Secure and Privacy-preserving CBDC Offline Payments using a Secure Element

Elli Androulaki, Angelo De Caro, Kaoutar El Khiyaoui, Romain Gay, Rebekah Mercer, and Alessandro Sorniotti

IBM Research - Zurich

Abstract. Offline payments present an opportunity for central bank digital currency to address the lack of digital financial inclusion plaguing existing digital payment solutions. However, the design of secure offline payments is a complex undertaking; for example, the lack of connectivity during the payments renders double spending attacks trivial. While the identification of double spenders and penal sanctions may curb attacks by individuals, they may not be sufficient against concerted efforts by states or well-funded institutions. It is hence important to also rely on preventive measures that reduce the scale of such attacks. An example of such a measure is secure elements. These however are limited in compute and storage, making the design of solutions that offer comparable privacy guarantees to those of physical cash challenging.

We address this with a protocol that offloads most of the payment computation to the user's mobile device and restricts the computation on the secure element to deleting spent tokens, and generating a signature with a computation equivalent to that of ECDSA. We claim that the use of mobile devices or enhanced smart card-based devices are required for secure consumer-to-consumer payments. To further harden the protocol, we enable the efficient identification of double spenders on the off-chance an attacker successfully double spends. Finally, we prove its security in the ideal/real world paradigm, and evaluate its performance to demonstrate its practicality.

1 Introduction

Central-Bank Digital Currency (CBDC) is a form of central bank money, on par with cash and central-bank reserves. In CBDC, central bank money is digitally represented as *tokens* with each token encoding an *owner* and a *value*. CBDCs are either *wholesale* or *retail*: the former is a payment instrument accessible only to financial institutions to settle their transactions, whereas the latter is available to the general public to conduct their retail payments.

With over 100 central banks currently exploring CBDC [24], and a few launching CBDC pilot programs and developing regulatory frameworks, CBDC systems may become a reality in the medium-term future. This increasing interest in CBDC is due in major part to the pressing need to address the inefficiencies prevalent in today's financial markets and payment systems. Indeed, wholesale

CBDC holds the potential to mitigate settlement delays, reduce counter-party risks, and expedite cross-border transactions. Retail CBDC is anticipated to slash transaction fees, ignite innovation in digital payment, and foster financial inclusion.

Retail CBDC is the most challenging to realize given the stringent requirements around resilience, scalability, privacy, and support for *offline payments*. The latter are a pre-requisite to achieving digital financial inclusion, whose goal is to empower individuals to successfully conduct digital payments irrespective of their location or the quality of their network coverage. Offline payments are also necessary to achieving resilient and robust digital payments that can operate even in the events of blackouts or power outages. In fact, offline payments assume a setting where the payers and the payees are offline, except when they withdraw or deposit their CBDCs.

To facilitate their adoption (against cash or other forms of offline payments), CBDC offline payments must mimic some of the properties of physical cash. More specifically, offline payments must be **(1) fast:** an offline payment should take only a few seconds to complete, **(2) transferable:** CBDCs received offline can be used in subsequent offline payments without restriction, and **(3) privacy-preserving:** the central bank and the commercial banks (usually referred to as intermediaries) should not learn any information about the offline payments beyond what is leaked necessarily at the withdrawal and the deposit of CBDCs (e.g., the value of and the identities of the participants in such a payment should be protected).

The complexity of offline payments stems from the inability of the payee to verify in real time whether the received CBDC tokens have never been spent. Without this verification, double spending is trivial: a malicious payer can spend – in an offline fashion – the same token over and over again without any restriction. It is, thus, crucial to curtail the ability of the payers to double spend. This can be achieved through *preventive* or *deterrent* measures or a combination of both. An example of a preventive measure is mobile apps that forbid double spending by design; for example, every time a payer sends a CBDC token that token is deleted, preventing its future use. A caveat is that an attacker can "hack" the app and bypass the restrictions put in place. An alternative is to leverage a *secure element* to ensure that such restrictions cannot be easily bypassed. Regarding deterrent measures an example is identifying double spenders when double spending is detected at time of deposit. In the presence of the right punitive measures, this approach can discourage double spending by individuals. This, nevertheless, may not deter state actors.

In this paper, we therefore combine secure elements and the identification of double spenders to achieve the highest levels of security. The secure element guarantees that double spending attacks are expensive and cannot be executed at scale, whereas the identification of double spenders ensures that on the off-chance such an attack is successful responsible parties will be identified and held accountable. It should be noted, however, that a secure element has limited storage and compute, and while it can accommodate an offline solution that uses

ECDSA signatures to *authenticate CBDCs* and *authorize offline payments*, such a solution will fail to address the privacy requirements discussed above. In fact, ECDSA signatures will leak the identities of the payer and the payee unless the payer and the payee use one-time public keys. This, on the other hand, will make double spender identification challenging, if not impossible.

Our approach instead relies on **(1) anonymous credentials** to allow the payer and the payee to transact anonymously such that no one else can learn their identities, **(2) blind signatures** and **(3) zero-knowledge proofs of knowledge of signatures** to ensure that neither the central bank nor the intermediaries can link a withdrawal to a deposit. Finally, to *efficiently* identify double spenders, we leverage **(4) verifiable encryption** to encrypt the identities of the payers for authorized auditors, which are tasked with decrypting ciphertexts and de-anonymizing double spenders.

Unfortunately, these types of cryptographic primitives are too expensive for off-the-shelf secure elements. To mitigate this, we divide the computation to send and receive an offline payment into two categories: one that is *computationally heavy* but does not require any secret keys, and the other that is *lightweight* (comparable to an ECDSA signature) and which makes use of a signing key. The heavy computation will be outsourced to a mobile device, and the lightweight computation will be performed on the secure element, which – in addition to authorizing payments using its signing key – is also responsible for ensuring that a CBDC token is only used once. To demonstrate the viability of our approach, we conducted benchmarks using smart cards as a stand in for the secure element and evaluated the incurred computational cost. Our evaluation shows that it takes the smart card under $400ms$ to authorize an offline payment.

2 Retail CBDC & Offline Payment Challenges

Central-bank digital currency (CBDC) is central bank money *issued* as tokens, such that each token encodes the identity of the owner and the value of currency it holds. There exist wholesale and retail CBDC: The former can only be owned by financial institutions, whereas the ownership of the latter is open to the general public. Once a CBDC token is issued, it can be used by its owner (acting as a payer) in a payment. While the regulatory frameworks for CBDC systems vary across jurisdictions, they all must meet the following requirements. **(1) Unforgeability:** Only tokens issued by the central bank can be used in payments. **(2) Theft-prevention:** Only the owner of a CBDC token can use it in a payment. **(3) Double-spending resistance:** In a payment, a payer cannot spend more than what she owns. **(4) Privacy:** CBDC payments should not leak info about the payer and the payee to unauthorized parties, including the central bank. **(5) Availability:** CBDC payments should be available at all times, even in the presence of accidental failures or malicious attacks.

Moreover, retail CBDC must also guarantee high performance and universal access. **High performance** refers to the requirement that retail CBDC should be as performant as existing legacy systems; in particular, the experienced latency of a retail CBDC payment should not exceed a few seconds. **Universal access**, on the other hand, refers to the property that the completion of a retail CBDC payment should not be conditioned on the environment of the payer and the payee (e.g., their access to the internet).

A common approach to satisfy the universal access property is by supporting **offline payments**, which are payments that do not necessitate the payer or the payee be online (i.e., *the digital counterpart of cash payments*). This is in contrast with existing digital payment solutions that require that at least one of the participants connect to the internet to learn the status of the payment (whether it was successful or not).

This paper focuses on offline retail CBDC payments and on how to realize the aforementioned security and privacy properties without taxing the performance or the user experience. Hereafter, we conflate, in what follows, "CBDC" with "retail CBDC", and we use "token" to refer to "CBDC token".

2.1 System Entities

The **central bank** issues the tokens for retail users and manages the reserves of the commercial banks, which we call intermediaries. The issuance is usually triggered by a *withdrawal* request from an intermediary, and when successful, the issuance results in debiting the issued value from the reserves of the requesting intermediary. The central bank also redeems tokens as a result of a *deposit* request originating from an intermediary. A successful redemption credits the redeemed value to the reserves of the requesting intermediary.

The **users** hold bank deposits at the intermediaries, which they can exchange for tokens. A user withdraws a token by submitting a *withdrawal request* to her intermediary, which routes the request to the central bank. A successful withdrawal results in debiting the withdrawn value from the user's bank deposits and from the intermediary's reserves. Once the user receives a token she can use it in offline payments. The lifecycle of the token ends with a *deposit* that enables the owner of the token to claim bank deposits in exchange for the deposited token. A successful deposit credits the deposited value to the bank deposits of the token's owner and the intermediary's reserves. A deposit of a token is only successful if the token was issued by the central bank, the deposit was initiated by the token's owner, and the token had not been deposited before.

The **intermediaries** maintain the users' bank deposit accounts and intermediate the interactions of the users with the central bank. They also onboard the users into the system by opening accounts for them and helping them obtain *long-term credentials* from the registration authority.

The **registration authority** provides the users with *long-term credentials*, which will be subsequently used to authorize and authenticate payments.

The **auditor** is authorized to trace a fraudulent offline payment to its originator. An example of such a payment is a payment that double spends a token.

2.2 Challenges in Offline Payments

Offline payments take place in a setting where the payer and the payee are both disconnected from the internet. The main challenge that arises is the inability of the payee to verify that the token she is receiving is one that was not spent before. While the payee can easily verify that the token was actually created by the central bank and holds the advertised value using for example signature schemes, it is impossible for her to check in real time if the payer did not previously spend the token. As a result, a rational payee is incentivized not to accept offline payments as she may fail to redeem the received tokens at time of deposit, due to double spending.

The identification of double spenders when combined with the right punitive measure may help discourage double spending attacks, as rational users will not double spend to avoid legal persecution and/or paying fines. This however may not thwart double spending by adversaries whose goal is not monetary gain but rather damaging the reputation of a given central bank and corroding trust in the currency.

The use of secure elements could help alleviate these concerns by constraining the actions that malicious users can perform. For example, the attempts of a malicious payer to spend the same token twice can be prevented by the secure element that enforces a strict *spend-once* policy. The rationale is that if the cost of bypassing the double spending prevention mechanisms in the secure element is too high, then double spending attacks will not be mounted at scale. In other words, the use of the secure element is equivalent to the use of micro-printing or holograms in banknotes; it ensures that a successful double spending is a rare occurrence as opposed to a common one.

Furthermore, to drive adoption, CBDC-based offline payments should be transferable, i.e., a payee should be able to use the tokens she receives in subsequent payments without restrictions. To satisfy this property together with double spender identification, the tokens must carry the history of their past payments so that prospective payees can verify their provenance and be assured of their authenticity, whereas authorized auditors can successfully pinpoint the double spender if the tokens were double spent. They must also offer privacy guarantees on par with those provided by cash. That is, the central bank should not learn the identities of the users withdrawing or depositing tokens. Additionally, when a user deposits a token, neither the user's intermediary nor the central bank should infer the identities of the payers and the payees involved in the token's past payments. One-time public keys can help the users preserve their anonymity, but they would hinder the identification of double spenders. A more suitable alternative is anonymous credentials [14, 23] that will allow the users to transact anonymously without sacrificing audit capabilities and the identification of double spenders.

Although effective, these techniques are computationally expensive, especially if one accounts for the fact that a payee must verify a series of consecutive payments to authenticate the tokens she is receiving. In particular, if these methods are implemented on today's constrained-resource secure elements, they will incur a too high latency that it will negatively impact the user experience.

Another difficulty that offline payment solutions face is the choice of the appropriate secure element. Mobile (smart) phones may be deemed a first-choice candidate as they are widespread and already embed a secure element, however, the embedded secure element is not programmable by application developers, and hence mobile phones cannot be used as is.At the same time, traditional smart cards are not sufficient to conduct peer-to-peer offline payments; both the payer and the payee need a trusted device that displays the payment amount and where smart card PINs can be securely entered. This device cannot simply be the mobile phone of one of the transactors, as she can tamper with her phone to show incorrect amounts or capture the PIN of the counter-party.

2.3 Our Approach

We equip each user with (1) a *mobile device* such as a smart phone and (2) a *secure element*, which enable us to resolve the aforementioned challenges in the following way. (1) Most of the payment computation and verification is offloaded to the more powerful mobile device, whereas the secure element is only required to update the status of the tokens, and authorize the payments via a simple signature. (2) If the secure element is a traditional smart card, then the users in the system can leverage their devices to display the values of the payments and enter their PIN codes.

Trust Assumptions The interactions of the system entities are governed by the following trust assumptions:

The **registration authority** is honest but curious. While it may collude with other system entities to learn offline payment information, it is trusted to assign each user in the system a unique identifier and generate a correctly-formed credential that will be accepted by all other system participants. We further assume that the credential is issued according to well-established KYC processes.

The **central bank** is honest but curious: it executes withdrawal and deposit requests correctly, but it may collude with other system participants to learn information about the payments.

The **intermediaries** are honest but curious, since non-cryptographic incentives can be used to deter them from misbehaving. Intermediaries' goal is to infer information about the offline payments either from the transcripts of withdrawals and deposits or by colluding with other system entities.

The **users** can be malicious and are interested in increasing their holdings. They may collude to forge tokens, double spend them tokens, or steal the tokens of honest users. They may also collude with other system entities to undermine the privacy of honest users. We note that while the users have full control of their **mobile devices**, we assume that they cannot easily tamper with the **secure elements** they possess.

The **auditor** is fully trusted to only inspect fraudulent payments and only de-anonymize misbehaving users.

3 Security Formalization

We formalize the security of offline payments using the *ideal/real world* paradigm, in which the ideal world captures the desired behavior that a protocol running in the real world should mimic. In more detail, the ideal world makes use of an *ideal functionality* representing an incorruptible trusted party, which receives the inputs of each task of the protocol, and returns the prescribed outputs. An adversary in this ideal world interacts with the ideal functionality only through the parties they corrupt. This ensures that the adversary can only determine the inputs of the corrupt parties, and receive the corresponding outputs, and cannot make the ideal functionality deviate from its specifications. In the literature, this adversary is usually called a *simulator* and denoted by $\mathcal{S}$.

We say that a protocol prot securely realizes an ideal functionality $\mathcal{F}$, if for any adversary $\mathcal{A}$ that controls a subset of the parties and interacts with prot, there exists a simulator $\mathcal{S}$ that controls the same parties as $\mathcal{A}$ and interacts with $\mathcal{F}$ such that for any input of the corrupt parties the resulting output of $\mathcal{A}$ in the real world is indistinguishable from the output of $\mathcal{S}$ in the ideal world. More formally, we say that prot securely realizes $\mathcal{F}$ if and only if: $\textsc{real}_{\mathcal{A},\text{prot}} \approx \textsc{ideal}_{\mathcal{S},\mathcal{F}}$, where $\textsc{real}_{\mathcal{A},\text{prot}}$ is the view of $\mathcal{A}$ in the real world while interacting with prot, and $\textsc{ideal}_{\mathcal{S},\mathcal{F}}$ is the view of $\mathcal{S}$ in the ideal world when interacting with $\mathcal{F}$.

Before we describe the ideal functionality $\mathcal{F}$ for an offline payment system (cf. Section 3.2), we first recall the security goals that such a system should achieve.

3.1 Security Goals

Offline payments systems must protect against forgery, token theft, and double-spending, and preserve the privacy of the users against the central bank and the intermediaries. More specifically, an offline payment system must ensure:

Unforgeability: An honest payee should not be tricked – during a payment – into accepting a token that was not issued by the central bank. Similarly, the honest-but-curious central bank will not declare a deposit as successful unless it has previously issued the corresponding token.

Theft-prevention: A malicious payer should not be able convince an honest payee to accept a token that the payer does not own. In the same vein, a malicious user should not be able to successfully deposit a token that they do not own.

Double-spending detection and identification: An honest user will use a token only once, either in a payment or a deposit, after which the token is deleted. A malicious user may instead use the token in multiple payments or deposits. A secure offline payment system must ensure that such double spending is swiftly

```
Before any call to the offline payment interfaces, $\mathcal{S}$ issues corruption queries:

Corrupt Upon an active corruption request $(\textsc{Corrupt}, P)$ for $P \in \mathbf{MD} \cup \mathbf{SE}$ from $\mathcal{S}$ do: $\mathbf{C} \leftarrow \mathbf{C} \cup P$.

InitAcc Upon receiving for the first time $(\textsc{InitAcc}, \mathrm{U}, v)$ from $\mathrm{I} \in \mathbf{I}$ for $\mathrm{U} \in \mathbf{U}$, send $(\textsc{InitAcc}, \mathrm{U}, \mathrm{I}, v)$ to $\mathcal{S}$ and wait for its go-ahead. On receipt of the go-ahead do:
  1. If $\textsc{accounts}[\mathrm{U}] \neq \perp$ then stop
  2. Else
     (a) $\textsc{accounts}[\mathrm{U}] \leftarrow \langle \mathrm{I}, v \rangle$
     (b) $\textsc{reserves}[\mathrm{I}] \leftarrow \textsc{reserves}[\mathrm{I}] + v$
     (c) return $\textsc{fin}$ to I

Register Upon receiving for the first time $(\textsc{Register}, \mathrm{U})$ from RA for $\mathrm{U} \in \mathbf{U}$ that already has an account – send $(\textsc{Register}, \mathrm{U})$ to $\mathcal{S}$ and wait for its go-ahead. On receipt of the go-ahead do:
  1. $\mathbf{R} \leftarrow \mathbf{R} \cup \mathrm{U}$
  2. return $\textsc{fin}$ to RA

Withdraw On receiving a $(\textsc{Withdraw}, \mathrm{U} = (\mathrm{SE}, \mathrm{MD}), v)$ from I for registered user U that has an account with I, send message Withdraw to $\mathcal{S}$ and await its go-ahead. On receipt of the go-ahead, fetch $\textsc{accounts}[\mathrm{U}] \rightarrow \langle \mathrm{I}, v^* \rangle$, and do:
  1. If $v^* < v$ or $\textsc{reserves}[\mathrm{I}] < v$ then stop
  2. Else do:
     (a) $\textsc{accounts}[\mathrm{U}] \leftarrow \langle \mathrm{I}, v^* - v \rangle$
     (b) $\textsc{reserves}[\mathrm{I}] \leftarrow \textsc{reserves}[\mathrm{I}] - v$
     (c) If $\mathrm{SE} \notin \mathbf{C}$ then randomly select $\rho$
     (d) Else receive $\rho$ from $\mathcal{S}$
     (e) If $\textsc{tokens}[(\rho)] \neq \perp$ then stop
     (f) Else do:
         i. $\textsc{tokens}[(\rho)] \leftarrow \langle \mathrm{U}, v \rangle$
         ii. If $\mathrm{SE} \in \mathbf{C}$ or $\mathrm{MD} \in \mathbf{C}$ then return $(\textsc{Withdraw}, \mathrm{U}, \rho, v)$ to $\mathcal{S}$
         iii. Else return $(\textsc{Withdraw}, \mathrm{U}, v)$ to $\mathcal{S}$
         iv. return $(\textsc{Withdraw}, \rho, v, \textsc{fin})$ to U

Pay On receiving $(\textsc{Pay}, \boldsymbol{\rho}, v, \mathrm{U}' = (\mathrm{SE}', \mathrm{MD}'))$ from registered user $\mathrm{U} = (\mathrm{SE}, \mathrm{MD})$, send $(\textsc{Pay}, \boldsymbol{\rho}, v)$ to $\mathrm{U}'$. On receiving the response Accept from $\mathrm{U}'$, send $(\textsc{Pay}, \boldsymbol{\rho}, v)$ to $\mathcal{S}$ and await the go ahead. On receipt of the go-ahead do:
  1. If $\mathrm{SE}' \notin \mathbf{C}$ and $\mathrm{MD}' \notin \mathbf{C}$ then do:
     (a) If $\textsc{tokens}[\boldsymbol{\rho}] \neq \langle \mathrm{U}, v \rangle$ then stop
     (b) Else do:
         i. randomly select a unique random number $\rho'$
         ii. $\textsc{tokens}[\boldsymbol{\rho'}] \leftarrow \langle \mathrm{U}', v \rangle$ where $\boldsymbol{\rho'} = (\boldsymbol{\rho}, \rho')$
         iii. If $\mathrm{SE} \in \mathbf{C}$ or $\mathrm{MD} \in \mathbf{C}$ then return $(\textsc{Pay}, \boldsymbol{\rho'}, v, \mathrm{U}, \mathrm{U}')$ to $\mathcal{S}$
         iv. Else return $(\textsc{Pay}, \boldsymbol{\rho'}, v)$ to $\mathcal{S}$
  2. Else do:
     (a) If $\mathrm{SE}' \in \mathbf{C}$ then receive $\rho'$ from $\mathcal{S}$
     (b) Else randomly select $\rho'$
     (c) If $\textsc{tokens}[\boldsymbol{\rho}] = \langle \mathrm{U}, v \rangle$ and $\mathrm{U}' \in \mathbf{R}$ then $\textsc{tokens}[\boldsymbol{\rho'}] \leftarrow \langle \mathrm{U}', v \rangle$
     (d) send $(\textsc{Pay}, \boldsymbol{\rho'}, v, \mathrm{U}, \mathrm{U}')$ to $\mathcal{S}$
  3. send $(\textsc{Pay}, \boldsymbol{\rho'}, v, \textsc{fin})$ to $\mathrm{U}'$

Deposit On receiving $(\textsc{Deposit}, \boldsymbol{\rho}, v, \mathrm{U})$ from I where U is registered, send $(\textsc{Deposit}, \boldsymbol{\rho}, v, \mathrm{I}, \mathrm{U})$ to $\mathcal{S}$ and await the go ahead. On receipt of the go-ahead do:
  1. If $\textsc{tokens}[\boldsymbol{\rho}] \neq \langle \mathrm{U}, v \rangle$ then stop
  2. Else do:
     (a) $\textsc{deposits}[\boldsymbol{\rho}[0]] \leftarrow \textsc{deposits}[\boldsymbol{\rho}[0]] \cup \{\boldsymbol{\rho}\}$
     (b) If $|\textsc{deposits}[\boldsymbol{\rho}[0]]| > 1$ then stop
     (c) Else do:
         i. $\textsc{accounts}[\mathrm{U}] \rightarrow \langle \mathrm{I}, v^* \rangle$
         ii. $\textsc{accounts}[\mathrm{U}] \leftarrow \langle \mathrm{I}, v^* + v \rangle$
         iii. $\textsc{reserves}[\mathrm{I}] \leftarrow \textsc{reserves}[\mathrm{I}] + v$
         iv. send $(\textsc{Deposit}, \textsc{fin})$ to U

Audit On receiving $(\textsc{Audit}, \rho)$ from A, send $(\textsc{Audit}, \rho)$ to $\mathcal{S}$ and await the go ahead. On receipt of the go-ahead do:
  1. If $|\textsc{deposits}[\boldsymbol{\rho}[0]]| \leq 1$ then send $(\textsc{Audit}, \textsc{fin})$ to A
  2. Else do:
     (a) $\textsc{deposits}[\rho] \rightarrow \{\boldsymbol{\rho}_1, ..., \boldsymbol{\rho}_n\}$
     (b) $\mathbf{DS} \leftarrow \emptyset$
     (c) $\forall i \neq j$ do:
         i. let $\boldsymbol{\rho}_{ij}$ denote the common prefix, i.e., $\boldsymbol{\rho}_{ij} = (\rho_0, ..., \rho_{k-1})$ such that $\boldsymbol{\rho}_i[l] = \boldsymbol{\rho}_j[l]$ for all $l < k$ and $\boldsymbol{\rho_i}[k] \neq \boldsymbol{\rho_j}[k]$
         ii. $\textsc{tokens}[\boldsymbol{\rho}_{ij}] \rightarrow \langle v, \mathrm{U}_{ij} \rangle$
         iii. $\mathbf{DS} \leftarrow \mathbf{DS} \cup \mathrm{U}_{ij}$
     (d) send $(\textsc{Audit}, \mathbf{DS}, \textsc{fin})$ to A and CB
```

Fig. 1. Offline Payments' Ideal Functionality $\mathcal{F}$.

detected at time of deposit and that the malicious user is identified by the authorized auditor.

Non-frameability: An honest user should never be accused of double spending by the authorized auditor, even if all the other users, the intermediaries and the central bank collude.

User anonymity: Only the intermediary of the honest user making a withdrawal should be able to learn her identity. During a payment, the history of the token being transferred does not leak any information about the honest users involved in the previous payments, beyond what's revealed naturally (e.g., the payee in the last payment is the payer in the current payment). We assume that the payer and the payee know each other, and this knowledge is not derived from the transcript of the payment, instead it is learned through auxiliary channels. Similarly, during a deposit, the history of the deposited token does not leak any information about the honest users involved in the payments preceding the deposit, other than what's leaked to the users who took part in the payments, and the intermediary of the depositor that identifies the depositor for accounting purposes.

Token unlinkability: The intermediaries and the central bank should not be able to link a withdrawal of an honest user to a deposit, beyond what is revealed by the value of the withdrawal and the deposit, even if the intermediaries and the central bank collude with all the other users in the system.

3.2 Ideal Functionality for Offline Payments

Figure 1 depicts ideal functionality $\mathcal{F}$ that reflects the aforementioned security goals.

Participants. We identify the following offline payment participants: $\mathbf{U} = \{\mathrm{U}_1, ..., \mathrm{U}_n\}$ are the users (who can act as both payers or payees), $\mathbf{I} = \{\mathrm{I}_1, ..., \mathrm{I}_m\}$ are the intermediaries, A is the auditor, RA the registration authority, and CB the central bank. For the sake of simplicity, we assume that each user U_i has one bank account in the system, and is equipped with a mobile device MD_i and a secure element SE_i. Notably, U_i is defined as pair $(\mathrm{MD}_i, \mathrm{SE}_i)$. The case where an individual has multiple bank accounts is simply accommodated by mapping the individual to multiple users, each associated with one bank account. We denote by $\mathbf{SE} = \{\mathrm{SE}_1, ..., \mathrm{SE}_n\}$ and $\mathbf{MD} = \{\mathrm{MD}_1, ..., \mathrm{MD}_n\}$ the sets of the secure elements and the mobile devices respectively. For ease of exposition, we assume that CB, RA, and the intermediaries $\mathbf{I}$ are all honest-but-curious. That is, the simulator $\mathcal{S}$ accesses by default their inputs and outputs. Finally, we assume a *static corruption model*, in which the simulator $\mathcal{S}$ corrupts the parties of its choosing before they interact with ideal functionality $\mathcal{F}$.

Data Stores. $\mathcal{F}$ makes use of a number of data stores: RESERVES is a map that tracks the reserves that each intermediary holds at CB. Entries are of type RESERVES[I] $= v$, where v is the value of the reserves of intermediary I. $\mathbf{R}$ is the set of registered users. ACCOUNTS is a map that tracks the accounts of the users in the system with entries $\textsc{accounts}[\mathrm{U}] = \langle \mathrm{I}, v\rangle$. Each entry indicates that U has an account of value v with intermediary I. $\textsc{tokens}$ is a map that keeps track of the history of the token ownership, whose entries are of the form $\textsc{tokens}[\boldsymbol{\rho}] = \langle \mathrm{U}, v\rangle$. This translates to U is the owner of the token whose value is v and whose history is identified by $\boldsymbol{\rho}$. $\textsc{deposits}$ is a map that tracks the history of the deposited tokens by storing entries $\textsc{deposits}[\rho] = \{\boldsymbol{\rho}_1, ..., \boldsymbol{\rho}_n\}$ such that ρ identifies a withdrawn token and $\boldsymbol{\rho}_i$ identifies a series of payments involving the withdrawn token and ending with a deposit. Finally, $\mathbf{C}$ is the set of corrupted parties in either $\mathbf{MD}$ or $\mathbf{SE}$. We note that $\mathbf{R}$, $\textsc{accounts}$, $\textsc{tokens}$, $\textsc{deposits}$, and $\mathbf{C}$ are all initially empty.

Interfaces. $\mathcal{F}$ also provides a number of interfaces through which it interacts with $\mathcal{S}$ and the participants.

$\underline{\textsc{Corrupt}}$. When $\mathcal{S}$ would like to compromise a party P where P is either a mobile device MD $\in \mathbf{MD}$ or a secure element SE $\in \mathbf{SE}$, $\mathcal{S}$ issues a call $\textsc{Corrupt}$ to $\mathcal{F}$, which results in adding P to $\mathbf{C}$. We note that though the users are equivalent to a given $(\mathrm{SE}, \mathrm{MD})$, it is possible to corrupt a user's MD without corrupting the corresponding SE. Since corrupting a secure element is much harder than corrupting a device, we assume, for simplicity, that if a secure element SE is corrupt, so is the associated MD.

$\underline{\textsc{InitAcc}}$. An intermediary I calls $\textsc{InitAcc}$ to create an account for user U $\in \mathbf{U}$ with initial value v. This call results in adding entry $\textsc{accounts}[U] \leftarrow \langle \mathrm{I}, v\rangle$ and updating I's reserves. During the call, $\mathcal{F}$ enforces that U has one account in the system.

$\underline{\textsc{Register}}$. The registration authority RA calls this interface to enrol a user – for which an account was already created – in the offline payment system. Users can only be registered once, and their registration results in adding U to the set $\mathbf{R}$.

$\underline{\textsc{Withdraw}}$. An intermediary I indicates a value and identifies a registered user U whose account is hosted by I. If the call is successful, U obtains a token which is identified by unique random number ρ, and can be used subsequently in offline payments authorized by the U's secure element. The call is successful if U's account and the I's reserves have enough holdings, as reflected in Fig. 1 $\textsc{Withdraw}$ step (1).

To ensure that the withdrawal of a token is not linked to its future deposit, the *unique identifier* ρ must not be known to either I or CB. Instead it is selected by $\mathcal{F}$ if U's secure element is not corrupt (step (2.c)), or by $\mathcal{S}$ otherwise (step (2.d)). Once ρ is selected and its uniqueness verified (step (2.e)), $\mathcal{F}$ adds entry $\textsc{tokens}[(\rho)] \leftarrow \langle \mathrm{U}, v\rangle$ (step (2.f.i)). Moreover, since I and CB are honest-but-curious, $\mathcal{S}$ learns by default the identity of the user and the value of the withdrawal (step (2.f.iii)). Finally, if either the secure element or the mobile device of the user is corrupt, then $\mathcal{S}$ also learns the value ρ (step (2.f.ii)).

$\underline{\textsc{Pay}}$. A registered user U invokes this interface to transfer ownership of a token she owns to some user U$'$. A call to this interface identifies the payer U and the payee U$'$, and contains the value of the token to be transferred and a vector

$\boldsymbol{\rho}$. The latter helps $\mathcal{F}$ identify the token in the map TOKENS and to trace its history. $\mathcal{F}$ checks if U$'$ would like to accept the payment and if her mobile device and secure element are not corrupt. In this case, $\mathcal{F}$ checks if the token identified by $\boldsymbol{\rho}$ is actually owned by U and has value v (cf. PAY step (1.a)). If so, then $\mathcal{F}$ produces a random identifier ρ' that together with $\boldsymbol{\rho}$ uniquely identify this payment (step (1.b.i)), and adds entry TOKENS$[\boldsymbol{\rho}\|\rho'] \leftarrow \langle \mathrm{U}', v\rangle$ (step (1.b.ii)). This indicates a successful payment and enables a *registered* U$'$ to subsequently transfer the ownership of the token. If either the mobile device or the secure element of U$'$ is corrupt, then $\mathcal{F}$ does not condition the payment success on the validity of the token (step (2)). This reflects that a malicious payee may choose to accept a payment that uses an invalid token. If it is the secure element of U$'$ that is corrupt, then the identifier ρ' is selected by $\mathcal{S}$ (step (2.a)); otherwise, it is selected by $\mathcal{F}$ (step (2.b)). Finally, if the token being transferred is actually valid (i.e., it exists in TOKENS, is owned by U and its value is v), and U$'$ is registered, then $\mathcal{F}$ adds entry TOKENS$[(\boldsymbol{\rho}, \rho')] \leftarrow \langle \mathrm{U}', v\rangle$ (step (2.c)). However, if the token is invalid, then $\mathcal{F}$ does not update TOKENS. This enables $\mathcal{F}$ to assure that if a malicious payee accepts an invalid token, she will not be able to transfer it to honest users, as it will not show in TOKENS. Now if the mobile device or the secure element of either U or U$'$ are corrupt, then $\mathcal{S}$ learns the identities of U and U$'$ (steps (1.b.iii) and (2.d)). Otherwise, $\mathcal{S}$ only learns the value and $\boldsymbol{\rho}\|\rho'$ (step (1.b.iv)). Finally, we stress that while an unregistered payee can accept a payment and receive a token, she will never be able to transfer its ownership as it is not added to TOKENS (cf. (step (2.c))).

DEPOSIT. An intermediary I invokes this interface to allow one of its registered users to exchange a token for commercial bank money. The call identifies the user U making the deposit, includes a vector $\boldsymbol{\rho}$ that refers to the token and its history, and specifies the value of the deposit. $\mathcal{F}$, therefore, verifies whether the token being deposited is indeed owned by U and has the right value (DEPOSIT step (1)). It then updates DEPOSITS$[\boldsymbol{\rho}[0]] \leftarrow$ DEPOSITS$[\boldsymbol{\rho}[0]] \cup \{\boldsymbol{\rho}\}$ (step (2.a)), and checks if the size of DEPOSITS$[\boldsymbol{\rho}[0]]$ is larger than 1. If that's the case, then this signals a double spending and $\mathcal{F}$ stops (step (2.b)). Otherwise, $\mathcal{F}$ updates the holdings of U and I (steps (c.i) – (c.iv)). Since CB and I are honest-but-curious, $\mathcal{S}$ learns the deposit information, which is the identity of U, the value v, and the vector $\boldsymbol{\rho}$.

AUDIT. The auditor A calls this interface with a token identifier ρ (this corresponds to the identifier used during the withdrawal). Upon such a call, $\mathcal{F}$ retrieves entry DEPOSITS$[\rho]$. If DEPOSITS$[\rho]$ contains nothing or just one vector, then $\mathcal{F}$ returns nothing (cf. AUDIT step (1)). This signifies that there is no double spender to identify. Otherwise, $\mathcal{F}$ retrieves the set of vectors $\{\boldsymbol{\rho_1}, ..., \boldsymbol{\rho}_l\}$ stored in DEPOSITS$[\rho]$; for each two vectors ρ_i and ρ_j, $\mathcal{F}$ identifies the double spender as the user U_{ij} stored in entry TOKENS$[\boldsymbol{\rho}_{ij}]$, where $\boldsymbol{\rho}_{ij}$ is the common prefix of $\boldsymbol{\rho}_i$ and $\boldsymbol{\rho}_j$ (steps (2.c.i) – (2.c.iii)). $\mathcal{F}$ concludes by returning the set of such users to A and CB.

4 Overview

In this section, we provide a short overview of our solution and the underpinning design principles. We start with the description of a simple solution whose primary goals are the prevention of forgeries and the detection of double spending. A token in this solution is a triple $\tau = (\mathrm{pk}, v, s)$, whereby pk is the public key of its owner, v is its value, and s is a random number that will uniquely identify it.

Withdrawals. User U_0 sends to her intermediary I_0 a withdrawal request $\langle v, s_0 \rangle$, where v is the value of the token to be withdrawn and s_0 is a random number selected by U_0. After checking that U_0 has enough funds, I_0 sends triple $\langle \mathrm{pk}_0, v, s_0 \rangle$ to the central bank. The latter checks if I_0 has enough funds, and responds with signature ψ_0 on $\langle \mathrm{pk}_0, v, s_0 \rangle$, which is then handed over to U_0. With signature ψ_0, U_0 can perform payments with token $\tau_0 = (\mathrm{pk}_0, v, s_0)$. A payment, in this paper, is a transfer of ownership of a token from a payer to a payee. In other words, a payment does not split nor merge tokens.

Payments. To pay user U_1 with τ_0, U_0 computes a signature σ_0 on $(v, s_0, \mathrm{pk}_0, \mathrm{pk}_1)$, where pk_1 is the public key of U_1, and sends tuple $\langle v, s_0, \mathrm{pk}_0, \mathrm{pk}_1, \sigma_0, \psi_0 \rangle$. U_1 verifies if signatures σ_0 and ψ_0 are valid with respect to the information in the response, and if so, accepts the payment. Similarly, U_1 can transfer the ownership of the received token to another user U_2. After n consecutive payments, the n^{th} payee, referred to as U_n, receives $\langle v, s_0, \mathrm{pk}_0, ..., \mathrm{pk}_n, \sigma_0, ..., \sigma_{n-1}, \psi_0 \rangle$. U_n accepts the payment if $\forall i \in [n]$ signature σ_i is a valid signature on $(v, s_0, \mathrm{pk}_i, \mathrm{pk}_{i+1})$ under public key pk_i, and ψ_0 is a valid signature of the central bank on (pk_0, v, s_0).

Deposits. To deposit the token, U_n computes a signature σ_n on (v, s_0, pk_n) and sends $\langle v, s_0, \mathrm{pk}_0, ..., \mathrm{pk}_n, \sigma_0, ..., \sigma_n, \psi_0 \rangle$ to her intermediary I_n. The latter forwards the deposit request to the central bank, which in turn, verifies that σ_n is a valid signature on (v, s_0, pk_n), ψ_0 is a signature issued by the central bank on (pk_0, v, s_0), and $\forall i \in [n]$, σ_i is a valid signature on $(v, s_0, \mathrm{pk}_i, \mathrm{pk}_{i+1})$ relative to pk_i. If all checks pass, the central bank accepts the deposit. If there is another deposit request that starts with (v, s_0, pk_0), then the central bank rejects the deposit due to double spending.

Identifying the double spenders. Unfortunately, the above solution is prone to framing attacks. Let $\langle v, s_0, \mathrm{pk}'_0, ..., \mathrm{pk}'_n, \sigma'_0, ..., \sigma'_n, \psi'_0 \rangle$ be the second deposit request. One can argue that the double spender's public key corresponds to the public key at index j that verifies $\forall i \leq j : \mathrm{pk}_i = \mathrm{pk}'_i$, and $\mathrm{pk}_{j+1} \neq \mathrm{pk}'_{j+1}$. This indicates that the user with public key pk_j has sent the token to users with public keys pk_{j+1} and pk'_{j+1}. However, this inherently assumes that the double spender will not spend the same token to the same payee. If they do, following the argument above, the *innocent* payee will be framed for double spending. A way to counter this is to have stateful payees that remember all the payments they have been involved in and rejects duplicates. This is clearly unreasonable, though. An alternative is to have the payee selects a random number that helps uniquely identify a received payment, and which will be carried in subsequent payments (a similar approach is adopted in transferable e-cash [8, 20, 2]). This ensures that two payments initiated by the same payer to the same payee with the same token will still carry two different random numbers, and this will signal that the payee received the same token twice, and cannot be framed for double spending. More specifically, U_i wishing to pay U_{i+1} receives a fresh random number s_{i+1} from U_{i+1}, computes a signature σ_i on $(v, s_i, s_{i+1}, \mathrm{pk}_i, \mathrm{pk}_{i+1})$, and sends $\langle v, s_0, \mathrm{pk}_0, s_1, \mathrm{pk}_1, ..., s_{i+1}, \mathrm{pk}_{i+1}, \sigma_0, \sigma_1, ..., \sigma_i, \psi_0\rangle$ to U_{i+1}. At deposit, U_n computes a signature σ_n on (v, s_n, pk_n), and sends deposit request $\langle v, s_0, \mathrm{pk}_0, s_1, \mathrm{pk}_1, ..., s_n, \mathrm{pk}_n, \sigma_0, \sigma_1, ..., \sigma_n, \psi_0\rangle$.

Assume that there is another deposit request that spends (v, s_0, pk_0). Let $\langle v, s_0, \mathrm{pk}_0, s'_1, \mathrm{pk}'_1, ..., s'_n, \mathrm{pk}'_n, \sigma'_0, \sigma'_1, ..., \sigma'_n, \psi'_0\rangle$ be such a request. The double spender is the user with public key pk_j such that $\forall i \leq j : (\mathrm{pk}_i, s_i) = (\mathrm{pk}'_i, s'_i)$, and $(\mathrm{pk}_{j+1}, s_{j+1}) \neq (\mathrm{pk}'_{j+1}, s'_{j+1})$. One can easily see that this user sent two different payments; one destined for user with public key pk_{j+1} with random number s_{j+1} and the other for user with public key pk'_{j+1} with random number s'_{j+1}.

Adding anonymity. The solution described so far ensures the following properties: (1) token unforgeability thanks to the central bank signature, (2) double spending detection, and (3) double spender identification. However, it does not protect the anonymity of the users: the chain of payments reveals the public keys of the payers and the payees. To mitigate this, users can leverage **anonymous credentials** thanks to which they obtain *long-term* credentials from the registration authority that bind their unique identifiers with their secret key (using BBS+ signatures [7] for example). Instead of defining the token owner as a public key, it is defined as a pseudonym, which is a hiding commitment to the user's unique identifier (i.e., a token $\tau = (v, C)$, where $C = P_0^{\mathrm{id}} P_1^r$, id is the identifier of the owner and r is random). During payments and deposits, the user produces an *anonymous* signature by computing a zero-knowledge proof that proves that the committed identifier was signed by the registration authority, and generating a signature of knowledge σ that shows that the signer knows the corresponding secret key.

This will yield a chain of payments of the form: $\langle v, s_0, C_0, ..., s_n, C_n, \sigma_0, ..., \sigma_n, \psi_0\rangle$, where ψ_0 is a signature of $\tau_0 = (C_0, v, s_0)$ instead of (pk_0, v, s_0). The size of this chain of payments can be further optimized if s_i is not revealed and used instead to compute $C_i = P_0^{\mathrm{id}_i} P_1^{s_i}$.The chain of payments will correspond thus to $\langle v, C_0, ..., C_n, \sigma_0, ..., \sigma_n, \psi_0\rangle$ with ψ_0 being a signature of $\tau_0 = (v, C_0)$.

Adding Auditability. To identify double spenders at time of deposit, we enhance each payment with a *verifiable Elgamal encryption* of the payer's unique identifier, which is intended for an **authorized auditor**. Assume there are two deposit requests $\langle v, C_0, \mathcal{E}_0, ..., C_n, \mathcal{E}_n, \sigma_0, ..., \sigma_n, \psi_0\rangle$ and $\langle v, C_0, \mathcal{E}'_0, ..., C'_n, \mathcal{E}'_n, \sigma'_0, ..., \sigma'_n, \psi_0\rangle$ starting with (v, C_0). The double spender can be identified by decrypting the Elgamal ciphertext $\mathcal{E}_j$ such that $\forall i \leq j : C_i = C'_i$ and $C_{j+1} \neq C'_{j+1}$.

Given the ability of the auditor to de-anonymize payments, we recommend distributing it using **threshold decryption**.The latter will guarantee that decryption is only successful if more than a threshold t of auditors cooperate.

Hiding the link between withdrawals and deposits. The solution so far does not prevent linking a withdrawal to a deposit. This may not look as a violation of user privacy given that payments are anonymized, but in reality, it is. We recall that at each withdrawal and deposit, the users identify themselves to their intermediaries. If one considers the case where a token is only transferred once before being deposited, then if the intermediaries of the payer and the payee collude, then they will be able to de-anonymize the payment (i.e., identify the payer and the payee). To avoid this, we rely on Poincheval-Sanders blind signatures [28, 30] to help the central bank sign the token $\tau_0 = (v, C_0)$ without leaking any information about C_0 or the resulting signature ψ_0 to the central bank or the intermediaries. We emphasize that the withdrawal of a token and its deposit can be linked through its value. This can be averted by forcing a single denomination, or mitigated to some extent by using fixed denominations and/or aggregating deposits. Aggregating deposits helps the depositor not to disclose the value of each deposited token, but only the aggregate value. This is the approach we follow (cf. Appendix D).

Using secure elements to limit double spending attacks. As discussed previously, secure elements can be leveraged to render double spending attack too impractical to mount. The challenge is that secure elements today come with limited compute and storage, and cannot accommodate the computation we have detailed. To address this, we divide the computation into two parts: one, lightweight, executed by the secure element, and another, computationally heavier, offloaded to the user's mobile device. In more details, the secure element will be in charge of three tasks: (1) generating the randomness s_i and computing the commitment C_i, (2) producing the signature of knowledge of the user's secret key, which can only be accessed by the secure element, and (3) deleting the tokens after they are spent. The mobile device, on the other hand, is responsible for everything else; in particular, the verification of the received payments, the computation of the verifiable encryption, and the generation of zero-knowledge proofs that show that the user is enrolled into the system.

5 Solution Description

Notations. Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_t$ be three cyclic groups of large prime order p that admit a non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$. Let $H_p : \{0,1\}^* \to \mathbb{Z}_p$ and $H_{\mathbb{G}_1} : \{0,1\}^* \to \mathbb{G}_1$ be two cryptographic hash functions.

Let P and $\widetilde{P}$ denote two generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. We denote elements in $\mathbb{G}_1$ by upper-case letters, whereas elements in $\mathbb{G}_2$ are denoted by upper-case letters with tilde. Elements in $\mathbb{Z}_p$ are denoted by lower-case letters.

We use the notation $\Pi \leftarrow \text{POK}[\mathbb{x}, \mathbb{w} : \mathcal{R}(\mathbb{x}, \mathbb{w}) = 1]$ to indicate that Π is non-interactive zero-knowledge (ZK) proof of knowledge that shows that public input $\mathbb{x}$ and witness $\mathbb{w}$ satisfy the binary relation $\mathcal{R}$. Similarly, we use the notation $\sigma \leftarrow \text{SOK}[m, (\mathbb{x}, \mathbb{w} : \mathcal{R}(\mathbb{x}, \mathbb{w}) = 1)]$ to indicate that σ is a signature of knowledge on message m that shows that public input $\mathbb{x}$ and witness $\mathbb{w}$ satisfy the binary relation $\mathcal{R}$. Appendix B.1 briefly describes ZK proofs and signatures of knowledge.

5.1 Setup

The **central bank** CB runs the key generator for Pointcheval-Sanders signatures [27, 28] $\mathrm{PSKeyGen}(1^\kappa, 3) \rightarrow (\mathrm{SK_{CB}}, \mathrm{PK_{CB}})$ and obtains a pair of secret and public keys that sign vectors of 3 messages. The description of Pointcheval-Sanders signatures can be found in Appendix B.6. CB also selects 3 random generators $(P_0, P_1, P_2) \in \mathbb{G}_1^3$ that will be used to compute Pedersen commitments. The **auditor** A produces a pair of secret and public Elgamal keys $\mathrm{SK_A} = a \in \mathbb{Z}_p^*$ and $\mathrm{PK_A} = (P, A = P^a) \in \mathbb{G}_1^2$. The **registration authority** RA, on the other hand, runs the key generator for BBS+ signatures [7] $\mathrm{BBSKeyGen}(1^\kappa, 2) \rightarrow (\mathrm{SK_{RA}}, \mathrm{PK_{RA}})$ and gets a pair of secret and public keys for BBS+ signatures that will be used to issue user credentials. Details on BBS+ signatures are deferred to Appendix B.4. The generators (P_0, P_1, P_2), the public keys of CB, A and RA, plus a description of two cryptographic hashes $H_p : \{0,1\}^* \rightarrow \mathbb{Z}_p$ and $H_{\mathbb{G}_1} : \{0,1\}^* \rightarrow \mathbb{G}_1$ are advertised to system participants (via a distributed ledger).

When **user** U opens an account with an intermediary I, I checks if U already has an account in the system by interacting with other intermediaries. If not, then U's **secure element** SE is provisioned with a secret key sk that only SE knows. SE is also assigned a unique random identifier id, and registered into the system by calling RA with the pair $(\mathrm{id}, \mathrm{pk})$, where pk is the public key matching sk. RA, accordingly, checks if id has not been registered before. If so, it rejects the registration request. Else, it engages with SE in an interactive ZK proof of knowledge to verify that SE knows the secret key sk matching public key pk. Upon a successful interaction, RA registers id and provides SE with signature $\phi \leftarrow \mathrm{BBSSign}(\mathrm{SK_{RA}}, \mathrm{id}, \mathrm{sk})$[1]. SE stores credential $\mathrm{cred} = (\mathrm{id}, \mathrm{pk}, \phi)$ in its long-term storage. Next, U pairs her secure element SE with **mobile device** MD to ensure that only instructions originating from MD will be accepted by SE. More specifically, MD is equipped with a public key communicated to SE at time of pairing, and which will be used subsequently to authenticate MD. Upon a successful pairing, MD stores SE's public credential $\mathrm{cred} = (\mathrm{id}, \mathrm{pk}, \phi)$.

From this point onward, we assume that all communications between the participants are mutually authenticated.

5.2 Withdrawal

User U_0 withdraws a token of value v as follows.

Preparation U_0 instructs her device MD_0 to initiate a withdrawal with value v. The device communicates to the secure element SE_0 that a withdrawal of value v is to take place. SE_0, as a result, selects random value s_0 computes a Pedersen commitment $C_0 = P_0^{\mathrm{id}_0} P_1^{s_0}$ and transmits s_0 and C_0 to MD_0. MD_0 then computes another Pedersen commitment $\mathrm{com} = P_0^{\mathrm{id}_0} P_1^{v} P_2^{s_0}$. MD_0 afterwards

[1] Note that the registration authority can sign sk without accessing its value.

prepares a signature request for a Pointcheval-Sanders signature over committed inputs using com and (id_0, v, s_0), following the description in Appendix B.7, where id_0 and s_0 are hidden and v is disclosed. This request, therefore, consists of $(R, \mathrm{com}, \mathcal{E}_0, v, \mathcal{E}_2, P, B, \Pi)$, whereby $R = H_{\mathbb{G}_1}(\mathrm{com})$, (P, B) is a one-time Elgamal encryption key computed by U_0, Π is a ZK proof that shows that $\mathrm{com} = P_0^{\mathrm{id}_0} P_1^v P_2^s$, and that $\mathcal{E}_0$ and $\mathcal{E}_2$ are Elgamal encryption of R^{id_0} and R^{s_0} respectively under encryption key (P, B). Next MD_0 sends to U_0's intermediary I_0 the tuple $\mathrm{req} = (\mathrm{id}_0, R, \mathrm{com}, \mathcal{E}_0, v, \mathcal{E}_2, P, B, \Pi, \rho_0)$, where ρ_0 is the randomness used to compute $\mathcal{E}_0$.

Issuance Given req, I_0 computes $R = H_{\mathbb{G}_1}(\mathrm{com})$ and checks that $\mathcal{E}_0$ encrypts R^{id_0} using randomness ρ_0 and public key (P, B) (i.e., $\mathcal{E}_0 = (P^{\rho_0}, R^{\mathrm{id}_0} B^{\rho_0})$). If so, then I_0 locks v in U_0's account, and sends to the central bank CB the tuple $(R, \mathrm{com}, \mathcal{E}_0, v, \mathcal{E}_2, P, B, \Pi)$. CB checks whether this is the first time it receives a withdrawal request with commitment com, and whether Π is a valid ZK proof for the relation described above. If so, then CB stores com (which plays the role of a withdrawal identifier), debits I_0's reserves, and returns to I_0 a response resp containing the expected blind Pointcheval-Sanders signature (see Appendix B.7). I_0, in turn, forwards resp to MD_0 and debits U_0's account.

Receipt MD_0 un-blinds resp following the description in Appendix B.7 and obtains a Pointcheval-Sanders signature $\psi \leftarrow \mathrm{PSSign}(\mathrm{SK}_{\mathrm{CB}}, (\mathrm{id}_0, v, s_0))$. Next, MD_0 persists tuple (v, s_0, C_0, ψ) for subsequent use and notifies SE_0 that the withdrawal is complete. SE_0, in return, persists in its storage token $\tau_0 = (v, C_0)$, which we call hereafter the original token. In the remainder of this paper, we use the following convention for token $\tau = (v, C) = (v, P_0^{\mathrm{id}} P_1^s)$: **1)** the secure element with identifier id is the owner of τ; **1)** v is the value of τ; **1)** s is a random value that ensures that τ is unique and does not leak any information about id.

Token Creation MD_0 then prepares the token data structures. First, it produces a ZK proof of knowledge Σ that shows that there is a signature ψ on τ_0 (i.e., τ_0 was issued by CB). More formally, MD_0 computes $\Sigma \leftarrow \textsc{pok}[\mathbb{x}, \mathbb{w} : \mathcal{R}(\mathbb{x}, \mathbb{w}) = 1]$, where $\mathbb{x} = (P_0, P_1, \mathrm{PK}_{\mathrm{CB}}, v, C_0)$, $\mathbb{w} = (u, \mathrm{id}_0, s_0, \psi)$, and $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$ corresponds to satisfying Eq. 1, with PSVerify being the verifier algorithm of Pointcheval-Sanders signatures.

$$1 \leftarrow \mathrm{PSVerify}(\mathrm{PK}_{\mathrm{CB}}, (\mathrm{id}_0, v, s_0), \psi) \ \wedge \ C_0 = P_0^{\mathrm{id}_0} P_1^{s_0} \tag{1}$$

Then, MD_0 stores tuple $T = (v, s_0, C_0, \mathrm{hist}_0 = \Sigma)$. hist_0 is the history of token $\tau_0 = (v, C_0)$. In Appendix C, we describe how to generate and verify the proof of withdrawal Σ.

5.3 Payment

U_i pays U_{i+1} with token $(v, s_i, C_i, \mathrm{hist}_i)$ as follows.

Preparation U_{i+1} instructs her device MD_{i+1} to receive a payment of value v. MD_{i+1} communicates to SE_{i+1} that a payment of value v is to be received. SE_{i+1}, consequently, selects a random value s_{i+1}, computes $C_{i+1} = P_0^{\mathrm{id}_{i+1}} P_1^{s_{i+1}}$ and sends s_{i+1} and C_{i+1} to MD_{i+1}. MD_{i+1} then transmits C_{i+1} to MD_i.

Transfer MD_i informs SE_i that a payment of value v is ongoing. SE_i in response selects and locks $\tau_i = (v, C_i)$. MD_i then computes ciphertext $\mathcal{E}_i = (P^{\rho_i}, P^{\mathrm{id}_i} A^{\rho_i})$ (recall $\mathrm{PK}_\mathrm{A} = (P, A)$ is the auditor's public key) and a ZK proof of knowledge $\mathit{\Gamma}_i \leftarrow \textsc{pok}[\mathbb{x}, \mathbb{w} : \mathcal{R}(\mathbb{x}, \mathbb{w}) = 1]$ where $\mathbb{x} = (C_i, \mathcal{E}_i)$, $\mathbb{w} = (\mathrm{id}_i, s_i, \rho_i)$, and $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$ is equivalent to:

$$C_i = P_0^{\mathrm{id}_i} P_1^{s_i} \ \wedge\ \mathcal{E}_i = (P^{\rho_i}, P^{\mathrm{id}_i} A^{\rho_i})$$

Put differently, $\mathit{\Gamma}_i$ proves that $\mathcal{E}_i$ correctly encrypts P^{id_i}, under A's public key PK_A, where id_i is committed in C_i. Next, SE_i and MD_i compute (following the description in Appendix B.5) $\sigma_i \leftarrow \textsc{sok}[(v, C_i, C_{i+1}), \mathbb{x}, \mathbb{w} : \mathcal{R}(\mathbb{x}, \mathbb{w}) = 1]$ where $\mathbb{x} = (\mathrm{PK}_\mathrm{RA}, P_0, P_1, C_i)$, $\mathbb{w} = (\phi_i, \mathrm{sk}_i, \mathrm{id}_i, s_i)$, and $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$ corresponds to satisfying Eq. 2. We note that BBSVerify is the verification algorithm of BBS+ signatures and PK_RA is the public key of the registration authority.

$$1 \leftarrow \mathrm{BBSVerify}(\mathrm{PK}_\mathrm{RA}, (\mathrm{id}_i, \mathrm{sk}_i), \phi_i) \ \wedge\ C_i = P_0^{\mathrm{id}_i} P_1^{s_i} \tag{2}$$

Roughly speaking, σ_i is a signature of knowledge over triple (v, C_i, C_{i+1}) that shows that that transfer of ownership of token $\tau_i = (v, C_i)$ has been authorized by its legitimate owner and that that owner is enrolled in the system. Lastly, MD_i sends payment $\mathfrak{p}_i = (v, C_{i+1}, C_i, \mathcal{E}_i, \sigma_i, \mathit{\Gamma}_i, \mathrm{hist}_i)$ to MD_{i+1}.

Receipt MD_{i+1} first checks if hist_i is valid. We distinguish between two cases: $i = 0$ and $i > 0$. If $i = 0$, then $\mathrm{hist}_0 = \mathit{\Sigma}$ and its validation consists of checking that $\mathit{\Sigma}$ is a valid proof of the relation depicted in Equation 1. Otherwise, $\mathrm{hist}_i = ((C_k, \mathcal{E}_k, \sigma_k, \mathit{\Gamma}_k)_{k=0}^{i-1}, \mathit{\Sigma})$, and the validation corresponds to verifying that $\forall k \in [i]$ $\mathit{\Sigma}$, $\mathit{\Gamma}_k$ and σ_k are valid. Notice that by verifying the validity of hist_i, U_{i+1} establishes the chain of provenance of token $\tau_i = (v, C_i)$. If $\mathit{\Gamma}_i$ and σ_i are also valid, then MD_{i+1} accepts the payment. After accepting the payment, MD_{i+1} stores tuple $(v, s_{i+1}, C_{i+1}, \mathrm{hist}_{i+1})$, with $\mathrm{hist}_{i+1} = (C_i, \mathcal{E}_i, \sigma_i, \mathit{\Gamma}_i) \| \mathrm{hist}_i$ and informs SE_{i+1}. SE_{i+1}, consequently, persists in its storage token $\tau_{i+1} = (v, C_{i+1})$.

Finally, on receiving the acknowledgment from MD_i that the payment was successful, SE_i deletes C_i from its storage.

5.4 Deposit and Reconciliation

Assume that user U_n wishes to deposit token $\tau_n = (v, C_n) = (v, P_0^{\mathrm{id}_n} P_1^{s_n})$. U_n, correspondingly, instructs her device MD_n to initiate a deposit of τ_n. MD_n, in response, informs SE_n, which locks token τ_n. Next MD_n and SE_n produce a signature of knowledge σ_n over token τ_n that shows that SE_n is actually the owner of τ_n. As shown in the previous section, $\sigma_n \leftarrow \text{SOK}[\tau_n, (\mathbb{x}, \mathbb{w} : \mathcal{R}(\mathbb{x}, \mathbb{w}) = 1)]$, where $\mathbb{x} = (\text{PK}_{\text{RA}}, P_0, P_1, C_n)$, $\mathbb{w} = (\phi_n, \text{sk}_n, \text{id}_n, s_n)$, and $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$ is defined as:

$$1 \leftarrow \text{BBSVerify}(\text{PK}_{\text{RA}}, (\text{id}_n, \text{sk}_n), \phi_n) \ \wedge \ C_n = P_0^{\text{id}_n} P_1^{s_n}$$

MD_n then sends the deposit request $\mathfrak{d}_n = (\tau_n, \sigma_n, \text{hist}_n)$ to U_n's intermediary I_n. I_n forwards $\mathfrak{d}_n$ to central bank CB, which initiates a reconciliation process that consists of first checking if σ_n is a valid signature of knowledge with respect to the above relation on τ_n. If not, then the transaction is rejected. Otherwise, CB checks if hist_n is valid. If the check succeeds, then CB verifies whether there exists another deposit $\mathfrak{d}'_m = (\tau'_m, \text{hist}'_m, \sigma'_m)$ such that hist'_m also starts with a payment spending the original token $\tau_0 = (v, C_0)$. If so, then the transaction is recorded but deemed invalid due to double spending. Next, CB notifies I_n, which also notifies MD_n, and triggers auditor A to identify the double spender. If no such a deposit exists, then CB accepts the transaction and credits I_n's account, whereas I_n credits U_n's account and notifies MD_n. Upon notification from I_n, MD_n informs SE_n to delete τ_n.

In Appendix D, we extend the protocol to support the aggregation of deposits.

5.5 Identifying the Double Spenders

For any two deposit transactions[2] $\mathfrak{d}_n = (\tau_n, \sigma_n, \text{hist}_n)$ and $\mathfrak{d}'_m = (\tau'_m, \sigma'_m, \text{hist}'_m)$ involving the same original token $\tau_0 = (v, C_0)$, we denote $\text{hist}_n = ((C_k, \mathcal{E}_k, \sigma_k, \varGamma_k)_{k \in [n]}, \varSigma)$ and $\text{hist}'_m = ((C'_k, \mathcal{E}'_k, \sigma'_k, \varGamma'_k)_{k \in [m]}, \varSigma')$. Auditor A identifies the double spender in these two transactions as follows. Let j be the index such that $\forall 0 \leq k \leq j$, $C_k = C'_k$ and $C_{j+1} \neq C'_{j+1}$. The double spender in this case corresponds to the owner of token $\tau_j = (v, C_j)$, and the auditor retrieves her identity by decrypting either $\mathcal{E}_j$ or $\mathcal{E}'_j$.

While U_n may attempt to deposit token τ_n twice, we do not consider this as double spending since this can happen accidentally, and will never result in crediting U_n's account.

6 Evaluation

We establish the practicality of the scheme by integrating the protocol in a mobile wallet and by benchmarking its performance on both iOS and Android platforms. Figures 2 and 3 show the implementation of both withdrawal and payment schemes from Sections 5.2 and 5.3. Four entities are involved in the protocols: a mobile phone, a secure element (SE), an intermediary and a central bank. In this section we focus on mobile phone and secure element, to establish

[2] If there exist more than two deposit transactions spending the same original token, the auditor will run the protocol for identifying double spenders for each pair of such transactions.

whether they can implement the protocol efficiently given their limitations on hardware and storage. The intermediary and the central bank need no such verification given that they are expected to run on servers or the cloud. The mobile phone and the secure element belong to three different personas in the protocol: the holder, which receives funds during withdrawal; payer and payee, which exchange possession of a token during payment.

6.1 Implementation

Secure Element. A fundamental aspect of a realistic benchmark is the choice of the secure element. There are four different options: 1) software emulation of the secure element in the wallet app; 2) TEE-based secure element in the mobile platform; 3) HSM-based secure element in the mobile platform; 4) external secure element. We discard the first approach on the grounds that it would present unrealistically good performance since it would not account for the resource-constrained computational environment of SEs, nor would it include the communication overhead with SEs. We discard the second approach since we do not consider it sufficiently secure: TEE platforms offer a wider attack surface owing to their programmability and have been compromised numerous times [32]. The third option would deliver adequate security, but we must discard it because SEs integrated on Apple and Android platforms do not implement our scheme and are not extensible/programmable. We thus adopt the fourth approach and implement the SE component of the scheme on *smartcards*.

A smartcard is a physical card embedded with an integrated circuit (IC) that can process and securely store data. The IC can include a secure element, which is a dedicated security component providing robust protection for sensitive data and cryptographic operations. In particular we use Javacard [17], a Java-based framework for smartcards. The protocol running on the SE is implemented as an *applet*, a small Java-based application supported by a subset of the Java runtime. Javacard applets communicate through Application Protocol Data Units (APDUs), which include identifiers used to select the applet function which should process that request. The Javacard framework supports operations on elliptic curves.

In our implementation, the smartcard exposes two functions: *receive* and *sign*. *receive* implements the secure element part of the protocol in Section 5.2 and the protocol in Section 5.3, which are the same protocol, involving the sampling of randomness and the generation of a Pedersen commitment. *sign* implements the smartcard part of the protocol in Section 5.3 (described in more details in Appendix B.5).

Recall that with the aid of the smartcard the payer produces a signature of knowledge over two commitments: the payer's and the payee's (referred to C_i and C_{i+1}, respectively, in Section 5.3). To prevent double-spending, the smartcard is programmed to produce signatures on (v, C_i, C_{i+1}) pairs, provided that: 1) C_i is the commitment generated by the card at the time of the receipt of the token; and 2) only produce signatures against a single destination commitment C_{i+1} when said token is further spent The first constraint ensures that the smartcard

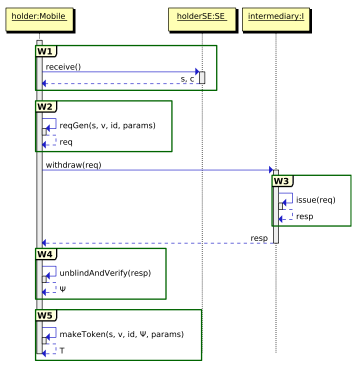

Fig. 2. Sequence diagram for token withdrawal.

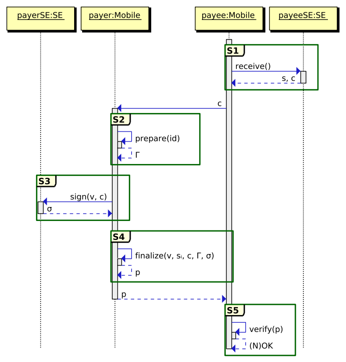

Fig. 3. Sequence diagram for token payment.

will only sign off payments for the token that was last received, thus avoiding re-spending past ones. The second constraint ensures that the current token can only be transferred to a single recipient. In order to cater for errors on the communication channel between the smartcard and the mobile phone (which are not uncommon in NFC transmissions), we permit multiple invocations of the smartcard's *sign* function, provided that the supplied destination commitment C_{i+1} never changes. All such invocations will generate valid signature of knowledge for the same payer/payee commitment pair. The smartcard can then be reset by invoking *receive*, which simply deletes the previous commitment and generates a new one. To handle multiple tokens, we introduce the concept of slot, each able to hold one pair of payer/payee commitments. The protocols in the figure just need to be augmented to receive the slot identifier.

Mobile Wallet. The mobile wallet is an app running on a mobile platform, implementing the necessary functionality for a user to withdraw tokens and transfer them. The app handles the communications between the user and the intermediary (which runs as a cloud service), and between two users upon payment. The app also handles the communications with the smartcard. The system parameters and the credentials for each transacting party are generated by a registration authority, also exposed as a cloud service.

To implement withdraw and payment protocols, the mobile wallet exposes the following functions: *reqGen* (the holder device part of the protocol in Section 5.2), *unblindAndVerify* (the protocol in Section 5.2) and *makeTokens* (the protocol in Section 5.2) to handle withdrawal; and *prepare* (the encryption and encryption correctness proof of Section 5.3), *finalize* (the payer device part of the protocol in Section 5.3) and *verify* (the protocol in Section 5.3) to handle payments.

6.2 Protocols

The message sequence flow for the withdrawal and payment protocols is shown in Figure 2 and Figure 3, respectively.

Withdraw. Withdrawing a token involves local invocations of the mobile function, NFC communications between the mobile and the smartcard and one remote call to the intermediary (which in turn contacts the central bank – we have omitted this step from the message sequence diagram for the sake of space). The protocol starts with the holder scanning her card to obtain the values s and c, and using the returned values to generate a withdrawal request (steps **W1** and **W2**), as described in Section 5.2. The holder then issues a request to the REST endpoint of her intermediary via HTTPs; at this point the intermediary interacts directly with the central bank who runs the protocol in Section 5.2 and generates a response which is sent back to the holder (step **W3**). The holder then runs the protocols in Sections 5.2 and 5.2 to create a token, which is stored for future use (steps **W4** and **W5**).

Payment. Before the offline payment protocol in Figure 3 can start, payer and payee carry out a negotiation to agree on the terms of the payment and to set up the BLE communication channel securely. The first message is a request, embedded in a base64 encoded JSON object shown as QRCode to the payer, containing the terms of the payment (amount, currency) and a BLE service identifier. The payee also starts a BLE GATT Server on the phone which advertises the BLE service identifier indicated within the QRCode. Under this service it provides a *Notify* BLE characteristic used to send data to the client (i.e. send data to the payer) and a *Write without response* BLE characteristic used to receive data from the client (i.e. send data to the payee). The QRCode also contains the cryptographic material to perform a secure BLE exchange based on Diffie-Hellman. The payer scans the QRCode to retrieve the BLE information as well as the payment details. If she agrees to pay, she looks for the BLE service identifier provided by the payee and she sends a payload to that BLE service completing the Diffie-Hellman exchange and confirming the terms of the payment. All subsequent messages (including this) are encrypted and encoded using CBOR.

Now the protocol in the figure begins with the payee scanning her card (step **S1**) to obtain the values s and C (Section 5.3). The payee then sends C through BLE to the payer. At this point the payer prepares the payment (step **S2**) by generating the audit information (the encryption of her identifier together with the proof of correctness, see Section 5.3). Then the payer's device and secure element engage in the joint proof of knowledge protocol from Appendix B.5 (steps **S3** and **S4**). At this point, the output of the payment $\mathfrak{p}$ is sent to the payee over BLE. Here the payee verifies the validity of the received token (step **S5**) by running the verification algorithm of Section 5.3 and stores the received token upon success.

6.3 Results

We implement the protocol on both Android (Samsung Galaxy A34 with Android 14) and Apple (iPhone 13 mini with iOS 17.6.1) platforms both running a crypto library using Go 1.21.8. We have conducted two different versions of the benchmarks, with and without auditing (no verifiable encryption of the payer's

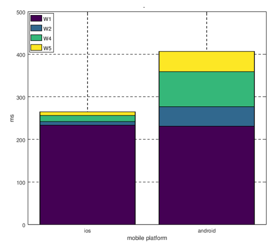

Fig. 4. Performance of withdrawal protocol, broken into component steps.

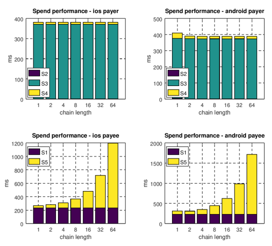

Fig. 5. Performance of payment protocol, broken into steps.

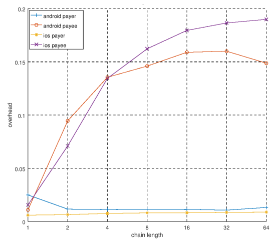

Fig. 6. Overhead of auditing of the payment protocol.

identifier for the auditor), in order to be able to clearly show the overhead imposed by auditing. The smartcard is an NXP 1ID white plastic cards with a Smart MX D600 chip (400KiB of available memory) running JCOP 4.5 OS with NXP's JCOPx extensions at version 1.1.4. Card and phone communicate using NFC.

Figures 4 and 5 show the performance of the no-audit version of the withdrawal and payment protocol, respectively. Withdrawal and the payer side of payments complete in less than 0.5s on both platforms; the running time of the payee on payment instead depends on the length of the payment chain, since the payee must verify its correctness. We can see that for a chain of up to 32 hops, the running time is below 1 second. For both protocols, the dominant operation are steps **W1** for withdraw, and **S1** and **S3** for payment: these steps are dominated by the point addition and scalar multiplication steps required to compute the Pedersen commitment in the card. We can also see how step **S5** is negligible for short chains.

Finally, Figure 6 shows the overhead of auditing on the payment protocol for both payer and payee. We can see how auditing does not impact the payer, and how it imposes an overhead between 10 and 20 percent on the running time of the payee. We speculate that the lower overhead for shorter chains is justified by the more moderate usage of memory when the payee verifies shorter chains, thus masking the effects of the go garbage collector kicking in for longer chains.

7 Related Work

Original e-cash. e-cash has been studied for several decades, beginning with Chaum's solutions [13, 15], which focus on the privacy of users and double spending detection. Privacy is achieved through the use of RSA blind signatures that breaks the link between withdrawals and deposits, whereas double spending detection is realized by using unique serial numbers. Although efficient, these solutions guarantee neither transferability nor theft prevention.

Divisible e-cash. Divisible e-cash [26, 25] allows a user to withdraw a token (generally with value 2^n), and split it among different recipients in multiple independent transactions with value $v = 2^l$ with $0 \leq l \leq n$. This however comes at the cost of a higher complexity of withdrawals, payments, and deposits, which scale with the token value. This is partially addressed in [11], where constant computation and communication complexity for the withdrawal and payment is achieved, while deposits have communication costs of $O(2^n)$ and computational costs of $O(2^{n+1})$. In [12], communication and computational costs of the deposits are improved to $O(2^l)$ and $O(2^{l+1})$ respectively. We stress that while divisibility is a useful feature, we leave it as a future work, and focus instead on transferability.

Transferable e-cash. In [26], they produce an e-cash scheme that is both transferable and divisible. However, the scheme high computational complexity, whereas we focus on efficiency and scalability. In [16], another divisible and transferable e-cash scheme is presented. Token size increases linearly with the number of transfers, in order to identify double spenders. Recipients are required to obtain tokens from the bank before being able to spend received ones, potentially limiting the transferability. Fuchsbauer et al. streamlined token size by having users store ownership receipts locally, maintaining constant token size regardless of transfers, but requiring user interaction for identifying double spenders [20]. However, when a double spending is detected, all previous owners of the token are identifiable. Baldimtsi et al. proposed a method where token size grows linearly with number of transfers, and reveals only the double spender's identity [2]. Due to the costs of this approach, we instead rely on a trusted auditor, that can be distributed, to de-anonymize payments as needed.

In [10, 3], the authors introduced the property of coin transparency, which ensures that a payee cannot tell if she is receiving a token that she has previously owned. In [5], this property is achieved thanks to an extensive use of zero-knowledge proofs and a trusted party that tracks all the tokens in the system and thereby can detect and identify double spending attempts [5]. While capturing the highest level of privacy, we choose not to focus on coin transparency due to the resulting computational overhead, and the fact that physical cash does not satisfy it.

CBDC offline payments. More recently, Rial and Piotrowska propose and implement divisible solutions while focusing on distributing the role of the bank via threshold issuance [29]. They consider divisible e-cash and more efficient withdrawals, while we focus on efficient transferable e-cash. Another recent contribution is found in [4], which offers a transferable offline CBDC solution. Their approach uses more complex cryptography compared to ours and assigns the central bank the responsibility for covering double-spent values, potentially increasing the total value in the system. We prioritize equipping users with a secure element to prevent double spending rather than focusing on detecting it.

8 Conclusion and Future Work

This paper introduces a solution for offline payments that protects against double spending while preserving the privacy of honest users. The solution relies on cryptographic techniques to ensure the anonymity of users while enabling the identification of double spenders by authorized auditors. It also leverages tamper-resistant secure elements to increase the difficulty of mounting double spending attacks. Given the constrained nature of secure elements, the solution divides the computation into one that's lightweight and performed by the secure element, and one that's more expensive and offloaded to the more powerful smart phone. Our benchmarks confirm the viability of our approach and showcase its practicality.

References

1. Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-TAA. In Roberto De Prisco and Moti Yung, editors, *Security and Cryptography for Networks*, pages 111–125, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
2. Foteini Baldimtsi, Melissa Chase, Georg Fuchsbauer, and Markulf Kohlweiss. Anonymous transferable e-cash. In Jonathan Katz, editor, *Public-Key Cryptography – PKC 2015*, pages 101–124, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
3. Balthazar Bauer, Georg Fuchsbauer, and Chen Qian. Transferable e-cash: A cleaner model and the first practical instantiation. In Juan A. Garay, editor, *Public-Key Cryptography – PKC 2021*, pages 559–590, Cham, 2021. Springer International Publishing.
4. Carolin Beer, Sheila Zingg, Kari Kostiainen, Karl Wüst, Vedran Capkun, and Srdjan Capkun. Payoff: A regulated central bank digital currency with private offline payments. *arXiv preprint arXiv:2408.06956*, 2024.
5. Olivier Blazy, Sébastien Canard, Georg Fuchsbauer, Aline Gouget, Hervé Sibert, and Jacques Traoré. Achieving optimal anonymity in transferable e-cash with a judge. In Abderrahmane Nitaj and David Pointcheval, editors, *Progress in Cryptology – AFRICACRYPT 2011*, pages 206–223, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
6. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, pages 41–55, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
7. Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation using the strong diffie hellman assumption revisited. In Michael Franz and Panos Papadimitratos, editors, *Trust and Trustworthy Computing*, pages 1–20, Cham, 2016. Springer International Publishing.
8. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, pages 302–321, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
9. Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Giuseppe Persiano, and Clemente Galdi, editors, *Security in Communication Networks*, pages 268–289, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
10. Sébastien Canard and Aline Gouget. Anonymity in transferable e-cash. In Steven M. Bellovin, Rosario Gennaro, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 207–223, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

11. Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. Divisible e-cash made practical. In Jonathan Katz, editor, *Public-Key Cryptography – PKC 2015*, pages 77–100, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
12. Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. Scalable divisible e-cash. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *Applied Cryptography and Network Security*, pages 287–306. Springer International Publishing, 2015.
13. David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology*, pages 199–203, Boston, MA, 1983. Springer US.
14. David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.
15. David Chaum. Privacy protected payments-unconditional payer and/or payee untraceability. In *Proceedings of the IFIP WG 11.6 International Conference on SMART CARD 2000: The Future of IC Cards, Oct. 1987*, pages 69–93. Elsevier, 1988.
16. David Chaum and Torben Pryds Pedersen. Transferred cash grows in size. In Rainer A. Rueppel, editor, *Advances in Cryptology — EUROCRYPT' 92*, pages 390–407, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.
17. Oracle Corporation. Oracle java card technology. 2023.
18. T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
19. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO '86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.
20. Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Transferable constant-size fair e-cash. In Juan A. Garay, Atsuko Miyaji, and Akira Otsuka, editors, *Cryptology and Network Security*, pages 226–247, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
21. Julia Hesse, Nitin Singh, and Alessandro Sorniotti. How to bind anonymous credentials to humans. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3047–3064, Anaheim, CA, August 2023. USENIX Association.
22. Benoît Libert, Fabrice Mouhartem, Thomas Peters, and Moti Yung. Practical "signatures with efficient protocols" from simple assumptions. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '16, page 511–522, New York, NY, USA, 2016. Association for Computing Machinery.
23. Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard M. Heys and Carlisle M. Adams, editors, *Selected Areas in Cryptography, 6th Annual International Workshop, SAC'99, Kingston, Ontario, Canada, August 9-10, 1999, Proceedings*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 1999.
24. Igor Mikhalev, Kaj Burchardi, Igor Struchkov, Bihao Song, and Jonas Gross. Cbdc tracker. `https://cbdctracker.org/`, 2024.
25. Tatsuaki Okamoto. An efficient divisible electronic cash scheme. In Don Coppersmith, editor, *Advances in Cryptology — CRYPTO' 95*, pages 438–451, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.
26. Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO' 91*, pages 324–337. Springer, 1991.

27. David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *Topics in Cryptology - CT-RSA 2016*, pages 111–126, Cham, 2016. Springer International Publishing.
28. David Pointcheval and Olivier Sanders. Reassessing security of randomizable signatures. In Nigel P. Smart, editor, *Topics in Cryptology – CT-RSA 2018*, pages 319–338, Cham, 2018. Springer International Publishing.
29. Alfredo Rial and Ania M Piotrowska. Compact and divisible e-cash with threshold issuance. *Proceedings on Privacy Enhancing Technologies*, 4:381–415, 2023.
30. Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.
31. Stefano Tessaro and Chenzhi Zhu. Revisiting bbs signatures. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 691–721, Cham, 2023. Springer Nature Switzerland.
32. Stephan van Schaik, Alex Seto, Thomas Yurek, Adam Batori, Bader AlBassam, Christina Garman, Daniel Genkin, Andrew Miller, Eyal Ronen, and Yuval Yarom. SoK: SGX.Fail: How stuff get eXposed. 2022.

A Cryptographic Assumptions

In this section, we present the cryptographic assumptions that the building blocks of our solution rely on.

Definition 1 (Discrete Logarithm (DL) Assumption). *Let $\mathbb{G}$ be a cyclic group and P one of its generators. Let X be a random element in $\mathbb{G}$. We say that the DL assumption holds in $\mathbb{G}$ if for any p.p.t. algorithm $\mathcal{A}$ we have:*

$$Pr(\mathcal{A}(P, X) \to x : P^x = X) \leq \varepsilon(\kappa)$$

where ϵ is a negligible function.

Definition 2 (Decisional Diffie-Hellman (DDH) assumption). *Let $\mathbb{G}$ be a cyclic group and P one of its generators. We say that the DDH assumption holds in $\mathbb{G}$ if for any p.p.t. algorithm $\mathcal{A}$ we have:*

$$Pr(\mathcal{A}(P^x, P^y, P^{xy}) \to 1 : x, y \leftarrow \mathbb{Z}_p^*) = 1 \approx_c \\ Pr(\mathcal{A}(P^x, P^y, P^z) \to 1 : x, y, z \leftarrow \mathbb{Z}_p^*) = 1$$

Definition 3 (q-Strong Diffie-Hellman (q-SDH) Assumption). *Let $\boldsymbol{X} = (P^x, ..., P^{x^q}) \in \mathbb{G}_1^q$ and $\tilde{X} = \tilde{P}^x \in \mathbb{G}_2$ for a randomly-selected $x \in \mathbb{Z}_p$. We say that the q-SDH assumption holds if for any p.p.t. algorithm $\mathcal{A}$ we have:*

$$Pr(\mathcal{A}(P, \boldsymbol{X}, \tilde{P}, \tilde{X}) \to (e, P^{1/x+e})) \leq \varepsilon(\kappa)$$

where ϵ is a negligible function.

Definition 4 (q-Modified SDH (q-MSDH) Assumption). *Let $\boldsymbol{X} = (P^x, ..., P^{x^q})$, $\tilde{\boldsymbol{X}} = (\tilde{P}^x, ..., \tilde{P}^{x^q})$, and $(P^y, \tilde{P}^y, \tilde{P}^{xy})$ for a randomly-selected pair $(x, y) \in \mathbb{Z}_p^2$.*

We say that the q-MSDH assumption holds if for any p.p.t. algorithm $\mathcal{A}$ we have:

$$\begin{aligned} Pr(\mathcal{A}(P, \boldsymbol{X}, \tilde{P}, P^y, \tilde{P}^y, \tilde{P}^{xy}) \rightarrow (c, \mathfrak{f}, Q, Q^{1/c+x}, Q^{a/\mathfrak{f}(x)}) :& \\ & Q \in \mathbb{G}_1 \setminus \{1\} \\ & \wedge\ \mathfrak{f}(z) = \sum_{i=0}^{q} f_i z^i \wedge f_q \neq 0 \wedge \mathfrak{f}(-c) \neq 0) \\ & \leq \varepsilon(\kappa) \end{aligned}$$

B Cryptographic Primitives

This section provides an overview of the cryptographic primitives underpinning our solution.

Notations. Let [n] denote the integer interval $[0..n-1]$. Let $\boldsymbol{x} = (x_0, ..., x_{n-1})$ be a vector in $\mathbb{Z}_p^n$ and y an element in $\mathbb{Z}_p$. For all $m < n-1$, we denote by $\boldsymbol{x}[: m]$ sub-vector $(x_0, ..., x_{m-1})$, whereas by $\boldsymbol{x}||y$, we refer to the vector $(x_0, ..., x_{n-1}, y)$. Similar notations apply to vectors in $\mathbb{G}_1$ and $\mathbb{G}_2$. Let $\boldsymbol{y} = (y_0, ..., y_{n-1})$ be a vector in $\mathbb{Z}_p^n$. We denote by $\boldsymbol{x} \cdot \boldsymbol{y}$ the inner product of $\boldsymbol{x}$ and $\boldsymbol{y}$ (i.e., $\sum_{i=0}^{n-1} x_i y_i$). Let $\boldsymbol{P} = (P_0, ..., P_{n-1})$, $\tilde{\boldsymbol{P}} = (\tilde{P}_0, ..., \tilde{P}_{n-1})$ and $\boldsymbol{x} = (x_0, ..., x_{n-1})$ denote vectors in $\mathbb{G}_1^n$, $\mathbb{G}_2^n$ and $\mathbb{Z}_p^n$ respectively. We denote by $\boldsymbol{P}^{\boldsymbol{x}} = \prod_{i=0}^{n-1} P_i^{x_i}$ and by $\tilde{\boldsymbol{P}}^{\boldsymbol{x}} = \prod_{i=0}^{n-1} \tilde{P}_i^{x_i}$.

B.1 Zero-knowledge Proofs of Knowledge

Definition 5. *A binary relation $\mathcal{R}$ is defined by pairs $(\mathbb{x}, \mathbb{w})$ where $\mathbb{x}$ is called the instance, and $\mathbb{w}$ the witness. We say that $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$, if $(\mathbb{x}, \mathbb{w})$ satisfies $\mathcal{R}$.*

Interactive Arguments of Knowledge Let $(\mathcal{P}, \mathcal{V})$ be two algorithms defined as follows. *Prover* $\mathcal{P}$ and *Verifier* $\mathcal{V}$ are interactive algorithms. $\mathcal{P}$ takes as input $\mathcal{R}$, $\mathbb{x}$ and $\mathbb{w}$, whereas $\mathcal{V}$ takes as input $\mathcal{R}$ and $\mathbb{x}$. We denote by $\text{tr} \leftarrow \langle \mathcal{P}(\mathcal{R}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathcal{R}, \mathbb{x}) \rangle$ the transcript of the interaction between $\mathcal{P}$ and $\mathcal{V}$.

$\mathcal{V}$ concludes the interaction with $\mathcal{P}$ by returning a bit $b = \langle \mathcal{P}(\mathcal{R}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathcal{R}, \mathbb{x}) \rangle$. $b = 1$ indicates that $\mathcal{V}$ accepts tr; otherwise, we say that $\mathcal{V}$ rejects.

The pair $(\mathcal{P}, \mathcal{V})$ is an interactive argument of knowledge if it satisfies the following properties.

- **Completeness.** $(\mathcal{P}, \mathcal{V})$ is complete **iff** for any $(\mathbb{x}, \mathbb{w})$ such that $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$, $\langle \mathcal{P}(\mathcal{R}, \mathbb{x}, \mathbb{w}), \mathcal{V}(\mathcal{R}, \mathbb{x}) \rangle = 1$.
- **Knowledge Soundness.** $(\mathcal{P}, \mathcal{V})$ is knowledge-sound **iff** whenever $\langle \mathcal{P}(\cdot), \mathcal{V}(\mathcal{R}, \mathbb{x}) \rangle = 1$, one can build an extractor, which with access to $\mathcal{P}$, outputs $\mathbb{w}$ such that $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$.

We also say that $(\mathcal{P}, \mathcal{V})$ is zero-knowledge iff $\text{tr} \leftarrow \langle \mathcal{P}(\mathbb{x}, \mathbb{w}), \mathcal{V}(\mathcal{R}, \mathbb{x}) \rangle$ does not leak any information about witness $\mathbb{w}$.

Definition 6 (Public-Coin Interactive Arguments of Knowledge). *An interactive argument of knowledge $(\mathcal{P}, \mathcal{V})$ is public coin if all messages that $\mathcal{V}$ sends to $\mathcal{P}$ are generated uniformly at random.*

In the random oracle model (ROM), public-coin interactive (zero-knowledge) arguments of knowledge can be transformed into non-interactive arguments via the Fiat-Shamir heuristic [19]. More specifically, $\mathcal{V}$'s messages are computed as the hash of $\mathcal{P}$'s preceding messages.

Furthermore, zero-knowledge non-interactive arguments of knowledge can be made into **signatures of knowledge** of witness $\mathbb{w}$ over a message m by including m as input to the hashes.

B.2 Pedersen Commitments

A Pedersen commitment allows one to commit to a vector of messages $\boldsymbol{m}$ without leaking any information about the committed vector, thanks to the following algorithms:

- ComKeyGen$(1^\kappa, n)$ On input of security parameter κ and integer n, ComKeyGen outputs commitment key $\text{ck} = \boldsymbol{P} \| P_n = (P_0, ..., P_{n-1}, P_n) \in \mathbb{G}_1^{n+1}$.
- Commit$(\text{ck}, \boldsymbol{m}, r)$ On input of commitment key ck, vector of messages $\boldsymbol{m} \in \mathbb{Z}_p^n$, and randomness $r \in \mathbb{Z}_p$, Commit computes $\text{com} = \boldsymbol{P}^{\boldsymbol{m}} P_n^r$.
- Open$(\text{ck}, \text{com}, \boldsymbol{m}, r)$ On input of commitment key ck, commitment com, vector $\boldsymbol{m} \in \mathbb{Z}_p^n$, and randomness r, Open outputs 1 if $\text{com} = \boldsymbol{P}^{\boldsymbol{m}} P_n^r$; otherwise, it outputs 0.

Pedersen commitments are *unconditionally hiding* and *computationally binding* under the discrete logarithm assumption.

B.3 Elliptic Curve (EC) Elgamal Cryptosystem

The EC Elgamal cryptosystem [18] is an IND-CPA encryption that encrypts elements of elliptic curves. It consists of the following algorithms.

- EncKeyGen(1^κ) On input of security parameter κ, EncKeyGen outputs secret key $\text{SK} = a \in \mathbb{Z}_p^*$ and public key $\text{PK} = (P, A = P^a)$.
- Enc(PK, M, ρ) On input of public key epk, a message $M \in \mathbb{G}_1$ and randomness $\rho \in \mathbb{Z}_p^*$, Enc computes $\mathcal{E} = (P^\rho, MA^\rho)$.
- Dec$(\text{SK}, \mathcal{E})$ On input of secret key SK and ciphertext $\mathcal{E}$, Dec parses $\mathcal{E}$ as (E_1, E_2) and outputs $M = \frac{E_2}{E_1^a}$.

EC Elgamal is IND-CPA under the Decisional Diffie-Hellman (DDH) assumption. EC Elgamal is also homomorphic. Given $\mathcal{E}_1 = \text{Enc}(\text{PK}, M_1, \rho_1)$ and $\mathcal{E}_2 = \text{Enc}(\text{PK}, M_2, \rho_2)$, one obtains the encryption of $M_1 M_2$ by multiplying $\mathcal{E}_1$ and $\mathcal{E}_2$ component-wise. More specifically, $\mathcal{E} = \mathcal{E}_1 \circ \mathcal{E}_2 = \text{Enc}(\text{epk}, M_1 M_2, \rho_1 + \rho_2)$

B.4 BBS+ Signatures

A BBS+ signature [6, 1, 7] is a signature scheme that allows a signer to sign multiple messages at once using the algorithms below:

- BBSKeyGen$(1^\kappa, n)$ On input of security parameters κ and an integer n, BBSKeyGen randomly selects $t \in \mathbb{Z}_p$ and a vector of $n+2$ generators $(Q_0, ..., Q_n, H) \in \mathbb{G}_1^{n+2}$. We denote by $\boldsymbol{Q} = (Q_0, ..., Q_n)$. Finally, BBSKeyGen computes $\tilde{T} = \tilde{P}^t$ and outputs secret key SK $= t$ and public key PK $= (\tilde{T}, \tilde{P}, \boldsymbol{Q}, H)$.
- BBSSign(SK, $\boldsymbol{m}$) On input of secret key SK and a vector $\boldsymbol{m}$ of n messages, BBSSign randomly selects $(e, s) \in \mathbb{Z}_p$ such that $e + t \neq 0$, then computes $C = H\boldsymbol{Q}^{\boldsymbol{m}\|s}$ and outputs $\phi = (e, s, E = C^{1/t+e})$.
- BBSVerify(PK, $\boldsymbol{m}, \phi$) On input of public key PK, vector of messages $\boldsymbol{m}$ and signature ϕ, BBSVerify computes $C = \boldsymbol{Q}^{\boldsymbol{m}\|s}H$, and checks if the following equality holds:
$$e(E, \tilde{P}^e\tilde{T}) = e(C, \tilde{P})$$

BBS signatures are existentially unforgeable under the q-strong Diffie-Hellman (q-SDH) assumption.

B.5 Joint Proof of Knowledge of BBS signature

Let $\phi = (e, s, E) \leftarrow$ BBSSign(SK, $\boldsymbol{m}$) be a BBS signature. By construction, $E = (H\boldsymbol{Q}^{\boldsymbol{m}\|s})^{1/t+e}$, where SK $= t$. Let $\mathbb{x} =$ PK $= (\tilde{P}, \tilde{T}, \boldsymbol{Q}, H)$ and $\mathbb{w} = (\boldsymbol{m}, \phi)$.

Proof of Knowledge of a BBS Signature A proof of knowledge of a BBS signature corresponds to showing that the pair $(\mathbb{x}, \mathbb{w})$ satisfies the following relation:
$$e(E, \tilde{P}^e\tilde{T}) = e(H\boldsymbol{Q}^{\boldsymbol{m}\|s}, \tilde{P})$$
As shown in [31], an alternative is for a prover to compute $U = E^{-x/y}$, $V = (H\boldsymbol{Q}^{\boldsymbol{m}\|s})^{-1/y}$, and $W = U^eV^x$, for randomly selected x and y, and prove that for $\mathbb{x} = (\text{PK}, U, V, W)$ and $\mathbb{w} = (e, s, x, y, \boldsymbol{m})$ the following relation holds
$$e(U, \tilde{T})e(W, \tilde{P}) = 1 \wedge W = U^eV^x \wedge H^{-1} = V^y\boldsymbol{Q}^{\boldsymbol{m}\|s} \tag{3}$$
Let $\mathcal{P}$ be such a prover; $\mathcal{P}$ follows a Sigma protocol, which is made non-interactive using Fiat-Shamir Heuristics. Hereafter, we refer to this type proof as Schnorr proofs. In more details, $\mathcal{P}$ selects elements $\alpha, \beta, \delta, \gamma_0, ..., \gamma_{n-1}, \gamma$ in $\mathbb{Z}_p$ and computes
$$A = U^\alpha V^\beta \ ; \ B = V^\delta Q_n^\gamma \prod_{i=0}^{n-1} Q_i^{\gamma_i} \ ; \ \theta = H_p(\mathbb{x}, A, B)$$
$$\eta = \alpha + \theta e \ ; \ \nu = \beta + \theta x \ ; \ \xi = \delta + \theta y \ ;$$
$$\pi_i = \gamma_i + \theta m_i, \ \forall i \in [n] \ ; \ \pi = \gamma + \theta s$$

$\mathcal{P}$ outputs $\varPi = (A, B, \eta, \nu, \xi, \pi_0, ..., \pi_{n-1}, \pi)$.

Upon receipt of proof $\varPi$, a verifier $\mathcal{V}$ first checks whether $e(U, \tilde{T})e(W, \tilde{P}) = 1$. If not, it rejects. Then it computes the challenge $\theta = H(\mathbb{x}, A, B)$ and verifies if the following holds:

$$AW^{\theta} = U^{\eta}V^{\nu} \ ; \ BH^{-\theta} = V^{\xi}Q_n^{\pi}\prod_{i=0}^{n-1} Q_i^{\pi_i}$$

If that's not the case, then $\mathcal{V}$ rejects. Otherwise, it accepts.

Joint Proof of Knowledge of a BBS Signature We now show how two provers $\mathcal{P}_0$ and $\mathcal{P}_1$ jointly produce the proof $\varPi$, such that $\mathcal{P}_0$ is a lightweight device which knows m_{n-1}, and $\mathcal{P}_1$ is computationally more powerful and knows $(e, s, m_0, ..., m_{n-2})$. The proof technique detailed below follows the approach introduced in [21]. More specifically, instead of proving the relation depicted in Equation 3, $\mathcal{P}_0$ and $\mathcal{P}_1$ prove that the following relation is satisfied:

$$\text{com} = Q_{n-1}^{m_{n-1}}Q_n^{\rho} \ \wedge e(U, \tilde{T})e(W, \tilde{P}) = 1 \ \wedge$$
$$W = U^eV^x \ \wedge \ (\text{com}H)^{-1} = V^yQ_n^{s'}\prod_{i=0}^{n-2} Q_i^{m_i}$$

for $\mathbb{x} = (\text{PK}, U, V, W, \text{com})$ and $\mathbb{w} = (e, s', x, y, \boldsymbol{m}, \rho)$.

Notably, $\mathcal{P}_0$ first computes commitment com $= Q_{n-1}^{m_{n-1}}Q_n^{\rho}$, and produces a non-interactive zero-knowledge proof $\varPi_0$ that shows that instance $\mathbb{x}_0 = (Q_{n-1}, Q_n, \text{com})$ and witness $\mathbb{w}_0 = (m_{n-1}, \rho)$ satisfy the relation com $= Q_{n-1}^{m_{n-1}}Q_n^{\rho}$. That is, $\mathcal{P}_0$ selects randomly (γ_{n-1}, υ), and computes $C = Q_{n-1}^{\gamma_{n-1}}Q_n^{\upsilon}$, $\theta_0 = H_p(\mathbb{x}_0, C)$, $\pi_{n-1} = \gamma_{n-1} + \theta_0 m_{n-1}$, and $\zeta = \upsilon + \theta_0\rho$. Finally $\mathcal{P}_0$ sends to $\mathcal{P}_1$ tuple: $(\text{com}, \rho, \varPi_0)$, where $\varPi_0 = (C, \pi_{n-1}, \zeta)$. Note that π_{n-1} and ζ are Schnorr proofs.

$\mathcal{P}_1$ computes a zero-knowledge proof $\varPi_1$ (based on Schnorr proofs) that proves that Equation 4 holds for $\mathbb{x}_1 = (\text{PK}, U, V, W, \text{com})$ and $\mathbb{w}_1 = (e, , s', x, y, \boldsymbol{m}[: n-1])$, where $s' = s - \rho$.

$$W = U^eV^x \ \wedge \ (\text{com}H)^{-1} = V^y\prod_{i=0}^{n-2} Q_i^{m_i}Q_n^{s'} \tag{4}$$

In fact, $\mathcal{P}_1$ selects random pair (α, β) and random vector $(\delta, \gamma_0, ..., \gamma_{n-2}, \gamma)$, and computes $A = U^{\alpha}V^{\beta}$ and $B = V^{\delta}Q_n^{\gamma}\prod_{i=0}^{n-2} Q_i^{\gamma_i}$. Then $\mathcal{P}_1$ computes $(\theta_1, \eta, \nu, \xi, \pi_0, ..., \pi_{n-2}, \pi)$ as follows:

$$\theta_1 = H_p(\mathbb{x}_1, A, B) \ ; \ \eta = \alpha + \theta_1 e; \ ; \nu = \delta + \theta_1 x \ ; \xi = \delta + \theta_1 y \ ;$$
$$\pi_i = \gamma_i + \theta_1 m_i, \ \forall i \in [n-1] \ ; \ \pi = \gamma + \theta_1 s'$$

$\mathcal{P}_1$ terminates by outputting

$$\varPi = (\text{com}, A, B, C, \eta, \nu, \xi, \pi_0, ..., \pi_{n-1}, \pi, \zeta).$$

Mapping to Our Solution In our solution, the secure element SE and the mobile device MD are required to produce $\sigma \leftarrow \textsc{sok}[\mathbb{x}, \mathbb{w} : \mathcal{R}(\mathbb{x}, \mathbb{w}) = 1]$, whereby $\mathbb{x} = (P_0, P_1, \mathrm{PK}_{\mathrm{RA}}, U, V, W, c)$, $\mathrm{PK}_{\mathrm{RA}} = (\tilde{P}, \tilde{T}, Q_0, Q_1, Q_2, H)$, $\mathbb{w} = (e, x, y, s, \mathrm{id}, \mathrm{sk}, r)$, and $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$ is equivalent to satisfying equations

$$e(U, \tilde{T})e(W, \tilde{P}) = 1 \ \wedge \ W = U^e V^x$$
$$\wedge \, H^{-1} = V^y Q_0^{\mathrm{id}} Q_1^{\mathrm{sk}} Q_2^s \ \wedge \ C = P_0^{\mathrm{id}} P_1^r$$

which corresponds to instantiating Equation 3 with $\boldsymbol{m} = (\mathrm{id}, \mathrm{sk})$ and where $C = P_0^{\mathrm{id}} P_1^r$.

SE in this scenario executes $\mathcal{P}_0$ and outputs $\mathrm{com} = Q_1^{\mathrm{sk}} Q_2^{\rho}$, ρ, proof Π_0 which is computed as described above except that the challenge θ_0 also hashes the message to be signed.

MD, on the other hand, computes a zero-knowledge proof $\Pi_1 \leftarrow \textsc{pok}[\mathbb{x}_1, \mathbb{w}_1 : \mathcal{R}(\mathbb{x}_1, \mathbb{w}_1) = 1]$, where $\mathbb{x}_1 = (P_0, P_1, \mathrm{PK}_{\mathrm{RA}}, U, V, W, \mathrm{com}, C)$, $\mathbb{w} = (e, x, y, \mathrm{id}, s', r)$, and $\mathcal{R}$ is defined by the following:

$$W = U^e V^x \wedge (\mathrm{com} H)^{-1} = V^y Q_0^{\mathrm{id}} Q_2^{s'} \wedge C = P_0^{\mathrm{id}} P_1^r$$

Π_1 will be computed using Schnorr proof systems, similar to proof Π_1 in Appendix B.5.

B.6 Pointcheval-Sanders Signatures

The Pointcheval-Sanders signature scheme [27, 28] is a scheme which like BBS allows the signing of vector of messages as opposed to single messages. This is achieved by executing the following algorithms.

- PSKeyGen$(1^\kappa, n)$ On input of security parameter κ and integer n, PSKeyGen randomly selects vector $(x, y_0, ..., y_{n-1}, z) \in \mathbb{Z}_p^{n+2}$, and computes $\tilde{X} = \tilde{P}^x$, $\tilde{Y}_i = \tilde{P}^{y_i}$ $\forall i \in [n]$ and $\tilde{Z} = \tilde{P}^z$. Finally, PSKeyGen outputs $\mathrm{SK} = (x, \boldsymbol{y}, z)$ and $\mathrm{PK} = (\tilde{X}, \tilde{\boldsymbol{Y}}, \tilde{Z})$, where $\boldsymbol{y} = (y_0, ..., y_{n-1})$ and $\tilde{\boldsymbol{Y}} = (\tilde{Y}_0, ..., \tilde{Y}_{n-1})$.
- PSSign$(\mathrm{SK}, \boldsymbol{m})$ On input of secret key SK and a vector of n messages $\boldsymbol{m}$, PSSign randomly selects R and u in $\mathbb{G}_1$ and $\mathbb{Z}_p$ respectively, computes $S = R^{x + \boldsymbol{y} \cdot \boldsymbol{m} + zu}$ and outputs signature $\psi = (u, R, S)$.
- PSVerify$(\mathrm{PK}, \boldsymbol{m}, \psi)$ On input of public key PK, vector of messages $\boldsymbol{m}$ and signature ψ, Verify checks if the following equality holds: $e(R, \tilde{X}\tilde{\boldsymbol{Y}}^{\boldsymbol{m}} \tilde{Z}^u) = e(S, \tilde{P})$

The resulting signatures are existentially unforgeable under the q-Modified Strong Diffie-Hellman (q-MSDH) assumption [28]. Furthermore, the Pointcheval-Sanders signatures can be easily extended to support **blind signing**; more specifically, a Pointcheval-Sanders signer can sign a committed vector without learning any information about said vector, cf. [30].

B.7 Pointcheval-Sanders Signatures over Committed Inputs

Let $\boldsymbol{P} = (P_0, ..., P_n) \in \mathbb{G}_1^{n+1}$ be a vector of $n+1$ random generators and $\boldsymbol{m} = (m_0, ..., m_{n-1}) \in \mathbb{Z}_p^n$ a vector of n messages.

Signatures over committed inputs [9, 22] is an interactive protocol between a **signer** and a **recipient**, in which the signer signs a vector of *committed* messages in such a way that only the recipient knows the content of these messages.

In the case of Pointcheval-Sanders signatures, the recipient submits to the signer a Pedersen commitment $\boldsymbol{P}^{\boldsymbol{m}||r}$, an Elgamal encryption of $\boldsymbol{m}$ to the signer, and a zero-knowledge proof that shows that the encryption is correct. The signer, in turn, verifies the validity of the zero-knowledge proof with respect to the transmitted commitment and ciphertexts. If the proof is valid, the signer signs and aggregates the ciphertexts and returns the result to the recipient. The recipient decrypts the signer's response using the secret key of Elgamal encryption, and derives a signature on $\boldsymbol{m}$. More specifically:

- The recipient first selects an Elgamal secret key $b \in \mathbb{Z}_p$ and computes the corresponding public key $(P, B = P^b)$. Then she computes $\mathrm{com} = \boldsymbol{P}^{\boldsymbol{m}||r}$, $R = H_{\mathbb{G}_1}(\mathrm{com})$, and n Elgamal ciphertexts $\mathcal{E}_i = (P^{\rho_i}, R^{m_i} B^{\rho_i}) : i \in [n]$. Afterwards, she generates a zero-knowledge proof of knowledge $\Pi \leftarrow \mathrm{POK}[\mathbb{x}, \mathbb{w} : \mathcal{R}(\mathbb{x}, \mathbb{w}) = 1]$ where $\mathbb{x} = (\boldsymbol{P}, P, B, \mathrm{com}, R, (\mathcal{E}_i)_{i\in[n]})$, $\mathbb{w} = ((m_i, \rho_i)_{i\in[n]}, r)$ and $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$ corresponds to satisfying the following:

$$1 = \boldsymbol{P}^{\boldsymbol{m}||r} \ \wedge \ \mathcal{E}_i = (P^{\rho_i}, R^{m_i} B^{\rho_i}), \forall i \in [n]$$

  and sends to the signer the tuple: $(\mathrm{com}, \mathcal{E}_0, ..., \mathcal{E}_{n-1}, P, B, \Pi)$
- The signer recomputes $R = H_{\mathbb{G}_1}(\mathrm{com})$, and using Π verifies that each ciphertext $\mathcal{E}_i$ is computed correctly in relation to commitment com, hash R and Elgamal public key (P, B). If not, the signer rejects the request. Else, she parses each ciphertext $\mathcal{E}_i$ as $(E_{i,1}, E_{i,2})$, randomly selects $u \in \mathbb{Z}_p$, computes

$$\mathcal{E} = (E_1, E_2) = (\prod_{i=0}^{n-1} E_{i,1}^{y_i}, R^x \prod_{i=1}^{n} E_{i,2}^{y_i} R^{uz})$$

  and finally sends $(u, \mathcal{E})$ to the recipient.
- The recipient decrypts $\mathcal{E} = (E_1, E_2)$ by computing $S = E_2/E_1^b$, and verifies if $\psi = (u, R, S)$ is a valid Pointcheval-Sanders signature on vector $\boldsymbol{m}$. If so, then she outputs ψ. Notice that if the signer honestly follows the protocol then

$$S = R^x \left(\prod_{i=0}^{n-1} R^{y_i m_i} \right) R^{uz}$$

We note that this protocol can be easily adjusted to support the blind signing of partially opened vectors. Let $\mathcal{D}$ be a subset of $[n]$ such that $(m_i)_{i\in\mathcal{D}}$ are revealed to the signer. The protocol proceeds as above except that the recipient computes $\mathrm{com} = \prod_{i\in[n]\setminus\mathcal{D}} P_i^{m_i}$, produces zero-knowledge proof $\Pi \leftarrow \mathrm{POK}[\mathbb{x}, \mathbb{w} :$

$\mathcal{R}(\mathbb{x},\mathbb{w}) = 1]$ for $\mathbb{x} = (\mathcal{D}, \boldsymbol{P}, P, B, \text{com}, R, (\mathcal{E}_i)_{i\in[n]\setminus\mathcal{D}})$, $\mathbb{w} = ((m_i, \rho_i)_{i\in[n]\setminus\mathcal{D}}, r)$, and $\mathcal{R}(\mathbb{x},\mathbb{w}) = 1$ being equivalent to the following equalities:

$$\text{com} = P_n^r \prod_{i\in[n]\setminus\mathcal{D}} P_i^{m_i} \ \wedge \mathcal{E}_i = (P^{\rho_i}, R^{m_i} B^{\rho_i}), \forall i \in [n] \setminus \mathcal{D}$$

and sends $m_i \forall i \in \mathcal{D}$ in the clear, as opposed to being encrypted. The signer then computes $\mathcal{E}$ as:

$$\mathcal{E} = (E_1, E_2) = (\prod_{i\in[n]\setminus\mathcal{D}} E_{i,1}^{y_i}, R^x \prod_{i\in\mathcal{D}} R^{m_i y_i} \prod_{i\in[n]\setminus\mathcal{D}} E_{i,2}^{y_i} R^{uz})$$

We note that the above protocol can be easily tweaked to support the withdrawal protocol; more specifically, we execute it with vector of messages (id_0, v, s_0) and randomness $r = 0$.

C Proof of Withdrawal Σ

Let $\tau = (v, C = P_0^{\text{id}} P_1^s)$ be a token. Let $\text{PK}_{\text{CB}} = (\tilde{X}, \tilde{Y}_0, \tilde{Y}_1, \tilde{Y}_2, \tilde{Z})$ be the public key of central bank CB. Let $\psi = (u, R, S)$ be the Pointcheval-Sanders signature that verifies $1 \leftarrow \text{PSVerify}(\text{PK}_{\text{CB}}, (\text{id}, v, s), \psi)$. That is, (u, R, S) verifies the following equality: $e(R, \tilde{X}\tilde{Y}_0^{\text{id}}\tilde{Y}_1^v\tilde{Y}_2^s\tilde{Z}^u) = e(S, \tilde{P})$.

To produce the proof Σ that τ was issued by CB, the owner of τ uses its mobile device MD to execute the following steps. (1) select a random number $r \in \mathbb{Z}_p$ and compute $(R^*, S^*) = (R^r, S^r)$. (2) Generate $\Delta \leftarrow \text{POK}[\mathbb{x}, \mathbb{w} : \mathcal{R}(\mathbb{x},\mathbb{w}) = 1]$, whereby $\mathbb{x} = (P_0, P_1, \tilde{X}, \tilde{Y}_0, \tilde{Y}_1, \tilde{Y}_2, \tilde{Z}, v, C, R^*, S^*)$, witness $\mathbb{w} = (u, \text{id}, s)$, and $\mathcal{R}(\mathbb{x},\mathbb{w}) = 1$ corresponds to the equalities:

$$e(S^*, \tilde{P}) = e(R^*, \tilde{X}\tilde{Y}_0^{\text{id}}\tilde{Y}_1^v\tilde{Y}_2^s\tilde{Z}^u) \ \wedge \ C = P_0^{\text{id}} P_1^s.$$

(3) Finally, return $\Sigma = (R^*, S^*, \Delta)$. It should be noted that since s_0 is random and not known, the pair (R^*, S^*) leaks no information about the identifier id_0.

Now we detail how MD computes Δ. It first computes commitment $\tilde{C} = \tilde{Y}_0^{\text{id}}\tilde{Y}_2^s\tilde{Z}^u$. Then it randomly selects α, β, δ in $\mathbb{Z}_p$, computes $A = P_0^\alpha P_1^\beta$, $\tilde{A} = \tilde{Y}_0^\alpha \tilde{Y}_2^\beta \tilde{Z}^\delta$ and challenge $\gamma = H_p(\mathbb{x}, \tilde{C}, A, \tilde{A})$. Finally, it computes three proofs $\pi_{\text{id}} = \alpha + \gamma\text{id}$, $\pi_s = \beta + \gamma s$ and $\pi_u = \delta + \gamma u$, and sets Δ to tuple $(\tilde{C}, A, \tilde{A}, \pi_{\text{id}}, \pi_s, \pi_u)$.

We remark that π_{id}, π_s and π_u are standard Schnorr proofs for the relation $\mathcal{R}$ defined for $\mathbb{x}' = (P_0, P_1, \tilde{Y}_0, \tilde{Y}_2, \tilde{Z}, C, \tilde{C})$, witness $\mathbb{w}' = (u, \text{id}, s)$, and $\mathcal{R}(\mathbb{x}',\mathbb{w}') = 1$ is equivalent to

$$C = P_0^{\text{id}} P_1^s \ \wedge \ \tilde{C} = \tilde{Y}_0^{\text{id}}\tilde{Y}_2^s\tilde{Z}^u.$$

We also remark that if (u, R^*, S^*) is a valid Pointcheval-Sanders signatures on (id, v, s) under public key PK_{CB}, then $e(S^*, \tilde{P}) = e(R^*, \tilde{X}\tilde{Y}_0^{\text{id}}\tilde{Y}_1^v\tilde{Y}_2^s\tilde{Z}^u)$, and therefore $e(S^*, \tilde{P}) = e(R^*, \tilde{X}\tilde{C}\tilde{Y}_1^v)$. Therefore, given proof $\Sigma = (R^*, S^*, \Delta)$ and token $\tau = (v, C)$, a verifier can check that τ was issued by central bank CB by parsing Δ as $(\tilde{C}, A, \tilde{A}, \pi_{\text{id}}, \pi_s, \pi_u)$, computing $\gamma = H_p(\mathbb{x}', A, \tilde{A})$ and verifying that the following holds:

$$e(S^*, \tilde{P}) = e(R^*, \tilde{X}\tilde{C}\tilde{Y}_1^v) \wedge C^\gamma A = P_0^{\pi_{\text{id}}} P_1^{\pi_s} \wedge \tilde{C}^\gamma \tilde{A} = \tilde{Y}_0^{\pi_{\text{id}}} \tilde{Y}_2^{\pi_s} \tilde{Z}^{\pi_u}$$

D Aggregating Deposits

To allow the aggregation of deposits in a way that hides the values of individual tokens, we change the protocol described in Section 5 slightly in the following way. A payment from U_i to U_{i+1} will consist of $\mathfrak{p}_i = (v, r, C_v, C_{i+1}, C_i, \mathcal{E}_i, \sigma_i, \Gamma_i, \mathrm{hist}_i)$, where $C_v = P_0^v P_1^r$ is a hiding commitment to the value, computed by U_0 at time of token creation, and σ_i is a signature of knowledge on (C_v, C_i, C_{i+1}) instead of (v, C_i, C_{i+1}).

Recall that $\mathrm{hist}_i = ((C_k, \mathcal{E}_k, \sigma_k, \Gamma_k)_{k\in[i]}, \Sigma)$. Now, $\Sigma \leftarrow \textsc{pok}[\mathbb{x}, \mathbb{w} : \mathcal{R}(\mathbb{x}, \mathbb{w})]$, with $\mathbb{x} = (P_0, P_1, \tilde{P}, \tilde{X}, \tilde{Y}_0, \tilde{Y}_1, \tilde{Y}_2, \tilde{Z}, C_v, C_0)$, $\mathbb{w} = (u, \mathrm{id}_0, v, s_0, r, \psi)$, and $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$ corresponds to

$$1 \leftarrow \mathrm{PSVerify}(\mathrm{PK_{CB}}, (\mathrm{id}_0, v, s_0), \psi) \ \wedge\ C_0 = P_0^{\mathrm{id}_0} P_1^{s_0} \ \wedge\ C_v = P_0^v P_1^r$$

If user U_n would like to deposit l tokens at once, then she proceeds as following. Let $\mathfrak{d}_{n,i} = (C_{v,i}, C_{n,i}, \mathrm{hist}_{n,i})$ be the deposit request of the i^{th} token where $C_{v,i} = P_0^{v_i} P_1^{r_i}$ and $\mathrm{hist}_{n,i} = ((C_{k,i}, \mathcal{E}_{k,i}, \sigma_{k,i}, \Gamma_{k,i})_{k\in[n]}, \Sigma_i)$. U_n prepares an aggregate deposit request $\mathfrak{d}_n = (v, r, \mathfrak{d}_{n,1}, ..., \mathfrak{d}_{n,l})$ such that $v = \sum_{i=1}^{l} v_i$ and $r = \sum_{i=1}^{l} r_i$.

The central bank verifies this request by checking first that $\prod_{i=1}^{l} C_{v,i} = P_0^v P_1^r$, then the validity of the signatures and the zero-knowledge proofs. The central bank checks for double spending for each deposit $\mathfrak{d}_{n,i}$ by verifying whether there exists another deposit $\mathfrak{d}' = (C'_v, C', \mathrm{hist}')$ such that $\mathrm{hist}' = ((C'_k, \mathcal{E}'_k, \sigma'_k, \Gamma'_k)_{k\in[m]}, \Sigma')$ and $C'_0 = C_{0,i}$. Finally, the double spender identification proceeds as described in Section 5 for each individual deposit request.

Since double spending is decided based solely on the commitment C and not the token $\tau = (v, C)$ as is the case in Section 5, we need to change the withdrawal process to avoid cases where a user *inadvertently* reuses the same randomness for two withdrawals with different values, and ends up being accused of double spending. Notably, during withdrawal, instead of checking the uniqueness of the received $\mathrm{com} = P_0^{\mathrm{id}} P_1^v P_2^s$, the central bank checks whether $\mathrm{com}' = \mathrm{com}/P_1^v$ is unique. This guarantees that if a user chooses the same randomness in two withdrawals, then this will be detected by the central bank and rejected.

E Security Analysis

Let prot denote the protocol described in Section 5. We use a series of indistinguishable games to prove that simulator $\mathcal{S}$ interacting with functionality $\mathcal{F}$ is able to successfully simulate an execution of our protocol (denoted hereafter prot) with adversary $\mathcal{A}$. For ease of exposition, we reason about users whose secure element and device are both corrupt, and we call them corrupt users. Also for simplicity purposes, we assume that the randomness ρ generated by $\mathcal{F}$ during Withdraw and Pay is group element in $\mathbb{G}_1$.

Game 0. This corresponds to an execution of prot.

Game 1. $\mathcal{S}$ runs prot on behalf of all honest users. This means $\mathcal{S}$ receives inputs from honest users, executes the functions of prot, and distributes the corresponding outputs to all necessary users. This is indistinguishable from Game 0.

Game 2. $\mathcal{S}$ now uses a dummy functionality $\mathcal{F}$ to execute prot on behalf of the honest users. The honest users and $\mathcal{S}$ communicate via $\mathcal{F}$ rather than directly with one another. This is also indistinguishable from Game 1.

Game 3. $\mathcal{F}$ now answers InitAcc queries from honest users. When an honest user U wishes to initialize an account with value v with some honest-but-curious intermediary I, $\mathcal{S}$ invokes $\mathcal{F}$ with request $(\textsc{InitAcc}, \mathrm{U}, v)$ as opposed to running prot. This game is indistinguishable from Game 2 as requests of honest users are always accepted whether one calling $\mathcal{F}$ or executing prot, and the information leaked during a run of prot when creating an account and that leaked to $\mathcal{S}$ when invoking InitAcc are identical.

Game 4. When a corrupt user U initiates an account creation with value v with some honest-but-curious intermediary I, $\mathcal{S}$ invokes $\mathcal{F}$ with the request $(\textsc{InitAcc}, \mathrm{U}, v)$. If $\mathcal{F}$ aborts, $\mathcal{S}$ instructs I to abort. Note that Game 4 is distinguishable from Game 3 only if $\mathcal{S}$ orders I to abort, but an actual execution of prot does not lead to an abort. However, since $\mathcal{F}$ aborts only when U already has an account and the same check is enforced by prot, this discrepancy will not occur.

Game 5. $\mathcal{F}$ now also answers Register queries. When the honest-but-curious registration authority receives a *valid* registration request from some user U, $\mathcal{S}$ invokes $\mathcal{F}$ with request $(\textsc{Register}, \mathrm{U}, \mathrm{SE}, \mathrm{MD})$. A valid registration request is a request where the user proves knowledge of the secret key corresponding to her advertised public key. If $\mathcal{F}$ rejects the request, $\mathcal{S}$ instructs the registration authority to abort the execution of prot. Now Game 5 is distinguishable from Game 4 only if $\mathcal{S}$ makes the registration authority abort, but an execution of prot does not result in an abort. Since $\mathcal{F}$ aborts only when U is already registered, and this is also enforced by prot, the event above will never occur.

Game 6. $\mathcal{F}$ now answers Withdraw queries from honest users. When an honest user U wishes to withdraw a token of value v, her intermediary I directly invokes $\mathcal{F}$ with request $(\textsc{Withdraw}, \mathrm{U}, v)$. $\mathcal{S}$ in this case learns the identity of the user U behind the withdrawal and the value v. Given this information, $\mathcal{S}$ generates a withdrawal transcript that matches this information. More specifically, it generates a withdrawal request $(\mathrm{id}, R, \mathrm{com}, \mathcal{E}_0, v, \mathcal{E}_2, P, B, \Pi, \rho_0)$ such that id is the identifier of U, com is a random group element in $\mathbb{G}_1$, (P, B) is a one-time Elgamal public key, $\mathcal{E}_0$ is an Elgamal encryption of id using randomness ρ_0, $\mathcal{E}_2$ is a random an Elgamal ciphertext, and Π is a simulated zero-knowledge proof that proves that the withdrawal request is well formed. Thanks to the hiding property of Pedersen commitments, the semantic security of Elgamal encryption, and the simulatability of the Schnorr zero-knowledge proofs in the random oracle model, this withdrawal request is indistinguishable from one that's generated according to prot. Therefore, this game is indistinguishable from Game 5.

Game 7. When a corrupt and registered user U sends a withdrawal request to their honest-but-curious intermediary I, $\mathcal{S}$ invokes $\mathcal{F}$ with request $(\textsc{Withdraw}, \mathrm{U}, v)$, and $\mathcal{S}$ instructs I and CB to process the withdrawal request only if $\mathcal{F}$ accepts. We first remark that the accounting checks during withdrawal are identical in Game 6 and Game 7: prot and $\mathcal{F}$ perform the same checks. It follows that if the withdrawal request is rejected because of the lack of funds by $\mathcal{F}$, it will also be rejected during an execution of prot. Assuming that the withdrawal request passes the accounting checks, $\mathcal{S}$ – given the withdrawal request and the attached zero-knowledge proof – first extracts s, where s the randomness used by U during withdrawal. $\mathcal{S}$ then provides $\mathcal{F}$ with $\rho = P_0^{\mathrm{id}} P_1^s P_2^v$. We recall that $\mathcal{F}$ rejects the withdrawal request if ρ is not unique. We also recall that a withdrawal request is rejected in prot if $\mathrm{com} = P_0^{\mathrm{id}} P_1^v P_2^s$ contained in the withdrawal request is not unique. Hence, if com is duplicated, so would ρ, so if the request is rejected by $\mathcal{F}$, it will also be rejected by prot. One can see that this game is indistinguishable from Game 6.

Game 8. $\mathcal{F}$ now answers Pay queries from honest users. For simplicity, we start by assuming that this payment is the first payment after a withdrawal. An honest payer U transfers the ownership of a token of value v to a payee U$'$ by invoking $\mathcal{F}$ with $(\text{Pay}, \rho, v, \mathrm{U}')$. There are two cases. 1) If U$'$ is honest, then $\mathcal{S}$ only learns $(\text{Pay}, v, \rho, \rho')$ with ρ' produced by $\mathcal{F}$. In this case, $\mathcal{S}$ generates payment $\mathfrak{p}_0 = (v, C_1, C_0, \mathcal{E}_0, \sigma_0, \Gamma_0, \mathrm{hist}_0 = \Sigma)$ such that $C_0 = P_0^{r_0} P_1^{s_0}$, $C_1 = P_0^{r_1} P_1^{s_1}$, $\mathcal{E}_0$ is a random Elgamal ciphertext under the auditor's public key, σ_0 is a simulated signature of knowledge, and (Γ_0, Σ) are simulated zero-knowledge proofs. 2) If U$'$ is corrupt, then $\mathcal{S}$ learns the identities of U and U$'$, and proceeds as follows: it computes $\rho' = C_1 P_2^v$, where C_1 is the commitment that U$'$ supplies when receiving the payment request during an execution of prot, and provides it to $\mathcal{F}$, and simulates the payment following the description above, with the only difference being that $\mathcal{E}_0$ is an Elgamal encryption of the identifier of U. $\mathcal{S}$ concludes by storing the payment in a map $\mathbb{P}$ dedicated to payments; that is, $\mathcal{S}$ adds entry $\mathbb{R}[(C_0, C_1)] \leftarrow (\rho, \rho')$ and $\mathbb{C}[(\rho, \rho')] \leftarrow (C_0, C_1)$.

Now we move onto the general case where the call to Pay corresponds to the i^{th} transfer of ownership of a token. The honest payer U transfers ownership of a token of value v to a payee U$'$ by invoking $\mathcal{F}$ with $(\text{Pay}, \boldsymbol{\rho}, v, \mathrm{U}')$ where $\boldsymbol{\rho} = (\rho_0, \rho_1, ..., \rho_i)$. $\mathcal{S}$ learns $\boldsymbol{\rho}$ and ρ'. If U$'$ is honest, then $\mathcal{S}$ receives ρ' from $\mathcal{F}$, otherwise it provides $\rho' = C_{i+1} P_2^v$, where C_{i+1} is the commitment provided by U$'$ during an execution of prot to receive the payment. $\mathcal{S}$ then finds entry $\mathbb{C}[\boldsymbol{\rho}] \leftarrow (C_0, ..., C_i)$, and produces a simulated payment $\mathfrak{p}_i = (v, C_{i+1}, C_i, \mathcal{E}_i, \sigma_i, \Gamma_i, \mathrm{hist}_i)$ such that the zero-knowledge proofs are simulated and the ciphertexts are random Elgamal ciphertexts under the auditor's public key. $\mathcal{S}$ terminates by adding entry $\mathbb{R}[(C_0, ..., C_i, C_{i+1})] \leftarrow (\rho_0, ..., \rho_i, \rho')$ and entry $\mathbb{C}[(\rho_0, ..., \rho_i, \rho')] \leftarrow (C_0, ..., C_{i+1})$.

It is clear that this game is indistinguishable from Game 7.

Game 9. $\mathcal{F}$ now also processes payments initiated by $\mathcal{S}$ on behalf of corrupt users. Assume that corrupt payer U wishes to send a token to payee U$'$, and that this is the first payment following a withdrawal operation. Let pair (v, C_0) denote the token used in this payment. $\mathcal{S}$ accordingly computes $\rho = C_0 P_2^v$ and sends request $(\text{Pay}, \rho, v, \mathrm{U}')$ to $\mathcal{F}$. Let $\mathfrak{p}_0 = (v, C_1, C_0, \mathcal{E}_0, \sigma_0, \Gamma_0, \mathrm{hist}_0)$ be the payment from U to U$'$ in an execution of prot. In the following, we demonstrate that if this payment is rejected by $\mathcal{F}$, then its counterpart executed using prot will also be rejected by U$'$. There are two cases to consider.

U$'$ **is honest.** If U$'$ accepts $\mathfrak{p}_0$, then $\mathcal{S}$ (simulating U$'$) sends Accept to $\mathcal{F}$, and then instructs it to continue its execution. If $\mathcal{F}$ accepts the payment then $\mathcal{S}$ receives ρ' from $\mathcal{F}$ and adds entry $\mathbb{R}[(C_0, C_1)] \leftarrow (\rho, \rho')$ and entry $\mathbb{C}[(\rho, \rho')] \leftarrow (C_0, C_1)$. Otherwise, $\mathcal{S}$ does nothing. We recall that $\mathcal{F}$ accepts the payment if and only if U is registered, ρ identifies a previously-withdrawn token with value v, and that token belongs to the payer U. Otherwise, $\mathcal{F}$ rejects. Assume that $\mathcal{F}$ rejects whereas U$'$ accepts when executing prot. Four cases arise.
a) $\mathcal{F}$ rejects because tokens$[\rho] \leftarrow \perp$. That is, there was no valid call Withdraw that matches ρ. This only happens if U is able to generate a valid zero-knowledge proof of knowledge Σ of a Pointcheval-Sanders signature ψ on some triple (id, v, s). Given the extractability property of the zero-knowledge proofs, $\mathcal{S}$ extracts the signature ψ and the triple (id, v, s). We show next that this breaks the existential unforgeability of Pointcheval-Sanders signatures if Σ is a sound zero-knowledge proof. We recall that when corrupt users engage in a withdrawal by executing prot, $\mathcal{S}$ issues Withdraw calls whereby ρ is computed as $P_0^{\mathrm{id}} P_1^s P_2^v$, where id is the user's identifier, v the value of the token, and s is its randomness. Therefore, if no token matching ρ is found, then this implies that there was no withdrawal request in the real world that matches the triple (id, v, s) committed in ρ. That is, the central bank did not compute the corresponding blind signature. Thus, it suffices to output $(\psi, \mathrm{id}, v, s)$ to break the existential forgery of Pointcheval-Sanders signatures. Given the security of Pointcheval-Sanders signatures, this event will never happen.
b) $\mathcal{F}$ rejects because tokens$[\rho] \leftarrow \langle \mathrm{U}, v^* \rangle$ with $v \neq v^*$. Notice that by construction $\rho = P_0^{\mathrm{id}^*} P_1^{s^*} P_2^{v^*}$ (from previous Withdraw call), but also $\rho = C_0 P_2^v$. Given the soundness of the zero-knowledge proof Σ, we have $C_0 = P_0^{\mathrm{id}'} P_1^{s'}$, and hence, $\rho = P_0^{\mathrm{id}'} P_1^{s'} P_2^v$. If $v \neq v^*$, then one breaks the binding property of Pedersen commitments by outputting triples $(\mathrm{id}^*, s^*, v^*)$ and (id', s', v) (the pair (id', s') is extracted using the extractability property of Σ).
c) $\mathcal{F}$ rejects because tokens$[\rho] \leftarrow \langle \mathrm{U}^*, v^* \rangle$ with $\mathrm{U}^* \neq \mathrm{U}$. We show now that if this happens, then one breaks the existential unforgeability of BBS+ under the soundness of the signature of knowledge σ_0. If U* is corrupt, then by construction $\rho = P_0^{\mathrm{id}^*} P_1^{s^*} P_2^{v^*}$, but also $\rho = P^{\mathrm{id}'} P_1^{s'} P_2^v$. If $\mathrm{id}' \neq \mathrm{id}^*$, then this breaks the binding property of Pedersen commitments. If U* is honest, then the only way U is able to know C_0 is because U* used it in a previous payment that U has observed. Otherwise, U was able to guess C_0. The probability of this event is negligible. Therefore, C_0 is of the form $P_0^{r_0} P_1^{s_0}$. If U* is honest, then we recall that C_0 is computed as $P_0^{r_0} P_1^s$. If U$'$ accepts $\mathfrak{p}_0$, then this implies that σ_0 is a valid signature of knowledge that proves that the initiator of the payment knows a pair (r_0, sk) and a valid BBS+ signature ϕ on it, under the public key of the registration authority. Given the soundness of the signature of knowledge σ_0 and using the extractability property of the signature of knowledge σ_0, one can extract the valid BBS+ signature ϕ and the pair (r_0, sk). Given that r_0 is selected randomly by $\mathcal{S}$ during the previous game, the probability that the registration

authority signed it in one of the registration requests is negligible. Therefore, to break the existential unforgeability of BBS+, one outputs $(\phi, (\mathrm{id}, \mathrm{sk}))$.

d) $\mathcal{F}$ rejects because U is not registered. This will occur in an execution of prot only if U is able to produce a signature of knowledge σ_0 of BBS+ signature that shows that $C_0 = P_0^{\mathrm{id}} P_1^{s}$ and that there exists a valid BBS+ signature on pair $(\mathrm{id}, \mathrm{sk})$ under the public key of the registration authority that the latter did not produce. Given the extractability of σ_0, one can retrieve a BBS+ forgery.

U$'$ **is corrupt.** If U$'$ accepts $\mathfrak{p}_0$, then $\mathcal{S}$ (simulating U$'$) sends Accept to $\mathcal{F}$, and then instructs it to continue its execution. During this execution, $\mathcal{S}$ sends $\rho' = C_1 P_2^{v}$ to $\mathcal{F}$. Finally, $\mathcal{S}$ adds entry $\mathbb{R}[(C_0, C_1)] \leftarrow (\rho, \rho')$ and entry $\mathbb{C}[(\rho, \rho')] \leftarrow (C_0, C_1)$.

We now consider the case where the corrupt payer U sending a payment to a payee U$'$ and this payment is the i^{th} payment after a withdrawal. Let $\mathfrak{p}_i = (v, C_{i+1}, C_i, \mathcal{E}_i, \sigma_i, \Gamma_i, \mathrm{hist}_i)$ be this payment, and assume that U$'$ accepts $\mathfrak{p}_i$. $\mathcal{S}$ first checks if there exists an entry $\mathbb{R}[(C_0, ..., C_i)] \leftarrow (\rho_0, ..., \rho_i)$. If no such an entry exists, then this implies that there are payments prior to the current payments that happened between two corrupt users, and for which $\mathcal{F}$ was not called. $\mathcal{S}$ next finds $0 \leq j < i$ such that entry $\mathbb{R}[(C_0, ..., C_j)] \leftarrow (\rho_0, ..., \rho_j)$ is recorded but entry $\mathbb{R}[(C_0, ..., C_{j+1})] \leftarrow (\rho_0, ..., \rho_{j+1})$ is not. $\mathcal{S}$ then uses the extractability property of zero-knowledge proof Γ_j to retrieve the identifiers of U_j and U_{j+1}, and issues an Pay request on behalf of U_j for payee U_{j+1} following the description above. Note that in this case, U_{j+1} is corrupt, otherwise, the payment would have been recorded. $\mathcal{S}$ then adds entry $\mathbb{R}[(C_0, ..., C_{j+1})] \leftarrow (\rho_0, ..., \rho_{j+1})$, and repeats the process, until $j = i - 1$. Now entry $\mathbb{R}[(C_0, ..., C_i)] \leftarrow (\rho_0, ..., \rho_i)$ must be stored, and we accordingly consider two cases.

U$'$ **is honest.** If U$'$ accepts $\mathfrak{p}_i$, then $\mathcal{S}$ (simulating U$'$) sends Accept to $\mathcal{F}$, and then instructs it to continue its execution. If $\mathcal{F}$ accepts the payment then $\mathcal{S}$ receives ρ' from $\mathcal{F}$, and adds entry $\mathbb{R}[(C_0, ..., C_{i+1})] \leftarrow (\rho_0, ...\rho_i, \rho')$ and entry $\mathbb{C}[(\rho_0, ..., \rho_i, \rho')] \leftarrow (C_0, ..., C_{i+1})$. Otherwise, $\mathcal{S}$ does nothing. We recall that $\mathcal{F}$ accepts the payment if and only if $(\rho_0, ..., \rho_i)$ identifies a token that has value v, and that token belongs to the payer U. That is, it corresponds to a series of valid payments that were recorded by $\mathcal{F}$. Otherwise, $\mathcal{F}$ rejects. Assume that $\mathcal{F}$ rejects whereas U$'$ accepts when executing prot. Four cases arise.

a) $\mathcal{F}$ rejects because tokens$[(\rho_0, ..., \rho_i)] \leftarrow \langle \mathrm{U}^*, v^* \rangle$ and $v^* \neq v$. We can show that this breaks the binding property of Pedersen commitments. That is, by construction $\rho_i = P_0^{\mathrm{id}^*} P_1^{s^*} P_2^{v^*}$, but also $\rho_i = P_0^{\mathrm{id}} P_1^{s} P_2^{v}$ (see $\mathfrak{p}_0$, U$'$ honest, **b)**).

b) $\mathcal{F}$ rejects because tokens$[(\rho_0, ..., \rho_i)] \leftarrow \langle \mathrm{U}^*, v \rangle$ and $\mathrm{U}^* \neq \mathrm{U}$. We use a similar argument to the one depicted previously (see $\mathfrak{p}_0$, U$'$ honest, **c)**) to prove that U is able to break the existential unforgeability of BBS+.

c) $\mathcal{F}$ rejects because tokens$[(\rho_0, ..., \rho_i)] \leftarrow \perp$. This only occurs if ρ_0 does not identify a token that was withdrawn (see $\mathfrak{p}_0$, U$'$ honest, **a)**) or there exists $0 < j \leq i$ such that $(\rho_0, ..., \rho_j)$ identifies a payment that was previously rejected by $\mathcal{F}$, but it is now accepted by U$'$. For the latter case, we distinguish between the following. i) $\mathcal{F}$ rejects because tokens$[(\rho_0, ..., \rho_j)] \leftarrow \langle \mathrm{U}^*, v^* \rangle$ and $v^* \neq v$: we use the same arguments as in **a)**. ii) $\mathcal{F}$ rejects because tokens$[(\rho_0, ..., \rho_i)] \leftarrow \langle \mathrm{U}^*, v \rangle$

and $\mathrm{U}^* \neq \mathrm{U}$: we use the same argument as in **b)**. iii) $\mathcal{F}$ rejects because the payee U_{j+1} was not registered at time of receiving payment $\mathfrak{p}_j$: if this is the case, then U' will also reject $\mathfrak{p}_i$. Note that when U_{j+1} registers, she will be assigned a random identifier. If $\mathfrak{p}_i$ is accepted by U', then this implies that either U_{j+1} successfully guessed her random identifier before registration (i.e., at time of her accepting the payment $\mathfrak{p}_j$) or she was able to forge a BBS+ signature on the identifier she used to receive $\mathfrak{p}_j$.
d) $\mathcal{F}$ rejects because U is not registered. Similar argument to above (see $\mathfrak{p}_0$, U' honest, **d)**).

U' **is corrupt.** If U' accepts $\mathfrak{p}_i$, then $\mathcal{S}$ (simulating U') sends ACCEPT to $\mathcal{F}$, and then instructs it to continue its execution. During this execution, $\mathcal{S}$ sends $\rho' = C_{i+1}P_2^v$ to $\mathcal{F}$. Finally, $\mathcal{S}$ adds entry $\mathbb{R}[(C_0, ..., C_{i+1})] \leftarrow (\rho_0..., \rho_i, \rho')$ and $\mathbb{C}[(\rho_0, ..., \rho_i, \rho')] \leftarrow (C_0, ..., C_{i+1})$.

Since $\mathcal{F}$ verifies a payment only when the payee sends ACCEPT and since the probability that an honest payee executing prot accepts a payment that $\mathcal{F}$ subsequently rejects is negligible, this game is indistinguishable from Game 8.

Game 10. Functionality $\mathcal{F}$ now handles DEPOSIT queries from intermediaries who send requests (DEPOSIT, $\boldsymbol{\rho}, v, \mathrm{I}, \mathrm{U}$) on behalf of honest users. An honest user will have only accepted a payment with correctly verifying signatures and zero-knowledge proofs of knowledge. However, they are unable to detect double-spending while offline, and double-spending must not lead to an increase in total reserves, even if an honest user is making the deposit. There are two cases where the functionality $\mathcal{F}$ would reject the deposit: the first is if TOKENS$[\boldsymbol{\rho}] \neq \langle \mathrm{U}, v \rangle$, and the second is if a token with the same starting entry $\boldsymbol{\rho}[0]$ has already been deposited. An honest user will not try to deposit a token which does not belong to them, and the handling of a repeated deposit will happen identically as in prot.

An honest user U deposits a token of value v through their intermediary I invoking $\mathcal{F}$ with (DEPOSIT, $\boldsymbol{\rho}, v, \mathrm{I}, \mathrm{U}$). $\mathcal{S}$ uses $\boldsymbol{\rho}$ to look up commitments $\mathbb{C}[(\rho_0, \ldots, \rho_n)] \rightarrow (C_0, \ldots, C_n)$. $\mathcal{S}$ can then produce a simulated deposit $(\tau_n, \sigma_n, \mathrm{hist}_n)$ with $\tau_n = (v, C_n) = (v, P_0^{\mathrm{id}_n}, P_1^{s_n})$, with the zero-knowledge proofs are simulated and the ciphertexts random Elgamal ciphertexts under the auditor's public key.

We next show that an honest user depositing a double-spent token will not alow the double-spending to go undetected. As each ρ_i and C_i are computed deterministically from one another, we have that each ρ_i maps to one fixed C_i, as a result of the binding property of Pedersen commitments. During withdrawal, a Pointcheval Sanders signature is issued over $P_0^{\mathrm{id}_0} P_1^v P_2^{s_0}$, only if this value is unique, which fixes the value of the commitment. As this commitment is then signed, the value of the commitment and hence id_0, v, s_0 are also unable to be altered due to unforgeability of Pointcheval Sanders signatures. This shows that a corrupt user is unable to change the initial commitment identifying a token, and therefore each double spending will be identified upon deposit. Hence this game is indistinguishable from Game 9.

Game 11. Now $\mathcal{F}$ also processes DEPOSIT requests for corrupt users. When a corrupt user U sends the deposit request to their intermediary bank I, I calls $\mathcal{F}$

with $(\textsc{Deposit}, \boldsymbol{\rho}, v, \mathrm{U})$. Again there are two distinct reasons which may lead to rejecting the deposit, either $\textsc{tokens}[\boldsymbol{\rho}] \neq \langle \mathrm{U}, v\rangle$, or a token beginning $\boldsymbol{\rho}[0]$ has been previously deposited. In this case, as the depositing user is corrupt, the case where $\textsc{tokens}[\boldsymbol{\rho}] \neq \langle \mathrm{U}, v\rangle$ can occur. We can again split this into four cases: **a)** $\textsc{tokens}[\boldsymbol{\rho}] \leftarrow \perp$; **b)** $\textsc{tokens}[\boldsymbol{\rho}] \neq \langle \mathrm{U}^*, v^*\rangle$ with $v^* \neq v$; **c)** $\textsc{tokens}[\boldsymbol{\rho}] \neq \langle \mathrm{U}^*, v^*\rangle$ with $\mathrm{U}^* \neq \mathrm{U}$; **d)** U is not registered.

We assume here that the deposit is the first action after a withdrawal. The cases are handled similarly to calls to Pay with a corrupt U and honest U′.
a) $\mathcal{F}$ rejects because $\textsc{tokens}[\boldsymbol{\rho}] = \perp$. This indicates that there is no valid call to Withdraw with $\boldsymbol{\rho}[0]$. This is only possible if U is able to generate a valid zero-knowledge proof of knowledge Σ for a Pointcheval-Sanders blind signature ψ. Assuming Σ is a sound zero-knowledge proof of knowledge, one can follow a similar logic to the one depicted in Game 9 (see argument for $\mathfrak{p}_0$, U′ honest, **a)**) to extract from Σ a signature ψ and a triple of messages (id, v, s), and then output as a Pointcheval-Sanders forgery $(\psi, \mathrm{id}, v, s))$
b) $\mathcal{F}$ rejects because $\textsc{tokens}[\boldsymbol{\rho}] \neq \langle \mathrm{U}^*, v^*\rangle$ with $v^* \neq v$. Here we need that both $\rho = P_0^{\mathrm{id}^*} P_1^{s^*} P_2^{v^*}$, and $\rho = C_0 P_2^v = P_0^{\mathrm{id}'} P_1^{s'} P_2^v$ with C_0 and id', s' extracted from Σ. If $v^* \neq v$, then one can break the binding property of Pedersen commitments by outputting triples $(\mathrm{id}^*, s^*, v^*)$ and (id', s', v). This follows the same argument as Game 9 for $\mathfrak{p}_0$ with U′ honest, **b)**.
c) $\mathcal{F}$ rejects because $\textsc{tokens}[\boldsymbol{\rho}] \neq \langle \mathrm{U}^*, v^*\rangle$ with $\mathrm{U}^* \neq \mathrm{U}$. The argument is again similar to the one in Game 9 for $\mathfrak{p}_0$ with U′ honest, **c)**. If U^* is corrupt, then as above we have $\rho = P_0^{\mathrm{id}^*} P_1^{s^*} P_2^{v^*}$, and $\rho = P_0^{\mathrm{id}'} P_1^{s'} P_2^{v}$, which would break the binding property of Pedersen commitments. Otherwise U has been able to produce a valid signature of knowledge σ_0, from which one can extract the correctly verifying BBS+ signature ϕ and r_0, sk, enabling one to output a BBS+ signature forgery $(\phi, (\mathrm{id}, \mathrm{sk}))$.
d) $\mathcal{F}$ rejects because U is unregistered. Similarly to in Game 9, for $\mathfrak{p}_0$, U′ honest, **d)**, this requires U to have produced a correctly verifying signature of knowledge σ_0 of a BBS+ signature. Therefore from the soundness of the signature of knowledge, we can extract a valid BBS+ signature ϕ along with id and sk, and so output forgery $(\phi, (\mathrm{id}, \mathrm{sk}))$.

Next we consider the case where the token has been transferred multiple times between withdrawal and deposit. If the payment does not align with the latest transaction and references an earlier one, the extractability of the zero-knowledge proof is used to identify the corrupt participants U_j and U_{j+1}. For each unrecorded payment with identifier $\boldsymbol{\rho}_j$, $(\textsc{Pay}, \boldsymbol{\rho}_j, v, \mathrm{U}_{j+1})$ is called to record the payment within $\mathcal{S}$. Here, the cases **b)**, **c)**, and **d)** are handled very similarly to the case $i = 0$ above. Case **a)** is handled as follows.
a) $\mathcal{F}$ rejects because $\textsc{tokens}[(\rho_0, \dots, \rho_i)] \leftarrow \perp$. We consider two cases. Either there was no valid call to Withdraw with $\boldsymbol{\rho}[0]$, in which case this being accepted in prot would enable a forgery of a Pointcheval-Sanders signature (see Game 9, U′ honest, **a)**). Otherwise, there exists $0 < j \leq i$ such that $(\rho_0, \dots, \rho_j)$ corresponds to a payment rejected by $\mathcal{F}$. We have the same as above that this may lead to a rejection, namely: i) $\textsc{tokens}[(\rho_0, \dots, \rho_j)] \neq \langle \mathrm{U}^*, v^*\rangle$ with $v^* \neq v$,

ii) $\textsc{tokens}[(\rho_0, \ldots, \rho_j)] \neq \langle \mathrm{U}^*, v^* \rangle$ with $\mathrm{U}^* \neq \mathrm{U}$, iii) U_{j+1} was not registered when receiving the payment. Each case is handled as explained in Game 9.

Finally, if the deposit corresponds to a token starting with $\boldsymbol{\rho}[0]$ such that another chain starting from the same $\boldsymbol{\rho}[0]$ has been deposited before, the logic for handling this in $\mathcal{F}$ is identical to that in prot. We can show as in Game 10 that a corrupt user is unable to change the initial commitment identifying a token, as $\boldsymbol{\rho}[0]$ maps to one fixed C_0 value due to the binding of the Pedersen commitment, and this commitment is signed with a Pointcheval Sanders signature meaning that the commitment value and therefore values id_0, v, s_0 are unable to be altered due to unforgeability of Pointcheval Sanders signatures. Hence this game and Game 10 are indistinguishable.

Game 12. Functionality $\mathcal{F}$ also addresses the $\textsc{Audit}$ queries, receiving $(\textsc{Audit}, \rho)$ from A. We must ensure that the outputs $(\textsc{Audit}, \mathbf{DS}, \textsc{fin})$ given to A and CB are indistinguishable in the real protocol and ideal functionality. Note that if a token is double-spent multiple times in its history, each two deposits of the token will identify one double spender: the user directly before the point where those two tokens' histories diverge. First note that the argument for randomness ρ being fully determined by C_i and vice versa, and unchangeable under the binding of Pedersen commitments and unforgeability of Pointcheval Sanders signatures, as described in Game 10, also holds here.

The outputs of $\mathcal{F}$ and prot could then only be different in the following cases: **a)** prot returns a different registered user; or **b)** prot returns an unregistered user.

In prot, the double-spending user is identified by decrypting $\mathcal{E}_i$. For **a)** to occur, we would need $\mathcal{E}_i$ to decrypt in prot to id^*, different to id associated with the token $\boldsymbol{\rho}$ in $\mathcal{F}$. First we show that a corrupt user cannot change the value of C_i once the payment has been received. This is because σ_{i-1} is a signature of knowledge over (v, C_{i-1}, C_i). This means that due to the soundness of σ_{i-1}, the value of C_i is fixed. As $\mathcal{E}_i$ is formed $(P^{\rho_i}, P^{\mathrm{id}_i} A^{\rho_i})$ and $\mathit{\Gamma}_i$ proves that $C_i = P_0^{\mathrm{id}_i} P_1^{s_i}$ and $\mathcal{E}_i = (P^{\rho_i}, P^{\mathrm{id}_i} A^{\rho_i})$, this would mean that either we have $P_0^{\mathrm{id}} P_1^{s} = P_0^{\mathrm{id}^*} P_1^{s^*}$ with $\mathrm{id} \neq \mathrm{id}^*$, breaking the binding of Pedersen commitments, or we have C_i and $\mathcal{E}_i$ with different id, breaking the soundness of $\mathit{\Gamma}_i$.

A payee cannot be framed by a corrupt payer picking $C_i = C'_j$ as the payee picks the value C_i themselves. As C_i is signed by the payer in σ_{i-1}, the payee cannot equivocate once C_i is chosen. This ensures that $C_i = C'_j$ if and only if the chooser of C_i is the double spender.

For **b)** to occur, we would need that $\mathcal{E}_i$ decrypt to id of an unregistered user. In this case, assuming soundness of signature of knowledge σ_i, we can extract a valid BBS+ signature forgery for unregistered user $(\phi, (\mathrm{id}, \mathrm{sk}))$.

The probability of these events occurring is negligible, and so Game 12 is indistinguishable from Game 11. This concludes our proof.