








Reusable, Instant and Private Payment Guarantees for Cryptocurrencies

Akash Madhusudan¹ , Mahdi Sedaghat¹ , Samarth Tiwari² ,
Kelong Cong¹ , and Bart Preneel¹ 

¹ imec-COSIC, KU Leuven, Leuven, Belgium

{[akash.madhusudan](mailto:akash.madhusudan@esat.kuleuven.be), [ssedagha](mailto:ssedagha@esat.kuleuven.be), [kelong.cong](mailto:kelong.cong@esat.kuleuven.be), [bart.preneel](mailto:bart.preneel@esat.kuleuven.be)}@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 s.

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 [25]. 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 [7, 11, 20]. 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 [18], and the route discovery mechanisms [33]. 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 [4]. 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 ~ 14 min [21]. 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 [29]. 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 [27]. 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. [31] 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 s.

Outline. The rest of this paper is organized as follows: Sect. 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 Sect. 5 we evaluate the performance of this instantiation while focusing on latency. In Sect. 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 Sect. 7 we conclude our paper.

2 Our Construction

2.1 Our Solution

The strawman solution shown in the full version [28] 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) [16] 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 Sect. 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**.

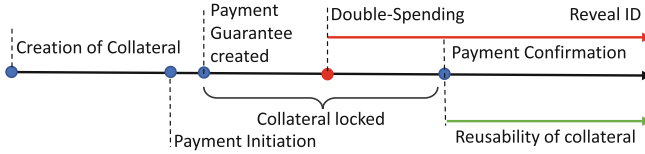


Fig. 1. Timeline of a transaction.

Finally, our construction also allows a customer to *reuse* their collaterals after a certain time interval in order to achieve **P4**. As illustrated in Fig. 1, 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 [1, 33]. 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 2 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 Fig. 2 with dotted and solid arrows respectively. The protocol proceeds as follows:

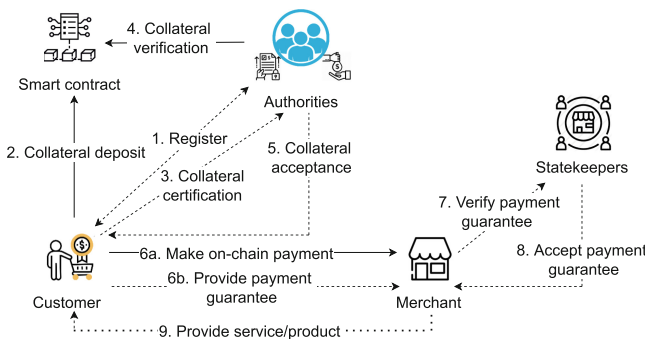


Fig. 2. Our construction.

- 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 Fig. 3.

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

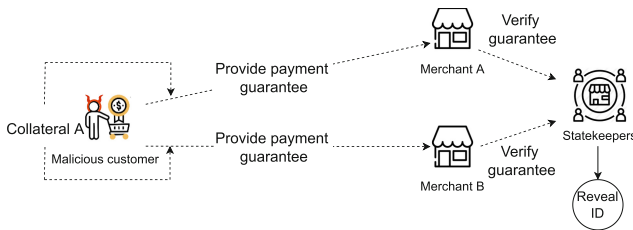


Fig. 3. Proactive double-spending detection.

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 Sect. 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 $\text{negl}(\lambda)$ 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., $\hat{\mathbf{s}}\mathbf{k}$) 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 the full version [28] 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, \mathbf{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 Fig. 2:

- 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 \mathbf{mpk} .
- AuKeyGen(\mathcal{A}) is executed by \mathcal{A} that returns $(\hat{\mathbf{sg}}k_{ai}, \mathbf{vk}_{ai})$ for $i \in [n]$ along with a global verification key \mathbf{vk}_a . $\mathcal{A}[\hat{\mathbf{sg}}k_{ai}, \mathbf{vk}_{ai}, \mathbf{vk}_a]_{i=1}^n$ represents the list of credentials for \mathcal{A} .
- MKeyGen(M_m) is executed by merchant $M_m \in \mathcal{M}$ in order to join the network. It initially generates a pair of signing/verification keys $(\hat{\mathbf{sg}}k_{bm}, \mathbf{vk}_{bm})$ and returns a tuple $(\hat{\mathbf{sg}}k_{bm}, \mathbf{vk}_{bm}, \mathbf{pk}_{bm} = \perp)$. The list of keys belonging to the group of merchants is recorded in $\mathcal{M}[\hat{\mathbf{sg}}k_{bi}, \mathbf{vk}_{bi}]_{i=1}^\ell$.
- MRegister($\mathcal{A}[\hat{\mathbf{sg}}k_{ai}]_{i=1}^t, \mathcal{M}[\mathbf{vk}_{bi}]_{i=1}^\ell$) is executed by any subset of \mathcal{A} of size at least t to register the merchants who deposit a collateral to join the merchant consortium and assign them a public key \mathbf{pk}_{bm} . For each merchant $M_m \in \mathcal{M}$, it takes the secret signing key of the authorities $\mathcal{A}[\hat{\mathbf{sg}}k_{ai}]$ for $1 \leq i \leq t$, and returns public key \mathbf{pk}_{bm} as output. After this phase, the list of parameters for the m^{th} merchant can be updated as $\mathcal{M}[\hat{\mathbf{sg}}k_{bm}, \mathbf{vk}_{bm}, \mathbf{pk}_{bm}]$.
- CuKeyGen(C_n) is executed by the customers, that for each customer $C_n \in \mathcal{C}$, a Pseudo-ID generator function (PID) generates an initial secret key $\hat{\mathbf{sk}}_{cn}$ and its corresponding public key \mathbf{pk}_{cn} . It returns a tuple of $(\mathbf{pk}_{cn}, \hat{\mathbf{sk}}_{cn}, \hat{\mathbf{cert}}_{cn} = \perp)$ with a NIZK proof π_1 to prove that the relation \mathbf{R}_{L_1} fulfills. The list of customers' keys are kept in $\mathcal{C}[\hat{\mathbf{sk}}_{ci}, \mathbf{pk}_{ci}, \hat{\mathbf{cert}}_{ci} = \perp]_{i=1}^k$.

Protocol as Depicted in Fig. 2:

- CuRegister($\mathcal{A}[\hat{\mathbf{sg}}k_{ai}]_{i=1}^t, \mathcal{C}[\mathbf{pk}_{cn}], \pi_1$) is depicted in step 1 of Fig. 2. It is executed by any subset of \mathcal{A} of size at least t to certify the public key \mathbf{pk}_{cn} of a customer C_n corresponds to some secret value $\hat{\mathbf{sk}}_{cn}$ under the relation \mathbf{R}_{L_1} . Once the customer C_n is registered by the authorities, it receives a certificate $\hat{\mathbf{cert}}_{cn}$ and it updates $\mathcal{C}[\mathbf{pk}_{cn}, \hat{\mathbf{sk}}_{cn}, \hat{\mathbf{cert}}_{cn}]$.
- CuCreate($\mathcal{C}[\hat{\mathbf{sk}}_{cn}, \hat{\mathbf{cert}}_{cn}]$) is depicted in step 2–3 of Fig. 2. It is executed by the customer to request for certification of their collateral. A successfully registered customer C_n , with certificate $\hat{\mathbf{cert}}_{cn} \neq \perp$, 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}_{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}_{L_2} fulfills.

Algorithm 1: OUR CONSTRUCTION.

Function PGen($1^\lambda, \mathbf{R}_L$):

```

  (cṛs, t̂s, t̂e) ← ZK.Kcṛs( $1^\lambda, \mathbf{R}_L$ )
  (pp) ← TSPS.Setup( $1^\lambda$ )
  return mpk := (pp, cṛs)

```

Function AuKeyGen(\mathcal{AU}):

```

  (sgka, vḱa, vka) ←
    TSPS.KGen(mpk, n, t)
  return (AU[sgkai, vkai, vka]ni=1)

```

Function MKeyGen(M_m):

```

  (sgkbm, vkbm) ← DS.KGen(pp)
  return (M[sgkbm, vkbm])

```

Function MRegister(\mathcal{AU} [sgk_{ai}]^t_{i=1}, \mathcal{M} [vk_{bi}]^ℓ_{i=1}):

```

  for j ∈ range(ℓ) do
    (pkbj, pkb) ←
      RRTÉ.KGen(mpk, ℓ, t, 2)
  return (M[pkbi]ℓi=1, pkb)

```

Function CuKeyGen(C_n):

```

  skcn := PID(Cn) ∈ Z*p
  sk'cn ← CO.Com(pp, skcn)
  pkcn := (sk'cn, M1, M2) ←
    iDHH(sk'cn, skcn)
  x1 = (pkcn)
  ŵ1 = (skcn)
  π1 ← ZK.P( $\mathbf{R}_{L_1}$ , cṛs, x1, ŵ1)
  return (C[pkcn, sk̂cn], π1)

```

Function CuRegister(\mathcal{AU} [sgk_{ai}]^t_{i=1}, C[pk_{cn}], π₁):

```

  if ZK.Vf( $\mathbf{R}_{L_1}$ , cṛs, x1, π1) = 1 then
    (cêrtcn) ←
      TSPS.Sign(AU[sgkai]ti=1, pkcn)
  return (C[cêrtcn])

```

Function CuCreate(\mathcal{C} [sk̂_{cn}, cêrt_{cn}):

```

  k̂j ← PRF.KGen(pp)
  k'j ← CO.Com(pp, k̂j)
  Mj := (k'j, M1, M2) ← iDHH(k'j, kj)
  x2 = (k'j)
  ŵ2 = (k̂j, sk̂cn, cêrtcn)
  π2 ← ZK.P( $\mathbf{R}_{L_2}$ , cṛs, x2, ŵ2)
  return (C[k̂j, Mj], C[cert'cn], π2)

```

Function AuCreate(\mathcal{AU} [sgk_{ai}, vk_a]^t_{i=1}, C[cert'_{cn}], \mathcal{CL} [k̂_j, k'_j], π₂):

```

  if ZK.Vf( $\mathbf{R}_{L_2}$ , cṛs, x2, π2) = 1 then
    (cêrtj) ←
      TSPS.ParSign(AU[sgkai]ti=1, Mj)
  return (C[cêrtj])

```

Function Spend(\mathcal{C} [cêrt_{cn}, sk̂_{cn}], \mathcal{CL} [k̂_j, cêrt_j], \mathcal{M} [pk_{bm}], t):

```

  (rt) ← PRFk̂j(t)
  Rt := e(rt, h) ▷ h is the generator of G2.
  Ctm ← RRTÉ.Enc(pkbm, skcn; rt)
  ŵ3 = (cêrtj, k̂j, rt, sk̂cn)
  x3 = (Rt, Ctm, t)
  π3 ← ZK.P( $\mathbf{R}_{L_3}$ , cṛs, x3, ŵ3)
  return (T[π3, x3, TxID])

```

Function Vf(\mathcal{M} [pk_{bm}], T[π₃, x₃, Tx_{ID}], t):

```

  if ZK.Vf( $\mathbf{R}_{L_3}$ , cṛs, x3, π3) = 1 then
    for i ∈ StK do
      if Rt ∉ Li then
        (σRt, i) ← DS.Sign(sgkbi, Rt)
      if DS.Vf(vkbi, σRt, i) = 1 ∧ |σRt| ≥
        (|StK|/2) + 1 then
        return 1

```

Function RevealID(Ct_m, Ct_{m'}, v):

```

  (skcn) ← RRTÉ.Dec(Ctm, Ctm', v)
  return (skcn)

```

- AuCreate(\mathcal{AU} [sgk_{ai}]^t_{i=1}, C[cert'_{cn}], \mathcal{CL} [k̂_j, k'_j], π₂) is depicted in step 4–5 of Fig. 2. It is executed by a group of authorities \mathcal{AU} of size at least t . It takes the authorities' secret signing keys (sgk_{ai}), an indexed DH message space of PRF key and a NIZK proof π₂ as inputs. To create a certified collateral, it checks the validity of the proof π₂ and whether this collateral exists in the smart contract, and returns certificate cêrt_j as output. The list of parameters for each collateral is kept by \mathcal{CL} [k̂_j, k'_j, cêrt_j].
- Spend(\mathcal{C} [sk̂_{cn}, cêrt_{cn}], \mathcal{CL} [k̂_j, cêrt_j], \mathcal{M} [pk_{bm}], t) is depicted in step 6a and 6b of Fig. 2. It is executed by a customer $C_n \in \mathcal{C}$ who performs a payment to

- the merchant $M_m \in \mathcal{M}$ using the underlying cryptocurrency of their choice at time t . The registered customer uses a certified collateral $\mathcal{CL}[\hat{k}_j, \hat{\text{cert}}_j]$ 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 $\mathcal{T}[\chi_3, \pi_3, Tx_{ID}]$, which contains a pair of instance and proof (χ_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.¹
- $\text{Vf}(\mathcal{M}[\text{pk}_{bm}], \mathcal{T}[\pi_m, \chi_m, Tx_{ID}], t)$ is depicted in step 7–9 of Fig. 2. The merchant $M_m \in \mathcal{M}$ 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 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 Fig. 3:

- $\text{RevealID}(\text{Ct}_m, \text{Ct}_{m'}, v)$ is a deterministic algorithm that takes two ciphertexts Ct_m and $\text{Ct}_{m'}$ generated under the public key of two distinct merchants M_m and $M_{m'}$ 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 $\hat{\text{sk}}_{cn}$. We depict this language formally below, which is used during $\text{CuKeyGen}(\mathcal{C}_n)$.

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

¹ For a public blockchain such as Bitcoin or Ethereum, Tx_{ID} could be the hash of a transaction ($\text{H}[Tx]$); 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 [19].

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

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

- 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 $\text{Spend}(\mathcal{C}[\hat{\text{sk}}_{cn}, \hat{\text{c}}\text{ert}_{cn}], \mathcal{CL}[\hat{k}_j, \hat{\text{c}}\text{ert}_j], \mathcal{M}[\text{pk}_{bm}], \mathbf{t})$.

$$\begin{aligned} \mathbf{L}_3 = \text{NIZK}\{(\hat{\text{c}}\text{ert}_j, \hat{k}_j, r_{\mathbf{t}}, \hat{\text{sk}}_{cn}) \mid r_{\mathbf{t}} \leftarrow \text{PRF}_{\hat{k}_j}(\mathbf{t}), R_{\mathbf{t}} := e(r_{\mathbf{t}}, h), \\ \text{Ct}_{\mathbf{t}_m} := \mathcal{RRTE}.\text{Enc}(\text{pk}_{bm}, \hat{\text{sk}}_{cn}; r_{\mathbf{t}}), \mathcal{TSPS}.\text{Vf}(M_j, \hat{\text{c}}\text{ert}_j) = 1\} \end{aligned}$$

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 $\text{AllGen}(\cdot)$ algorithm (see Fig. 4) 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** $\mathcal{O}_{\text{AuCorrupt}}(Au_i)$: 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** $\mathcal{O}_{\text{CuCorrupt}}(\cdot)$: Adversary \mathcal{A} can corrupt any customer $C_n \in \mathcal{C}$ by querying this oracle, and receive its uncertified secret key $\hat{\text{sk}}_{cn}$.
- **Oracle** $\mathcal{O}_{\text{ColCorrupt}}(\cdot)$: \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** $\mathcal{O}_{\text{MCorrupt}}(\cdot)$: Adversary \mathcal{A} can corrupt a minority set of merchants (statekeepers) like $M_m \in \mathcal{M}$ and receives its pair of public key pk_{bm} and secret signing key $\hat{\text{sg}}k_{bm}$. The list of corrupted merchants is denoted by \mathcal{M}' s.t. we have, $|\mathcal{M}'| < |\text{stk}|/2$.
- **Oracle** $\mathcal{O}_{\text{Revoke}}(\cdot)$: Adversary \mathcal{A} can revoke at most q_R certified collaterals $CL_j \in \mathcal{CL}$ and redeem the deposited money.
- **Oracle** $\mathcal{O}_{\text{Spend}}(\cdot)$: \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 $\mathcal{O}_{\text{AuCorrupt}}, \mathcal{O}_{\text{CuCorrupt}}, \mathcal{O}_{\text{MCorrupt}}, \mathcal{O}_{\text{Spend}}$ oracles, $\mathcal{O}_{\text{ANON}}$ in short, and with advantage of $\text{Adv}_{\mathcal{A}}^{\text{ANON}}(\lambda, \beta) = 2((\text{Exp}_{\mathcal{A}}^{\text{ANON}}(1^\lambda, \beta) = 1) - 1/2)$, has a non-negligible chance of winning the experiment described in Fig. 4, i.e. we have, $|\text{Adv}_{\mathcal{A}}^{\text{ANON}}(\lambda, \beta = 0) - \text{Adv}_{\mathcal{A}}^{\text{ANON}}(\lambda, \beta = 1)| \leq \text{negl}(\lambda)$.*

$\text{EXP}_{\mathcal{A}}^{\text{ANON}}(1^\lambda, \beta)$ <hr/> 1 : $(\text{mpk}, H, \mathcal{AU}[\hat{\text{sgk}}_{ai}, \text{vk}_{ai}, \text{vk}_a]_{i=1}^n, \mathcal{M}[\hat{\text{sgk}}_{bi}, \text{vk}_{bi}, \text{pk}_{bi}]_{i=1}^\ell) \leftarrow \text{AllGen}(1^\lambda, \mathbf{R}_L)$ 2 : $(M_m, \hat{\text{sk}}_0^*, \hat{\text{sk}}_1^*, \hat{k}_0^*, \hat{k}_1^*) \leftarrow \mathcal{A}^{\text{ANON}}(\text{mpk})$ 3 : $\beta \leftarrow \$_\{0, 1\}$ 4 : $(\pi_\beta, x_\beta, R_{t,\beta}) \leftarrow \text{Spend}(\mathcal{C}[\hat{\text{cert}}_{c\beta}^*, \hat{\text{sk}}_\beta^*], \mathcal{CL}[\hat{k}_\beta^*, \hat{\text{cert}}_\beta^*], \mathcal{M}[\text{pk}_{bm}])$ 5 : $\beta' \leftarrow \$_{\mathcal{A}^{\text{ANON}}}(\mathcal{T}[\pi_\beta, x_\beta, R_{t,\beta}])$ 6 : return $\beta' == \beta$
$\text{EXP}_{\mathcal{A}}^{\text{PC}}(1^\lambda)$ <hr/> 1 : $(\text{mpk}, H, \mathcal{AU}[\hat{\text{sgk}}_{ai}, \text{vk}_{ai}, \text{vk}_a]_{i=1}^n, \mathcal{M}[\hat{\text{sgk}}_{bi}, \text{vk}_{bi}, \text{pk}_{bi}]_{i=1}^\ell) \leftarrow \text{AllGen}(1^\lambda, \mathbf{R}_L)$ 2 : $(\pi^*, x^*, R_t^*) \leftarrow \mathcal{A}^{\text{PC}}(\text{mpk})$ 3 : if $\mathcal{ZK}.\text{Vf}(\mathbf{R}_L, \text{c}\hat{\text{r}}\hat{\text{s}}, \pi^*, x^*) == 1 \wedge q_D < q_S + q_R$: return 1 4 : else return 0

Fig. 4. 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 $\mathcal{O}_{\text{AuCorrupt}}$, $\mathcal{O}_{\text{CuCorrupt}}$, $\mathcal{O}_{\text{ColCorrupt}}$, $\mathcal{O}_{\text{Revoke}}$, $\mathcal{O}_{\text{Spend}}$ oracles, \mathcal{O}_{PC} in short, can win the experiment described in Fig. 4 with a non-negligible advantage in λ and we can write, $\text{Adv}_{\mathcal{A}}^{\text{PC}}(\lambda) := \Pr[\text{Exp}_{\mathcal{A}}^{\text{PC}}(1^\lambda) = 1] \leq \text{negl}(\lambda)$.*

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 $\mathcal{T}[\pi, x, R_t, H(Tx)]$ 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 $(\hat{sk}_0^*, \hat{k}_0^*)$ and $(\hat{sk}_1^*, \hat{k}_1^*)$, can distinguish between $(\pi_0, x_0, R_{t,0})$ and $(\pi_1, x_1, R_{t,1})$ 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, $\pi'_\beta \leftarrow \mathcal{ZK}.\text{Sim}(\text{c}\hat{r}s, \hat{t}\hat{s}, x_\beta)$.

The Zero-Knowledge property of NIZK arguments 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_2^β** : In this game, we modify H_1^β s.t. for generating the index *id* the challenger commits $\hat{sk}_{1-\beta}^*$ instead of \hat{sk}_β^* .

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

- **Hybrid H** : In this game, we modify H_2^β 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_2^β . To be more concrete, \mathcal{A} cannot distinguish between Ct_β and $\text{Ct}_{1-\beta}$ 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 $R_{t,0}$ and $R_{t,1}$. \square

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. $|\sigma_{R_t}| \geq (|\text{StK}|/2) + 1$.

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 probability of adversary in “**Case 1**” is negligible. Thus having played a sequence of indistinguishable games between $\mathcal{B}_{\text{WR}}^{\text{PRF}}(1^\lambda)$, $\mathcal{B}_{\text{EUF-CIMA}}^{\text{TSPS}}(1^\lambda)$, $\mathcal{B}_{\text{KS}}^{\text{NIZK}}(1^\lambda)$ 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 $\sum \mathcal{L}_c + \tau^* > \text{col}_{\mathcal{A}}$ 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, $\text{Adv}_{\mathcal{A}}^{\text{PC}}(\lambda) = \Pr[\mathcal{A} \text{ Wins } G_0] \geq \epsilon$.
- **Game G_1 :** In this game, we modify G_0 such that we assume the existence of an efficient extractor $\text{Ext}(\cdot)$. In this case, there exists an extractor that takes the extraction trapdoor $\hat{\text{te}}$ and the received challenge tuple (π^*, π_j^*, x^*) as inputs, and returns the corresponding witness $(w^*) \leftarrow \text{Ext}(\hat{\text{te}}, x^*, \pi^*)$ s.t. $w^* = (\text{cert}^*, \mu^*, \text{sk}_c^*, r_t^*, (M_j^*, k^*))$. To be more precise, the extractor first extracts the indexed DH message $M_j^* := (id_j^*, M_{j1}^*, M_{j2}^*)$, and then can extract 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. This property guarantees the existence of the defined extractor under non-falsifiable assumptions and we can write, $\text{Adv}_{\mathcal{A}}^{\text{PC}}(\lambda) = \Pr[\mathcal{A} \text{ Wins } G_0] \approx \Pr[\mathcal{A} \text{ Wins } G_1]$ and this advantage consequently depends on two possible cases,

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

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

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

Under the assumption that the given NIZK is KS, the adversary \mathcal{A} can win the game when the event of $(w^*, x^*) \in \mathbf{R}_L$ 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. $k^* \notin \mathcal{C}\mathcal{L}'$ or it is generated under a random key $k^* \notin \mathcal{K}$. The latter case will be bounded by the advantage of $\mathcal{B}_{\text{WR}}^{\text{PRF}}(1^\lambda)$ attacker, then we can write,

$$\Pr[\mathcal{A} \text{ Wins } G_2] = \Pr[\mathcal{A} \text{ Wins } G_2 : k^* \in \mathcal{K} \wedge k^* \notin \mathcal{C}\mathcal{L}'] +$$

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

- **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 $\mathcal{S}_{(\star, M_{j_2}^*)}$. If $|\mathcal{S}_{(\star, M_{j_2}^*)} \cup \mathcal{AU}'| < t$, $\mathcal{B}_{\text{EUF-CiMA}}^{\text{TSPS}}(1^\lambda)$ returns the pair (M_j^*, cert^*) as a valid forgery for the defined threshold EUF-CiMA security game in Def. [16, 4.3]. Thus, we can write,

$$\begin{aligned} Adv_{\mathcal{A}}^{\text{PC}}(\lambda) &\leq Adv_{\mathcal{B}_{\text{KS}}}^{\text{NIZK}}(\lambda) + Adv_{\mathcal{B}_{\text{WR}}}^{\text{PRF}}(\lambda) + \Pr[\mathcal{A} \text{ Wins } G_3 : |\mathcal{S}_{(\star, M_{j_2}^*)} \cup \mathcal{AU}'| < t] \\ &+ \Pr[\mathcal{A} \text{ Wins } G_3 : |\mathcal{S}_{(\star, M_{j_2}^*)} \cup \mathcal{AU}'| \geq t] \leq Adv_{\mathcal{B}_{\text{KS}}}^{\text{NIZK}}(\lambda) \\ &+ Adv_{\mathcal{B}_{\text{WR}}}^{\text{PRF}}(\lambda) + Adv_{\mathcal{B}_{\text{EUF-CiMA}}}^{\text{TSPS}}(\lambda) + \Pr[\mathcal{A} \text{ Wins } G_3 : |\mathcal{S}_{(\star, M_{j_2}^*)} \cup \mathcal{AU}'| \geq t]. \end{aligned}$$

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 $G_3 \wedge |\mathcal{S}_{(\star, M_{j_2}^*)} \cup \mathcal{AU}'| \geq t$ ”, is equal to zero. Then we can write,

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

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 $|\mathcal{M}'| < |\text{StK}|/2$, then we can form an efficient algorithm $\mathcal{B}_{\text{EUF-CMA}}^{\text{DS}}(1^\lambda)$ to break the EUF-CMA property of the underlying DS scheme. Then we can write:

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

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

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 the full version [28]. 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.

4.1 Randomness-Reusable Threshold Encryption

(ℓ, t, k) -RRTE is a new observation on threshold encryption (TE) schemes (see the full version [28] for the definition) 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:*

- $(\vec{\text{pk}}, \text{pk}) \leftarrow \text{RRTE.KGen}(\text{pp}, \ell, t, k)$: *Key generation is a probabilistic and distributed algorithm that takes pp along with three integers $\ell, t, k \in \text{poly}(\lambda)$ as inputs. It then returns a vector of public key $\vec{\text{pk}}$ of size ℓ and a general public key pk as outputs.*
- $(\text{Ct}_j, v) \leftarrow \text{RRTE.Enc}(\text{pp}, \text{pk}, m, \text{pk}_j)$: *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 $\text{Ct}_j \in \mathcal{C}$ associated with the recipient $j \in \mathcal{R}$ and an auxiliary value v as outputs.*
- $(\perp, m) \leftarrow \text{RRTE.Dec}(\text{pp}, \{\text{Ct}_j\}_{j \in \mathcal{K}}, v)$: *The decryption algorithm takes pp , a set of ciphertexts $\{\text{Ct}_j\}_{j \in \mathcal{K}}$ 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 the full version [28] we elaborate more on the security requirements and then we propose an efficient construction based on threshold ElGamal encryptions.

4.2 Pseudo-Random Function

We utilise a weakly-robust PRF proposed by Dodis and Yampolskiy [17] in order to make customers’ collaterals reusable. This PRF enables us to define a time of payment, i.e. x in $\text{PRF}_k(x) = g^{1/(k+x)}$ 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 Sect. 1, we can block a customer from reusing the same collateral for a pre-defined time period.

4.3 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 [16]. 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 [8]. 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.

4.4 Commitment Scheme

In our construction, we use the Pedersen commitment scheme [32] due to the following reasons; firstly, the TSPS construction is defined over the indexed DH message spaces (see the full version for the exact definition) 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.

4.5 NIZK Proofs

To instantiate the described NP-relations in Sect. 3.1, we utilize three main proof systems: Sigma protocols [34], range-proofs [10] and GS proof systems [24] (see the full version). Sigma protocols are an efficient choice as the main proof system in our implementation; we use the Fiat-Shamir heuristic [22] to make them 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 [26].

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.

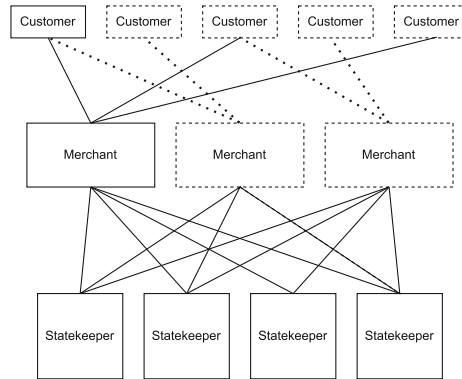


Fig. 5. Experimental setup. Dotted lines and shapes represent parties and workload that do not affect the leftmost customer or the leftmost merchant. Our experiment only considers the solid lines and shapes. From the perspective of the leftmost client and the leftmost merchant, this is equivalent to running the full system. While statekeepers are the same as merchants, we make these two sets distinct for clarity.

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 transactions². Specifically, our implementation uses the Charm-Crypto framework [15], 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 1000 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 Fig. 5.

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 [3]. In this pairing group, the base field order is 256 bits.

² The open-source implementation can be found in [this repository](#).

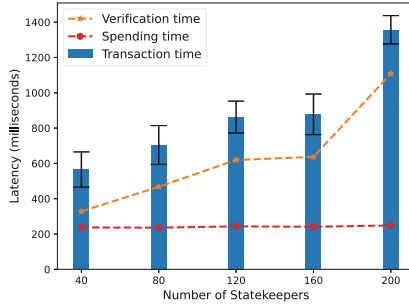


Fig. 6. Latency comparison. Total transaction time for 1000 tx per second vs the number of statekeepers.

Latency. As illustrated in Fig. 6, 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., ~240 ms. However, in the case of 40 statekeepers, our construction allows a customer and merchant to successfully transact within ~550 milliseconds (ms). In contrast, in the case of 200 statekeepers, a transaction takes ~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 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 s.

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.

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 Sect. 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 reg.	Pay	Merchant reg.	Reclaim	Withdraw Bal.
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

6.1 Collateral Reusability

This property can be understood by comparing to PCNs such as Lightning [33], Raiden [30]. Our work is similar to PCNs since we are both reliant on collaterals for guaranteeing security of transactions. PCNs enable quick and cheap transactions over established channels between parties (routes), but suffer from route availability issues in case any involved party is unresponsive. They are capital dependent, since each channel in a path must individually have sufficient capital to route a certain transaction [25].

The main point of difference between our work and PCNs is the suitability for supporting retail markets. The capital locked into a payment channel is only suitable for a specific kind of payments, while our collaterals allow customers to

pay any merchant in the system. This is because PCNs aren't designed to tackle the unilateral nature of payments in retail markets. The funds locked in a channel deplete quickly, and hence their capability to act as intermediaries decreases [29]. Although fund rebalancing techniques [4] exist to mitigate channel depletion, they require a user to have multiple channels and are ineffective for managing unilateral payments. Rebalancing is generally not possible for a user that only employs their channels for payments and not for getting paid. Our protocol is tailor made to this economic environment, and so we believe it supplements PCNs, rather than directly competing with them.

6.2 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.

The performance of our construction is based 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 [31] with the crucial distinction that 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" where balance is achieved through cryptographic innovations. We call for further cryptographic innovations and welcome research into even more trustless, robust and secure systems.

A similar trade-off has been observed in PCN, a promising scalability solution for cryptocurrencies, by Avarikioti et al. [2], who suggest that PCN are more stable and efficient when centralized structures are present. In an empirical survey, Zabka et al. [35] observe the rising centrality in the Lightning Network as the capacity and capabilities of Lightning grew over time.

6.3 Privacy

The main idea underlying our private double-spending protection, goes all the way back to the e-cash schemes introduced by Chaum [14]. The 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 [5, 9, 12, 13, 23]. 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 [6]), 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. A detailed explanation is given in Sect. 2 of the full version [28].

6.4 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 Sect. 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.

6.5 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 s.

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. Akash Madhusudan, Mahdi Sedaghat and Bart Preneel were supported in part by the Flemish Government through the FWO SBO project SNIPPET S007619, 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 and by CyberSecurity Research Flanders with reference number VR20192203. 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. Aumayr, L., Abbaszadeh, K., Maffei, M.: Thora: Atomic and privacy-preserving multi-channel updates. *IACR Cryptol. ePrint Arch.* 317 (2022). <https://eprint.iacr.org/2022/317>
2. Avarikioti, Z., Heimbach, L., Wang, Y., Wattenhofer, R.: ride the lightning: the game theory of payment channels. In: Bonneau, J., Heninger, N. (eds.) *FC 2020*. LNCS, vol. 12059, pp. 264–283. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_15
3. 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 (2006). https://doi.org/10.1007/11693383_22
4. 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>
5. 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 (2015). https://doi.org/10.1007/978-3-662-46447-2_5
6. Bauer, B., Fuchsbauer, G., Qian, C.: Transferable E-cash: a cleaner model and the first practical instantiation. In: Garay, J.A. (ed.) *PKC 2021*. LNCS, vol. 12711, pp. 559–590. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75248-4_20
7. Benmeleh, Y.: Blockchain firm starkware valued at \$2 billion in funding round (2021). <https://www.bloomberg.com>
8. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. *J. Cryptol.* **17**(4), 297–319 (2004). <https://doi.org/10.1007/s00145-004-0314-9>
9. Brands, S.: Untraceable off-line cash in wallet with observers. In: Stinson, D.R. (ed.) *CRYPTO 1993*. LNCS, vol. 773, pp. 302–318. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_26
10. 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 (2018). <https://doi.org/10.1109/SP.2018.00020>
11. Buterin, V.: An incomplete guide to rollups (2021). <https://vitalik.ca/general/2021/01/05/rollup.html>
12. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact E-cash. In: Cramer, R. (ed.) *EUROCRYPT 2005*. LNCS, vol. 3494, pp. 302–321. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_18
13. 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 (2010). https://doi.org/10.1007/978-3-642-14577-3_9

14. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) CRYPTO 1982, pp. 199–203. Plenum Press, New York (1982)
15. Joseph, A.A., et al.: Charm: a framework for rapidly prototyping cryptosystems. *J. Cryptograph. Eng.* **3**, 111–12 (2013). <https://doi.org/10.1007/s13389-013-0057-3>
16. 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>
17. 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 (2005). https://doi.org/10.1007/978-3-540-30580-4_28
18. 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 (2019). <https://doi.org/10.1109/SP.2019.00020>
19. Electric Coin Company: Explaining viewing keys. <https://electriccoin.co/blog/explaining-viewing-keys/>. Accessed 13 Feb 2023
20. Ethereum.org: Layer 2 rollups (2021). <https://ethereum.org/en/developers/docs/scaling/layer-2-rollups/>
21. ethos.dev: The beacon chain ethereum 2.0 explainer you need to read first. <https://ethos.dev/beacon-chain>. Accessed 13 Feb 2023
22. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
23. Frankel, Y., Tsiounis, Y., Yung, M.: “Indirect discourse proofs”: achieving efficient fair off-line e-cash. In: Kim, K., Matsumoto, T. (eds.) ASIACRYPT 1996. LNCS, vol. 1163, pp. 286–300. Springer, Heidelberg (1996). <https://doi.org/10.1007/BFb0034855>
24. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_24
25. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: SoK: layer-two blockchain protocols. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 201–226. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_12
26. Herold, G., Hoffmann, M., Kloof, 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 (2017). <https://doi.org/10.1145/3133956.3134068>
27. 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 30 Aug 2022
28. Madhusudan, A., Sedaghat, M., Tiwari, S., Cong, K., Preneel, B.: Reusable, instant and private payment guarantees for cryptocurrencies. *Cryptology ePrint Archive*, Paper 2023/583 (2023). <https://eprint.iacr.org/2023/583>
29. Mavroudis, V., Wüst, K., Dhar, A., Kostianen, K., Capkun, S.: Snappy: fast on-chain payments with practical collaterals. In: NDSS 2020. The Internet Society (2020)
30. Network, Raiden: What is the raiden network (2019). <https://raiden.network/101.html>

31. 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>
32. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_9
33. Poon, J., Dryja, T.: The Bitcoin lightning network: scalable off-chain instant payments (2016). <https://lightning.network/lightning-network-paper.pdf>
34. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_22
35. 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>