Towards a Two-Tier Hierarchical Infrastructure: An Offline Payment System for Central Bank Digital Currencies

Mihai Christodorescu[*], Wanyun Catherine Gu[**], Ranjit Kumaresan[*], Mohsen Minaei[*], Mustafa Ozdayi[*], Benjamin Price[**], Srinivasan Raghuraman[*], Muhammad Saad[*], Cuy Sheffield[**], Minghua Xu[*], and Mahdi Zamani[*]

[*] *Visa Research, Palo Alto, CA*
[**] *Visa Crypto Product, Palo Alto, CA*

Abstract

Digital payments traditionally rely on online communications with several intermediaries such as banks, payment networks, and payment processors in order to authorize and process payment transactions. While these communication networks are designed to be highly available with continuous uptime, there may be times when an end-user experiences little or no access to network connectivity.

The growing interest in digital forms of payments has led central banks around the world to explore the possibility of issuing a new type of central-bank money, known as *central bank digital currency (CBDC)*. To facilitate the secure issuance and transfer of CBDC, we envision a CBDC design under a two-tier *hierarchical trust infrastructure*, which is implemented using public-key cryptography with the central bank as the root certificate authority for generating digital signatures, and other financial institutions as intermediate certificate authorities. One important design feature for CBDC that can be developed under this hierarchical trust infrastructure is an “offline” capability to create secure point-to-point offline payments through the use of authorized hardware. An offline capability for CBDC as digital cash can create a resilient payment system for consumers and businesses to transact in any situation.

In this paper, we propose an *offline payment system (OPS)* protocol for CBDC that allows a user to make digital payments to another user while both users are temporarily offline and unable to connect to payment intermediaries (or even the Internet). OPS can be used to *instantly* complete a transaction involving any form of digital currency over a point-to-point channel without communicating with any payment intermediary, achieving virtually unbounded throughput and real-time transaction latency. One needs to ensure funds cannot be double-spent during offline payments as no trusted intermediary is present in the payment loop to protect against replay of payment transactions. Our OPS protocol prevents double-spending by relying on digital signatures generated by *trusted execution environments (TEEs)* which are already available on most computer devices, including smartphones and tablets. While a TEE brings the primary point of trust to an offline device, an OPS system requires several cryptographic protocols to enable the secure exchange of funds between multiple TEE-enabled devices, and hence a reliable financial ecosystem that can securely support CBDC at scale.

Contents

1 Introduction

Digital payments today represent an account-based system of debiting and crediting accounts operated by the financial institutions, where the ownership of a payment account is tied to a user's public identity. With the emergence of distributed ledger technology, there has been growing interest in a new form of token-based digital payment, where the token itself represents the medium of exchange or money, and the ownership of the token is determined by a user's access to a private cryptographic key that provides access to the user's *digital wallet*. Access to these wallets are typically facilitated by entities, known as *wallet providers*, that offer secure access to the cryptographic keys as well as some banking and other financial capabilities [1–3].

The growing interest in token-based payments have led central banks around the world to explore the possibility of issuing a new type of central-bank money, known as *central bank digital currency (CBDC)*. Some of these central banks have proposed designs that would issue this new money in the form of cryptographic tokens backed directly by central bank reserves to enable consumers and businesses to make payments in the form of "digital cash" [4–6].

In a CBDC model, the money in transit should remain the liability of its trusted issuer (e.g., a central bank in the case of CBDC), meaning that its value is always guaranteed by the issuer as long as the recipient can easily verify the authenticity of the money. This ensures that (1) the money was issued properly according to a specific set of rules (aka, a monetary policy that has parallels to the existing financial system) set by the issuer, and (2) the money maintained its value in transit (i.e., was neither double spent nor counterfeited). In the digital world, this can be implemented using public-key cryptography, where the money in transit carries a digital signature that can only be generated directly by the central bank or indirectly by one of the central bank's certified delegates. Any recipient of the money can then simply authenticate it by verifying the signature against the public key of the central bank and/or the certificate of its trusted delegate. Public-key cryptography could offer significant advantages for security and compliance over cash.

We envision a two-tier *hierarchical trust infrastructure* delivered through certified delegation, which allows the central bank to outsource the complexity of managing digital certificates for CBDC tokens to a set of potentially regulated, permissioned entities that derive their authority from the central bank, through a hierarchy of digital certificates originated from the central back at the root. This hierarchical trust design resembles the hierarchy of *certificate authorities (CAs)* in a *public-key infrastructure (PKI)* [7] that plays a vital role in enabling the secure transfer of information over the Internet. A PKI model for CBDC can significantly facilitate the secure issuance and transfer of CBDC funds with the central bank serving as the root CA and supervised financial institutions (FIs) serving as intermediate CAs under regulatory oversight. These intermediate CAs have two roles: (1) Vetting wallet providers based on regulatory compliance; (2) Issuing digital certificates to vetted wallet providers to facilitate CBDC payments securely.

The core advantage of the two-tier model is that it decouples the certificate infrastructure (Tier 1) from the critical latency path of CBDC payments (Tier 2), allowing wallet providers such as banks and other FIs to securely process CBDC payments at a high scale without imposing extra overhead on the highly-protected PKI nodes. This is a particularly relevant question at the heart of the current CBDC debate.

In the past year, several central banks have begun to research and ask how offline CBDC payments could occur [4, 8–10]. **An important feature that can be developed under the hierarchical trust infrastructure for CBDC is an "offline" capability to create secure point-to-point offline payments using authorized hardware.** An offline protocol can be a potential feature of CBDC by bringing the primary point of trust to an offline device under this infrastructure. CBDC as digital cash can move instantly across multiple payment rails and condition without necessarily needing to directly involve any intermediary during transfers. For example, if the sender and the recipient of a payment have relationships with different wallet providers, they should still be able to transact with each other instantly in an ad-hoc, point-to-point fashion without communicating with their wallet providers. This enables a significantly higher throughput as payments can still happen under congested network conditions. Moreover, the design provides a higher level of privacy to the clients by avoiding the need to share unnecessary payment information with the intermediaries.

One technical challenge in creating secure offline payments is to protect the system from financial crimes and to avoid exposing either the buyer, seller, or the central bank to the risk that the payment may not ultimately be settled [4]. Existing card payment networks such as Visa and Mastercard can provide some form of offline payments for situations where the acceptance device (e.g., a card terminal) cannot connect to payment providers for authorization in real time [11,12]. Payments without issuer authorization require the merchant to bear some counterparty risk because the payer may not actually have the necessary funds to fulfill the transaction [13].

> *Would it be possible to allow offline payments between two parties without exposing counterparty risk on either party?*

Today, most mobile devices (e.g., smartphones and tablets) are equipped with *secure hardware* to store keys and other sensitive material that can only be accessed through strong user authentication measures (e.g., biometrics). It has been shown that compromising these hardware-protected mobile devices without help from their manufacturers is difficult [14]. This can potentially make mobile devices a viable option to store the user's CBDC funds and to issue offline payments using hardware-protected credentials provisioned by the central bank or one of its delegates. As long as the secure hardware remains secure, (1) the keys for signing offline payments are protected from malicious access, and (2) the same funds cannot be spent offline more than once (i.e., no double spending).

While secure hardware provides a simple and efficient medium for delegation of trust in the digital setting [14], the possibility of device compromise would not only expose the involved users to the risk of funds loss but also could, at a much larger scale, jeopardize the functionality of the entire CBDC ecosystem. In a recent study, Allen et al. [15] identify three main challenges in the use of secure hardware. First, there is a strong economic incentive for users to compromise their secure hardware in order to counterfeit CBDC funds. Second, compromising one device could allow a user to double spend funds an unlimited number of times (i.e., by default there is no *graceful degradation*). And third, a user's funds could be totally lost in case of device loss (due to, e.g., damage or failure).

We observe that these challenges are not exclusive to digital offline payments and are applicable to virtually any form of offline payments, including physical cash. Therefore, we envision that measures similar to those used for physical cash could be employed to protect the security of CBDC in an offline setting while still maintaining the practical benefits of offline payments. In this paper, we initiate the formal study of offline digital payments and propose a system that provides the basic functionalities for offline payments assuming secure hardware cannot be compromised. In subsequent work, we will explore extended techniques for offline payments to alleviate the above challenges by addressing economic incentives, graceful degradation, and funds recovery.

1.1 Our Contribution

In this paper, we initiate the study of offline digital payments by defining the notion of an *offline payment system (OPS)* that allows a user (e.g., a customer) to make digital payments in CBDCs to another user (e.g., a merchant) while both users are temporarily offline from payment intermediaries (or even the Internet). We then construct the first OPS protocol that allows point-to-point authorization of offline payments using open source technology and public key infrastructure to significantly reduce the overhead of onboarding new participants compared to existing digital payment systems. Once provisioned, OPS wallets can securely sign and transmit transaction messages directly with each other over any communication channel they prefer without requiring an intermediary to authorize and settle it. Recipients can submit signed, offline payment messages to an authorized wallet provider with guaranteed settlement of those transactions in order to withdraw funds from the offline payment system.

1.2 Overview of Our Solution

Consider two *clients* A and B who hold online accounts with a *server* S. We assume an account maintains information about the amount of money (aka, balance) that the client holds at S. We assume that the server is a digital wallet provider that has already obtained a digital certificate from the central bank through the hierarchical trust infrastructure for CBDC, as described in Section 1. This certificate can be used to attest to the clients that S is a trusted entity. In the rest of this paper, we describe the offline payment protocol that happens between the clients and the trusted server.

We assume a client's account maintains information about the amount of money (aka, balance) that the client holds at S. The goal of our OPS protocol is to allow client A (aka, the *sender*) to pay client B (aka, the *receiver*) an amount of money denoted by x from A's account with S without either client communicating with S during the payment. We assume A (or any other client who wishes to send money) owns a secure device D_A that can securely store data and execute code via a *trusted execution environment (TEE).* However, we do not require B (or any other client who only wants to receive money) to own a secure device. In the following, we first describe our TEE model and the main components of our protocol. Next, we briefly describe the main OPS protocols to set up the clients and perform offline payments.

TEE Model. A TEE is a software stack stored on a read-only memory within a secure device. This software stack should adhere to privacy, security, and high software development standards which are already adopted by most secure devices today. The stack consists of a set of resources to access the secure device, a *trusted operating system (TOS)* that provides developer access to the underlying secure device, and one or more *trusted applications (TA)* that implement application-specific functionalities to be executed securely by the TEE (see Figure 4).

Untrusted Applications. Every client (with or without TEE) has an application-specific *untrusted application (UA)* that resides in the "untrusted" region of the device and thus could be malicious. A benign UA provides user-facing functionalities to receive, verify, and store payments on the device as well as to submit the payments to the server whenever the client goes online. In case of a TEE-enabled client, the UA also interacts with the TA to provide wallet operations to the user, such as creating new offline payments, adding/collecting received payments into the secure wallet, etc. (see Section 2.3 for more details).

OPS Components. Our OPS consists of the following four main components:

- *OPS Server TA:* Deployed on the server and provides the functionalities to register and set up client devices and manage client accounts.
- *OPS Sender UA:* Deployed on sender's device and provides the OPS user interface to create offline payments by interacting with the OPS TA, and to interact with the server to register the UA and the TEE.
- *OPS Receiver UA:* Deployed on the receiver's device and provides the OPS user interface to receive and verify offline payments, and to interact with S to register the UA and claim offline payments. This UA does not interact with any TEE. If the receiver is also wishing to receive money from another client, then she needs to deploy the OPS Receiver UA on her device.
- *OPS TA:* Deployed on the sender's secure device (within TEE) and provides OPS-specific functionalities to securely access the secure device and manage the client's offline balance. We denote the TA deployed on client A's device by T_A.

Setup Protocol. Our OPS protocol requires both clients to register with S during a one-time, online setup to establish asymmetric cryptographic keys that are later used to issue and verify offline payments. The online setup also allows A to initialize her TEE jointly by S and D_A's manufacturer. The TEE setup consists of three phases: (1) Remote attestation to allow either the manufacturer or S to remotely verify the validity of the TEE stack; (2) TA provisioning to allow either the manufacturer or S to securely deploy a TA inside TEE; and (3) TA registration to allow the TA to establish a signing key pair, register it with S, and obtain a certificate attesting to the validity of the key pair. See Section 3.1 for the complete description of the setup protocol.

Deposit Protocol. Client A needs to initially deposit funds into her secure device when she is online to be able to send offline payments later. Namely, A requests server S to deposit an amount of x money from her online balance stored at S into her offline balance stored in T_A. The server responds with a signature on showing that x was deducted from A's online balance. The client TA verifies the signature with the server's public verification key and adds x to the offline balance stored in T_A. See Section 3.4 for details.

Offline Payment Protocol. An offline payment is initiated by the receiver B who sends a payment request to A, including B's certificate in the request. Upon receiving the request, A invokes T_A.`Pay` to securely deduct the payment amount from T_A's balance and create a signed payment message P containing the payment amount and the certificates of both clients. A sends P to B who verifies A's signature and her certificate, and checks that the payment contains B's certificate as the recipient. If all checks pass, B accepts the payment and stores P on his device. Note that by deducting the payment amount from A's balance (which is stored on the TEE storage), the TEE prevents double spending of that amount. See Section 3.5 for details.

Claim Protocol. If B wants to add the amount of P that he received offline from A to his online balance stored at S, he can invoke the Claim protocol in which S verifies the validity of P and checks if it was not previously marked as spent using a payment log stored by S. If all checks pass, S adds the amount of P to B's online balance and adds P to the log. See Section 3.6 for details.

Collect Protocol. Imagine that B also has a secure device with T_B set up similar to what was described before. If B wishes to make an offline payment out of the money he previously received in P from A without going online, then he can invoke the Collect protocol to add the money in P into T_B's balance. This allows B to spend the funds offline in exactly the same way A made the offline payment P. See Section 3.6 for details.

Withdraw Protocol. If A wishes to move funds from T_A to her online balance stored at S, then she can invoke the Withdraw protocol which invokes T_A.`Withdraw` to deduct the funds from T_A and return a message signed with T_A's signing key. The client then forwards the signed withdraw message to S who adds the fund to A's online balance after verifying the signature. See Section 3.4 for details.

Replay/Rollback Protection. To protect against malicious intermediaries (such as a malicious UA) replay the messages exchanged between S and T_A as well as between A and B, each party maintains monotonically-increasing counters that are incremented after every round of communication between a pair of parties. Both S and T_A (as well as A and B) include the latest value of their counter in their signed messages so that the receiver can verify the uniqueness and ordering of all messages according to their local counter which is synchronized after every exchange.

2 Our Model

Consider a group of clients $\mathsf{A}, \mathsf{B}, \mathsf{C}, \ldots$ who can communicate with each other by exchanging messages through a communication network. We assume that a secure communication infrastructure is in place, that is, all parties may interact and send messages to each other in a secure way. In particular, this means that when a party communicates with another, the receiver of the communication will be able to ascertain the authenticity and validity of said communication.

Every client, say A, is associated with a non-negative numeric value known as its *wallet balance* (or simply *balance*) $\mathsf{bal}_{\mathsf{A}}$ indicating the amount of money possessed by A. A payment is represented in the form $P : \mathsf{A} \xrightarrow{x} \mathsf{B}$ indicating a transfer of x amount of money from client A (aka, the sender) to client B (aka, the receiver). A *payment protocol* is a protocol that processes a payment $P : \mathsf{A} \xrightarrow{x} \mathsf{B}$ by updating the clients' balances correspondingly, i.e., $\mathsf{bal}_{\mathsf{A}} = \mathsf{bal}_{\mathsf{A}} - x$ and $\mathsf{bal}_{\mathsf{B}} = \mathsf{bal}_{\mathsf{B}} + x$. A payment is called *authentic* if and only if the sender's balance at the time of payment is greater than or equal to x. A *payment system* PS consists of a network of clients who can verify the authenticity of any payment within PS through a designated authority, referred to as *server* S.

2.1 Problem Definition

An *offline payment system* (denoted by OPS) is a payment system that enables any pair of clients to pay each other while both are offline from S. More precisely, any client A is associated with an *online balance* $\mathsf{onBal}_{\mathsf{A}}$ and an *offline balance* $\mathsf{offBal}_{\mathsf{A}}$. Given a payment $P : \mathsf{A} \xrightarrow{x} \mathsf{B}$, an *online payment* is a protocol that ensures $\mathsf{onBal}_{\mathsf{A}} = \mathsf{onBal}_{\mathsf{A}} - x$ and $\mathsf{onBal}_{\mathsf{B}} = \mathsf{onBal}_{\mathsf{B}} + x$. Given a payment $P : \mathsf{A} \xrightarrow{x} \mathsf{B}$, an *offline payment* is a protocol that ensures $\mathsf{offBal}_{\mathsf{A}} = \mathsf{offBal}_{\mathsf{A}} - x$ and $\mathsf{offBal}_{\mathsf{B}} = \mathsf{offBal}_{\mathsf{B}} + x$. The following properties are enabled by OPS:

- **Offline Verifiability:** The receiver must be able to independently verify the authenticity of any payment without communicating with the server during the payment.

- **Absolute Finality:** Once a payment is complete, the receiver must be instantly guaranteed to own the transferred funds.

- **Online Redeemability:** A client A must be able to convert any amount $y \leq \mathsf{offBal}_{\mathsf{A}}$ from their offline balance into their online balance, i.e., $\mathsf{offBal}_{\mathsf{A}} = \mathsf{offBal}_{\mathsf{A}} - y$ and $\mathsf{onBal}_{\mathsf{A}} = \mathsf{onBal}_{\mathsf{A}} + y$ are executed atomically, and vice versa.

- **Offline Transitivity:** After receiving an offline payment, the receiver must be able to spend the payment amount (or a portion of it) in the same offline session, i.e., without requiring to go online to redeem the payment amount and then spend it.

- **Security:** OPS is secure if it has the following properties:
  - *No Double Spending:* No malicious client (or a coalition of them) can spend the same money more than once.
  - *Wallet Security:* No malicious client (or a coalition of them) can spend/remove money from an honest client's wallet without her permission.
  - *Supply Conservation:* The total supply of money in the system always stays the same, i.e., a client can only add/remove money to/from the system via the deposit/withdraw functionalities provided by the server.

If client A wishes to spend her money offline, then she must have a TEE-enabled *secure device* $\mathsf{D_A}$ to store her offline balance $\mathsf{offBal_A}$ securely. The balance can only be modified by the OPS trusted application $\mathsf{T_A}$ stored in $\mathsf{D_A}$. The authenticity of such modifications are enabled via digital signatures generated by $\mathsf{T_A}$ using a secret key $\mathsf{sk_A}$ stored securely inside $\mathsf{D_A}$. If the client does not want to spend money offline, then she does not require any secure device.

2.2 Threat Model

We assume all parties communicate with each other via secure and authenticated communication channels. We consider a probabilistic, polynomial-time adversary who can corrupt any client in order to (1) prevent the protocol from achieving its defined properties; and (2) counterfeit money, for example, by double-spending the client's money or forging new money. A corrupt client may do so by arbitrarily tampering with and/or blocking messages exchanged between the server and the client's TEE. We assume that uncorrupt clients employ standard authentication mechanisms such as password and biometrics to prevent unauthorized access to their device in order to spend and/or erase their money without their approval. We finally assume that the server is fully trusted.

2.3 TEE Model

A TEE is an isolated execution environment with its own protected hardware resources (e.g., processor memory, and peripherals) as well as a software stack consisting of an operating system and trusted programs, known as *trusted applications (TAs)*, to access the TEE hardware resources securely [16,17]. The isolation provides strong integrity and confidentiality guarantees, where integrity ensures that unauthorized users cannot change the code of a TA or its behavior, while confidentiality guarantees that unauthorized access to private TA data is prohibited. The trusted OS provides API access for external programs, known as *untrusted applications (UAs)*, to call and execute public TA functions within the TEE while restricting external access to the rest of the TEE.

In practice, there are different TEE architectures depending on the platform they run and the hardware that provides the isolation. In this paper, we target offline payments for mobile devices; therefore, we adopt GlobalPlatform (GP) [18], a standardized TEE model adopted by ARM TrustZone technology [19,20] which itself is used in most Android smartphones today. As shown in Figure 1, the GP model provides a standardized API for UAs in the non-secure world (aka, the *rich execution environment – REE*) to interact with the isolated TAs via the trusted OS. We now describe GP's secure storage model that allows us to protect against OPS TA state rollback.

TEE Replay/Rollback Protection. Throughput this paper, we assume that the OPS TA has access to a secure storage to store its state, ensuring that the state cannot be rolled back by the adversary. The GP specification mandates the possibility to store general-purpose data and key material within a TEE with integrity, confidentiality, and atomicity[1] guarantees [21]. Typically, a *replay-protected memory block (RPMB)* partition[2] on an eMMC storage (e.g., the phone's persistent storage) is used to store TA's data securely [22, 23]. Any data written on the RPMB is protected against man-in-the-middle replay/rollback attacks using a *monotonically-increasing counter (MIC)* maintained by a dedicated hardware, known as the *RPMB engine*. The engine increments the MIC after every write to the RPMB and uses message authentication codes (MACs) to verify the validity of the write command by checking that (1) the counter was increased, and (2) the MAC that was sent by the sender (e.g., the TA) is identical to the MAC that the RPMB engine generated using its latest value of MIC. Finally, every read from RPMB is MAC-checked by

[1] Atomicity means that either the entire write operation completes successfully or no write is done.

[2] The RPMB partition is typically 4 MB in size [22].

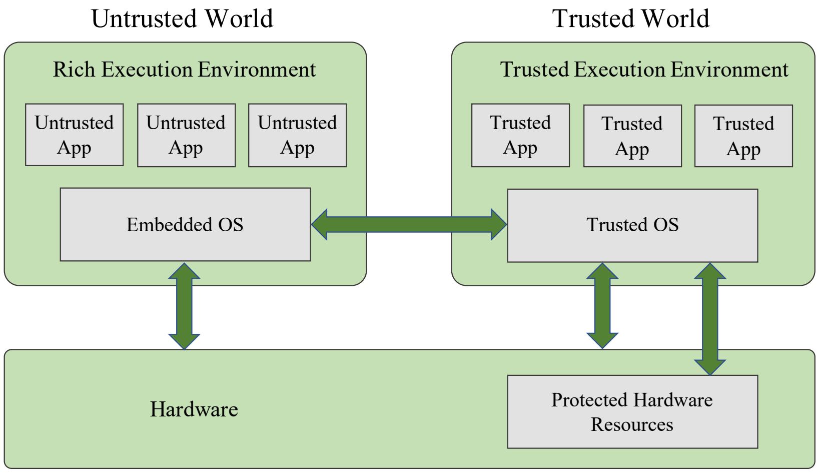

Figure 1: Our TEE Model

the reader (e.g., the TA) using the latest value of MIC maintained by the reader. For more details, we refer the reader to [22]. The GP specification also allows a secure storage to be implemented on the REE (i.e., non-secure) file system as long as suitable cryptographic protection is applied, which must be as strong as the means used to protect the TEE code and data itself [21].

3 The OPS Protocol

We now describe the OPS protocol explicitly. We break down the protocol to its components and explain each thoroughly.

3.1 Client Setup

Every client (TEE-enabled or not) needs to participate in a one-time setup protocol to register her device with the server (i.e., establish cryptographic keys and certificates) and to initialize her device's TEE stack in case of a TEE-enabled device. To register with the server, the client generates a local signing key pair denoted by $(\mathsf{vk}, \mathsf{sk})$ and submits the verification key to the server. In return, the server initializes the client's account information and returns a certificate denoted by cert to the client.

The certificate is essentially a signature by the server on the client's verification key so that the client can later prove to other entities that her device is registered with the server. The server maintains a registry (denoted by $\mathsf{S.Registry}$) of all registered clients. When a new client registers herself with the server, the verification key of the client is added to this registry. This allows the server to keep track of registered clients in the future to catch duplicate and bogus users. The server also stores the online balance (denoted by onBal) of each registered client. When a new client registers herself with the server, the server initializes an online balance of 0 for the client. The formal description of this protocol is presented in Figure 2.

Variable	**Description**	**Scope**
S.Registry	Server S's registry of valid UA certificates	S
S.onBal$_\mathsf{A}$	Client A's online balance stored at S	S
S.i$_\mathsf{A}$	Client A's index maintained by S	S
D$_\mathsf{A}$	Client A's secure device/hardware	A
T$_\mathsf{A}$	Client A's trusted application deployed on D$_\mathsf{A}$	A
$(\mathsf{sk_S}, \mathsf{vk_S})$	Server S's signing key pair	(S, Global)
$(\mathsf{sk_A}, \mathsf{vk_A})$	Client A's signing key pair	(A, Global)
$(\mathsf{T_A.sk}, \mathsf{T_A.vk})$	T$_\mathsf{A}$'s signing key pair	(T$_\mathsf{A}$, Global)
cert$_\mathsf{A}$	Certificate for client A consisting of vk$_\mathsf{A}$ and a signature on it by S certifying that vk$_\mathsf{A}$ was issued by S	Global
T$_\mathsf{A}$.cert	Certificate for T$_\mathsf{A}$ consisting of T$_\mathsf{A}$.vk and a signature on it by S certifying that T$_\mathsf{A}$.vk was issued by S	Global
T$_\mathsf{A}$.i	Client A's server index maintained by T$_\mathsf{A}$	T$_\mathsf{A}$
T$_\mathsf{A}$.j	Client A's payment index maintained by T$_\mathsf{A}$	T$_\mathsf{A}$
T$_\mathsf{A}$.bal	Client's A's offline balance	T$_\mathsf{A}$
T$_\mathsf{A}$.inPaymentLog	List of offline payments received by T$_\mathsf{A}$	A, T$_\mathsf{A}$, S
P.amount	Amount money transferred by payment P	Holder of P
P.sender	Certificate of the sender of payment P	Holder of P
P.receiver	Certificate of the receiver of payment P	Holder of P
P.index	Index of payment P	Holder of P
P.time	Time when payment P was created	Holder of P
P.type	Type of payment P (“`Basic`” or “`Conditional`”)	Holder of P
Function	**Description**	**Scope**
$\mathtt{H}(x)$	Outputs a cryptographic hash of x	Global
$\mathtt{Sign}(x, \mathsf{sk})$	Outputs a signature of x signed with signing key sk	Global
$\mathtt{SigVerify}(x, \sigma, \mathsf{vk})$	Outputs 1 iff signature σ over x using verification key vk is valid	Global
$\mathtt{CertVerify}(\mathsf{cert}, \mathsf{vk_S})$	Outputs 1 iff $\mathtt{SigVerify}(\mathsf{cert.vk}, \mathsf{cert.sig}, \mathsf{vk_S}) \overset{?}{=} 1$	Global
$\mathtt{TACertVerify}(\mathsf{cert}, \mathsf{vk_S})$	Outputs 1 iff $\mathtt{SigVerify}([\mathsf{cert.vk}, \text{“}\mathtt{TA}\text{”}], \mathsf{cert.sig}, \mathsf{vk_S}) \overset{?}{=} 1$	Global
$\mathtt{OEMVerify}(\mathsf{vk}, \mathsf{cert}, \mathsf{vk_D})$	Outputs 1 iff $\mathtt{SigVerify}([\mathsf{vk}, \text{“Secure Device”}\Vert\mathsf{M}], \mathsf{cert}, \mathsf{vk_D}) \overset{?}{=} 1$	Global
$\mathsf{T_A}.\mathtt{Deposit}(x, ...)$	Deposits x amount of money into client A's secure hardware T$_\mathsf{A}$	T$_\mathsf{A}$
$\mathsf{T_A}.\mathtt{Withdraw}(x, ...)$	Withdraws x amount of money from client A's secure hardware T$_\mathsf{A}$	T$_\mathsf{A}$
$\mathsf{T_A}.\mathtt{Pay}(x, ...)$	Debits x amount of money from T$_\mathsf{A}$ and outputs a payment P	T$_\mathsf{A}$
$\mathsf{T_A}.\mathtt{Collect}(P)$	Credits a payment P into T$_\mathsf{A}$'s balance	T$_\mathsf{A}$

Table 1: Protocol Notations

Client Registration Protocol

Client A requests server S create an account for her.

1. A sends [**RegisterClient**, vk_A] to S, where $(\mathsf{vk}_\mathsf{A}, \mathsf{sk}_\mathsf{A}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
2. Upon receiving [**RegisterClient**, vk_A], S performs the following steps:
   (a) Abort if $(\mathsf{vk}_\mathsf{A}, \cdot) \in \mathsf{S.Registry}$;
   (b) Add $(\mathsf{vk}_\mathsf{A}, \perp)$ to $\mathsf{S.Registry}$;
   (c) $\mathsf{S.onBal}_\mathsf{A} \leftarrow 0$;
   (d) Create cert_A such that $\mathsf{cert}_\mathsf{A}.\mathsf{vk} \leftarrow \mathsf{vk}_\mathsf{A}$ and $\mathsf{cert}_\mathsf{A}.\mathsf{sig} \leftarrow \mathtt{Sign}(\mathsf{vk}_\mathsf{A}, \mathsf{sk}_\mathsf{S})$;
   (e) Send cert_A to A.

Figure 2: Client Registration Protocol

We now describe how a TEE-enabled client can set up its TEE (according to the GP specification) using remote attestation and TA provisioning.

TEE Remote Attestation. The first step to set up a TEE is to obtain an attestation from the original equipment manufacturer (OEM) of the TEE to convince any verifier that the TEE hardware and the trusted OS are authentic. At a high-level, this is ensured by a read-only memory (ROM), and a device specific *device key-pair* $(\mathsf{D.vk}, \mathsf{D.sk})$. Both the ROM and the device keys are embedded into the hardware by the OEM. The trusted OS provides a signature (using D.sk from the ROM) on the TEE binaries using a method we denote $TOS.Attest$ along with D.vk to the remote party who forwards them to the OEM for verification. Since OEM knows the contents of the TEE stack, it can verify the signature, and hence attest to the authenticity of the TEE using a method we denote `OEMVerify`.

OPS TA Provisioning. Once the TEE is authenticated via remote attestation, the verifier needs to ensure that the OPS TA program (as shown on Figure 4) is provisioned (i.e., deployed) properly inside the TEE. This can be done via either local or remote provisioning [18]. In local provisioning, the OEM ships TA binaries on the device as part of the TEE software stack. In remote provisioning, a trusted party (e.g., Trustonic [24]) deploys the TA to the TEE remotely after the TEE has been authenticated via the remote attestation process. This is done by first establishing a secure channel with the TEE using the device's verification key, and then transmitting the TA binaries to the TEE over the secure channel. To ensure the TA is deployed properly, the TEE signs a hash of the binary with D.sk and returns the signature to the trusted party for verification.

3.2 OPS TA Registration

After registering with the server, TEE-enabled clients will need to perform two key steps in order to initialize their TEE. First, their TEE must be authenticated by means of (remote) attestation. Next, the OPS TA (as shown in Figure 4) must be provisioned within their TEE. After the TEE is validated and the OPS TA is setup via provisioning phase, the client's device and the OPS TA instance need to be registered with the server. To do this, the OPS TA first generates a signing key pair denoted $(\mathsf{T.vk}, \mathsf{T.sk})$, and returns to the client's device the verification key as well as the remote attestation. This process is described by the method `Init` described in the OPS TA program in Figure 4.

Next, the client transmits the TA's verification key and the attestation, along with its own device information to the server. The server, after verifying the attestation (using `OEMVerify`), certifies the key by signing it with the server's secret key and returning it to the device. The signed verification key is a certificate showing that the OPS TA key is generated by a genuine TEE and is registered with the server. Whenever this device makes a payment, it transmits the certificate along with other payment information, so that the receiver can independently verify the validity of the payment using the server's public (verification) key.

The certificate is also required to "activate" the client's TEE for offline payments. That is, only after receiving the certificate from the server ascertaining that it has been registered with the server will the OPS TA in the TEE perform any of its functions (other than `Init`). This check is performed by the method `CertInit` described in the OPS TA program (Figure 4). Specifically, initializing the variable T.cert by the method `CertInit` in the OPS TA is necessary for the invocation of other methods.

The server initializes a counter (denoted by i) for each OPS client. The OPS TA also maintains an internal variable denoted by T.i. Our protocol ensures that the two counters are "in sync" with one another. While the value of the counter would denote the number of deposits or withdrawals that have been performed by the client, the role of this counter is to distinguish various deposits (converting online funds to offline funds) and withdrawals (converting offline funds to online funds) and protect against replay attacks (e.g., replaying a deposit would allow a client to create offline funds out of thin air). This will be explained in further detail in Section 3.4.

The server makes use of its registry to ensure that the client has registered herself before the deposit step. The server also uses the registry to tag the TEE being registered along with the client who is registering it, by storing the pair (vk, T.vk) in its registry. The formal description of this protocol is presented in Figure 3.

TA Registration Protocol

Client A requests server S to register the instance of the OPS TA, denoted by T_A, on her device D_A. After the secure boot sets TEE, the following protocol is executed between A and S.

1. A obtains $(\mathsf{T}_\mathsf{A}.\mathsf{vk}, \sigma) \longleftarrow \mathsf{T}_\mathsf{A}.\mathtt{Init}()$ and sends $[\textbf{TARegister}, \mathsf{D}_\mathsf{A}.\mathsf{vk}, \mathsf{T}_\mathsf{A}.\mathsf{vk}, \mathsf{vk}_\mathsf{A}, \sigma]$ to S.
2. Upon receiving $[\textbf{TARegister}, \mathsf{D}_\mathsf{A}.\mathsf{vk}, \mathsf{T}_\mathsf{A}.\mathsf{vk}, \mathsf{vk}_\mathsf{A}, \sigma]$ from A, server S does the following steps:
   (a) Abort if any of the following conditions is true:
      - $(\mathsf{vk}_\mathsf{A}, \perp) \notin \mathsf{S.Registry}$;
      - $\mathtt{OEMVerify}(\mathsf{T}_\mathsf{A}.\mathsf{vk}, \sigma, \mathsf{D}_\mathsf{A}.\mathsf{vk}) \neq 1$;

   (b) $\mathsf{i}_\mathsf{A} \longleftarrow 0$;

   (c) Create a certificate cert such that
   $\mathsf{cert.vk} \longleftarrow \mathsf{T}_\mathsf{A}.\mathsf{vk}$;
   $\mathsf{cert.sig} \longleftarrow \mathtt{Sign}([\mathsf{T}_\mathsf{A}.\mathsf{vk}, \text{“}\mathtt{TA}\text{”}], \mathsf{sk}_\mathsf{S})$.

   (d) Send [cert] to A;

   (e) Replace $(\mathsf{vk}_\mathsf{A}, \perp)$ with $(\mathsf{vk}_\mathsf{A}, \mathsf{cert.vk})$ in S.Registry.
3. Upon receiving [cert] from S, client A invokes $\mathsf{T}_\mathsf{A}.\mathtt{CertInit}(\mathsf{cert})$.

Figure 3: TA Registration Protocol

3.3 OPS TA Program

In Figure 4, we present the methods provided by our OPS TA to the TEE-enabled client devices. We first briefly describe them. Note that the workings and the roles of the methods `Init` and `CertInit` have already been discussed in Sections 3.1 and 3.2.

- `Init`: Initializes the OPS TA, generates a key-pair along with attestation; this is the first method that must be invoked.

- `CertInit`: Processes certificate from the server; this is the second method that must be invoked, after which, other methods can be executed.

- `Deposit`: Converts online funds into offline funds, increases the offline balance.

- `Withdraw`: Converts offline funds into online funds, decreases the offline balance.

- `Pay`: Creates an offline payment object.

- `Collect`: Verifies an offline payment and applies it to the offline balance by increasing it with the payment amount.

- `Get-Balance`: Returns the current offline balance stored inside the TEE storage.

After registering herself with the server and provisioning the OPS TA on her TEE, the client invokes the `Init` method of the OPS TA. Using the results from the OPS TA, the client can then register her TEE with S, obtain the certificate from S, and invoke the `CertInit` method of the OPS TA. Now, the client can:

- Convert some/all of her online funds into offline funds in her OPS TA (by invoking the deposit protocol described in Section 3.4 which will involve invoking the `Deposit` method of the OPS TA);

- Use offline funds to make offline payments (by invoking the offline payment protocol described Section 3.5 which will involve invoking the `Pay` method of the OPS TA);

- Verify offline payments made to her either offline with her OPS TA (by invoking the collect protocol described Section 3.6 which will involve invoking the `Collect` method of the OPS TA) or online with the server (by invoking the claim protocol described Section 3.6);

- Convert some/all of her offline funds into online funds (by invoking the withdraw protocol described Section 3.4 which will involve invoking the `Withdraw` method of the OPS TA).

With the overall flow of operations in mind, we now describe the OPS TA program in more detail. The OPS TA maintains some variables whose functions are as follows:

- $(\mathsf{T.vk}, \mathsf{T.sk})$ is the singing key-pair used to authenticate outputs of the OPS TA.

- $\mathsf{T.bal}$ is the offline balance that is maintained within the OPS TA.

- $\mathsf{T.cert}$ is the certificate issued by the server on registering the TEE. This certificate allows the OPS TA to be convinced that the TEE has been registered with the server. It is also used by the OPS TA when it generates offline payments to identify itself as an authentic registered sender of offline funds.

- $\mathsf{T.inPaymentLog}$ is the log of offline payments received from other users. It is used to protect a client from a malicious sender who may be replaying a previous payment in an attempt to double-spend.

- $\mathsf{T.i}$ is a counter for deposits and withdrawals. The role and workings of this counter were alluded to briefly in Section 3. Recall that the server also maintains a copy of this counter and the two copies are kept in sync with one another. It is used to prevent replay attacks in the context of deposits and withdrawals. Further details are described in Section 3.4.

- T.j is a counter for payments. It is used to make every payment unique. This (in conjunction with the payment log inPaymentLog) prevents a client from replaying a previous payment in an attempt to double-spend.

We now describe in detail the various sub-protocols that are involved in our OPS protocol.

3.4 Deposit and Withdraw Protocols

In the deposit protocol presented in Figure 5, the client converts some/all of her online funds into offline funds. That is, the client deducts some amount from her online balance, as maintained by the server, and deposits the amount to her offline balance which is maintained by the OPS TA within her device. The withdraw protocol presented in Figure 6 works in the opposite direction converting offline funds into online funds. That is, the client withdraws some amount from her offline balance and transfers it to the server to add the amount to her online balance.

The deposit protocol works as follows. The client wishing to deposit an amount x of online funds into her offline balance, sends the request $[\textbf{Deposit}, x]$ to the server. The server on identifying the client[3] checks that the client has sufficient (greater than x) online funds. If so, the server deducts an amount of x from the clients online balance and generates a deposit confirmation that contains the amount x. Aside from the amount, the confirmation contains two other key pieces of information. The first is the counter i for deposits and withdrawals, and the second is a signature σ by the server on T.vk, x and i. We will describe the need for each of these ahead. On receiving the deposit confirmation, the client can invoke the Deposit method of the OPS TA with the confirmation. The method checks that the its local copy T.i is "in sync" with that of the server (technically, they would be off by 1 at this stage, but equal to each other once Deposit completes) and that the signature is valid. If so, it increments the offline balance by x and syncs up T.i.

We now describe the role of i and σ. The counter i is used to uniquely identify deposits and withdrawals and thus prevent a client using a particular deposit confirmation more than once. In particular, if a client attempts to invoke the method Deposit using a particular deposit confirmation more than once, the counter i would be out of sync with that of the server that is present in the confirmation itself. But what if the client attempts to spoof the value of i in the deposit confirmation in order to make it seem as though it is in sync? This is prevented by the signature σ.

Concretely, the presence of σ authenticates the confirmation sent by the server. It is not possible for the client to modify the particulars of the confirmation (the TEE, the amount or i) without resulting in an invalid signature in the confirmation. By signing on T.vk, deposit confirmations can only be processed by one single (intended) client TEE. By signing on x, the TEE is convinced of the authenticity of the deposit amount. By signing on i, as explained before, a deposit confirmation cannot be replayed in an attempt to generate offline funds out of thin air. The formal description of this protocol is presented in Figure 5.

[3] The explicit reference to these details has been omitted in the presentation in Figure 5. This would involve the client identifying themselves for instance using the certificate or verification key which the server could check for in its registry. The server would also need to determine if the client has registered her TEE which would be necessary in order to deposit online funds offline. The client would have to explicitly identify their TEE which the server could then check for in its registry.

OPS Trusted Application

```
Init():
    1. $(\mathsf{T.vk}, \mathsf{T.sk}) \leftarrow \mathsf{KeyGen}(1^{\lambda})$;  $\mathsf{T.bal} \leftarrow 0$;  $\mathsf{T.cert} = \perp$;  $\mathsf{T.inPaymentLog} \leftarrow \perp$;  $\mathsf{T.i} \leftarrow 0$;  $\mathsf{T.j} \leftarrow 0$;
    2. $\sigma \leftarrow TOS.Attest(\mathsf{T.vk})$
    3. Output $(\mathsf{T.vk}, \sigma)$.

CertInit(cert):
    1. Abort if $\texttt{TACertVerify}(\mathsf{cert}, \mathsf{T.vk_S}) \neq 1$.
    2. $\mathsf{T.cert} \leftarrow \mathsf{cert}$.

Deposit($x$, i, $\sigma_S$):
    1. Abort if $\mathsf{T.cert} = \perp$ or $\mathsf{i} \neq \mathsf{T.i} + 1$ or $\mathsf{SignVerify}([\mathsf{T.vk}, x, \mathsf{i}], \sigma_S, \mathsf{vk_S}) \neq 1$;
    2. $\mathsf{T.bal} \leftarrow \mathsf{T.bal} + x$;
    3. $\mathsf{T.i} \leftarrow \mathsf{T.i} + 1$.

Withdraw($x$):
    1. Abort if $\mathsf{T.cert} = \perp$ or $x > \mathsf{T.bal}$;
    2. $\mathsf{T.bal} = \mathsf{T.bal} - x$;
    3. $\mathsf{T.i} = \mathsf{T.i} + 1$;
    4. Output $[x, \mathsf{T.i}, \sigma]$, where $\sigma = \texttt{Sign}([x, \mathsf{T.i}], \mathsf{T.sk})$.

Pay($x$, receiver):
    1. Abort if $\mathsf{T.cert} = \perp$ or $\mathsf{T.bal} < x$;
    2. $\mathsf{T.bal} \leftarrow \mathsf{T.bal} - x$;
    3. $\mathsf{T.j} \leftarrow \mathsf{T.j} + 1$;
    4. $P.\mathsf{amount} \leftarrow x$;  $P.\mathsf{sender} \leftarrow \mathsf{T.cert}$;  $P.\mathsf{receiver} \leftarrow \mathsf{receiver}$;  $P.\mathsf{index} \leftarrow \mathsf{T.j}$;
    5. Output $P$, where $P.\mathsf{sig} \leftarrow \texttt{Sign}([P.\mathsf{amount}, P.\mathsf{sender}, P.\mathsf{receiver}, P.\mathsf{index}], \mathsf{T.sk})$.

Collect($P$):
    1. Abort if $\mathsf{T.cert} = \perp$ or $\texttt{PayVerify}(P) \neq 1$ or $P.\mathsf{receiver} \neq \mathsf{T.cert}$ or $P \in \mathsf{T.inPaymentLog}$;
    2. $\mathsf{T.bal} \leftarrow \mathsf{T.bal} + P.\mathsf{amount}$;
    3. Append $P$ to $\mathsf{T.inPaymentLog}$.

Get-Balance():
    1. Abort if $\mathsf{T.cert} = \perp$;
    2. Output $[\mathsf{T.bal}, \mathsf{T.i}, \sigma]$, where $\sigma = \texttt{Sign}([\mathsf{T.bal}, \mathsf{T.i}], \mathsf{T.sk})$.
```

Figure 4: OPS Trusted Application

Deposit Protocol

Client A requests server S to deposit an amount of x money from her online balance stored at S into her offline balance stored in T_A.

1. A sends [**Deposit**, x] to S.
2. Upon receiving [**Deposit**, x] from A, server S does the following steps:
   (a) Abort if $x > \mathsf{S.onBal}_\mathsf{A}$;
   (b) $\mathsf{S.onBal}_\mathsf{A} \longleftarrow \mathsf{S.onBal}_\mathsf{A} - x$;
   (c) $\mathsf{S.i}_\mathsf{A} \longleftarrow \mathsf{S.i}_\mathsf{A} + 1$;
   (d) Send [**DepositConfirmed**, $x, \mathsf{S.i}_\mathsf{A}, \sigma$] to A, where $\sigma \longleftarrow \mathtt{Sign}([\mathsf{T}_\mathsf{A}.\mathsf{vk}, x, \mathsf{S.i}_\mathsf{A}], \mathsf{S.sk}_\mathsf{S})$.
3. Upon receiving [**DepositConfirmed**, x, i, σ] from S, client A invokes $\mathsf{T}.\mathtt{Deposit}(x, \mathsf{i}, \sigma)$.

Figure 5: Deposit Protocol (Online → Offline)

The withdraw protocol works in exactly the same way as the deposit protocol, only in reverse. The client wishing to withdraw an amount x of offline funds into her online balance, invokes the `Withdraw` method of the OPS TA with the amount x. The OPS TA checks that the client has sufficient (greater than x) offline funds. If so, the OPS TA deducts an amount of x from the clients offline balance and generates a withdraw confirmation that contains the amount x. As in the case of the deposit protocol, the withdraw confirmation also contains the counter i and a signature σ by the OPS TA on x and i. We will describe the need for each of these ahead.

The client on receiving the withdraw confirmation, sends the request [**Withdraw**, x, i, σ] to the server. The server on identifying the client checks that the its local copy $\mathsf{S.i}$ is "in sync" with that of the OPS TA and that the signature is valid. If so, it increments the online balance by x and syncs up $\mathsf{S.i}$. The counter i prevents a client using a particular withdraw confirmation more than once. The presence of σ authenticates the confirmation sent by the OPS TA. It is not possible for the client to modify the particulars of the confirmation (the amount or i) without resulting in an invalid signature in the confirmation. By signing on x, the TEE is convinced of the authenticity of the deposit amount. By signing on i, as explained before, a withdraw confirmation cannot be replayed in an attempt to generate online funds out of thin air. The formal description of this protocol is presented in Figure 6.

Withdraw Protocol

Client A requests server S to withdraw an amount of x money from her offline balance maintained by T_A into her online balance maintained by S.

1. Client A sends [**Withdraw**, x, i, σ] to S, where $[x, \mathsf{i}, \sigma] \longleftarrow \mathsf{T}_\mathsf{A}.\mathtt{Withdraw}(x)$.
2. Upon receiving [**Withdraw**, x, i, σ] from A, server S performs the following steps:
   (a) Abort if $\mathsf{i} \neq \mathsf{S.i}_\mathsf{A} + 1$ or $\mathtt{SigVerify}([x, \mathsf{i}], \sigma, \mathsf{T}_\mathsf{A}.\mathsf{vk}) \neq 1$;
   (b) $\mathsf{S.onBal}_\mathsf{A} \longleftarrow \mathsf{S.onBal}_\mathsf{A} + x$;
   (c) $\mathsf{S.i}_\mathsf{A} \longleftarrow \mathsf{S.i}_\mathsf{A} + 1$;
   (d) Send [**WithdrawConfirmed**] to A;

Figure 6: Withdraw Protocol (Offline → Online)

3.5 Offline Payment Protocol

In the offline payment protocol presented in Figure 7, a client makes an offline payment to another client in the following way. First, the receiver of the payment sends a payment request [**RequestPayment**, x, receiver] to the sender of the payment. The payment request contains the payment amount as well as the certificate of the receiver (denoted by receiver). Upon receiving a payment request, the sender invokes the method Pay of the OPS TA using x and receiver. The OPS TA checks that the sender has sufficient (greater than x) offline funds. If so, the OPS TA deducts an amount of x from the clients offline balance and generates a payment confirmation that contains the amount x. Aside from the amount x, the payment confirmation also contains the certificates of the sender (recall, T.cert) and the receiver (recall, receiver), the payment counter j and a signature σ by the OPS TA on all of these particulars.

On receiving the payment confirmation, the receiver checks that the payment is valid using PayVerify which verifies the certificates and σ. In addition, the receiver checks that the payment was intended for her (i.e., matches the receiver) and that it is a fresh payment, i.e., the receiver can maintain a log inPaymentLog of payments she has received, much like the OPS TA does, in order to prevent malicious senders from replaying payments in an attempt to double-spend. If all checks pass, the receiver is convinced of receiving the payment – at this point, the receiver is ensured to obtain the payment funds. The receiver adds this payment to her log of payments and sends a confirmation message [**ReceivedPayment**] to S. This completes the portion of the protocol that involves the sender making an offline payment.

In order to obtain the payment funds, the receiver must either invoke the Collect method of the OPS TA, or she is not a TEE-enabled client, she may engage in the claim protocol. Further details regarding this are described in Section 3.6. It is to be noted that offline payments made to TEEs must be collected and not claimed, while payments made to clients must be claimed and not collected. We now describe the need for the various particulars of the payment confirmation. The presence of the sender and receiver certificates in the payment confirmation convince the receiver that the payment is coming from a registered sender and that the payment has indeed been made to herself. The counter j prevents a client using a particular payment confirmation more than once. Notice that the Pay method of the OPS TA increments j. Thus, every payment confirmation generated by a given TEE is unique (at the bare minimum, the value of j would differ between them).

The payment confirmation also contains the sender certificate, this also means that every payment confirmation every generated is unique (either the sender certificate would differ, and if not, j would). The presence of σ authenticates the confirmation sent by the OPS TA. It is not possible for the client to modify the particulars of the confirmation (the amount, sender and receiver certificates or j) without resulting in an invalid signature in the confirmation. By signing on x and the certificates of the sender and the receiver, the authenticity of the payment amount and parties involved in the payment is guaranteed. By signing on j, as explained before, a payment confirmation cannot be replayed in an attempt to double-spend offline funds. The formal description of this protocol is presented in Figure 7.

Offline Payment Protocol

Client B requests client A to send B a payment $P : \mathsf{A} \xrightarrow{x} \mathsf{B}$ transferring x amount of money from A to B while both clients are offline from S.

1. Client B sets $\mathsf{receiver} \leftarrow \mathsf{T_B.cert}$ if $\mathsf{T_B} \neq \bot$. Otherwise, $\mathsf{receiver} \leftarrow \mathsf{cert_B}$.
   He then sends [**RequestPayment**, x, receiver] to A.
2. Upon receiving [**RequestPayment**, x, receiver] from B, client A sends $P \leftarrow \mathsf{T_A}.\texttt{Pay}(x, \mathsf{receiver})$ to B.
3. Upon receiving P from A, client B performs the following steps:

   (a) Abort if any of the following conditions is true:
   - $\texttt{PayVerify}(P) \neq 1$,
   - $P.\mathsf{receiver} \neq \mathsf{receiver}$, or
   - $P \in \mathsf{B.inPaymentLog}$;

   (b) B adds P to B.inPaymentLog and sends [**ReceivedPayment**] to A;

   (c) If P.receiver.type = "TA", then B calls $\mathsf{T_B}.\texttt{Collect}(P)$;

   (d) Otherwise, B engages in the Claim protocol with S (Figure 9) as soon as B is online.

Figure 7: Offline Payment Protocol

$\texttt{PayVerify}(P)$:

Return 1 if and only if all of the following conditions hold:

1. $\texttt{TACertVerify}(P.\mathsf{sender}) \overset{?}{=} 1$, and
2. $\texttt{SigVerify}([P.\mathsf{amount}, P.\mathsf{sender}, P.\mathsf{receiver}, P.\mathsf{index}], P.\mathsf{sig}, P.\mathsf{sender.vk}) \overset{?}{=} 1$.

Figure 8: Payment Verification Method

3.6 Claim and Collect Protocols

Once a client has obtained a payment confirmation, they can obtain the funds by either invoking the `Collect` method of the OPS TA, or if she is not a TEE-enabled client, she may engage in the claim protocol presented in Figure 9 as soon as she is online. As noted in Section 3.5, offline payments made to TEEs must be collected and not claimed, while payments made to clients must be claimed and not collected. This is to prevent a malicious client from collecting and claiming a single payment confirmation in an attempt to generate online or offline funds out of thin air. Thus, a single payment confirmation may only be either collected or claim, but not both. This means that it is sufficient to ensure that a single confirmation cannot be collected more than once, nor claimed more than once.

Collect Protocol. A client can collect a payment by invoking the `Pay` method of the OPS TA using the payment confirmation. The OPS TA checks that the payment is valid (performed using `PayVerify` which verifies the certificates and σ), that it was intended for itself (matching receiver) and that it is a fresh payment (the OPS TA maintains a log inPaymentLog of payments it has received in order to prevent malicious senders from replaying payments in an attempt to double-spend; this means that a payment cannot be collected more than once). If all checks pass, the OPS TA increments the offline balance by x and adds this payment to its log of payments.

Claim Protocol. A client can claim a payment by engaging in the claim protocol as shown in Figure 9. The client wishing to claim a payment P sends the request [**Claim**, P] to S who checks that the payment is valid and that it is a fresh payment. To achieve this, S maintains a log inPaymentLog of payments that have been claimed, much like the OPS TA does, in order to prevent malicious clients from replaying payments in an attempt to generate online funds out of thin air. This means that a payment cannot be claimed more than once. If all checks pass, S increments the online balance by x and adds this payment to its log of payments. Finally, S sends a confirmation message [**ClaimConfirmed**] to the client.

Claim Protocol

Client B submits an offline payment P that he received from A to server S to have the money credited into his online balance stored at S.

1. B sends [**Claim**, P] to S.
2. Upon receiving [**Claim**, P] from B, server S performs the following steps:
   - (a) Abort if any of the following conditions is true:
     - P.receiver.type ≠ "UA";
     - PayVerify(P) ≠ 1;
     - $P \in$ S.paymentLog;
   - (b) $\mathsf{S.onBal}_{P.\mathsf{receiver}} \leftarrow \mathsf{S.onBal}_{P.\mathsf{receiver}} + P.\mathsf{amount}$;
   - (c) Add P to S.paymentLog;
   - (d) Send [**ClaimConfirmed**] to B.

Figure 9: Claim Protocol (Offline → Online)

References

[1] United States House of Representatives Committee on Financial Services: Task Force on Financial Technology. Inclusive banking during a pandemic: Using fedaccounts and digital tools to improve delivery of stimulus payments. https://www.congress.gov/116/meeting/house/110778/witnesses/HHRG-116-BA00-Wstate-GiancarloJ-20200611.pdf, June 2020. (Accessed on 09/15/2020).

[2] European Parliament. Crypto-assets: Key developments,regulatory concerns and responses. https://www.europarl.europa.eu/RegData/etudes/STUD/2020/648779/IPOL_STU(2020)648779_EN.pdf, April 2020. (Accessed on 09/15/2020).

[3] G7 Working Group on Stablecoins. Investigating the impact of global stablecoins. https://www.bis.org/cpmi/publ/d187.pdf, October 2019. (Accessed on 09/15/2020).

[4] Discussion paper - central bank digital currency: Opportunities, challenges and design. https://bankofengland.co.uk/-/media/boe/files/paper/2020/central-bank-digital-currency-opportunities-challenges-and-design.pdf, 03 2020. (Accessed on 06/29/2020).

[5] Colm Fulton. Sweden starts testing world's first central bank digital currency - reuters. https://reuters.com/article/us-cenbank-digital-sweden/sweden-starts-testing-worlds-first-central-bank-digital-currency-idUSKBN20E26G, 02 2020. (Accessed on 06/29/2020).

[6] China aims to launch the world's first official digital currency | finance & economics | the economist. https://economist.com/finance-and-economics/2020/04/23/china-aims-to-launch-the-worlds-first-official-digital-currency, 04 2020. (Accessed on 06/29/2020).

[7] Stephen S. Wu, Randy V. Sabett, Dr. Santosh Chokhani, Dr. Warwick S. Ford, and Charles (Chas) R. Merrill. Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework. RFC 3647, November 2003.

[8] The Fed - comparing means of payment: What role for a central bank digital currency? https://www.federalreserve.gov/econres/notes/feds-notes/comparing-means-of-payment-what-role-for-a-central-bank-digital-currency-20200813.htm, August 2020. (Accessed on 09/22/2020).

[9] Raphael Auer and Rainer Böhme. The technology of retail central bank digital currency. https://www.bis.org/publ/qtrpdf/r_qt2003j.pdf, March 2020. (Accessed on 09/22/2020).

[10] Payment and settlement systems report - Annex – Bank of Japan reports and research papers. https://www.boj.or.jp/research/brp/psr/data/psrb200702.pdf, July 2020. (Accessed on 09/22/2020).

[11] Card acceptance guidelines for Visa merchants. https://usa.visa.com/dam/VCOM/download/merchants/card-acceptance-guidelines-for-merchants.pdf, 2015. (Accessed on 06/30/2020).

[12] Mastercard transaction processing rules. https://mastercard.us/content/dam/mccom/global/documents/transaction-processing-rules.pdf, 12 2019. (Accessed on 06/30/2020).

[13] Square Editorial Team. Offline credit card processing - accept credit cards offline. https://squareup.com/us/en/townsquare/offline-credit-card-processing, 2020. (Accessed on 11/18/2020).

[14] Felix Wu. No easy answers in the fight over iphone decryption. *Commun. ACM*, 59(9):20–22, August 2016.

[15] Sarah Allen, Srdjan Capkun, Ittay Eyal, Giulia Fanti, Bryan Ford, James Grimmelmann, Ari Juels, Kari Kostiainen, Sarah Meiklejohn, Andrew Miller, Eswar Prasad, Karl Wüst, and Fan Zhang. Design choices for central bank digital currency – policy and technical considerations. https://www.brookings.edu/wp-content/uploads/2020/07/Design-Choices-for-CBDC_Final-for-web.pdf, July 2020. (Accessed on 12/10/2020).

[16] M. Sabt, M. Achemlal, and A. Bouabdallah. Trusted execution environment: What it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages (57–64), 2015.

[17] G. Arfaoui, S. Gharout, and J. Traoré. Trusted execution environments: A look under the hood. In *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 259–266, 2014.

[18] GlobalPlatform – the Trusted Execution Environment: Delivering enhanced security at a lower cost to the mobile market. https://globalplatform.org/wp-content/uploads/2018/04/GlobalPlatform_TEE_Whitepaper_2015.pdf, June 2015. (Accessed: 2020-06-21).

[19] ARM TrustZone. https://developer.arm.com/ip-products/security-ip/trustzone. (Accessed: 2020-06-21).

[20] Sandro Pinto and Nuno Santos. Demystifying ARM TrustZone: A comprehensive survey. *ACM Comput. Surv.*, 51(6), January 2019.

[21] GLOBALPLATFORM. GlobalPlatform: TEE internal API specification". https://globalplatform.org/wp-content/uploads/2018/06/GPD_TEE_Internal_Core_API_Specification_v1.1.2.50_PublicReview.pdf.

[22] Einav Zilberstein and Adi Klein. Western digital whitepaper: e.MMC security methods. https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/collateral/white-paper/white-paper-emmc-security.pdf, July 2017.

[23] Liang Cai. Guard your data with the Qualcomm Snapdragon mobile platform. https://www.qualcomm.com/media/documents/files/guard-your-data-with-the-qualcomm-snapdragon-mobile-platform.pdf, April 2019.

[24] Trustonic: World leading embedded cybersecurity & patented technology Trustonic. https://www.trustonic.com/. (Accessed on 11/24/2020).

Disclaimers

Case studies, comparisons, statistics, research and recommendations are provided "AS IS" and intended for informational purposes only and should not be relied upon for operational, marketing, legal, technical, tax, financial or other advice. Visa Inc. neither makes any warranty or representation as to the completeness or accuracy of the information within this document, nor assumes any liability or responsibility that may result from reliance on such information. The information contained herein is not intended as investment or legal advice, and readers are encouraged to seek the advice of a competent professional where such advice is required. All trademarks are the property of their respective owners, are used for identification purposes only, and do not necessarily imply product endorsement or affiliation with Visa.