verification keys $(\hat{\mathsf{sgk}}_i, \mathsf{vk}_i)$ associated with a message space $\mathcal{M}$ and party i.*
- $\sigma_i \leftarrow \mathcal{DS}.\mathsf{Sign}(\mathsf{pp}, \hat{\mathsf{sgk}}_i, m)$*: On input of the signing key $\hat{\mathsf{sgk}}_i$ and a message $m \in \mathcal{M}$, this probabilistic algorithm outputs a signature σ_i.*
- $0/1 \leftarrow \mathcal{DS}.\mathsf{Vf}(\mathsf{pp}, \mathsf{vk}, \sigma, m)$*: This deterministic algorithm takes as inputs a set of verification keys $\mathsf{vk} := \{\mathsf{vk}_i\}$ for any $i \in [w]$, a signature σ and m and outputs either 1 (accept) or 0 (reject).*

Next we recall the primary security requirements for a digital signature: including *correctness* and *unforgeability against chosen message attack* and extendable to an aggregatable signature.

Definition 9 (Correctness). *A digital signature scheme, $\Psi_{\mathcal{DS}}$, is called correct, if we have,*

$$\Pr\left[\forall\ (\hat{\mathsf{sgk}}, \mathsf{vk}) \leftarrow \mathsf{KGen}(\mathsf{pp}), m \in \mathcal{M} : \mathsf{Vf}\left(\mathsf{pp}, \mathsf{vk}, m, \mathsf{Sign}(\mathsf{pp}, \hat{\mathsf{sgk}}, m)\right) = 1 \quad \right] \geq 1 - \mathsf{negl}(\lambda)\ .$$

Definition 10 (EUF-CMA**).** *A digital signature, $\Psi_{\mathcal{DS}}$, is called* EUF-CMA*-secure if for all PPT adversaries $\mathcal{A}$ with an access to the signing oracle $\mathcal{O}_{\mathsf{Sign}}()$ and the following winning advantage, $Adv^{\text{EUF-CMA}}_{DS,\mathcal{A}}(\lambda)$:*

$$\Pr\begin{bmatrix}\forall\ (\mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda), (\hat{\mathsf{sgk}}, \mathsf{vk}) \leftarrow \mathsf{KGen}(\mathsf{pp}), \\ (\sigma^*, m^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(\mathsf{pp}, \mathsf{vk}) : m^* \notin \mathcal{Q}_{msg} \wedge \mathsf{Vf}(\mathsf{vk}, \sigma^*, m^*) = 1\end{bmatrix}\ ,$$

where the signing oracle $\mathcal{O}_{\mathsf{Sign}}$ takes a message $m \in \mathcal{M}$, runs $\mathsf{Sign}(\mathsf{pp}, \hat{\mathsf{sgk}}, m)$ and adds the message to a query set $\mathcal{Q}_{msg}$. A digital signature is called EUF-CMA*-secure if $Adv^{\text{EUF-CMA}}_{DS,\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$.*

The term aggregatable signature refers to those that enable the aggregation of several distinct signatures into a single one. If all the aggregated signatures are valid then the verification algorithm fulfils on the aggregated signature. Next we formally define the aggregation algorithm as a supplementary algorithm.

- $\sigma \leftarrow \mathcal{DS}.\mathsf{Agg}(\mathsf{pp}, \{\sigma_i\}_{i\in[w]})$: To aggregate w different signatures σ_i for $i \in [w]$, it takes the public parameters pp as input and then returns as aggregated signature σ as output.

BLS Signatures: Based on the formal definition of a digital signature in Appendix A.3, we recall BLS signatures [14] as an efficient and aggregatable signature[6] For a given security parameter λ in its unary representation and cyclic group $(\mathbb{G}, \mathsf{p})$, the BLS signature consists of the following PPT algorithms:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: Given security parameter λ, it samples $h \leftarrow\!\!\$\ \mathbb{G}$ and a hash-to-curve function $\mathsf{H} : \{0,1\}^* \rightarrow \mathbb{G}$. It returns $\mathsf{pp} = (\mathsf{H}, h)$ as output.
- $(\mathsf{vk}, \hat{\mathsf{sgk}}) \leftarrow \mathcal{DS}.\mathsf{KGen}(\mathsf{pp})$: This algorithm takes public parameters pp and samples a random integer $s_i \leftarrow\!\!\$\ \mathbb{Z}^*_{\mathsf{p}}$ and returns the pair of signing/verification keys $(\hat{\mathsf{sgk}}_i, \mathsf{vk}_i) = (s_i, h^{s_i})$ for all $i \in [1, w]$.
- $\sigma_i \leftarrow \mathcal{DS}.\mathsf{Sign}(\mathsf{pp}, \hat{\mathsf{sgk}}_i, m)$: This probabilistic algorithm takes the pp, signing key $\hat{\mathsf{sgk}}_i$ and message $m \in \mathcal{M}$ as inputs and then computes $\sigma_i = \mathsf{H}(m)^{s_i}$ and returns the signature σ_i as output.
- $\sigma \leftarrow \mathcal{DS}.\mathsf{Agg}(\mathsf{pp}, \{\sigma_i\}_{i\in[1,w]})$: The aggregation algorithm takes pp, a set of distinct signatures $\{\sigma_i\}_{i\in[1,w]}$ and computes $\sigma = \prod_{i=1}^{w} \sigma_i$ and returns the aggregated signature σ as output.
- $0/1 \leftarrow \mathcal{DS}.\mathsf{Vf}(\mathsf{vk}, \sigma, m)$: The verification algorithm takes $\mathsf{vk} = \{\mathsf{vk}_i\}_{i=1}^{w}$, aggregated signature σ and message m as inputs. It return 1 (accept), if $\sigma \in \mathbb{G}_1$ and the equation $e(\sigma, h) = e\left(\mathsf{H}(m), \prod_{i=1}^{w} \mathsf{vk}_i\right)$ holds, otherwise it returns 0 (reject).

A digital signature is called structure-preserving [1], when it preserves the group structure over bilinear group setting, if it satisfies the following criteria:

- The verification key consists of $\mathbb{G}_1$ and $\mathbb{G}_2$ group elements.

[6] A rogue-key attack makes aggregatable BLS signatures insecure, but the technique described in [12] addresses this problem.

– The signature consists of group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$.
– The messages are composed of $\mathbb{G}_1$ and $\mathbb{G}_2$ elements.
– Only $\mathbb{G}_1$ and $\mathbb{G}_2$ membership and pairing product equations of the form of $\prod_i \prod_j \hat{e}(G_i, H_j)^{c_{i,j}} = 1_T$ need to be considered in the verification algorithm, where $G_i \in \mathbb{G}_1$ and $H_j \in \mathbb{G}_2$ and $c_{i,j} \in \mathbb{Z}_{\mathsf{p}}$.

By avoiding structure-destroying operations such as hash functions, SPS are able to construct efficient schemes when combined with other primitives such as Zero-Knowledge proof systems. In a SPS, both signed messages and signatures are group elements that can be used to verify the validity of a signature by performing pairing-product equations. These unique properties make the SPS schemes attractive for a variety of privacy-preserving applications, like anonymous credentials [42,19], anonymous e-cash [10] and access control encryptions [60]. Moreover, these signatures are efficiently re-randomizable under the knowledge of a secret randomness such that the re-randomized and original signatures are computationally indistinguishable. We utilise this property of re-randomization to ensure unlinkability of transactions in our construction.

Given the fact that the SPS relies on a single issuer then it does not meet our constructions desirable properties. In this aim, we recall the definition of Threshold Structure-Preserving Signatures from a recent work of Crites et al. [24]: it preserves the SPS's properties while mitigating the needed trust to a single entity.

Definition 11 (Threshold Structure-Preserving Signatures [24]). *For a given security parameter λ and an asymmetric bilinear group (given in Definition 4), a (n,t)-TSPS over message space $\mathcal{M}$, consists of the following PPT algorithms:*

– $\mathsf{pp} \leftarrow \mathcal{TSPS}.\mathsf{Setup}(1^\lambda)$*: Given the security parameter λ in its unary representation as input, it returns the set of global public parameters* pp *as the output.*
– $(\vec{\hat{\mathsf{sgk}}}, \vec{\mathsf{vk}}, \mathsf{vk}) \leftarrow \mathcal{TSPS}.\mathsf{KGen}(\mathsf{pp}, t, n)$*: Given the global public parameters* pp*, and integers $t, n \in \mathsf{poly}(1^\lambda)$ s.t. $1 \leq t \leq n$, as inputs it returns the vectors of secret signing keys $\vec{\hat{\mathsf{sgk}}} = (\hat{\mathsf{sgk}}_1, \ldots, \hat{\mathsf{sgk}}_n)$ and verification keys $\vec{\mathsf{vk}} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ along with a common verification key* vk *as output.*
– $\sigma_i \leftarrow \mathcal{TSPS}.\mathsf{ParSign}(\mathsf{pp}, \hat{\mathsf{sgk}}_i, m)$*: Given the public parameter* pp*, the i^{th} signing key $\hat{\mathsf{sgk}}_i$ and a message $m \in \mathcal{M}$ as inputs, it returns the partial signature σ_i as output.*
– $0/1 \leftarrow \mathcal{TSPS}.\mathsf{Par\text{-}Vf}(\mathsf{pp}, \mathsf{vk}_i, m, \sigma_i)$*: Given the i^{th} verification key vk_i, message $m \in \mathcal{M}$ and partial signature σ_i as inputs, it returns either 1 (accept) or 0 (reject).*
– $(\sigma, \perp) \leftarrow \mathcal{TSPS}.\mathsf{Reconst}(\mathsf{pp}, \{i, \sigma_i\}_{i \in \mathcal{T}})$*: Given* pp *and successfully verified partial signatures $\{i, \sigma_i\}$ over subset $\mathcal{T} \in \{1, \ldots, n\}$ as inputs, it returns a reconstructed signature σ if $|\mathcal{T}| \geq t$, otherwise it responds with $\perp$.*
– $0/1 \leftarrow \mathcal{TSPS}.\mathsf{Vf}(\mathsf{pp}, \mathsf{vk}, m, \sigma)$*: Given* pp*, verification key* vk*, a message $m \in \mathcal{M}$ and a reconstructed signature σ as inputs, it outputs either 1 (accept) or 0 (reject).*

As discussed in [24], two main security requirements for a TSPS scheme over the indexed Diffie-Hellman message spaces (as defined in Definition 5) are correctness and threshold existential unforgeability under chosen indexed message attacks. We refer the readers to [24] for more details.

We recall the proposed non-interactive TSPS in [24] based on the Pedersens's Distributed Key Generation [55] as follows.

– $(\mathsf{pp}) \leftarrow \mathcal{TSPS}.\mathsf{Setup}(1^\lambda)$: It executes the bilinear pairing group generator $\mathcal{BG}(1^\lambda)$ and returns $\mathsf{pp} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h, \mathsf{p})$ as output.

– $(\hat{\vec{\mathsf{sk}}}, \vec{\mathsf{vk}}, \mathsf{vk}) \leftarrow \mathcal{TSPS}.\mathsf{KGen}(\mathsf{pp}, t, n)$: For a given group of authorities $\{Au_1, \ldots, Au_n\}$, it acts as follows:
  1. Each authority Au_i, for $1 \leq i \leq n$, samples two initial random integers $(\alpha_{i0}, \beta_{i0}) \leftarrow\!\!\$\, \mathbb{Z}_{\mathsf{p}}^*$ and does the following:
     a) It samples t random pairs $\{\alpha_{ij}, \beta_{ij}\}_{j=1}^{t}$ and calculates two polynomials $A_i[X] = \alpha_{i0} + \alpha_{i1}X + \ldots + \alpha_{it}X^t \in \mathbb{Z}_{\mathsf{p}}[X]$ and $B_i[X] = \beta_{i0} + \beta_{i1}X + \ldots + \beta_{it}X^t \in \mathbb{Z}_{\mathsf{p}}[X]$ of degree t and then commits the randomly sampled coefficients by computing, $T_{ij} = (T_{1ij}, T_{2ij}) = (h^{\alpha_{ij}}, h^{\beta_{ij}})\ \forall j \in \{0, \ldots, t\}$.
     b) Au_i transfers the pair of $(A_i(\ell), B_i(\ell))$ to Au_ℓ s.t. $\ell \in \{1, \ldots, n\} \setminus \{i\}$ and keeps $(A_i(i), B_i(i))$ secret.
  2. Authority Au_i to checks the validity of the received shares from authority Au_ℓ, $(F_\ell(i), G_\ell(i))$, computes $h^{A_\ell(i)} = \prod_{j=0}^{t} T_{1\ell j}^{ij}$ and $h^{B_\ell(i)} = \prod_{j=0}^{t} T_{2\ell j}^{ij}$. It accepts the shares if these equations hold, else it rejects the shares and report the faulty authority Au_ℓ.
  3. An authority is called disqualified if it receives at least t complaints.Lastly t qualified authorities, $\mathcal{Q} \subset \mathcal{AU}$, continue the next steps.
  4. The qualified authorities compute the global verification key as $\mathsf{vk} := (\mathsf{vk}_1, \mathsf{vk}_2) := (\prod_{i\in\mathcal{Q}} T_{1i0}, \prod_{i\in\mathcal{Q}} T_{2i0}) = (h^{\sum_{i\in\mathcal{Q}} \alpha_{i0}}, h^{\sum_{i\in\mathcal{Q}} \beta_{i0}})$.
  5. Any $Au_i \in \mathcal{Q}$ sets its private key share $\hat{\mathsf{sk}}_i$ as a pair of $\hat{\mathsf{sk}}_i = (\mathsf{sk}_{i,1}, \mathsf{sk}_{i,2}) = \big(\sum_{\ell\in\mathcal{Q}} A_\ell(i), \sum_{\ell\in\mathcal{Q}} B_\ell(i)\big)$.
  6. Respectively the verification key vk_i can be generated by computing, $\mathsf{vk}_i = \big(h^{A(i)}, h^{B(i)}\big) = \Big(\prod_{\ell\in\mathcal{Q}} \prod_{j=0}^{t} (T_{1\ell j})^{i^j}, \prod_{\ell\in\mathcal{Q}} \prod_{j=0}^{t} (T_{2\ell j})^{i^j}\Big)$, where $A[X] = \sum_{\ell\in\mathcal{Q}} A_\ell[X]$ and $B[X] = \sum_{\ell\in\mathcal{Q}} B_\ell[X]$.
  7. For any $Au_j \notin \mathcal{Q}$, we have $\hat{\mathsf{sk}}_j = (0, 0)$ and corresponding verification key $\mathsf{vk}_j = (1_{\mathbb{G}_2}, 1_{\mathbb{G}_2})$.

  It then returns the vectors $\hat{\vec{\mathsf{sk}}} = (\hat{\mathsf{sk}}_1, \ldots, \hat{\mathsf{sk}}_n)$ and $\vec{\mathsf{vk}} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ for each party Au_i for $i \in [n]$ along with a common verification key vk.
– $(\sigma_i) \leftarrow \mathcal{TSPS}.\mathsf{ParSign}(\mathsf{pp}, \hat{\mathsf{sk}}_i, M)$: An authority $Au_i \in \mathcal{Q}$ possesses $\hat{\mathsf{sk}}_i$ given an indexed DH message $M := (id, M_1, M_2) \in \mathcal{M}_{\mathsf{iDH}}^{\mathsf{H}}$ as input, it runs the hash function $\mathsf{H}(id)$ to get the random basis $\boldsymbol{g}$. If $e(\boldsymbol{g}, M_2) = e(M_1, h)$, it computes the partial signature $\sigma_i = (\boldsymbol{g}, s_i) = (\boldsymbol{g}, \boldsymbol{g}^{\mathsf{sk}_{i1}} M_1^{\mathsf{sk}_{i2}})$ and returns σ_i as output, else it responds by $\perp$.
– $0/1 \leftarrow \mathcal{TSPS}.\mathsf{Par\text{-}Vf}(\mathsf{pp}, \mathsf{vk}_i, \tilde{M}, \sigma_i)$: The partial verification algorithm takes the i^{th} verification key, vk_i, a partial signature σ_i and message $\tilde{M} := (M_1, M_2) \in \mathcal{M}_{\mathsf{iDH}}$ as inputs. If all conditions: $M_1, s_i \in \mathbb{G}_1$, $\boldsymbol{g} \neq 1_{\mathbb{G}_1}$ and $M_2 \neq 1_{\mathbb{G}_2}$, $e(\boldsymbol{g}, M_2) = e(M_1, h)$ and $e(\boldsymbol{g}, \mathsf{vk}_{i1})e(M_1, \mathsf{vk}_{i2}) = e(s_i, h)$ hold, then it returns 1 and accepts the partial signature; otherwise it returns 0 and rejects it.
– $(\sigma, \perp) \leftarrow \mathcal{TSPS}.\mathsf{Reconst}(\mathsf{pp}, \{i, \sigma_i\}_{i\in\mathcal{T}})$: Given a set of well-formed partial signatures $\{\sigma_i\}_{i\in\mathcal{T}}$, it returns a reconstructed signature by computing $\sigma := (\boldsymbol{g}, s) := \Big(\boldsymbol{g}, \prod_{i\in\mathcal{T}} s_i^{L_i^{\mathcal{T}}(0)}\Big)$, where $L_i^{\mathcal{T}}(0)$ is the Lagrange coefficient for the i^{th} index corresponding to set $\mathcal{T}$ (see Appendix A.2) and returns the aggregated signature σ as output iff $|\mathcal{T}| \geq t$, otherwise it returns $\perp$.
– $0/1 \leftarrow \mathcal{TSPS}.\mathsf{Vf}(\mathsf{pp}, \mathsf{vk}, \tilde{M}, \sigma)$: If all conditions including: $\boldsymbol{g}, M_1, s \in \mathbb{G}_1$, $\boldsymbol{g} \neq 1_{\mathbb{G}_1}$ and $M_2 \neq 1_{\mathbb{G}_2}$, $e(\boldsymbol{g}, M_2) = e(M_1, h)$ and $e(\boldsymbol{g}, \mathsf{vk}_1)e(M_1, \mathsf{vk}_2) = e(s, h)$ hold, then the verification algorithm responds by 1 (accept), else it returns 0 (reject).

The correctness and Threshold EUF-CiMA-security with adaptive adversary is formally proved in [24]. Note that in Algorithm 1 no $\mathcal{TSPS}.\mathsf{Reconst}(.)$ algorithm is operated upon, and a user may execute this algorithm if it receives at least t partial signatures from the authorities.

A.4 Commitment schemes

A commitment scheme [16] is a strong cryptographic primitive, which allows a committer to commit to a secret value with two main security properties, i.e., *Perfect Hiding* and *Computational Binding*. Informally, perfect hiding guarantees that the commitment does not reveal any information about the hidden committed value. Computational hiding ensures that a committer cannot open a commitment under two distinct messages.

Definition 12 (Commitment schemes [16]). *A commitment scheme, Ψ_{CO}, over the message space of $\mathcal{M}$ and opening space of $\mathcal{T}$ consists of the following PPT algorithms:*

- $(\mathsf{pp}) \leftarrow \mathcal{CO}.\mathsf{Setup}(1^\lambda)$*: The setup algorithm takes the security parameter λ in its unary representation as input and returns the public parameters* pp *as output.*
- $(\mathsf{com}) \leftarrow \mathcal{CO}.\mathsf{Com}(\mathsf{pp}, m)$*: The commitment algorithm takes the public parameters* pp *and a message $m \in \mathcal{M}$ as inputs, and outputs a commitment $\mathsf{com} \in \mathcal{C}$ computed under the random opening value $\tau \in \mathcal{T}$.*
- $0/1 \leftarrow \mathcal{CO}.\mathsf{Vf}(\mathsf{pp}, \mathsf{com}, m', \tau')$*: The verification algorithm is a deterministic algorithm that given commitment $\mathsf{com} \in \mathcal{C}$, public parameters* pp*, a message $m' \in \mathcal{M}$ and an opening value $\tau' \in \mathcal{T}$ as inputs, returns a bit that indicates either accept (1) or reject (0).*

The primary security requirements for a commitment can be defined as follows:

Definition 13 (Correctness). *A commitment scheme, Ψ_{CO}, satisfies* correctness*, if we have:*

$$\Pr\left[\forall\ m \in \mathcal{M}\ \wedge\ (\mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda) : \mathsf{Vf}\left(\mathsf{pp}, m, \mathsf{Com}(\mathsf{pp}, m; \tau), \tau\right) = 1 \quad \right] \geq 1 - \mathsf{negl}(\lambda)\ .$$

Definition 14 (Computationally Hiding). *A commitment, Ψ_{CO}, satisfies Computationally Hiding, if for all PPT adversaries $\mathcal{A}$ we have:*

$$\left| 2\Pr\left[\begin{matrix}(\mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda), (m_0, m_1) \leftarrow\!\$\ \mathcal{A}^{\mathsf{Com}(.)}(\mathsf{pp}), b \leftarrow\!\$\ \{0,1\}, \\ (\mathsf{com}_b) \leftarrow \mathsf{Com}(\mathsf{pp}, m_b), b' \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{com}_b) : b == b'\end{matrix}\right] - 1 \right|\ .$$

The commitment scheme is called computationally and perfectly hiding if the above probability is $\leq \mathsf{negl}(\lambda)$ and equal to 0, respectively.

Definition 15 (Computationally Binding). *A cryptographic commitment scheme, Ψ_{com}, meets computationally binding security, if for all PPT adversaries $\mathcal{A}$ we have,*

$$\Pr\left[\begin{matrix}(\mathsf{pp}) \leftarrow \mathsf{Setup}(1^\lambda), ((\mathsf{com}, m_0, \tau_0), (\mathsf{com}, m_1, \tau_1)) \leftarrow\!\$\ \mathcal{A}^{\mathsf{Com}(.)}(\mathsf{pp}) : \\ \mathsf{Vf}(\mathsf{pp}, \mathsf{com}, m_0, \tau_0) = \mathsf{Vf}(\mathsf{pp}, \mathsf{com}, m_1, \tau_1) = 1\ \wedge\ m_0 \neq m_1\end{matrix}\right] \leq \mathsf{negl}(\lambda)\ .$$

The commitment scheme is called perfectly binding if the above probability is 0.

Our efficient instantiation utilizes Pedersen commitment [56] that is defined as follows:

- $\mathsf{pp} \leftarrow \mathcal{CO}.\mathsf{Setup}(1^\lambda)$: It takes the security parameter, λ, as input, picks two generators $g \leftarrow\!\$\ \mathbb{G}$ and $g_1 \leftarrow\!\$\ \mathbb{G}$ uniformly at random and returns the public parameters $\mathsf{pp} = (\mathbb{G}, \mathsf{p}, g, g_1)$ as output.
- $\mathsf{com} \leftarrow \mathcal{CO}.\mathsf{Com}(\mathsf{pp}, m)$: It takes the public parameters pp and a message $m \in \mathbb{Z}_\mathsf{p}$ as inputs, picks random opening $\tau \leftarrow\!\$\ \mathbb{Z}_\mathsf{p}^*$ and computes and outputs $\mathsf{com} = g^\tau g_1^m$.
- $0/1 \leftarrow \mathcal{CO}.\mathsf{Vf}(\mathsf{pp}, \mathsf{com}, m', \tau')$: It computes $\mathsf{com}' = g^{\tau'} g_1^{m'}$, if $\mathsf{com} = \mathsf{com}'$ it accepts and returns 1: else it responds with 0 and rejects the commitment.

A.5 Threshold Encryption

Next, recall the definition of threshold encryption schemes and then slightly modify it and define the notion of randomness reusable threshold encryption schemes.

Definition 16 (Threshold Encryption). *Given a set of public parameters* pp *along with security parameter* λ*, a* (n,t)*-threshold encryption,* Ψ_{TE}*, over the message space* $\mathcal{M}$ *and ciphertext space* $\mathcal{C}$ *consists of four main PPT algorithms defined as follows:*

- $(\hat{\vec{\mathsf{sk}}}, \mathsf{pk}) \leftarrow \mathcal{TE}.\mathsf{KGen}(\mathsf{pp}, n, t)$*: Key generation is a probabilistic algorithm that takes the set of public parameter* pp *along with two integers* $t, n \in \mathsf{poly}(\lambda)$ *as inputs. It then returns a vector of secret key* $\hat{\vec{\mathsf{sk}}}$ *of size* n *and a general public key* pk *as outputs.*
- $\mathtt{Ct} \leftarrow \mathcal{TE}.\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m)$*: The encryption algorithm as a probabilistic algorithm takes the public parameters* pp*, global public key* pk *and a message* $m \in \mathcal{M}$ *as inputs. It returns ciphertext* $\mathtt{Ct} \in \mathcal{C}$ *as output. When we want to assign a specific value to the random integer* r*, we write* $\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m; r)$*.*
- $\mathsf{pd}_j \leftarrow \mathcal{TE}.\mathsf{PDec}(\mathsf{pp}, \mathsf{sk}_j, \mathtt{Ct})$*: The partial decryption algorithm is run by receiver* $j \in \mathcal{R}$ *and takes the public parameters* pp*, the secret key* $\hat{sk}_j$ *of the receiver* j *and a ciphertext* $\mathtt{Ct}$ *as inputs. It returns a partially decrypted ciphertext* pd_j *as output.*
- $(\perp, m) \leftarrow \mathcal{TE}.\mathsf{Dec}(\mathsf{pp}, \mathtt{Ct}, \{\mathsf{pd}_j\}_{j\in\mathcal{K}})$*: The decryption algorithm takes the public parameters* pp*, a ciphertext* $\mathtt{Ct}$ *and the partially decrypted ciphertexts* $\{\mathsf{pd}_j\}_{j\in\mathcal{K}}$ *as inputs. If* $|\mathcal{K}| \geq t$*, it returns* $m \in \mathcal{M}$*, else it responds by* $\perp$*.*

The primary security requirements for a (n,t)-threshold encryption are correctness and *static semantic security* and *partial decryption simulatability* based on the static security definitions of Reyzen et al. [58].

Definition 17 (Correctness). *A* (n,t)*-threshold encryption,* Ψ_{TE}*, for all security parameters* λ *and messages* $m \in \mathcal{M}$ *and set of receivers* $\mathcal{R}$ *is called correct if for any* $|\mathcal{K}| \geq t$ *we have:*

$$\Pr\left[\begin{array}{l}\forall \mathsf{pp}, \lambda, n, t, (\hat{\vec{\mathsf{sk}}}, \mathsf{pk}) \leftarrow \mathsf{KGen}(\mathsf{pp}, n, t) : \\ \mathsf{Dec}\left(\mathsf{pp}, \mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m), \{\mathsf{PDec}(\mathsf{pp}, \hat{\mathsf{sk}}_j, \mathtt{Ct})\}_{j\in\mathcal{K}}\right) = m\end{array}\right] \geq 1 - \mathsf{negl}(\lambda) \ .$$

Definition 18 (Static Semantic Security [58]). *A* (n,t)*-threshold encryption,* Ψ_{TE}*, is said to be* (n,t)*-statically semantic secure (SSS) if for all PPT adversaries* $\mathcal{A}$ *in winning the following experiment we have* $\Pr[\mathrm{Exp}^{SSS}_{\mathcal{A}}(1^\lambda, \mathcal{R}, t) = 1] \leq 1/2 - \mathsf{negl}(\lambda)$*. Where adversary* $\mathcal{A}$ *has access to a partial decryption oracle,* $\mathcal{O}_{\mathsf{PDec}}$*, and can obtain up to* $t-1$ *partially decrypted values of the given ciphertext.*

$\text{Exp}^{SSS}_{\mathcal{A}}(1^\lambda, \mathcal{R}, t)$

$\mathcal{C} \leftarrow \mathcal{A}(1^\lambda, \mathcal{R}, t)$

$(\hat{\mathsf{sk}}_i) \leftarrow \mathcal{TE}.\mathsf{KGen}(\mathsf{msk}, i)$ For $i \in [n]$;

$(m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{PDec}}}(\{\hat{\mathsf{sk}}_i\}_{i\in\mathcal{C}})$;

$b \leftarrow\$ \{0, 1\}$,

$(\mathtt{Ct}_b) \leftarrow \mathcal{TE}.\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m_b)$;

$b' \leftarrow\$ \mathcal{A}^{\mathcal{O}_{\mathsf{PDec}}}(\mathtt{Ct}_b)$;

return $(b' = b \wedge |m_0| \neq |m_1| \wedge |\mathcal{R} \cap \mathcal{K}| < t)$

Fig. 9: Static Semantic Security Experiment.

A.6 Collaborative Key Generation and RRTE

Collaborative Key Generation (CKG) is a slightly modified variant of DKG that uses threshold cryptography to achieve distributed key generation. The (n, t)-DKG generates n pairs of secret and public keys s.t. at least t parties among n are required to execute a key oriented operation, whereas any subset of size smaller than t is not able to execute it. Likewise in a (n, t, k)-CKG, a global public key pk corresponding to a secret key sk is shared among n parties s.t. any subset of larger than k can rebuild pk and any subset of larger than t, where $k < t \leq n$ can reconstruct the sk. The main difference here is that any operation that needs the secret key sk requires the cooperation of at least t collaborators, whereas the public key pk itself can be reconstructed by at least $k < t$ parties. For a given public parameters $\mathsf{pp} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, h, \mathsf{p}, e)$, we instantiate a CKG construction as follows:

– $(\vec{\mathsf{pk}}, \mathsf{pk}) \leftarrow \mathsf{CKG}(\mathsf{pp}, k, t, k)$: The CKG is a collaborative algorithm that is executed by the group of authorities $\mathcal{AU}$ of size n to generate public keys for ℓ merchants, defined in the Algorithm 1 with a fixed threshold $k = 2$, while their corresponding secret keys remain hidden as long as the majority of the authorities are honest ($t \geq n/2 + 1$).

1. Each authority $\mathcal{AU}_i$ samples an initial random value $x_{i0} \leftarrow\$ \mathbb{Z}^*_{\mathsf{p}}$ and does the following:
   a) It samples a random integer $\{x_{i1}\}$, forms a polynomial $F_i[X] = x_{i0} + x_{i1}X \in \mathbb{Z}_{\mathsf{p}}[X]$ and commits the coefficients by publishing, $V_{ij} = h^{x_{ij}}\ \forall\ j \in \{0, 1\}$.
   b) It broadcasts $F_i(j)$ to the rest of authorities as a share corresponding to the j^{th} merchants.
2. Each authority checks the consistency of the received shares, $F_i(j)$, from $\mathcal{AU}_i$ by computing the equations $g^{F_i(j)} = V_{i0}V_{i1}^j$ for all merchants' label $j \in [\ell]$. If this equation holds, the shares generated by $\mathcal{AU}_i$ will be accepted, otherwise it will reject and then report the faulty authority $\mathcal{AU}_i$.
3. Any faulty authority that receives at least $t \geq n/2 + 1$ complaints is labelled as disqualified. At the end of this phase t parties from the set of qualified authorities, $\mathcal{Q} \subset \mathcal{AU}$ perform the next steps.
4. The global public key is determined as $\mathsf{pk} := \prod_{i\in\mathcal{Q}} V_{i0} = h^{\sum_{i\in\mathcal{Q}} x_{i0}}$.
5. Each merchant $\mathcal{M}_j$ is assigned by a public key pk_j that is obtained by computing, $\mathsf{pk}_j = h^{F(j)} = \prod_{i\in\mathcal{Q}} \left(V_{i0}(V_{i1})^j\right)$, where $F[X] = \sum_{i\in\mathcal{Q}} F_i[X]$.

These steps complete the CKG phase and return the set of public keys $\{\mathsf{pk}_j\}_{1\leq j\leq\ell}$ along with the global public key pk.

An efficient RRTE. We propose an efficient RRTE scheme that relies on CKG constructions Appendix A.6.

- $(\vec{\mathsf{pk}}, \mathsf{pk}) \leftarrow \mathcal{RRTE}.\mathsf{KGen}(\mathsf{pp}, \ell, t, 2)$: It runs the CKG algorithm, $\mathsf{CKG}(\mathsf{pp}, \ell, k, t, 2)$, as described in Appendix A.6.
- $(\mathtt{Ct}_j, v) \leftarrow \mathcal{RRTE}.\mathsf{Enc}(\mathsf{pp}, \mathsf{pk}, m, \mathsf{pk}_j)$: The encryption algorithm takes public parameters pp, global public key pk, the message m and public key pk_j as inputs. It samples $r \leftarrow\!\!\$\, \mathbb{G}_1$ uniformly at random and generates the ciphertext underlying each recipient by computing $(\mathtt{Ct}_j, v) := \big(e(r, \mathsf{pk}_j), m \cdot e(r, \mathsf{pk})\big)$.
- $(m, \perp) \leftarrow \mathcal{RRTE}.\mathsf{Dec}(\mathsf{pp}, \mathtt{Ct}_j, \mathtt{Ct}_{j'}, v)$: The decryption algorithm takes twin ciphertexts $(\mathtt{Ct}_j, v)$ and $(\mathtt{Ct}_{j'}, v)$ along with public parameters pp as inputs. Let $\mathcal{J} = \{j, j'\}$, it computes $g_T^{rs} : z = \left(\mathtt{Ct}_j^{L_j^{\mathcal{J}}(0)} \cdot \mathtt{Ct}_{j'}^{L_{j'}^{\mathcal{J}}(0)}\right)$ and then returns $m := v/z$, otherwise, it responds with $\perp$.

A.7 Non-Interactive Zero-Knowledge proofs

Zero-Knowledge proofs [38] are two-party protocols, which are a fundamental and powerful cryptographic tool. They allow a prover to convince the verifier about the validity of a statement without revealing any other information. Non-Interactive Zero-Knowledge proofs remove the interaction between the parties in two possible settings, either the Random Oracle Model (ROM) [34] or the Common Reference String (CRS) model [11]. We recall the definition of NIZK arguments[7] in the CRS model, in which the prover is computationally bounded to ensure the soundness. Hence, for security parameter λ, let $\mathcal{R}$ be a relation generator, such that $\mathcal{R}(1^\lambda)$ returns an efficiently computable binary relation $\mathbf{R_L} = \{(\mathsf{x}, \mathsf{w})\}$, where x is the instance (statement) and w is the corresponding witness. Let $\mathbf{L} = \{\mathsf{x} : \exists\, \mathsf{w} \mid (\mathsf{x}, \hat{\mathsf{w}}) \in \mathbf{R_L}\}$ be the NP-language consisting of the statements in relation $\mathbf{R_L}$.

Definition 19 (NIZK arguments). *Formally, a NIZK argument, Ψ_{NIZK}, for $\mathcal{R}$ consists of a tuple of PPT algorithms $\mathcal{ZK}.(\mathsf{K}_{\vec{\mathsf{crs}}}, \mathsf{P}, \mathsf{Vf}, \mathsf{Sim})$, defined as follows:*

- $(\vec{\mathsf{crs}}, \hat{\vec{\mathsf{ts}}}, \hat{\vec{\mathsf{te}}}) \leftarrow \mathcal{ZK}.\mathsf{K}_{\vec{\mathsf{crs}}}(1^\lambda, \mathbf{R_L})$*: The CRS generator as a probabilistic algorithm takes the security parameter 1^λ and relation $\mathbf{R_L}$ as inputs. It then generates common reference string $\vec{\mathsf{crs}}$ by sampling a simulation trapdoor $\hat{\vec{\mathsf{ts}}}$ and an extraction trapdoor $\hat{\vec{\mathsf{te}}}$. It keeps the trapdoors $(\hat{\vec{\mathsf{te}}}, \hat{\vec{\mathsf{ts}}})$ hidden while publishes $\vec{\mathsf{crs}}$.*
- $(\pi, \perp) \leftarrow \mathcal{ZK}.\mathsf{P}(\mathbf{R_L}, \vec{\mathsf{crs}}, \mathsf{x}, \hat{\mathsf{w}})$*: Prove as a probabilistic algorithm takes the CRS, $\vec{\mathsf{crs}}$, and a pair of statement and witness $(\mathsf{x}, \hat{\mathsf{w}})$ as inputs. If $(\mathsf{x}, \hat{\mathsf{w}}) \in \mathbf{R_L}$ it returns a proof π, otherwise it responds with $\perp$. This algorithm sometimes is denoted by $\mathsf{PoK}\{\hat{\mathsf{w}} \mid (\mathsf{x}, \hat{\mathsf{w}}) \in \mathbf{R_L}\}$.*
- $0/1 \leftarrow \mathcal{ZK}.\mathsf{Vf}(\mathbf{R_L}, \vec{\mathsf{crs}}, \mathsf{x}, \pi)$*: Verification as a deterministic algorithm takes CRS, $\vec{\mathsf{crs}}$, and a pair of statement and proof (x, π) as inputs. It either returns 1 (accept) or 0 (reject).*
- $\pi' \leftarrow \mathcal{ZK}.\mathsf{Sim}(\mathbf{R_L}, \vec{\mathsf{crs}}, \hat{\vec{\mathsf{ts}}}, \mathsf{x})$*: The Simulator algorithm takes the tuple $(\mathbf{R_L}, \vec{\mathsf{crs}}, \vec{\mathsf{ts}}, \mathsf{x})$ as input and without knowing the corresponding secret witness, outputs a simulated proof π' s.t. it is computationally indistinguishable from π.*

[7] The CRS does not depend on the language distribution or language parameters.

Definition 20 (Completeness). *A NIZK argument, Ψ_{NIZK}, is called* complete *for relation $\mathbf{R_L} \in \mathcal{R}$, if for all security parameters 1^λ and $(\mathsf{x}, \hat{\mathsf{w}}) \in \mathbf{R_L}$, we have:*

$$\Pr\left[(\vec{\mathsf{crs}}, \hat{\vec{\mathsf{ts}}}, \hat{\vec{\mathsf{te}}}) \leftarrow \mathsf{K}_{\vec{\mathsf{crs}}}(1^\lambda, \mathbf{R_L}) : \mathsf{Vf}(\mathbf{R_L}, \vec{\mathsf{crs}}, \mathsf{x}, \mathsf{P}(\mathbf{R_L}, \vec{\mathsf{crs}}, \mathsf{x}, \hat{\mathsf{w}})) = 1 \quad \right] \geq 1 - \mathsf{negl}(\lambda) \ .$$

Definition 21 (Soundness). *A NIZK argument, Ψ_{NIZK}, is* Sound *for any relation $\mathbf{R_L} \in \mathcal{R}$, if for all PPT adversaries $\mathcal{A}$, we have:*

$$\Pr\left[(\vec{\mathsf{crs}}, \hat{\vec{\mathsf{ts}}}, \hat{\vec{\mathsf{te}}}) \leftarrow \mathsf{K}_{\vec{\mathsf{crs}}}(1^\lambda, \mathbf{R_L}), (\mathsf{x}, \pi) \leftarrow \mathcal{A}(\mathbf{R_L}, \vec{\mathsf{crs}}) : \mathsf{Vf}(\mathbf{R_L}, \vec{\mathsf{crs}}, \mathsf{x}, \pi) = 1 \wedge \mathsf{x} \notin \mathbf{L_R} \quad \right] \leq \mathsf{negl}(\lambda) \ .$$

Definition 22 (Statistically Zero-Knowledge). *A NIZK argument, Ψ_{NIZK}, is called* statistically Zero-Knowledge*, if for all security parameter 1^λ, and all PPT adversaries $\mathcal{A}$ we have, $\varepsilon_0^{unb} \approx_\lambda \varepsilon_1^{unb}$, where,*

$$\varepsilon_b = \Pr\left[(\vec{\mathsf{crs}}, \hat{\vec{\mathsf{ts}}}, \hat{\vec{\mathsf{te}}}) \leftarrow \mathsf{K}_{\vec{\mathsf{crs}}}(1^\lambda, \mathbf{R_L}) : \mathcal{A}^{\mathsf{O}_b(\cdot,\cdot)}(\mathbf{R_L}, \vec{\mathsf{crs}}) = 1\right] \cdot$$

Here, the oracle $\mathcal{O}_0(\mathsf{x}, \hat{\mathsf{w}})$ returns $\perp$ (reject) if $(\mathsf{x}, \hat{\mathsf{w}}) \notin \mathbf{R_L}$, otherwise it returns $\mathcal{ZK}.\mathsf{P}(\mathbf{R_L}, \vec{\mathsf{crs}}, \mathsf{x}, \hat{\mathsf{w}})$. Similarly, $\mathcal{O}_1(\mathsf{x}, \hat{\mathsf{w}})$ returns $\perp$ (reject) if $(\mathsf{x}, \hat{\mathsf{w}}) \notin \mathbf{R_L}$, otherwise it returns $\mathcal{ZK}.\mathsf{Sim}(\mathbf{R_L}, \vec{\mathsf{crs}}, \mathsf{x}, \hat{\vec{\mathsf{ts}}})$.

Definition 23 (Computational Knowledge-Soundness). *A NIZK argument, Ψ_{NIZK}, is called computationally (adaptively)* knowledge-sound for *$\mathcal{R}$, if for all PPT adversary $\mathcal{A}$ and $\mathbf{R_L} \in \mathcal{R}$, there exists an extractor $\mathsf{Ext}_{\mathcal{A}}$, s.t. for all 1^λ we have,*

$$\Pr\left[\begin{array}{l}(\vec{\mathsf{crs}}, \hat{\vec{\mathsf{ts}}}, \hat{\vec{\mathsf{te}}}) \leftarrow \mathsf{K}_{\vec{\mathsf{crs}}}(1^\lambda, \mathbf{R_L}), (\mathsf{x}, \pi) \leftarrow \mathcal{A}(\mathbf{R_L}, \vec{\mathsf{crs}}), \\ (\hat{\mathsf{w}}) \leftarrow \mathsf{Ext}_{\mathcal{A}}(\mathbf{R_L}, \vec{\mathsf{crs}}, \hat{\vec{\mathsf{te}}}, \pi) : (\mathsf{x}, \hat{\mathsf{w}}) \notin \mathbf{R_L} \ \wedge \ \mathsf{Vf}(\mathbf{R_L}, \vec{\mathsf{crs}}, \mathsf{x}, \pi) = 1\end{array}\right] \leq \mathsf{negl}(\lambda) \ .$$

Next, we overview the used proof systems in the implementation of the proposed efficient instantiation. As we already discussed in Section 4, we rely on three main proof systems including the Sigma protocols, Groth-Sahai proof systems and Range-proofs.

Sigma Protocols [59]. In what follows we recall the standard Sigma protocols and also some techniques used recently in [25,50]. All the protocols are interactive and we use the Fiat-Shamir paradigm to make make then non-interactive; additionally, the existence of a collision-resistance hash function $\mathsf{H} : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$ modeled in random oracle is assumed. Thus, the prover runs the hash function to obtain the challenge c instead of taking the random challenges from the verifier over an extra round of communication.

Proof of knowledge of DLog. Figure 10 outlines an interactive Sigma protocol to prove the knowledge of discrete logarithm, $x \in \mathbb{Z}_p$, of a public value g^x, where g is the generator of a cyclic group $\mathbb{G}$ with prime order p.

To prove a multiplicative relation between two committed values. Figure 11 describes an interactive Sigma protocol to prove the knowledge of three integers x_1, x_2, x_3 with public Pedersen commitments of them, i.e. $\mathsf{com}_i = g^{\tau_i} g_1^{x_i}$ for $i = 1, 2, 3$ and also showing the fact $x_3 = x_1 x_2 \mod p$.

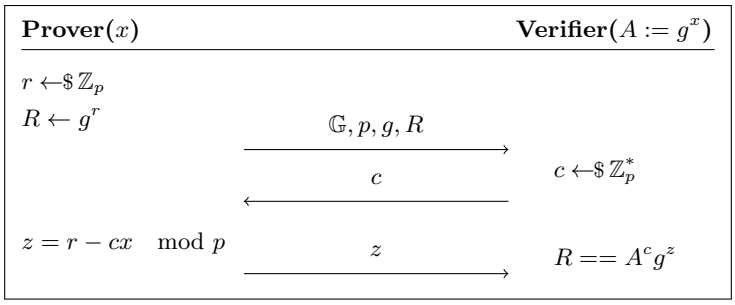


Fig. 10: Proving the knowledge of DLog.

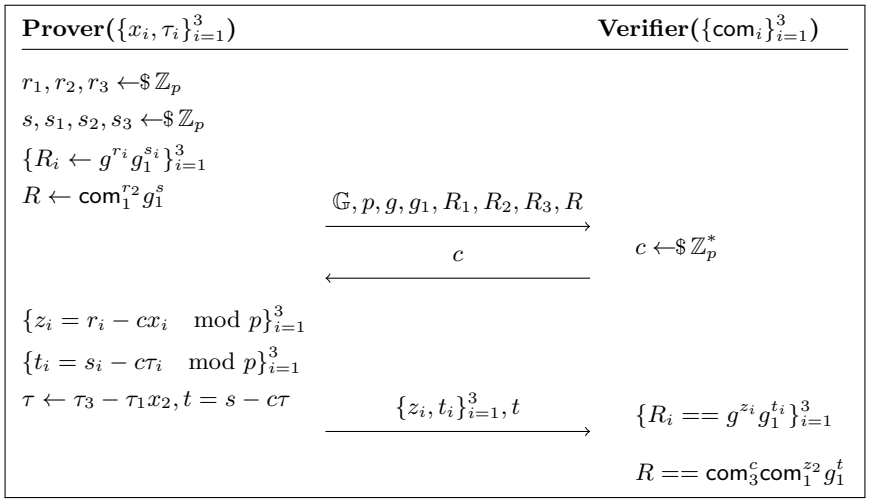


Fig. 11: To prove a multiplication relation between committed values.

To prove the well-formedness of the Dodis-Yampolskiy PRF. Figure 12 outlines a Sigma protocol to prove the knowledge of a seed $k \in \mathcal{K}_{\mathsf{PRF}}$ and showing the fact that it is correctly computed. For this, we rely on the above mentioned Sigma protocol and denote it by $\Sigma.\mathsf{MultiCom}(.)$.

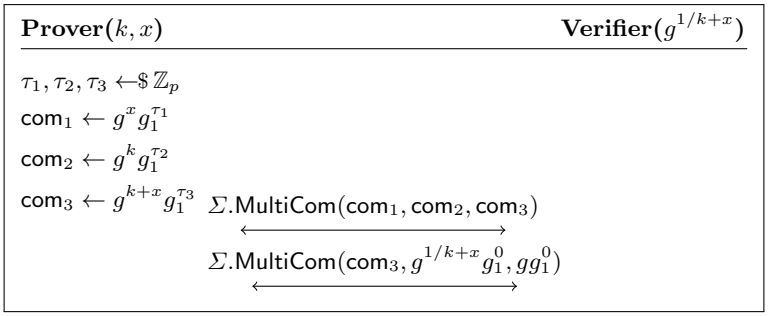


Fig. 12: Proving well-formedness of the DY's PRF.

Groth-Sahai (GS) proof system [40]. The GS proof system is a commit-and-prove system where the prover takes CRS and commits to the hidden variables (witnesses), subsequently providing proofs to convince the verifier that the PPE relation is valid under the generated commitments. In a subsequent work, Escala and Groth in [31] show that by replacing the commitment with ElGamal encryption [30] the prover's computation cost can be reduced. Thus in this paper the prover to commit to hidden values (witnesses) computes an ElGamal encryption of the witnesses. In the implementation, we take one of the instantiation proposed in [40] that relies on the symmetric external Diffie-Hellman (SXDH) assumption over prime order groups. Over an asymmetric bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{p}, e)$, the relation over variables $\mathcal{X}_1, \ldots, \mathcal{X}_m \in \mathbb{G}_1$, $\mathcal{Y}_1, \ldots, \mathcal{Y}_n \in \mathbb{G}_2$ and a constant $T \in \mathbb{G}_T$ can be any product-pairing equation (PPE) on the form:

$$\prod_{i=1}^{n} e(A_i, \mathcal{Y}_i) \prod_{i=1}^{m} e(\mathcal{X}_i, B_i) \prod_{j=1}^{m} \prod_{i=1}^{n} e(\mathcal{X}_j, \mathcal{Y}_i)^{\mu_{i,j}} = T \ ,$$

where $(A_1, \ldots, A_n) \in \mathbb{G}_1$, $(B_1, \ldots, B_m) \in \mathbb{G}_2$ and $\{\mu_{i,j}\}_{1 \leq i \leq m \ \& \ 1 \leq j \leq n} \in \mathbb{Z}_{\mathsf{p}}^{m \cdot n}$ are constant.

Range-proofs. These constructions enable a prover to prove that a hidden witness x lies within a certain interval like $(0, Max]$, where Max can be any integer upper bound in this relation. In the implementation of our construction we rely on Bulletproof proposed by Bunz et al. [17].

A.8 NIZK Language Realization

Next, we briefly discuss how the described languages in Section 3.1 are implemented.

- Language $\mathbf{L}_1$: Towards the realization of this language we rely on Sigma protocols. The customer runs the described Sigma protocol in Figure 10 on the proof of knowledge of $\hat{\mathsf{sk}}_{cm}$.
- Language $\mathbf{L}_2$: To realize this language, we rely on Sigma protocols, Range-proofs and GS proof systems. To show the commitment k'_j is formed correctly and it is indeed the Pedersen commitment of the hidden key $\hat{k}_j$, the prover runs the described protocol in Figure 11 on a single commitment. To be more precise, the customer can assign $x_2 = 0$ and perform this protocol. Additionally, the customer runs the range-proof on the committed PRF in the previous relation and depicts the fact that it samples correctly from the key distribution of the PRF, i.e. $\mathcal{K}_{\mathsf{PRF}}$. To show the customer has the knowledge of a valid certificate $\hat{\mathsf{cert}}_{cn}$ signed by at least t authorities, we rely on GS proof systems. As we already discussed, the verification equation of the given TSPS can be written as a pairing-product equation and the customer can convince the merchant on this. The main advantage of this proof system is that the prover does not need to necessarily know the discrete logarithm of the message and it is compatible with all group element witnesses. Moreover, this enables us to have a straight-line extraction of the witnesses and allows proof aggregation.
- Language $\mathbf{L}_3$: The instantiation of this language is done based on two proof systems: Sigma protocols and GS proof systems. To prove the well-formedness of the PRF evaluation function and showing the fact that the prover knows the key $\hat{k}_j$, the customer runs the described Sigma protocol in Figure 12. Since the remaining relations of this language can be written as a PPE, the prover can utilise the GS proof systems.

In all of the above cases, all relations and languages are proven solely and a bridge can be made between them whenever the issued commitments are all committing to the same hidden witness. Sigma protocols can be used to demonstrate that the committed values in two distinct commitments are the same.