A Secure and Flexible Blockchain-Based Offline Payment Protocol

Wanqing Jie, Wangjie Qiu, *Member, IEEE*, Arthur Sandor Voundi Koe, *Member, IEEE*, Jianhong Li, Yin Wang, Yaqi Wu, Jin Li, *Senior Member, IEEE*, and Zhiming Zheng

***Abstract*—Off-chain transactions seek to address the low on-chain scalability and enable blockchain-based payments over unreliable on-chain networks. The key problem with existing works is that they fail to balance security and flexibility in their designs. These studies would have been more useful if they could provide a sense of security without compromising their flexibility. We hypothesize that two offline parties having loosely synchronized clocks and channels with known bounded latency can conduct off-chain transactions while maintaining a high level of security and flexibility: we introduce a novel blockchain-based offline payment protocol that supports our hypothesis. Our work leverages on-chain smart contracts and offline wallet interactions to build resilience against intermittent on-chain connectivity. Our protocol achieves flexible and trusted computations with the use of platform-agnostic Trusted Execution Environments (TEEs) and open transactions. We empirically evaluate our design over the mainstream Intel Software Guard Extensions (SGX) and compare our protocol with state-of-the-art solutions. We found that our protocol attains high efficiency and exhibits an advanced level of security and flexibility in functionality. We evaluate our construction against several real-world attacks. We prove the security and robustness of our scheme based on a practical universally composable framework with synchronous settings. This work contributes to the existing knowledge of safe and user-friendly offline payment solutions for the blockchain technology.**

***Index Terms*—Blockchain, offline payment, smart contract, security, flexible, protocol.**

Manuscript received 12 January 2023; revised 12 July 2023; accepted 5 November 2023. Date of publication 13 November 2023; date of current version 15 January 2024. This work was supported by the National Key Research and Development Program of China under Grant 2021YFB2700300 and Beijing Natural Science Foundation Z230001. Recommended for acceptance by C. Li. *(Corresponding author: Wangjie Qiu.)*

Wanqing Jie, Wangjie Qiu and Zhiming Zheng are with the Institute of Artificial Intelligence, Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, Beihang University, Beijing 100191, China, also with Zhongguancun Laboratory, Beijing, China, also with the Institute of Artificial Intelligence and Blockchain, Guangzhou University, Guangdong 510006, China, and also with Health Blockchain Research Center, Binzhou Medical University, Yantai, Shandong 264003, China (e-mail: wanqing-jie@qq.com; wangjieqiu@buaa.edu.cn; zzheng@pku.edu.cn).

Arthur Sandor Voundi Koe is with the State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710071, China, and also with the Institute of Artificial Intelligence and Blockchain, Guangzhou University, Guangdong 510006, China (e-mail: 2517482859@qq.com).

Jianhong Li, Yin Wang, Yaqi Wu, and Jin Li are with the Institute of Artificial Intelligence and Blockchain, Guangzhou University, Guangdong 510006, China (e-mail: 1791269020@qq.com; 867042655@qq.com; 1550322596@qq.com; lijin@gzhu.edu.cn).

Digital Object Identifier 10.1109/TC.2023.3331823

I. Introduction

In the new global economy, the blockchain technology has become a key instrument in removing trusted intermediaries. The first proof of concept towards its practical use was put forward by the pseudonym Satoshi Nakamoto: the Bitcoin [1]. Ethereum [2], which supports the second most valuable cryptocurrency after Bitcoin, was developed with an intriguing property in mind: the ability to run smart contracts that satisfy Turing-completeness. What is striking is the growing amount of blockchain-based cryptocurrency projects [3] that leverage smart contracts to promote diverse use cases out of the sole financial realm. Governments, academicians and practitioners increasingly adopt the blockchain technology to address daily real-world problems in finance [4], e-health [5], cloud services [6], unmanned vehicle network [7], [8], etc.

Research in the field of blockchain technology considers security, scalability, and decentralization to be vital concepts. According to Vitalik's trilemma [9], existing blockchain implementations have yet to achieve these three properties simultaneously. To date, much study has concentrated on the application of blockchain technology in fully synchronous networks, rather than the robustness of blockchain solutions in adversarial networks featuring intermittent connectivity. Offline payments were introduced to promote transactions that take place offline and are reconciled with network connectivity. Off-chain transactions play a vital role in bringing about offline payments to blockchain-based scenarios.

Payment channels [10], [11], [12] stand as a significant contributory factor to the development of solutions towards layer-2 blockchain scalability issues. They leverage an off-chain network of channels to support offline payments. One criticism of much of the literature on payment channels is that the channels need to be pre-established and sufficient funds need to be locked in advance via a pay-to-multisig scheme (P2MS). Such requirements impede the flexibility of existing offline payment scenarios, introduce many security issues, and limit the wide adoption of blockchain technology in offline-first design for real-world applications. Payment channels would have been more relevant if they did not require to open on-chain settlement transactions (by locking up a chosen amount of cryptocurrency as a security deposit) and were not vulnerable to random failures and targeted attacks [13], [14]: channel exhaustion and node isolation attacks.

Previous published studies on offline payment solutions for blockchain lack standardized threat model and security model: such threat model should cover the various security threats in offline payment solutions for blockchain; such security model should guide the security analysis of blockchain-based offline payment schemes and assess the basic security requirements that are needed. To date, there has been no detailed investigation that simultaneously supports security and flexibility in existing blockchain-based offline payment constructions.

In this paper, we hypothesize that two offline parties, which experience intermittent on-chain connectivity, can engage in blockchain-based off-chain transactions without sacrificing security and flexibility: we introduce a secure and flexible protocol that realizes offline payments in blockchain-based solutions.

Our design builds upon the interactions between on-chain smart contracts and secure offline accounts. We transfer the on-chain trust from the blockchain anchor to offline accounts via the use of secure off-chain states, which are modified state channels that follow our protocol workflow. To support flexibility, we use open transactions that support coin redistribution together with instant settlement. To improve the security of our protocol, we leverage platform-agnostic Trusted Execution Environments (TEEs) that achieve trusted computations and secure offline wallet interactions in distributed settings. We provide a strong theoretical security analysis of our design over a universally composable framework that captures weakly synchronous off-chain interactions. We investigate several real-life attacks against our protocol: coin forgery, transaction data forgery, double-spend and double-deposit through transaction replay, man-in-the-middle (MITM) attacks and TEE related attacks. We evaluate the performances of our proposed construction over the Intel Software Guard Extensions (SGX).

Our contributions are as follows:

- **Effective trade-off between security and flexibility.** Different from previous studies, our scheme achieves a significant trade-off between security and flexibility in blockchain-based offline payment solutions.
- **Characterization of a comprehensive threat model.** We characterize a threat model that accounts for the various security issues in blockchain-based offline payment solutions, and prove the robustness of our construction under universal composability (UC) with synchronous settings.
- **Source code provisioning and evaluation methodology.** We analyze the source code lexicalities of our protocol and evaluate the performance based on specific metrics of real-world applications.

II. Related Work

This section revisits the literature on offline blockchain-based payments. Currently, research on the subject has been focusing on either security or flexibility: existing constructions fail to simultaneously meet the desired requirements of security and flexibility.

A. Offline Payment Constructions Based on Payment Channels

The concept of payment channel originates from the Lightning Network [15]: an off-chain expansion scheme of Bitcoin, which later evolved into more general off-chain channel schemes that exhibit Turing completeness. Two prominent examples of off-chain channel constructions are Raiden Network [16] and Counterfactual [17], which are based on the Ethereum blockchain.

Payment channels have become a de facto offline payment solution. In their latest work xLumi [11], Ying et al. described the specific workflow of offline wallets in payment channel based schemes: the payer and the payee must first create a channel online through a 2-of-2 multi-signature wallet, recharge the amount in the multi-signature wallet as offline spending and mortgage, then both parties can leverage the channel metadata to conduct offline transactions. Such metadata are broadcast on the chain for attestation, since the blockchain explicitly serves as the trust anchor. The two parties can check their respective account states to ensure the consistency of their interactions, both on and off the chain. They can opt to close the established channel.

Many recent works have made some improvements over the Lightning Network's payment channels. For example, Teechain [18], which is based on TEE trusted security hardware, empowers payment channel users to upload asynchronously to the chain. Teechain can tolerate read, write and dispute operations sent to the blockchain within the upper limit of the time window $\triangle$, which heightens its security and its efficiency. The schemes [12], [19] use one-way functions instead of digital signature schemes to reduce offline transaction fees. Nikolay Ivanov et al. implemented the VolgaPay payment system [20] to reduce the computational overhead of the original hash-based chain. They used a multi-hash chain-based micro-payment channel and leveraged polynomial price representation. Their work achieves security and efficiency trade-off towards payment transactions processing between merchants and customers.

Although offline payment solutions centered on payment channels are relatively mature and secure, the existing accounts fail to operate offline transactions without establishing a channel in advance. Existing payment channel constructions hardly support open transactions, which limits the flexibility of current designs.

B. Other Offline Payment Schemes

Recent offline payment protocols advocate flexible interactions between random payers and payees over intermittent on-chain connectivity. Dmitrienko A. et al. [21] proposed a classic three-phase protocol design for Bitcoin offline payment. In their protocol, the payer preloads online coins, recharges the offline wallet, and carries out offline payments with the payee. The payee performs online coin redemption and double-spend cancellation whenever the on-chain communication link sparkles. Such a scheme solves the double spending problem (launched by a malicious offline payer) through a tamper-proof wallet. It achieves the revocation of double-spend offline transactions and distributed offline wallets. Besides the fact that coin preloading results from earlier successful payer verification, Dmitrienko A. et al. [21] employ a time-based verification mechanism for transaction confirmation, which prevents coin forgery attacks. The follow-up work by Takahashi et al. [22] improved the coin

preloading process by requiring to verify the payee. Such a scheme gets rid of the timestamp server to ensure the security of offline coin recharge.

None of the above two solutions realizes instant settlement of the payee's offline account. Under poor on-chain connectivity, the above works fail to verify the status of emitted transactions and to timely update the balance of offline accounts. Ensuring the correct account balance at any time is crucial to maintaining the coherence of future transactions.

Igboanusi et al. [23] implemented the offline payment framework PureWallet (PW) using Ethereum smart contracts. In their work, the token manager converts cryptocurrencies into other tokens of interest, and both the sender and receiver perform offline transactions through Near Field Communication (NFC). Such a paradigm fails to suit real-world application scenarios as the system lacks a token forgery detection mechanism: resulting in offline transactions' fail to meet the basic security requirements. In addition, the tokens in Igboanusi et al. [23] cannot be re-distributed: on-chain network availability is necessary to update the state of any account. Such a requirement limits the inherent flexibility that stems from deploying offline transactions. Wang et al. designed the MOBT wallet model [24]. They generate key pairs for offline wallets through the master public key property of hierarchical deterministic (HD) wallets. In their work, the wallets only need to store and back up the master private keys (not the individual private keys generated for each offline transaction). They aimed to address storage scalability issues where the size of the offline wallet grows with the storage of individual private keys, when multiple offline transactions are performed. Besides, their work utilizes an interactive signature protocol to thwart the Kleptographic attack [25] on offline wallets.

Although recent offline payment solutions support open transactions and promote their flexibility, they still fall short of basic security requirements. In those works, the flexibility threshold to meet an acceptable level of security remains relatively low. To our knowledge, existing accounts fail to meet a high level of security and flexibility simultaneously. This is the first time a blockchain-based offline payment protocol is proposed to mitigate such a limitation in the literature.

III. Systemic Overview

A. System Model

Our protocol enables the interactions between five main entities of interest: the offline payer, the offline payee, the trusted execution environment (TEE), the rich execution environment (REE) and the on-chain smart contract. In this work, we refer to the offline payer and the offline payee as users. Fig. 1 highlights the interactions between users and the blockchain network (on-chain smart contract), while Fig. 2 captures the set of interactions between the blockchain network (on-chain smart contract), the REE, and the TEE, as well as the set of operations performed by the TEE and the REE, respectively. The following lines provide insights into the different roles each entity plays in the proposed construction.

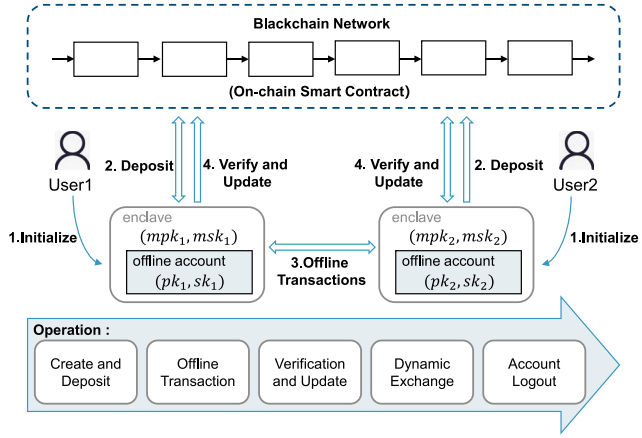


Fig. 1. Overview of our offline payment protocol.

(1) Users: Each user requires an offline account (a public/private key pair) to participate in our framework. Such an offline account possesses an address that is generated via the offline account's public key. The offline payer must perform an initial on-chain transaction that credits its offline account: there should be a confirmed on-chain transaction that deposits a strictly positive amount of cryptocurrency (for coherence) into the payer's offline account. Our protocol relaxes such a requirement on the offline payee, which is free from any mandatory initial on-chain deposit. In this work, users can deposit cryptocurrencies into offline accounts, as well as reconcile their off-chain and on-chain states during good on-chain connectivity. When the on-chain network becomes adversarial, users move off-chain. Our framework assumes that the offline payer initiates an offline transaction to the offline payee. Such an assumption is similar to many real-world payment protocols.

Our work requires the payer to embed its off-chain state metadata in every emitted offline transaction. Whenever the payee receives an offline transaction, it verifies the integrity and validity of the payer state using embedded metadata. Such metadata build upon the generation of an enclave measurement that relies on prior attestation from the on-chain smart contract. The offline payee updates its off-chain state if the verification succeeds. Our protocol empowers users to apply for account cancellation.

We require a weaker variant of synchrony between users to ensure input completeness and guaranteed termination. When the offline payer captures a receipt of transaction confirmation from the offline payee within the given lock time, it updates the balance of its off-chain state, and the offline transaction is considered successful. If no confirmation is received within the given lock time, the offline payer assumes that the transaction failed.

(2) TEE: The TEE operates in offline mode and disregards the on-chain connectivity state. It generates an offline account for the new user: a public/private key pair (pk, sk). It stores the off-chain state of the offline account and hosts the manufacturer's public key mpk. The TEE verifies the integrity of operations that aim to update the off-chain state.

(3) REE: The REE sits between the TEE and the on-chain smart contract and serves as an interface between both. It senses

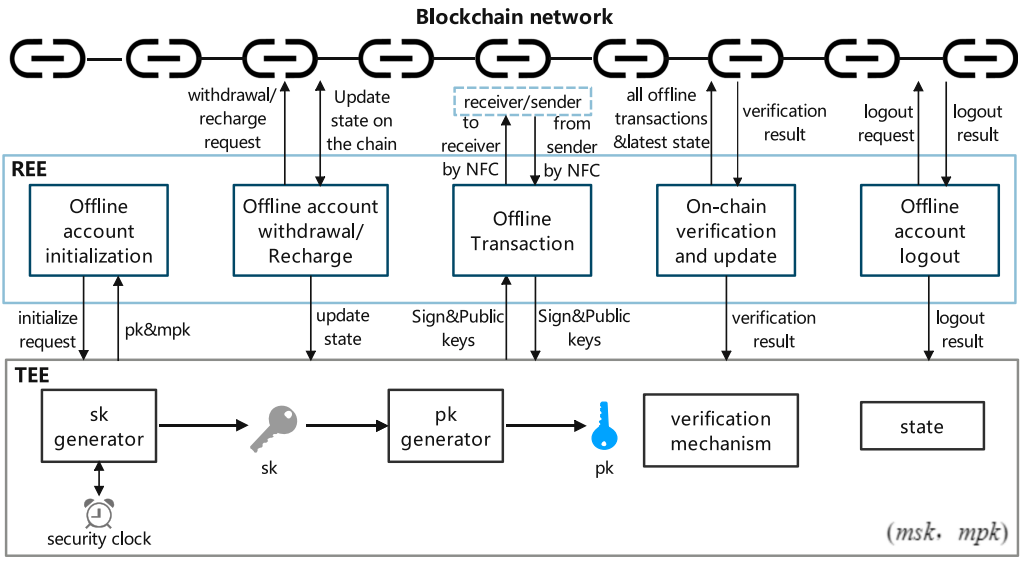


Fig. 2. Offline payment protocol workflow diagram.

the state of the on-chain connectivity; it initiates data transfer and calls to the on-chain smart contract during good on-chain network conditions. The REE provides functionalities that support the set of offline operations performed by users, such as the offline account initialization, the recharge and withdrawal of offline accounts, the processing of offline transactions, the on-chain verification of off-chain states, and the cancellation of user accounts. Section IV dives into the operations performed by REE and TEE in our construction.

(4) On-chain smart contract: Our scheme relies on an on-chain smart contract to update the on-chain state of users. In our protocol, every successful state transfer from on-chain to off-chain requires an attestation (a proof) from the on-chain smart contract: a transaction receipt. The on-chain smart contract is written in a contract-oriented language (COL) and performs verifiable calculations according to a predefined set of rules. The on-chain contract verifies the consistency and integrity of the received off-chain data during stable on-chain connectivity.

B. Threat Model and Security Assumptions

In this section, we briefly define the threat actors that participate in our protocol and state several assumptions that sustain the applicability of our novel framework.

1) Threat Model: In our proposed framework, the attacker is capable of carrying out coin forgery attacks, offline transaction data forgery, double-spend and double-deposit attacks, as well as man-in-the-middle attacks. The attacker may be a misbehaving participant, who adopts these malicious behaviors to obtain illicit gains.

Coin Forgery Attacks. Malicious users may launch coin forgery attacks during the initial on-chain deposit in their offline accounts. They may attempt to update the off-chain state without prior on-chain transaction confirmation; they may attempt to update the deposit balance in the offline wallet with a value greater than the corresponding one in the on-chain state.

Offline Transaction Data Forgery. The payer and the payee have incentives to tamper with data stored in the TEE. They might forge the amount, the generation time, or other data in offline transactions to get additional profits.

Double Spend and Double Deposit Attacks. A Malicious user may attempt sending the same offline transaction multiple times, or replay the same offline transaction to their TEE to launch double-spend and double-deposit attacks, respectively.

Man-in-the-Middle Attacks. Malicious users may try to impersonate the identities of honest users to shunt the nominal scenario in our construction.

2) Security Assumptions: Our scheme enforces the following security assumptions.

The on-chain operations exhibit correctness and soundness. The blockchain serves as trust anchor through transparent and immutable storage. This work assumes that on-chain functions are reliable.

The TEE is safe and reliable. Our protocol models the TEE as a secure abstraction that is platform-agnostic. The computations performed by the TEE are reliable.

Users operate on TEE-enabled platforms. This assumption is reasonable since recent mobile and computer devices are equipped with secure enclaves capable of trusted execution.

Cryptographic primitives are secure. This assumption is consistent with similar claims within the literature [26]. Primitives such as hash functions and digital signatures are secure and exhibit platform independence in this work.

The communication between users is reliable. Channels between users are weakly synchronous and authenticated: offline stakeholders possess loosely synchronized clocks and share authenticated channels with known bounded latency [27].

C. Design Goals

The overall goal of our manuscript is to design an efficient and secure offline payment protocol that achieves a significant trade-off between security and flexibility. The following lines depict the specific objectives we pursue in this work.

(1) SECURE BLOCKCHAIN-BASED OFFLINE PAYMENT PROTOCOL. We aim to achieve a secure construction that exhibits the following properties:

- **Coins and data unforgeability:** Our protocol should prevent malicious users from launching coin forgery attacks, and users should fail to forge transaction data in their offline wallets.
- **Consistency:** The state of unconfirmed on-chain transactions should not be considered valid in offline settings. An offline payee should be able to independently verify the authenticity of an offline transaction on-premises.
- **Double-spend, double-deposit, and man-in-the-middle attacks resistance:** Our scheme should thwart double-spend and multiple deposit initiatives from the same offline transaction. Our scheme should thwart offline identity spoofing and deter man-in-the-middle attacks.

(2) FLEXIBLE BLOCKCHAIN-BASED OFFLINE PAYMENT PROTOCOL. We aim to achieve a flexible protocol that exhibits the following properties:

- **Open transactions:** Our protocol should support open transactions, which overlook prior commitment to an on-chain multi-signature account whenever the users engage into off-chain transactions.
- **Coin redistribution:** Our protocol aims to achieve coin redistribution through divisible cryptocurrency: users can transact divisible units of cryptocurrency without the need for a change address, as opposed to the Bitcoin system.
- **Instant settlement and platform-agnostic design:** The offline payee should update its offline account balance within the shortest possible delay. Our protocol should uphold deployment across different TEE platforms.

(3) OFFLINE TRANSACTION ATOMICITY. An offline commit should correctly update user offline accounts with non-repudiation. An offline abort should revert the user's offline state to its previous set of values with coherence.

(4) OPTIMISTIC RESPONSIVENESS. To achieve a high degree of flexibility, our protocol should capitalize on-chain weaker synchrony, and strive during on-chain asynchrony.

IV. PROTOCOL DESIGN

We design our framework under universal composability (UC) with weak synchronous securities in the hybrid model: we express the real-world protocol as $\mathcal{P}$ and the ideal protocol as $\mathcal{F}$.

To specify our construction, we follow the interesting methodology of Camenish et al. [28], which provides a security framework for UC on top of the inexhaustible interactive Turing machine (IITM) model [29]. We specify our protocol via a set of machines. Each machine implements one or many roles (e.g. payer, payee). The running instance of a machine manages one or several entities.

We consider a player set P where each party $p_i \in P$ possesses a unique id pid_i, and supports one or many roles $role_i$. Players interact within sessions identified by a session id sid_i. An entity $(pid_i, sid_i, role_i)$ characterizes an instance of the role $role_i$ by party p_i within session id sid_i.

A. Participating Entities

Our system consists of the following types of interactive Turing machines (ITM): the environment $\mathcal{Z}$, the adversary $\mathcal{A}$, the real protocol $\mathcal{P}$, and the ideal functionality $\mathcal{F}$, which exhibits many ideal sub-functionalities.

THE ENVIRONMENT $\mathcal{Z}$. In our framework, the environment $\mathcal{Z}$ is the master meaning the first machine to run. $\mathcal{Z}$ is a mixed environment, both responsive and unresponsive. When responsive, $\mathcal{Z}$ must answer certain types of messages immediately: restricting messages [28]. The environment $\mathcal{Z}$ in this work is universally bounded: we place an upper bound on the amount of available computation power and storage capacity.$\mathcal{Z}$ provides initial parameters to all parties running the protocol under the common reference string (CRS) model [30].

THE IDEAL FUNCTIONALITY $\mathcal{F}$. Our work implements a single ideal functionality $\mathcal{F}$ that extends to several ideal sub-functionalities.

To achieve weak synchronous security in the UC (i.e., input-completeness and guaranteed termination), we leverage two ideal sub-functionalities depicted in [27]: $\mathcal{F}^{\delta}_{BD-SMT}$, which characterizes authenticated channels with known bounded-delay δ and eventual delivery properties, such that messages leading to an accumulated delay $T > \delta$ are ignored; $\mathcal{F}_{CLOCK}$, which establishes parties with loosely synchronized clocks enforced by a known upper bound σ on the maximum clock-drift, and ignores notifications from corrupted parties. To achieve authenticated and secure bounded-delay channel, this work assumes that $\mathcal{F}^{\delta}_{BD-SMT} = \{\mathcal{F}^{\delta}_{BD-SEC}$ [27], $\mathcal{F}^{\delta}_{BD-AUTH}$ [27]$\}$. To establish real-time delivery among players, we set $\delta = 1$.

To manage on-chain operations, we employ the ideal distributed ledger sub-functionality $\mathcal{F}_{ledger}$ [31]. $\mathcal{F}_{ledger}$ operates on chain and manages the blockchain initialization (i.e. to generate the genesis block), requests to read the global on-chain state, and updates to the on-chain state. It captures as well the full support for smart contract execution, the on-chain registration of offline parties, the consensus protocols, and other experimental on-chain operations to be set in the future.

To abstract and manage the TEEs, we employ the ideal sub-functionality $\mathcal{G}_{att}$ that captures all the platforms equipped with attested execution secure processors [32]. A secure processor embeds an internal source of randomness and stores the manufacturer-generated public/secret key pair (mpk/msk).$\mathcal{G}_{att}$ allows the installation of a new enclave using the command install, which loads the enclave program $prog_{enclave}$. To call the functions in $prog_{enclave}$, a player sends the command resume to $\mathcal{G}_{att}$.

As on-chain operations must be accessible to all participants, this work implements the ideal sub-functionality $\mathcal{F}_{ledger}$ using a single machine over a single instance that manages all entities. Such machine $\mathcal{M}_{ledger}$ implements the role ledger and concretizes the ideal sub-functionality $\mathcal{F}_{ledger}$.

We realize the ideal sub-functionality $\mathcal{G}_{att}$ through a single machine $\mathcal{M}_{install,resume}$ that spawns multiple instances. Each instance of $\mathcal{M}_{install,resume}$ manages exactly one entity. $\mathcal{G}_{att}$ supports a diverse set of operations that are detailed in [32].

Algorithm 1: Ideal Functionality $\mathcal{F}$

For each $p_i : (pk_i, ski) \leftarrow\$ \Sigma.genKey(1^n)$,
$state_{offchain_i} = state^0_{offchain_i}, addr_{pk_i} \leftarrow G_{att}.H(pk_i)$

function: onchainDeposit ($tx_{deposit}$, withdraw: boolean):
leak $(tx_{deposit}, (pk_{ledger}, sid_{ledger}$, ledger)) to $\mathcal{A}$
await $(state_{onchain} = tx^{receipt}_{deposit} \mid \perp) from \mathcal{A}$
verify $state_{onchain}$ abort if $\perp$
verify $tx^{receipt}_{deposit} : status$ abort if False
leak(**updateState**, "deposit", $tx_{deposit}, tx^{receipt}_{deposit}$, withdraw, $(pk_1, sid_1$, payer)) to $\mathcal{A}$

function: transfer (b_0, $addr_{pk_2}$):
leak ("transfer", $b_0, addr_{pk_2}, (pk_1, sid_1$, payer)) to $\mathcal{A}$
await tx_{off} from $\mathcal{A}$
leak(**transfer**, $tx_{off}, (pk_1, sid_1$, payer)) to $\mathcal{A}$

function: transfer (tx_{off}):
leak $(tx_{off}$, Payee $(pk_2, sid_2$, payee)) to $\mathcal{A}$
await $(ack_{win_2} \mid ack_{fail_2})$ from $\mathcal{A}$
if ack_{win_2}:
leak ("save", $ack_{win_2}, (pk_1, sid_1$, payer)) to $\mathcal{A}$
leak(**updateState**, "pay", $ack_{win_2}, (pk_1, sid_1$, payer)) to $\mathcal{A}$
if ack_{fail_2}:
leak(**updateState**, "abort", $ack_{fail_2}, (pk_1, sid_1$, payer)) to $\mathcal{A}$

function: updateState ("deposit", $tx_{deposit}$, $tx^{receipt}_{deposit}$, withdraw):
leak("deposit", $tx_{deposit}, tx^{receipt}_{deposit}$, withdraw, $(pk_1, sid_1$, payer)) to $\mathcal{A}$

function: updateState ("pay", ack_{win_2}):
leak("pay", $ack_{win_2}, (pk_1, sid_1$, payer)) to $\mathcal{A}$

function: updateState ("abort", ack_{fail_2}):
leak("abort", $ack_{fail_2}, (pk_1, sid_1$, payer)) to $\mathcal{A}$

function: leak():
leak("leakState",$(pk_i, sid_i$, (payer | payee))) to $\mathcal{A}$
if $(pk_i, sid_i$, (payer | payee)) is honest:
await (σ^i_{att}) from $\mathcal{A}$
if $(pk_i, sid_i$, (payer | payee)) is corrupted:
await $(\sigma^i_{att}, \{state^i_{offchain}\}$ without $\{msk_i\})$ from $\mathcal{A}$

function: receive(tx_{off}):
parse tx_{off} as $(pk_1, addr_{pk_1}, addr_{pk_2}, b_0, T_{gen}, \sigma^1_{att}, mpk_1, \sigma 1)$
if accept b_0: leak("receive", $tx_{off}, (pk_2, sid_2$, payee)) to $\mathcal{A}$
await $ack_{win_2} \mid ack_{fail_2}$ from $\mathcal{A}$
leak $(ack_{win_2} \mid ack_{fail_2}$, Payer $(pk_1, sid_1$, payer)) to $\mathcal{A}$
if refuse b_0: leak("reject", $tx_{off}, (pk_2, sid_2$, payee)) to $\mathcal{A}$
await ack_{fail_2} from $\mathcal{A}$
leak $(ack_{fail_2}$, Payer $(pk_1, sid_1$, payer)) to $\mathcal{A}$

function: reconciliation():
leak("onchainUpdate", $(pk_i, sid_i$, (payer | payee)) to $\mathcal{A}$
await $(tx^1_{off_j|\forall j\in\{1..n\}}, ack^j_{win_2}, ack^j_{fail_2}, \sigma^i_{att})$ from $\mathcal{A}$
leak $((tx^1_{off_j|\forall j\in\{1..n\}}, ack^j_{win_2}, ack^j_{fail_2}, \sigma^i_{att}), (pk_{ledger}, sid_{ledger}$, ledger)) to $\mathcal{A}$
await $(state_{onchain} = tx^{receipt}_{reconcile} \mid \perp) from \mathcal{A}$
verify $state_{onchain}$ abort if $\perp$
if $tx^{receipt}_{reconcile} : status$ is False:
$\mathcal{F}_{ledger}$ freezes $(addr_{pk_i}, pk_i)$
leak("freeze", $tx^{receipt}_{reconcile}, (pk_i, sid_i$, (payer | payee)) to $\mathcal{A}$
if $tx^{receipt}_{reconcile} : status$ is True:
$\mathcal{F}_{ledger}$ updates $state_{onchain_i}$
leak("finalize", $tx^{receipt}_{reconcile}, (pk_i, sid_i$, (payer | payee))) to $\mathcal{A}$

function: logout():
leak(**reconciliation**, $(pk_i, sid_i$, (payer | payee))) to $\mathcal{A}$
await $tx^{receipt}_{reconcile}$ from $\mathcal{A}$
parse $tx^{receipt}_{reconcile}$ as $tx^{receipt}_{reconcile} : status$
if status is True: leak("logout", $(pk_i, sid_i$, (payer | payee))) to $\mathcal{A}$ abort if False
await tx_{logout} from $\mathcal{A}$
parse tx_{logout} as $(pk_i, addr_{pk_i}, addr_{pk_x}, state_{offchain_i} : balance, T_{gen}, \sigma i, (\sigma^i_{att}, mpk_i))$
leak $(tx_{logout}, (pk_{ledger}, sid_{ledger}$, ledger)) to $\mathcal{A}$
await $(state_{onchain} = tx^{receipt}_{logout} \mid \perp) from \mathcal{A}$
verify $state_{onchain}$ abort if $\perp$
verify $tx^{receipt}_{logout} : status$ abort if False
leak("clean", $tx_{logout}, tx^{receipt}_{logout}, (pk_i, sid_i$, (payer | payee))) to $\mathcal{A}$

Algorithm 1: Ideal Functionality $\mathcal{F}$

(1) **On receive** ("deposit", $tx_{deposit}, tx^{receipt}_{deposit}$, withdraw):
parse $tx_{deposit}$ as $(pk_{pid}, pk_{caller}, addr_{pk_{caller}}, \sigma, b_0)$
parse $tx^{receipt}_{deposit}$ as $(status)$
verify σ and $status$, abort if False
if withdraw False:
$state_{offchain_i} : balance+ = b_0$
if withdraw True:
$state_{offchain_i} : balance- = b_0$
append $(tx_{deposit}, tx^{receipt}_{deposit}$, withdraw) to Storage[$\mathcal{P}_i$]

(2) **On receive** ("save", ack_{win_2}):
store ack_{win_2}

(3) **On receive** ("pay", ack_{win_2}):
parse ack_{win_2} as $(pk_2$,
$tx_{off} = (pk_1, addr_{pk_1}, addr_{pk_2}, b_0, T_{gen}, \sigma^1_{att}, mpk_1, \sigma 1), \sigma_2$,
$status_{tx_{off}} = ok)$
verify $(\sigma_2, status_{tx_{off}} = ok, \sigma^1_{att}, \sigma 1)$ and abort if False
$state_{offchain_1} : balance- = b_0$

(4) **On receive** ("receive", tx_{off}):
parse tx_{off} as $(pk_1, addr_{pk_1}, addr_{pk_2}, b_0, T_{gen}, \sigma^1_{att}, mpk_1, \sigma 1)$
if verify $(\sigma_1, \sigma^1_{att})$ is True:
build and store ack_{win2}; $state_{offchain_2} : balance+ = b_0$
leak ack_{win2} to $\mathcal{A}$
if verify $(\sigma_1, \sigma^1_{att})$ is False:
build ack_{fail2}
leak ack_{fail2} to $\mathcal{A}$

(5) **On receive** ("transfer", $b_0, addr_{pk_2}$) :
build tx_{off} as $(pk_1, addr_{pk_1}, addr_{pk_2}, b_0, T_{gen}, \sigma^1_{att}, mpk_1, \sigma 1)$
leak tx_{off} to $\mathcal{A}$

(6) **On receive** ("logout") :
build tx_{logout} as $(pk_i, addr_{pk_i}, addr_{pk_x}, state_{offchain_i} : balance, T_{gen}$,
$\sigma^i_{att}, mpk_i, \sigma_i)$
leak tx_{logout} to $\mathcal{A}$

(7) **On receive** ("reject", tx_{off}):
build ack_{fail2}
store ack_{fail2}
leak ack_{fail2} to $\mathcal{A}$

(8) **On receive** ("abort", ack_{fail_2}):
parse ack_{fail_2} as $(pk_2, tx_{off} = (pk_1, addr_{pk_1}, addr_{pk_2}, b_0, T_{gen}, \sigma^1_{att}$,
$mpk_1, \sigma 1), \sigma_2, status_{tx_{off}} = fail)$
verify $(\sigma_2, status_{tx_{off}} = fail, \sigma^1_{att}, \sigma 1)$ and abort if False
store ack_{fail_2}

(9) **On receive** ("clean", $tx_{logout}, tx^{receipt}_{logout}$):
verify $tx_{logout} : \sigma$ and $tx^{receipt}_{logout} : status$ abort if False
delete(pk_i, sk_i)

(10) **On receive** ("leakState"):
if **honest**: leak σ_{att} to $\mathcal{A}$
if **corrupted**: leak $(\sigma_{att}, (state_{offchain}$ without $msk))$ to $\mathcal{A}$

(11) **On receive** ("finalize", $tx^{receipt}_{reconcile}$):
leak $tx^{receipt}_{reconcile}$ to $\mathcal{A}$
parse $tx^{receipt}_{reconcile}$ as $tx^{receipt}_{reconcile} : status$
if $tx^{receipt}_{reconcile} : status$ is True:
free storage $(tx^1_{off_j|\forall j\in\{1..n\}}, ack^j_{win_2}, \sigma^i_{att})$
if $tx^{receipt}_{reconcile} : status$ is False: abort

(12) **On receive** ("onchainUpdate"):
collect params as $(tx^1_{off_j|\forall j\in\{1..n\}}, ack^j_{win_2}, ack^j_{fail_2}\ \sigma^i_{att})$
leak params to $\mathcal{A}$

This work extends $\mathcal{G}_{att}$ with the ability to sign messages, verify signatures, and hash data under the random oracle model. Algorithm 1 portrays the security objective of our offline protocol in the ideal functionality $\mathcal{F}$.

THE ADVERSARY $\mathcal{A}$ AND THE CORRUPTION MODEL. Our work adopts the static corruption model, where the dummy adversary $\mathcal{A}$ corrupts a subset of entities after their trusted initialization [28].

We consider an entity as corrupted if at least one of its subroutines is corrupted, rather than requiring all subroutines to be corrupted. Our work abstracts the corruption protocol to be used and insights to design an interesting corruption protocol are available in [28].

When an entity becomes corrupted, the internal state of such entity is leaked to the adversary that gains full control over the corrupted entity. In this work, $\mathcal{G}_{att}$ responds to a specific restricting message (**leakState**) that requests its complete internal state. We adopt the transparent enclave model $\hat{\mathcal{G}}_{att}$ [33] that leaks the full state of a corrupted player to the dummy adversary $\mathcal{A}$, while keeping the manufacturer signing key msk secret.

We feature a malicious adversary that instructs corrupted parties to send several messages on its behalf. We match those messages to the set of offline attacks we investigate in this manuscript: coin forgery, data tampering, double-spend, double-deposit, and man-in-the-middle attacks.

Algorithm 2: Setup sub-Protocol $\mathcal{P}_{setup}$

On receive ("install"):
For each instance G_{att}^{j}: $(mpk_j, msk_j) := \Sigma.genKey(1^n)$

On receive ("init"):
$p_i : (pk_i, ski) := \Sigma.genKey(1^n)$
$mpk_i := \mathcal{G}_{att}.getpk()$
$addr_{pk_i} := G_{att}.H(pk_i)[20]$
return $(pk_i, mpk_i, addr_{pk_i})$

THE OFFLINE PAYMENT PROTOCOL $\mathcal{P}$. We model our offline payment protocol as a real protocol $\mathcal{P}$ that exploits four ideal sub-functionalities as subroutines: $\mathcal{G}_{att}$, $\mathcal{F}_{BD-SMT}^{\delta=1}$, $\mathcal{F}_{CLOCK}$, and $\mathcal{F}_{ledger}$. $\mathcal{P}$ embodies three roles: setup, payer, and payee. We leverage the digital signature scheme Σ and the random oracle hash function H both embedded in $\mathcal{G}_{att}$. We illustrate our overall framework in Fig. 1.

To enable the secure generation and global distribution of system initial parameters, we implement the setup role as a single machine that spawns a single instance over multiple entities. We implement the roles payer and payee as two separate machines. We require every instance of those machines to handle exactly one entity. The instances above characterize an actual implementation of our protocol and each new run leads to a new program instance.

Setup role. The setup sub-protocol $\mathcal{P}_{setup}$ manages the initialization of players. It leverages system parameters analog to the global common reference string (GCRS) model.

During the execution of $\mathcal{P}_{setup}$, parties send the install command to their instance of $\mathcal{M}_{install,resume}$ to load the enclave program $prog_{enclave}$, and obtain a manufacturer key pair (mpk, msk). Parties send the init command to $\mathcal{M}_{install,resume}$ to generate an off-chain account with a unique public/private key pair (pk_{pid}, sk_{pid}), generate an account address based on pk_{pid} and initialize the off-chain state $state_{off-chain}$ with the value $state_{off-chain}^{0}$.

Algorithm 2 reveals the details of the sub-protocol $\mathcal{P}_{setup}$ in our construction.

Payer role. We define several sub-routines of the sub-protocol $\mathcal{P}_{payer}$ that aim at the following four operations: onchainDeposit, transfer, updateState, and leak. We combine existing operations to provide two additional functionalities for the role payer: withdraw and recharge. Algorithm 3 reveals the details of the sub-protocol $\mathcal{P}_{payer}$ in our scheme.

$\mathcal{P}_{payer}$ leverages $\mathcal{F}_{BD-SMT}^{\delta=1}$ and $\mathcal{F}_{CLOCK}$ to query a party through the authenticated channel, and wait for a guaranteed response within the threshold δ. Our scheme can be extended to support any arbitrary value of δ.

The onchainDeposit subroutine first checks the on-chain connectivity. It exploits the restricting message **checkConnection** sent to $\mathcal{M}_{ledger}$, which returns a Boolean value.

When **checkConnection** returns True, the onchainDeposit subroutine leverages $\mathcal{M}_{ledger}$ to send the input transaction $tx_{deposit}$ to the on-chain smart contract address. $tx_{deposit}$ includes the account address generated from pk_{pid} as the receiver address, a caller address, the caller's public key, the caller's signature σ, and the required cryptocurrency amount b_0. We consider σ was generated using $\mathcal{F}_{ledger}$ based on the private key of the on-chain contract caller. Such a caller, the sender in $tx_{deposit}$, must possess an existing on-chain account filled with cryptocurrencies.

The onchainDeposit subroutine queries the status of $tx_{deposit}$ using the algorithm **checkStatus**($tx_{deposit}$) over $\mathcal{M}_{ledger}$. **checkStatus**($tx_{deposit}$) aims to return the current on-chain state $state_{onchain}$ that is either none or the transaction receipt $tx_{deposit}^{receipt}$, after validation from the on-chain contract $contract_{onchain}$ (i.e., $balance(calleraddress) >= value + fees$). Based on the status in $tx_{deposit}^{receipt}$, the onchainDeposit subroutine can call the subroutine updateState that sends the message deposit to $\mathcal{M}_{install,resume}$ with $tx_{deposit}$ and $state_{onchain}$ as parameters. To support withdrawal and recharge operations, we leverage a Boolean variable $withdraw$ as input parameter to the onchainDeposit subroutine.

The transfer subroutine leverages the amount of cryptocurrency b_0 to be sent to the payee, and the payee's address $addr_{pk_2}$, which was generated through the payee's public key pk_2. The transfer subroutine sends the command transfer to $\mathcal{M}_{install,resume}$ with b_0 and $addr_{pk_2}$ as parameters. $\mathcal{M}_{install,resume}$ checks that the balance of the local off-chain state $state_{offchain_1}$:$balance$ is greater than b_0. It builds an offline transaction tx_{off} that includes pk_1, the address of the payer $addr_{pk_1}$, $addr_{pk_2}$, b_0, the timestamp T_{gen}, the enclave measurement σ_{att}^{1}, mpk_1, and the offline transaction signature σ_1.To generate σ_{att}^{1}, $\mathcal{M}_{install,resume}$ leverages pk_1, $state_{offchain_1}$:$balance$, and msk_1 for enclave signature. To generate σ_1, $\mathcal{M}_{install,resume}$ signs tx_{off} using sk_1.The transfer subroutine forwards tx_{off} to the payee's machine instance and waits for a guaranteed response: tx_{off} can succeed or fail. We define a failed offline transaction as one that was rejected by the payee. The payee generates an acknowledgment of failure ack_{fail_2} whenever it rejects an offline transaction. The payee generates an acknowledgment of success ack_{win_2} whenever it accepts an offline transaction. ack_{fail_2} contains pk_2, tx_{off}, $status_{tx_{off}}$=fail, and the signature σ_2 over $\{pk_2, tx_{off}, status_{tx_{off}} = fail\}$; ack_{win_2} contains pk_2, tx_{off}, $status_{tx_{off}}$ = ok, and the signature σ_2 over $\{pk_2, tx_{off}, status_{tx_{off}} = \text{ok}\}$.

If the transfer subroutine receives an acknowledgment of failure ack_{fail_2} from the payee, it invokes the updateState subroutine that sends the command abort to $\mathcal{M}_{install,resume}$ together with ack_{fail_2} as the parameter. If the transfer subroutine receives an acknowledgement of success ack_{win_2} from the payee: it sends the command save to $\mathcal{M}_{install,resume}$ with ack_{win_2} as parameter; it invokes the updateState subroutine that sends the command pay to $\mathcal{M}_{install,resume}$ with ack_{win_2} as parameter. Following the command save, $\mathcal{M}_{install,resume}$ stores ack_{win_2} locally.

The updateState subroutine aims to update the balance of $state_{offchain_1}$. It sends three types of commands to $\mathcal{M}_{install,resume}$: deposit, pay, and abort. The command deposit serves to update $state_{offchain_1}$:$balance$ after an on-chain deposit. It takes $tx_{deposit}$ and $state_{onchain}$ as input parameters. The command deposit instructs $prog_{enclave}$ to verify σ in $tx_{deposit}$ and to check the on-chain status of $tx_{deposit}$ using $state_{onchain}$. If both verification processes succeed, $\mathcal{M}_{install,resume}$ credits $state_{offchain_1}$:$balance$ by b_0.

Algorithm 3: Payer sub-Protocol $\mathcal{P}_{payer}$

$state_{offchain_1} := state^0_{offchain_1}$
Storage[p_1] := ϕ

function: onchainDeposit ($\mathbf{tx_{deposit}}$, **withdraw:bool**)**:**
$state_{onchain} :=$ upload $tx_{deposit}$ to $\mathcal{M}_{ledger}$
assert $state_{onchain} \neq \perp$
assert ($state_{onchain} := tx^{receipt}_{deposit}$) $\rightarrow status \neq$ False
call updateState ("deposit", $tx_{deposit}$, $tx^{receipt}_{deposit}$, withdraw:bool, (pk_1, sid_1, $payer$))
call leak()

function: transfer ($\mathbf{b_0}$, $\mathbf{addr_{pk_2}}$)**:**
tx_{off} = upload ("transfer", b_0, $addr_{pk_2}$, (pk_1, sid_1, $payer$)) to $\mathcal{M}^1_{install,resume}$
call transfer(tx_{off}, (pk_1, sid_1, $payer$))
call leak()

function: transfer ($\mathbf{tx_{off}}$)**:**
output := upload (tx_{off}, (pk_2, sid_2, payee)) to $\mathcal{M}_{payee}$
if output == ack_{win_2}:
 upload ("save", ack_{win_2}, (pk_1, sid_1, $payer$)) to $\mathcal{M}^1_{install,resume}$
 call updateState ("pay", ack_{win_2}, (pk_1, sid_1, payer))
if output == ack_{fail_2}:
 call updateState ("abort", ack_{fail_2}, (pk_1, sid_1, payer))
call leak()

function: updateState (**"deposit"**, $\mathbf{tx_{deposit}}$, $\mathbf{tx^{receipt}_{deposit}}$, $withdraw : bool$)**:**
upload ("deposit", $tx_{deposit}$, $tx^{receipt}_{deposit}$, withdraw:bool, (pk_1, sid_1, payer)) to $\mathcal{M}^1_{install,resume}$
call leak()

function: updateState (**"pay"**, ack_{win_2})**:**
upload ("pay", ack_{win_2}, (pk_1, sid_1, payer)) to $\mathcal{M}^1_{install,resume}$
call leak()

function: updateState (**"abort"**, ack_{fail_2})**:**
upload ("abort", ack_{fail_2}, (pk_1, sid_1, payer)) to $\mathcal{M}^1_{install,resume}$
call leak()

function: leak()**:**
output := upload ("leakState", (pk_1, sid_1, payer)) to $\mathcal{M}^1_{install,resume}$
if (pk_1, sid_1, payer) is honest:
 return σ^1_{att}
if (pk_1, sid_1, payer) is corrupted:
 return σ^1_{att}, $state^1_{offchain}$ = (mpk_1, $balance$, pk_1, sk_1, $addr_{pk_1}$, $tx^1_{off_{j|j\in\{1..n\}}}$, $ack^{j|j\in\{1..n\}}_{win_2}$, $ack^{j|j\in\{1..n\}}_{fail_2}$)
function: reconciliation()**:**
output := upload ("onchainUpdate", (pk_1, sid_1, payer)) to $\mathcal{M}^1_{install,resume}$
parse output as ($tx^1_{off_{j|j\in\{1..n\}}}$, $ack^{j|j\in\{1..n\}}_{win_2}$, $ack^{j|j\in\{1..n\}}_{fail_2}$, σ^1_{att})
receipt := upload output to $\mathcal{M}_{ledger}$
receipt := receipt || output
if $receipt \rightarrow status$ is False:
 upload ("freeze", receipt, (pk_1, sid_1, payer)) to $\mathcal{M}^1_{install,resume}$
if $receipt \rightarrow status$ is True:
 upload ("finalize", receipt, (pk_1, sid_1, payer)) to $\mathcal{M}^1_{install,resume}$
call leak()

function: logout()**:**
status:= **call** reconciliation((pk_1, sid_1, payer))
if status is True:
 tx_{logout} = upload ("logout", (pk_1, sid_1, payer) to $\mathcal{M}^1_{install,resume}$
 $state_{onchain} :=$ upload tx_{logout} to $\mathcal{M}_{ledger}$
 assert $state_{onchain} \neq \perp$ and assert $state_{onchain} \rightarrow status \neq$ False
 upload ("clean", tx_{logout}, $state_{onchain}$, (pk_1, sid_1, payer) to $\mathcal{M}^1_{install,resume}$
if status is False abort

function: recharge(b_0, $addr_{pk_j}$)**:**
status:= **call** reconciliation((pk_1, sid_1, payer))
if status is True:
 $tx_{recharge}$:= upload ("transfer", b_0, $addr_{pk_j}$, (pk_1, sid_1, $payer$)) to $\mathcal{M}^1_{install,resume}$
 call onChainDeposit ($tx_{recharge}$, withdraw:=False, (pk_1, sid_1, $payer$))
 call leak()
if status is False:
 call leak()
 abort

$\mathcal{M}_{install,resume}$ stores $tx_{deposit}$ and $state_{onchain}$ to reconcile off-chain and on-chain states during on-chain synchrony (or weak synchrony). The command pay serves to update the local off-chain state of the payer machine instance after a successful offline transaction. The command pay takes ack_{win_2} as parameter. It instructs $prog_{enclave}$ to perform several operations: $prog_{enclave}$ verifies the signature σ_2 using pk_2; $prog_{enclave}$ ensures tx_{off} was accepted by the payee; $prog_{enclave}$ verifies the enclave signature over σ^1_{att} using mpk_1; $prog_{enclave}$ verifies the payer signature over tx_{off} using sk_1.

Algorithm 3: Payer sub-Protocol $\mathcal{P}_{payer}$

function: withdrawal(b_0, $addr_{pk_j}$)**:**
status:= **call** reconciliation((pk_1, sid_1, payer))
if status is True:
 $tx_{withdraw}$:= upload ("transfer", b_0, $addr_{pk_j}$, (pk_1, sid_1, $payer$)) to $\mathcal{M}^1_{install,resume}$
 call onChainDeposit ($tx_{withdraw}$, withdraw:=True, (pk_1, sid_1, $payer$))
 call leak()
if status is False: **call** leak() and abort

(1) **On receive** ("deposit", $tx_{deposit}$, $tx^{receipt}_{deposit}$, withdraw:bool):
if withdraw is True:
 parse $tx_{deposit}$ as (pk_1, $addr_{pk_j}$, $addr_{pk_1}$, b_0, T_{gen}, σ^1_{att}, mpk_1, σ_1)
 verify (σ^1_{att}, σ_1) and assert ($status == True$) abort if false
 $state_{offchain_i}$: $balance- = b_0$
if withdraw is False:
 parse $tx_{deposit}$ as (pk_{pid}, pk_{caller}, $addr_{pk_{caller}}$, σ, b_0)
 verify σ and assert ($status == True$) abort if false
 $state_{offchain_i}$: $balance+ = b_0$
append ($tx_{deposit}$, $tx^{receipt}_{deposit}$, withdraw) to Storage[$\mathcal{P}_1$]

(2) **On receive** ("save", ack_{win_2}): store ack_{win_2} to Storage [p_1]

(3) **On receive** ("transfer", b_0, $addr_{pk_2}$) :
build $tx_{off} as (pk_1, addr_{pk_1}, addr_{pk_2}, b_0, T_{gen}, \sigma^1_{att}, mpk_1, \sigma 1)$
upload (tx_{off}, (pk_1, sid_1, payer)) to $\mathcal{M}_{payer}$

(4) **On receive** ("logout"):
build tx_{logout} as (pk_1, $addr_{pk_1}$, $addr_{pk_x}$, $state_{offchain_1}$: $balance$, T_{gen}, σ^1_{att}, mpk_1, $\sigma 1$)
upload (tx_{logout}, (pk_1, sid_1, payer)) to $\mathcal{M}_{payer}$

(5) **On receive** ("pay", ack_{win_2}):
parse ack_{win_2} as (pk_2, $tx_{off} = (pk_1, addr_{pk_1}, addr_{pk_2}, b_0, T_{gen}, \sigma^1_{att}, mpk_1, \sigma 1), \sigma_2$, $status_{tx_{off}} = ok$)
verify (σ_2, σ^1_{att}, $\sigma 1$) and assert ($status_{tx_{off}} == ok$) abort if False
$state_{offchain_1}$: $balance- = b_0$

(6) **On receive** ("abort", ack_{fail_2}):
parse ack_{fail_2} as (pk_2, $tx_{off} = (pk_1, addr_{pk_1}, addr_{pk_2}, b_0, T_{gen}, \sigma^1_{att}, mpk_1, \sigma 1), \sigma_2$, $status_{tx_{off}} = fail$)
verify (σ_2, σ^1_{att}, $\sigma 1$) and assert ($status_{tx_{off}} == fail$) abort if False
store ack_{fail_2} to Storage[p_1]

(7) **On receive** ("finalize", receipt):
parse receipt as (receipt $\rightarrow$ status, output= ($tx^1_{off_{j|j\in\{1..n\}}}$, $ack^{j|j\in\{1..n\}}_{win_2}$, $ack^{j|j\in\{1..n\}}_{fail_2}$ σ^1_{att}))
if receipt $\rightarrow$ status is True:
 free Storage[p_1](output) and upload (receipt, (pk_1, sid_1, payer)) to $\mathcal{M}_{payer}$
if receipt $\rightarrow$ status is False: abort

(8) **On receive** ("clean", tx_{logout}, $state_{onchain}$):
parse tx_{logout} as (pk_1, $addr_{pk_1}$, $addr_{pk_x}$, $state_{offchain_1}$: $balance$, T_{gen}, σ^1_{att}, mpk_1, σ_1)
verify (σ_1, σ^{att}_1) and assert ($state_{onchain} \rightarrow status$) abort if False
delete(pk_1, sk_1)

(9) **On receive** ("leakState"):
if (pk_1, sid_1, payer) is honest:
 upload (σ^1_{att}, (pk_1, sid_1, payer)) to $\mathcal{M}_{payer}$
if (pk_1, sid_1, payer) is corrupted:
 upload (σ^1_{att}, $state^1_{offchain}$=(mpk_1, $balance$, pk_1, sk_1, $addr_{pk_1}$, $tx^1_{off_{j|j\in\{1..n\}}}$, $ack^{j|j\in\{1..n\}}_{win_2}$, $ack^{j|j\in\{1..n\}}_{fail_2}$), ($pk_1$, sid_1, payer)) to $\mathcal{M}_{payer}$

(10) **On receive** ("onchainUpdate"):
get ($tx^1_{off_{j|\forall j\in\{1..n\}}}$, $ack^j_{win_2}$, $ack^{j|j\in\{1..n\}}_{fail_2}$, σ^1_{att}) from Storage[p_1]
upload ($tx^1_{off_{j|\forall j\in\{1..n\}}}$, $ack^j_{win_2}$, $ack^{j|j\in\{1..n\}}_{fail_2}$, σ^1_{att}, (pk_1, sid_1, payer)) to $\mathcal{M}_{payer}$

If all verification processes succeed, $\mathcal{M}_{install,resume}$ debits $state_{offchain_1}$:$balance$ by b_0. The command abort helps to process failed offline transactions. It takes as input ack_{fail_2} and instructs $prog_{enclave}$ to perform the following operations: $\mathcal{M}_{install,resume}$ verifies σ_2 using pk_2; $\mathcal{M}_{install,resume}$ checks that ack_{fail_2} contains rejection proof of tx_{off}; $\mathcal{M}_{install,resume}$ verifies σ_1 in tx_{off} using pk_1; $\mathcal{M}_{install,resume}$ verifies σ^1_{att} in tx_{off} using mpk_1. If no verification fails, $\mathcal{M}_{install,resume}$ simply stores ack_{fail_2} to resolve eventual disputes.

The leak subroutine takes advantage of the restricting message $leakState$ sent to $\mathcal{M}_{install,resume}$, which returns only the enclave measurement σ_{att} for uncorrupted parties (honest players). $leakState$ sent to $\mathcal{M}_{install,resume}$ returns both the

enclave measurement σ_{att} and the entirety of the off-chain state $state_{offchain}$, except msk_{pid}, for corrupted parties.

Payee role. We define subroutines of the sub-protocol $\mathcal{P}_{payee}$ that aim at the following two main operations: receive, and leak. Algorithm 4 exposes the details of the sub-protocol $\mathcal{P}_{payee}$ in this work.

The receive subroutine fetches the offline transaction tx_{off} from the network. It checks the cryptocurrency amount b_0 in tx_{off} and decides whether to accept or reject tx_{off}.If the receive subroutine accepts tx_{off}, it sends the command "receive" to $\mathcal{M}_{install,resume}$ with tx_{off} as parameter. $\mathcal{M}_{install,resume}$ checks the signature of the enclave measurement σ^1_{att} in tx_{off} using mpk_1. $\mathcal{M}_{install,resume}$ checks the payer signature σ_1 using pk_1. After the successful validation of both signatures, $\mathcal{M}_{install,resume}$ builds an acknowledgment of success ack_{win_2}, stores a copy locally, and returns ack_{win_2} to the receive subroutine; $\mathcal{M}_{install,resume}$ credits the payee's off-chain state balance $state_{offchain_2} : balance$ by b_0. The receive subroutine exploits $\mathcal{F}^{\delta=1}_{BD-SMT}$ and $\mathcal{F}_{CLOCK}$ to forward ack_{win_2} to the payer's machine instance. If the receive subroutine refuses tx_{off}, the receive subroutine sends the command "reject" to $\mathcal{M}_{install,resume}$ with tx_{off} as parameter. The following process also occurs if the validation of any signature fails. $\mathcal{M}_{install,resume}$ builds an acknowledgment of failure ack_{fail_2}, saves a local copy, and returns ack_{fail_2} to the receive subroutine that forwards it to the payer's machine instance (with guaranteed delivery).

The leak subroutine exploits the leakState restricting message sent to $\mathcal{M}_{install,resume}$. It obeys the same workflow as the leak subroutine in the payer's machine instance.

Reconciliation of off-chain and on-chain states. The reconciliation subroutine applies to both the payer and the payee.

The reconciliation subroutine first assays the on-chain network connectivity by sending the restricting message checkConnection to $\mathcal{M}_{ledger}$. If checkConnection returns True, the reconciliation subroutine sends the command "onchainUpdate" to $\mathcal{M}_{install,resume}$. $\mathcal{M}_{install,resume}$ gathers all the stored offline transactions $tx_{off_{j|\forall j\in\{1..n\}}}$, all the acknowledgments of success $ack^j_{win_2}$, all the acknowledgments of failure $ack^j_{fail_2}$, and the latest enclave measurement for the party initiating the on-chain reconciliation (i.e., σ^1_{att} or σ^2_{att}). $\mathcal{M}_{install,resume}$ returns those data to the reconciliation subroutine that leverages $\mathcal{M}_{ledger}$ to submit them to the on-chain smart contract. $\mathcal{M}_{ledger}$ responds to the reconciliation subroutine with a transaction receipt $tx^{receipt}_{reconcile}$ that indicates the success or failure of the operation through the status field. If $tx^{receipt}_{reconcile}$:$status$ outputs False (the on-chain verification of data failed), $\mathcal{M}_{ledger}$ freezes the on-chain balance of such party, and the reconciliation subroutine sends the command freeze to $\mathcal{M}_{install,resume}$ with $tx^{receipt}_{reconcile}$ as parameter. $\mathcal{M}_{install,resume}$ checks the status in $tx^{receipt}_{reconcile}$. If $tx^{receipt}_{reconcile}$:$status$ outputs False, $\mathcal{M}_{install,resume}$ freezes the off-chain state $state_{offchain}$, which prevents the generation of future enclave measurements during the freezing time slot. If $tx^{receipt}_{reconcile}$:$status$ outputs True (the on-chain verification of data succeeded), $\mathcal{M}_{ledger}$ updates the on-chain state $state_{onchain_{pid}}$ of the corresponding party pid, and the reconciliation subroutine sends the command finalize to $\mathcal{M}_{install,resume}$ with $tx^{receipt}_{reconcile}$ as parameter. $\mathcal{M}_{install,resume}$ checks the status in $tx^{receipt}_{reconcile}$. If $tx^{receipt}_{reconcile}$:$status$ outputs True, $\mathcal{M}_{install,resume}$ frees storage space related to offline transactions, acknowledgments of success and the enclave measurement used to generate $tx^{receipt}_{reconcile}$.

Offline account cancellation. The offline account cancellation relies on the logout subroutine to delete an existing offline account within the abstract TEE. It applies to both the payer and the payee. The logout subroutine begins by the reconciliation of off-chain and on-chain states of a specific party. The rationale behind such subroutine is to carry out an on-chain transfer of the complete balance of the offline account $state_{offchain_i} : balance$ to any address, and to require $\mathcal{M}_{install,resume}$ to delete the account (pk_i, sk_i) after successful on-chain transfer.

Regarding additional operations in our protocol, the withdraw functionality performs an on-chain transfer of cryptocurrency units from the offline account to a recipient address; it requires a prior reconciliation of off-chain and on-chain states. The recharge functionality replenishes the offline account in terms of cryptocurrency units: it requires an initial reconciliation of on-chain and off-chain states, and leverages the onchainDeposit subroutine.

B. Communication Model

In our protocol, the parties, the adversary, and the subfunctionalities are linked in a star topology through an additional router machine that takes instructions from the adversary $\mathcal{A}$. In our work, such a router machine is implemented using an ideal sub-functionality that ensures authenticated channels (authenticated channels prevent an adversary from modifying data in transit while enforcing secrecy through encryption) and weak synchrony among parties. The work of Canetti et al. [34] provides more insights into the role of such a router machine. According to [28], the entities can send and receive messages using the input/output and the network interfaces that belong to their respective roles. This work enforces partial synchrony on the off-chain communication model: we leverage a responsive environment [35] for off-chain settings where messages are delivered reliably within δ. Although we alleviate assumptions regarding the on-chain communication model and rely on underlying properties provided by $\mathcal{F}_{ledger}$, we assume the default asynchrony of the on-chain communication model: our work considers the environment as unresponsive for on-chain interactions.

V. Security Analysis and Proof

In this section, we dive into the security analysis of our proposed protocol. With respect to the threat model in Section III, we separate our analysis into two subsections: informal security analysis, which analyzes the various attacks in our protocol; formal security, which evaluates the security of our construction based on UC.

Algorithm 4: Payee sub-Protocol $\mathcal{P}_{payee}$

$state_{offchain_2} = state^0_{offchain_2}$
Storage[p_2] := ϕ

function: leak():
output := upload ("leakState", (pk_2, sid_2, payee)) to $\mathcal{M}^2_{install,resume}$
if (pk_2, sid_2, payee) is honest:
return σ^2_{att}
if (pk_2, sid_2, payee) is corrupted:
return σ^2_{att}, $state^2_{offchain}$ = ($mpk_2, balance, pk_2, sk_2, addr_{pk_2}, tx^2_{off_{j|j\in\{1..n\}}}, ack^{j|j\in\{1..n\}}_{win_2}, ack^{j|j\in\{1..n\}}_{fail_2}$)

function: receive(tx$_{off}$):
parse tx_{off} as ($pk_1, addr_{pk_1}, addr_{pk_2}, b_0, T_{gen}, \sigma^1_{att}, mpk_1, \sigma 1$)
if accept b_0:
output:=upload ("receive", tx_{off},(pk_2, sid_2, payee) to $\mathcal{M}^2_{install,resume}$
parse output as ack_{win_2} or ack_{fail_2}
upload ("save", ack_{win_2} | ack_{fail_2}, (pk_2, sid_2, payee)) to $\mathcal{M}^2_{install,resume}$
upload(output, (pk_1, sid_1, payer)) to $\mathcal{M}_{payer}$
if refuse b_0:
ack_{fail_2} :=upload ("reject", tx_{off}, (pk_2, sid_2, payee) to $\mathcal{M}^2_{install,resume}$
parse output := ack_{fail_2} as ($pk_2, tx_{off} = (pk_1, addr_{pk_1}, addr_{pk_2}, b_0, T_{gen}, \sigma^1_{att}, mpk_1, \sigma 1), \sigma_2, status_{tx_{off}} = fail$)
upload ("save", output, (pk_2, sid_2, payee)) to $\mathcal{M}^2_{install,resume}$
upload(output, (pk_1, sid_1, payer)) to $\mathcal{M}_{payer}$
call leak()

function: reconciliation():
output := upload ("onchainUpdate", (pk_2, sid_2, payee)) to $\mathcal{M}^2_{install,resume}$
parse output as ($tx^1_{off_{j|j\in\{1..n\}}}, ack^{j|j\in\{1..n\}}_{win_2}, ack^{j|j\in\{1..n\}}_{fail_2}, \sigma^2_{att}$)
receipt := upload output to $\mathcal{M}_{ledger}$
receipt := receipt || output
if $receipt \rightarrow status$ False:
upload ("freeze", receipt, (pk_2, sid_2, payee)) to $\mathcal{M}^2_{install,resume}$
if $receipt \rightarrow status$ True:
upload ("finalize", receipt, (pk_2, sid_2, payee)) to $\mathcal{M}^2_{install,resume}$
call leak()

function: logout():
status:= **call** reconciliation((pk_2, sid_2, payee))
if status is True:
tx_{logout} = upload ("logout", (pk_2, sid_2, payee) to $\mathcal{M}^1_{install,resume}$
$state_{onchain}$:= upload tx_{logout} to $\mathcal{M}_{ledger}$
assert $state_{onchain} \neq \perp$ and assert $state_{onchain} \rightarrow status \neq$ False
upload ("clean", tx_{logout}, $state_{onchain}$, (pk_2, sid_2, payee) to $\mathcal{M}^1_{install,resume}$
if status is False abort
call leak()

Algorithm 4: Payee sub-Protocol $\mathcal{P}_{payee}$

(1) **On receive** ("receive", tx_{off}):
parse tx_{off} as ($pk_1, addr_{pk_1}, addr_{pk_2}, b_0, T_{gen}, \sigma^1_{att}, mpk_1, \sigma 1$)
if verify (σ_1, σ^1_{att}) True:
build ack_{win_2} as (pk_2, $tx_{off} = (pk_1, addr_{pk_1}, addr_{pk_2}, b_0, T_{gen}, \sigma^1_{att}, mpk_1, \sigma 1), \sigma_2$, $status_{tx_{off}} = ok$)
store ack_{win_2} to Storage[p_2]
$state_{offchain_2}$: $balance += b_0$
upload(ack_{win_2}, (pk_1, sid_1, payer)) to $\mathcal{M}_{payer}$
if verify (σ_1, σ^1_{att}) False:
build ack_{fail2} as (pk_2, $tx_{off} = (pk_1, addr_{pk_1}, addr_{pk_2}, b_0, T_{gen}$, $\sigma^1_{att}, mpk_1, \sigma 1), \sigma_2, status_{tx_{off}} = fail$)
store ack_{fail2} to Storage[p_2]
upload(ack_{fail2}, (pk_1, sid_1, payer)) to $\mathcal{M}_{payer}$

(2) **On receive** ("reject", tx_{off}):
build ack_{fail2} and store ack_{fail2} to Storage[p_2]
upload(ack_{fail2}, (pk_1, sid_1, payer)) to $\mathcal{M}_{payer}$

(3) **On receive** ("finalize", receipt):
parse receipt as (receipt $\rightarrow$ status, output= ($tx^1_{off_{j|j\in\{1..n\}}}, ack^{j|j\in\{1..n\}}_{win_2}$, $ack^{j|j\in\{1..n\}}_{fail_2}, \sigma^2_{att}$))
if receipt $\rightarrow$ status True: free Storage[p_2](output) abort if False

(4) **On receive** ("logout"):
build tx_{logout} as ($pk_2, addr_{pk_2}, addr_{pk_x}, state_{offchain_2}$: $balance, T_{gen}$, $\sigma^2_{att}, mpk_2, \sigma 2$)
upload (tx_{logout}, (pk_2, sid_2, payee)) to $\mathcal{M}_{payee}$

(5) **On receive** ("clean", tx_{logout}, $state_{onchain}$):
parse tx_{logout} as ($pk_2, addr_{pk_2}, addr_{pk_x}, state_{offchain_2}$: $balance, T_{gen}$, $\sigma^2_{att}, mpk_2, \sigma_2$)
verify (σ_2, σ^{att}_2) and assert ($state_{onchain} \rightarrow status$) abort if False
delete(pk_2, sk^2_2)

(6) **On receive** ("leakState"):
if (pk_2, sid_2, payee) is honest:
upload (σ^2_{att}, (pk_2, sid_2, payee)) to $\mathcal{M}_{payee}$
if (pk_2, sid_2, payee) is corrupted:
upload (σ^2_{att}, $state^2_{offchain}$=($mpk_2, balance, pk_2, sk_2, addr_{pk_2}$, $tx^1_{off_{j|j\in\{1..n\}}}, ack^{j|j\in\{1..n\}}_{win_2}, ack^{j|j\in\{1..n\}}_{fail_2}$), ($pk_2$, sid_2, payee)) to $\mathcal{M}_{payee}$

(7) **On receive** ("onchainUpdate"):
get params := ($tx^1_{off_{j|\forall j\in\{1..n\}}}, ack^j_{win_2}, ack^j_{fail_2}, \sigma^2_{att}$) from Storage[$\mathcal{P}_2$]
upload (params, (pk_2, sid_2, payee)) to $\mathcal{M}_{payee}$

(8) **On receive** ("save", $input$): store $input$ to Storage [p_2]

A. Informal Security Analysis

Coin Forgery Attacks. A malicious user p_i may try to update $state_{offchain_i}$ via a fake transaction $tx^{'}_{deposit}$ that lacks receipt from the on-chain smart contract. Since $tx^{'receipt}_{deposit}$ must be produced by the on-chain smart contract, the execution of $tx^{'}_{deposit}$ fails as long as it is an invalid transaction. To modify the balance of a deposit transaction requires the ability to forge the on-chain digital signature scheme. Since on-chain operations are trusted, such endeavor exhibits unfeasibility in our scheme. We consider that our protocol avoids coin forgery attacks.

Forgery of Offline Transaction Data. A malicious player may attempt to tamper the amount b_0 or the transaction timestamp T_{gen} in an offline transaction received from the network. As our work relies on the transparent enclave model, no other entity except the enclave has knowledge of the manufacturer's secret key msk: verifying the signature of such transaction will fail in the secure enclave and the attack will be unsuccessful.

Double Spend and Double Deposit Attacks. A malicious payer could be motivated to send the same transaction multiple times to the payee in hopes to double spend. Since operations in the enclave are trusted and each enclave keeps track of received transactions, $prog_{enclave}$ will fail to process the same offline transaction twice on the recipient side. For multiple use of the same coin, the payer does not have access to msk and lacks the ability to masquerade a fake attestation as valid. A malicious payee may re-transmit the same transaction received from the network to itself (to its enclave), in hopes to double deposit and increase the balance of its offline account. Since the enclave keeps track of received offline transactions as a trusted offline ledger, attempts to double deposit will fail and the malicious payee may under go temporal account freeze. Our protocol thwarts double spend and double deposit attacks.

Man-in-the-middle Attacks. An eavesdropper can intercept an offline payment transaction tx_{off}. It may attempt to impersonate the payee to enjoy the amount of cryptocurrency units in such offline transaction. Since each party has a unique public/private key pair and enclave computations are trusted, it is infeasible for such adversary to confuse the enclave about the ownership of the rightful recipient's public key. Since the environment is responsive, the eavesdropper has no ability to continuously delay the execution of the protocol. Such transaction will be denied by the secure enclave and more actions could be taken such as temporal account freeze to deter man-in-the-middle attacks.

B. Formal Security Proof Based on UC

This subsection analyzes our offline payment solution under the UC framework with off-chain synchronous securities.

Theorem 1: Let $\mathcal{P}$ and $\mathcal{F}$ be two complete but resource-bounded protocols with the same set of public roles [28]. The protocol $\mathcal{P}$ realizes the ideal functionality $\mathcal{F}$ ($\mathcal{P} \leq \mathcal{F}$) if and only if for all adversaries $\mathcal{A}$ that control the network of $\mathcal{P}$, there exists

a simulator $\mathcal{S}$ that controls the network of $\mathcal{F}$ such that $\mathcal{A}$, $\mathcal{P}$ exhibits indistinguishability from $\{\mathcal{S}, \mathcal{F}\}$ for all environments $\mathcal{Z}$: $\{\mathcal{Z}, \mathcal{A}, \mathcal{P}\} \equiv \{\mathcal{Z}, \mathcal{S}, \mathcal{F}\}$.

Proof: Proof of indistinguishability. Let E be an event where the simulator $\mathcal{S}$ has access to all leaked messages as well as the internal state of corrupted parties, such that $\mathcal{S}$ can forge signatures, forge data in transit and data at rest. Since our UC model exhibits weakly synchronous and authenticated channels; since computations performed within the TEE are trusted (the hash function $\mathcal{G}_{att}.H$ in the random oracle model, and the digital signature scheme $\mathcal{G}_{att}.\Sigma$); since the environment $\mathcal{Z}$ is responsive and maintains the secrecy of msk even for corrupted parties, it is challenging to construct a PPT algorithm that finds collisions in $\mathcal{G}_{att}.H$ or collisions in random functions: the probability that event E occurs is negligible. Our protocol $\mathcal{P}$ can safely realize the ideal functionality $\mathcal{F}$ in the proposed hybrid UC model.

VI. Implementation and Evaluation

A. Offline Payment Protocol Implementation

We implement two components of our offline payment protocol: 1) on-chain smart contract and 2) offline account. We test the blockchain offline payment protocol using an eight-core 2.4 GHz Intel(R) Core(TM) i7-10510U CPU and 16 GB of RAM. The operating system we used is Ubuntu 20.04.4 TLS with Linux kernel version 5.15.0-43-generic.

On-chain smart contract: We developed the on-chain smart contract of our protocol based on the Ethereum platform [2], using Solidity [36] version 0.8.1. To guarantee the functional requirements of the on-chain smart contract in our work, we deployed such smart contract in a local go-Ethereum environment on our internal server. Users can interact with the deployed smart contract: they can invoke its functions to ensure the functional requirements of our protocol regarding the enclave and the consistency of operations.

Offline account: We use the open source framework Occlum 0.29.2 [37] for TEE OS based on the common TEE platform Intel SGX [38], and develop it using the python language. Occlum is an open-source TEE OS system, designed using the container concept and divided into enclave and host. The user loads and executes the appropriate application from the trusted image using the Occlum run command in the host. In Python, we utilized web3.py [39], which is compatible with Ethernet accounts, to hash the transactions using Keccak256 [40] and sign the transaction hash using ECDSA-secp256k1 [41]. The offline payment protocol we developed via Python is built into Intel SGX via Occlum to enable secure interaction between offline accounts and on-chain smart contracts.

B. Offline Payment Protocol Evaluation

1) Code Size: To develop the blockchain offline payment protocol, we migrate web3.py to SGX via Occlum. We use Python for the development of the offline payment protocol on the off-chain, relying on the web3 package and making relevant modifications to the functions within the package to meet the functional requirements of the protocol. As shown in Table I, we produced 560 LOC of overall Python code for the protocol and another 2849 LOC from dependencies. The protocol was made to run into Intel SGX by packing it with 63 LOC of one configuration file and two script files, using the Occlum build command and making a trusted image. The online contract deployed on the blockchain is implemented with Solidity and has 215 LOC.

TABLE I
Code Size

Component	Code	LOC
Offline payment protocol	Python	560
Dependent codes	Python	2849
Occlum setting	scripts	63
Online-contract	Solidity	215

TABLE II
Gas and Time Consumption of On-Chain Smart Contract

Function	Gas (GWEI)	Time (ms)	Remark
Deploy the Contract	987073	19.88	-
Verification and update	463948	8.80	one transaction
	551844	11.12	threee transactions
	639740	12.89	five transactions
Dynamic balance exchange	420000	8.46	withdrawal/recharge
Offline account cancellation	469884	9.47	-
Freeze account	21916	0.44	-

2) On-Chain Smart Contract Performance: We use Solidity to write on-chain smart contracts and deploy them to the Ethereum test network for testing, implementing the functions required for an offline payment protocol. The specific on-chain smart contract performance is shown in Table II. Deploying the on-chain smart contract to the blockchain consumes 987073gas and takes 19.88ms. The deployment of the contract takes place only once and the corresponding consumption occurs only once to invoke the functionality it is set up for. When the user has a network, offline transactions are uploaded to the chain to validate and update the status. We tested the performance of the contract to validate one, three, and five transactions respectively, and the consumption of gas is 463948, 551844, and 639740, and the time consumption is 8.80ms, 11.12ms, and 12.89ms. With the increasing number of transactions, the on-chain smart contract increases for the consumption of offline transaction information validation. Once the offline transaction information is verified, a transaction is executed on the blockchain to update the account state. In addition, for the dynamic balance exchange function of offline and on-chain accounts, the gas consumption for recharge/withdrawal is 420,000 and the time cost is 8.46ms; for the offline account cancellation function, the gas consumption is 469,884 and the time cost is 9.47ms; for the frozen offline account, the gas consumption is 21916 and the time cost is 0.44ms.

TABLE III
IMPLEMENTED FUNCTIONS COMPARISON WITH EXISTING WORKS

Implemented Functions	Scheme Name						
	Bitcoin1 [21]	Bitcoin2 [22]	LN [10]	SVLP [12]	xLumi [11]	PW [23]	Ours
Interact with smart contracts	×	×	×	✓	✓	✓	✓
Coin forgery attack	✓	✓	✓	✓	✓	✓	✓
Offline transaction data forgery	✓	✓	✓	✓	✓	×	✓
Double-spend and double-deposit attack	✓	✓	✓	✓	✓	✓	✓
Man-in-the-middle attack	✓	✓	✓	✓	✓	✓	✓
Open transaction	✓	✓	×	×	×	✓	✓
Coin redistribution	✓	✓	✓	✓	✓	×	✓
Instant settlement for offline payments	×	×	✓	✓	×	✓	✓

Through the analysis of the on-chain smart contract's gas and time consumption, the performance of our developed contract is sound and implementable.

3) Offline Account Performance: We implement the offline account functionalities in the enclave to support the proper working of the offline payment protocol. Fig. 3 shows the time consumption of the developed offline payment protocol running 50 times in the enclave. Specifically, the five specific operations of the offline payment protocol are included, namely from P1 to P5. P1 is the initialization of the offline account operation, where the time used in the enclave in generating the offline account public and private key pairs and initializing the account state is focused on 46 to 50 ms. P2 is the offline transaction operation, where the average time consumption is about 32 ms. The time consumption here only includes the time consumption for execution within the TEE, in practice the time consumption should also include the communication time between the TEE and the host, NFC or Bluetooth. P3 is the upload verification and update operations, which corresponds to the enclave time cost of submitting 3 offline transactions for upload verification and update, with an average time cost of 50ms. P4 is the dynamic balance exchange operation, with an average time cost of 37ms for recharge/withdrawal in the enclave. P5 is the offline account logout operation, with an average time cost of 55ms in the enclave.

By analyzing the time consumption of offline accounts operating in the enclave, our protocol performs well in the enclave and is implementable.

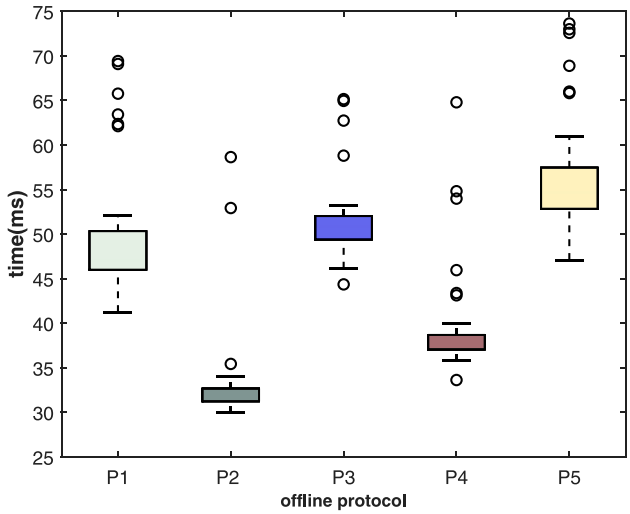


Fig. 3. Time consumption for 50 executions of the offline payment protocol in the enclave.

C. Comparison With Existing Works

This protocol implements functions in an offline payment scenario in blockchain and compares them with existing relevant blockchain offline payment solutions. The basic security requirements for offline payments are coin forgery attacks, offline transaction data forgery, double-spend and double-deposit attacks, and man-in-the-middle attacks. As well as the real flexibility requirements of users in practical application scenarios, including open transactions, offline coin redistribution, and instant settlement for offline payments, are addressed. The comparison results are shown in Table III.

From Table III, comparing with the latest blockchain offline payment solutions, it is clear that all existing protocols have some missing functionality, while our protocol is more complete than the others. This protocol meets the security requirements of offline payments while giving users the practical flexibility they need in real-world scenarios. Specifically, Bitcoin wallet [21], [22] can meet the basic security requirements, but lacks the function of instant settlement of offline payments; for [10], [11], [12], it is the payment channel that represents the offline payment solution, both have the disadvantage of not being able to achieve open transactions, and are greatly limited in the application scenario; Ethereum offline payment framework PW wallet [23] does not meet the basic security requirements, and the offline balance is non-distributable.

Our offline payment protocol can meet all the above functional requirements and satisfy both basic security requirements and flexibility requirements. We realize the advanced level of current blockchain-based offline payment solutions, with security and flexibility.

VII. Conclusion

The main goal of this paper was to determine whether two offline parties can proceed with blockchain-based offline payments without sacrificing security and flexibility. To achieve our aim, we designed a flexible and secure offline payment protocol under partial network asynchronism. Our work leverages on-chain smart contracts during good on-chain connectivity and secure off-chain transactions that leverage compatible TEEs under bad on-chain networks. We characterized a threat model for blockchain-based offline payment over an asynchronous network. We found that our construction meets the basic security requirements in practical offline payment scenarios: it is secure and robust against coin forgery attacks, offline transaction forgery, double-spend and double-deposit attacks, as well as man-in-the-middle attacks. As another finding, our protocol exhibits flexibility thanks to the use of open transactions that support coin redistribution and instant settlement. We evaluated our work under the Intel SGX and provided an empirical performance analysis of our work on the Ethereum blockchain. We provided a theoretical comparison between our design and state-of-the-art constructions. The results suggest that our protocol achieves a threshold between security and flexibility compared to existing works. The findings will be of interest to businesses, governments, academicians, and practitioners. The most important limitation lies in the fact that our scheme lack support for fully asynchronous mode: this would be a fruitful area for further work. Our work shows that there is a definite need for secure and flexible offline payment solutions that improve the user experience and widen the adoption of blockchain technology.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Bus. Rev.*, 2008, Art. no. 21260.

[2] V. Buterin et al., "A next-generation smart contract and decentralized application platform," *White Paper*, vol. 3, no. 37, pp. 2–1, 2014.

[3] "Biggest cryptocurrency in the world - Both coins and tokens - Based on market capitalization on November 11, 2022." Statista. Accessed: Nov. 2022. [Onlline]. Available: https://www.statista.com/statistics/1269013/biggest-crypto-per-category-worldwide/

[4] B. Sriman and S. G. Kumar, "Decentralized finance (DEFI): The future of finance and defi application for Ethereum blockchain based finance market," in *Proc. Int. Conf. Adv. Comput., Commun. Appl. Informat. (ACCAI)*, 2022, pp. 1–9.

[5] P. Kar, K. Chen, and J. Shi, "DMACN: A dynamic multi-attribute caching mechanism for NDN-based remote health monitoring system," *IEEE Trans. Comput.*, vol. 72, no. 5, pp. 1301–1313, May 2023.

[6] M. Xu, S. Liu, D. Yu, X. Cheng, S. Guo, and J. Yu, "Cloudchain: A cloud blockchain using shared memory consensus and RDMA," *IEEE Trans. Comput.*, vol. 71, no. 12, pp. 3242–3253, Dec. 2022.

[7] M. Xu, F. Zhao, Y. Zou, C. Liu, X. Cheng, and F. Dressler, "BLOWN: A blockchain protocol for single-hop wireless networks under adversarial SINR," *IEEE Trans. Mobile Comput.*, vol. 22, no. 8, pp. 4530–4547, Aug. 2023.

[8] M. Xu, C. Liu, Y. Zou, F. Zhao, J. Yu, and X. Cheng, "wChain: A fast fault-tolerant blockchain protocol for multihop wireless networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 10, pp. 6915–6926, Oct. 2021.

[9] V. Buterin. "Ethereum. On sharding blockchains." GitHub. Accessed: Nov. 19, 2019. [Online]. Avalable: https://github.com/ethereum/wiki/

[10] J. S. Bellagarda, "The potential effect off-chain instant payments will have on cryptocurrency scalability issues-the lightning network," in *Proc. Int. Conf. Inf. Resour. Manage.* (*CONFIRM)*, 2019, vol. 2, pp. 1–14.

[11] N. Ying and T. W. Wu, "xlumi: Payment channel protocol and off-chain payment in blockchain contract systems," 2021, *arXiv:2101.10621*.

[12] L. Zhong, Q. Wu, J. Xie, J. Li, and B. Qin, "A secure versatile light payment system based on blockchain," *Future Gener. Comput. Syst.*, vol. 93, pp. 327–337, Apr. 2019.

[13] E. Rohrer, J. Malliaris, and F. Tschorsch, "Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops (EuroS&PW)*, 2019, pp. 347–356.

[14] A. Mizrahi and A. Zohar, "Congestion attacks in payment channel networks," in *Financial Cryptography and Data Security*, N. Borisov and C. Diaz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2021, pp. 170–188.

[15] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[16] "The Raiden network." Raiden. Accessed: Jul. 2023. [Online]. Available: https://raiden.network/

[17] J. Coleman, L. Horne, and L. Xuanji. "Counterfactual: Generalized state channels." Available: https://www.bgp4.com/wp-content/uploads/2019/05/statechannels.pdf

[18] J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. Pietzuch, "Teechain: A secure payment network with asynchronous blockchain access," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, 2019, pp. 63–79.

[19] M. Elsheikh, J. Clark, and A. M. Youssef, "Short paper: Deploying payword on Ethereum," in *Proc. Int. Conf. Financial Cryptography Data Secur.*, New York, NY, USA: Springer, 2019, pp. 82–90.

[20] N. Ivanov and Q. Yan, "System-wide security for offline payment terminals," in *Proc. Int. Conf. Secur. Privacy Commun. Syst.*, New York, NY, USA: Springer, 2021, pp. 99–119.

[21] A. Dmitrienko, D. Noack, and M. Yung, "Secure wallet-assisted offline bitcoin payments with double-spender revocation," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 520–531.

[22] T. Takahashi and A. Otsuka, "Short paper: secure offline payments in bitcoin," in *Proc. Int. Conf. Financial Cryptography Data Secur.,* New York, NY, USA: Springer, 2019, pp. 12–20.

[23] I. S. Igboanusi, K. P. Dirgantoro, J.-M. Lee, and D.-S. Kim, "Blockchain side implementation of pure wallet (PW): An offline transaction architecture," *ICT Exp.*, vol. 7, no. 3, pp. 327–334, 2021.

[24] H. Wang, X. Li, J. Gao, and W. Li, "Mobt: A kleptographically-secure hierarchical-deterministic wallet for multiple offline bitcoin transactions," *Future Gener. Comput. Syst.*, vol. 101, pp. 315–326, Dec. 2019.

[25] S. Verbücheln, "How perfect offline wallets can still leak bitcoin private keys," *MCIS 2015 Proceedings*. *2*. [Online]. Available: https://aisel.aisnet.org/mcis2015/2

[26] W. Dai, J. Deng, Q. Wang, C. Cui, D. Zou, and H. Jin, "SBLWT: A secure blockchain lightweight wallet based on trustzone," *IEEE Access*, vol. 6, pp. 40638–40648, 2018.

[27] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Universally composable synchronous computation," in *Theory of Cryptography*, A. Sahai, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 477–498.

[28] J. Camenisch, S. Krenn, R. Küsters, and D. Rausch, "iUC: Flexible universal composability made simple," in *Proc. Adv. Cryptol. (ASIACRYPT)*, S. D. Galbraith and S. Moriai, Eds., Cham, Switzerland: Springer International Publishing, 2019, pp. 191–221.

[29] R. Kuesters, M. Tuengerthal, and D. Rausch, "The IITM model: A simple and expressive model for universal composability," Cryptol. ePrint Arch., Paper, 2013, 2013/025. [Online]. Available: https://eprint.iacr.org/2013/025

[30] R. Canetti, R. Pass, and A. Shelat, "Cryptography from sunspots: How to use an imperfect reference string," in *Proc. 48th Annu. IEEE Symp. Found. Comput. Sci. (FOCS)*, 2007, pp. 249–259.

[31] M. Graf, D. Rausch, V. Ronge, C. Egger, R. Küsters, and D. Schröder, "A security framework for distributed ledgers," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA: ACM, 2021, pp. 1043–1064.

[32] R. Pass, E. Shi, and F. Tramèr, "Formal abstractions for attested execution secure processors," in *Proc. Adv. Cryptol. (EUROCRYPT)*, J.-S. Coron and J. B. Nielsen, Eds., Cham, Switzerland: Springer International Publishing, 2017, pp. 260–289.

[33] F. Tramèr, F. Zhang, H. Lin, J.-P. Hubaux, A. Juels, and E. Shi, "Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, 2017, pp. 19–34.

[34] R. Canetti, A. Cohen, and Y. Lindell, "A simpler variant of universally composable security for standard multiparty computation," in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in*

Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9216. Berlin, Germany: Springer Verlag, 2015, pp. 3–22.
[35] J. Camenisch, R. R. Enderlein, S. Krenn, R. Küsters, and D. Rausch, "Universal composition with responsive environments," in *Proc. Adva. Cryptol. (ASIACRYPT)*, J. H. Cheon and T. Takagi, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 807–840.
[36] "Solidity is an object-oriented, high-level language for implementing smart contracts." Solidity. Accessed: Jul. 2023. [Online]. Available: https://docs.soliditylang.org/en/v0.8.17/
[37] Y. Shen et al., "Occlum: Secure and efficient multitasking inside a single enclave of intel SGX," in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 955–970.
[38] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *Proc. 2nd Int. Workshop Hardware Archit. Support Secur. Privacy*. New York, NY, USA: ACM, 2013, 13(7).
[39] "Web3.py is a python library for interacting with Ethereum." [Online]. Available: https://web3py.readthedocs.io/en/v5/
[40] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "The road from panama to Keccak via RadioGatún," in *Dagstuhl Seminar Proceedings* Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009.
[41] D. R. Brown, "Sec 2: Recommended elliptic curve domain parameters," *Standards for Efficient Cryptography*, 2010. [Online]. Available: https://www.secg.org/sec2-v2.pdf

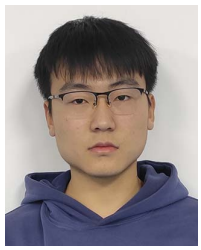
Wanqing Jie received the B.S. degree from the Department of Management Science and Engineering, Tianjin University of Finance and Economics, China, in 2020, and the M.S. degree from the Institute of Artificial Intelligence and Blockchain, Guangzhou University, China, in 2023. Currently, she is working toward the Ph.D. degree with Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, Beihang University, China. Her research interests include blockchain and privacy computing.

Wangjie Qiu (Member, IEEE) received the B.S. degree, in 2008, and the Ph.D. degree, in 2013, both in mathematics from Beihang University. He is currently an Associate Professor with Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, Beihang University. He is also the Deputy Secretary General of the blockchain committee, China Institute of Communications. His research interests include Information security, blockchain, and privacy computing. As the founding team, he successfully developed ChainMaker, an international advanced blockchain technology system. He has been authorized a number of invention patents in the fields of information security, blockchain, and privacy computing. He won the first prize of China National Technology Invention in 2014.

Arthur Sandor Voundi Koe (Member, IEEE) received the B.E. degree in network and computer maintenance from The African Institute of Computer Science (IAI), Cameroon branch, in 2010, the B.S. degree in fundamental computer science from the University of Yaoundé 1, Cameroon, in 2011, the M.S. degree in computer and application technology from Hunan University, China, in 2015, and the Ph.D. degree in computer science and application technology from Hunan University, China in 2020. He was a Postdoctoral Researcher with Guangzhou University, from 2020 to 2023. He is currently a Lecturer with Xidian University. He serves as a Reviewer for many renowned international journals. His research interests include security and privacy issues in blockchain technology, cloud computing security and privacy issues, and machine learning (adversarial learning and federated learning).

Jianhong Li was born in 1998. He received the bachelor of engineering degree in information security from Guangzhou University. He is now working toward the master's degree in cyberspace security with Guangzhou University. His research interests are blockchain and cybersecurity.

Yin Wang was born in 1999 and received the B.S. degree from the Computer Science Department of Jiangxi Agricultural University. He is now working toward the M.S. degree with Guangzhou University. His research interests are blockchain and consensus algorithms.



Yaqi Wu was born in 2000. He is currently working toward the M.Eng. degree with the Institute of Artificial Intelligence and Blockchain, Guangzhou University. His research interests include AI security and blockchain.

Jin Li (Senior Member, IEEE) received the B.S. degree, in 2002, M.S. degree, in 2004, both in mathematics, from Southwest University and Sun Yat-sen University respectively, and the Ph.D. degree in information security from Sun Yat-sen University, in 2007. He is currently a Professor and an Executive Dean of the Institute of Artificial Intelligence and Blockchain, Guangzhou University. His research interests include the design of secure protocols in cloud computing and cryptographic protocols. He has published more than 100 papers in international conferences and journals, including IEEE INFOCOM, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, and ESORICS. His work has been cited more than 18000 times at Google Scholar and the H-Index is 56. He also served as program chair and committee for many international conferences. He received NSFC Outstanding Youth Foundation in 2017.

Zhiming Zheng received the Ph.D. degree in mathematics from the School of Mathematical Sciences, Peking University, Beijing, China, in 1987. He is currently a Professor with the Institute of Artificial Intelligence, Beihang University, Beijing, China. His research interests include refined intelligence, blockchain, and privacy computing. He is one of the initiators of Blockchain-ChainMaker. He is a member of Chinese Academy of Sciences.