2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS) | 979-8-3503-8605-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDCS60910.2024.00038

CORE: Transaction Commit-Controlled Release of Private Data over Blockchains

Shan Wang[yx], Ming Yang[y], Jiannong Cao[x], Zhen Ling[y], Qiang Tang[z], Xinwen Fu[{]
[y] Southeast University. Email:{yangming2002, zhenling}@seu.edu.cn.
[x]The Hong Kong Polytechnic University. Email: {shan-cs.wang, jiannong.cao}@polyu.edu.hk.
[z]The University of Sydney. Email: qiang.tang@sydney.edu.au.
[{]University of Massachusetts Lowell. Email: Xinwen_Fu@uml.edu.

***Abstract*—In blockchain applications such as digital goods exchange, private data may be transmitted from a data owner to a recipient through a transfer transaction. However, these blockchain applications often assume the underlying blockchain system is secure and reliable, and thus do not consider transaction failures. We find that a failed transfer transaction may disclose the private data to the recipient, but the data owner may not receive tokens as payments or the ledger may not correctly record the data trail. To handle transaction failures and protect private data, we propose a novel transaction commit-controlled release (CORE) protocol. With CORE, the private data can only be obtained by an intended recipient after the transfer transaction is committed, the data owner receives tokens, and the ledger correctly records the data trail. We perform security analysis of CORE, implement CORE and evaluate its performance over representative public and permissioned blockchains. The results of our extensive experiments show CORE introduces minor overhead in terms of transaction latency and transaction fees. We are the first to identify and address the generic private data disclosure issues in both public and permissioned blockchains.**

***Index Terms*—Blockchain, Data Transfer, Private Data Leak**

I. Introduction

In blockchain applications such as digital goods exchange, cryptocurrency swap and big data sharing, *private data* may be transferred from a data owner to a recipient through a transaction, which can verify the private data via smart contracts, facilitate mandatory payment and document evidence for purposes such as auditing. For instance, the digital goods exchange protocol utilizes a hashed time-locked contract (HTLC) [1] to fairly exchange a secret key s on-chain [2], [3]. The recipient uses a hash h and a time lock T to lock tokens in the smart contract. Before time T, the owner can propose a transfer transaction that carries the secret key s, i.e., the preimage of the hash h, to withdraw the locked tokens. With a successful transfer transaction, the data owner receives the locked tokens as payment while the recipient receives s, i.e., *private data owned by the owner*. After time T, the HTLC does not permit the owner to withdraw locked tokens through transactions, and locked tokens can be refunded to the recipient.

Those blockchain applications for private data transfer often assume that the underlying blockchain system is secure and reliable, and thus do not consider transaction failures, which may cause private data leaks. In practice, a transaction may encounter failures due to various faults and vulnerabilities in a blockchain system, such as message delivery delays in an asynchronous network and execution faults at nodes. To ensure fairness [4] for an honest data owner, the recipient shall only learn the data when the corresponding transfer transaction is successfully committed to the blockchain for facilitating payment [3], [4] and documenting evidence [5]. However, in public blockchains, the private data within a failed transfer transaction may be disclosed to the public blockchain network. An adversary (including a malicious data recipient) may learn the private data but does not pay tokens. In particular, if a transfer transaction is delayed beyond the time lock T, the owner cannot receive the payment through transfer transactions any more. In permissioned blockchains, we find that the original private data is prematurely delivered to a recipient in a peer-to-peer fashion before transaction commit. When a transfer transaction fails, the recipient still obtains the private data, but the ledger fails to correctly document the trail.

In this paper, we systematically address the private data leak issues caused by transaction failures. Our major contributions are summarized as follows. We are the first to identify the generic private data disclosure issues because of failed transactions in existing applications and protocols over both public and permissioned blockchains. We propose a novel protocol named *transaction commit-controlled release (CORE)* protocol, which can protect the confidentiality of private data in case of transaction failures. CORE introduces a group of n witnesses to attest transaction commit events and employs bilinear pairing cryptography to keep the private data confidential from witnesses while ensuring that the private data can be obtained only by a specific recipient after the transfer transaction is committed. CORE also introduces threshold cryptography so as to tolerate a fraction of corrupted witnesses. A witness attests a transaction commit event by publishing a verifiable signature on the committed transaction. With any t signatures from n witnesses, a recipient can use its private key to derive the private data. In the case of transaction failure, the recipient cannot recover the private data, given well-studied cryptographic assumptions and a maximum of $t - 1$ corrupted witnesses.

In CORE, witnesses do not have access to the private data, and only provide publicly verifiable signatures on committed transactions and control the timing of private data release to a specific recipient. The private data is only shared within the

* *Corresponding author: Prof. Ming Yang of Southeast University, China.*

2575-8411/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDCS60910.2024.00038

owner and the recipient. This distinguishes CORE from secret sharing [6] and threshold cryptosystems [7], [8], as well as the witness encryption based on signatures [9]. In our context, these existing techniques cannot ensure the confidentiality of private data from witnesses and unintended parties.

We implement CORE in the representative public blockchain—Ethereum and the representative permissioned blockchain—Hyperledger Fabric [10], and evaluate its performance. We deploy seven cloud servers across different countries as witnesses with each running with limited computational resources. The overhead incurred by CORE is minor in terms of transaction latency and fees. We also conduct a large-scale analysis of the use of private data transfer transactions in mainstream blockchains, demonstrating the generality of private data transfer transactions and the need of CORE.

II. Background

In this section, we introduce the private data transfer over both public and permissioned blockchains.

A. Private Data Transfer in Public Blockchains

Hashed time-locked contracts (HTLCs) [1] are commonly used for conditional payments in public blockchains, and serve as building blocks of protocols for fair exchange of digital goods and HTLC-based atomic swap. A withdrawal transaction related to a HTLC involves transferring private data of high value.

Fair Exchange of Digital Goods. In a digital goods fair exchange protocol over blockchains, the private data transferred in a transaction is an encryption key s. In ZKCP [2] and ZKCPlus [3], a seller Alice first sends the ciphertext of digital goods off chain only to a buyer Bob, and uses zero knowledge proof (ZKP) to prove to Bob that h is the hash of the correct encryption key s without revealing s. Next, Bob and Alice follow the HTLC workflow as shown in Fig. 1 to finish the exchange of s on chain. Bob uses the hash h and a specific time T to deposit v tokens in the smart contract through a transaction $TX_{dep}(h; T; v; pk_A; sig_B)$, which specifies Alice as the intended recipient of v tokens via her public key pk_A and Bob as the sender via his signature sig_B. Before time T, Alice can propose a withdrawal transaction $TX_{wdr}(pre; sig_A)$, where the preimage pre is the key s, to withdraw the locked v tokens as the payment. The HTLC smart contract verifies if the preimage is consistent with hash h and whether the current time is prior to time T. If both conditions are met, the HTLC smart contract sends v tokens to Alice. Everyone including Bob can obtain s from the withdrawal transaction in the public blockchain. Bob can use s to recover the digital goods from the previously received ciphertext. If Alice fails to deliver a valid TX_{wdr} before T

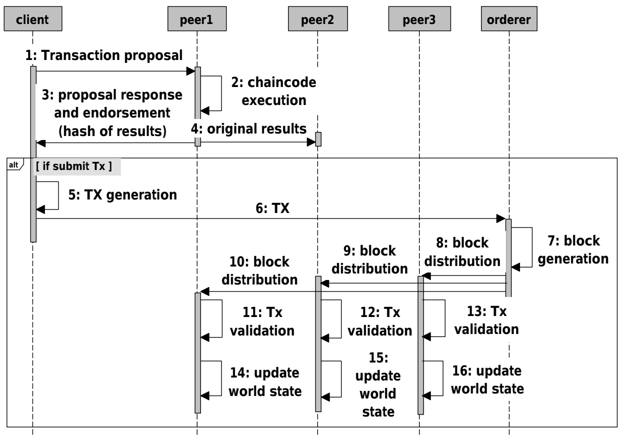

Fig. 2. Private Transaction Lifecycle in Fabric

Steps 11-16. Each peer validates transactions in the new block. Only the transaction which passes the validations, including the endorsement policy check and version check, is marked as valid [19]. Only the execution results of a valid transaction are updated to the world state database. *Peer* 2 updates the original private data (that is stored locally) to its world state. Other PDC non-member peer nodes only update the hash of the private data in the transaction to their world states.

Another permissioned blockchain **Enterprise Ethereum** previously received ciphertext, but not pay tokens to the seller. This is unfair for the seller who does not get the payment. Delivering the ciphertext of the secret key s in a transaction [11] cannot tolerate the transaction failures either, since a buyer can obtain the ciphertext from the failed transaction and use the pre-shared symmetric key to recover the secret key s without pay. (ii) In the HTLC-based atomic swap protocol, if Alice's withdrawal transaction fails, Alice cannot withdraw the Ethers locked by Bob in Ethereum after time t_2. But Bob may learn the preimage from the failed withdrawal transaction, and use the learned preimage to withdraw the BTC locked by Alice in Bitcoin. As a result, Alice loses money.

*2) **Private Data Disclosure in Permissioned Blockchains**:* The big data sharing protocol [5] in Section II-B assumes the underlying permissioned blockchain system and its private data mechanism are secure, and does not consider transaction failures either. According to the private data transaction workflow in Hyperledger Fabric in Fig. 2, the owner's *peer* 1 sends the original private data to the recipient peer in a peer-to-peer way in Step 4 before generation of the transaction. We find that the transaction may fail in multiple cases. (i) *Message delivery delay*. During Step 3, if the proposal response is delayed and not received within the timeout of 30 seconds by default, the client will not proceed with generating and submitting the transaction to orderers in Step 5. (ii) *Node execution faults*. In Steps 11-13, the transaction may be marked as invalid by peer nodes due to endorsement policy check errors or version check errors before updating the execution results to the ledger. For example, an endorsement policy may not permit the transfer transaction with only one endorsement [32].

The private data transaction in an Enterprise Ethereum network has the similar private data disclosure problem as Fabric, since the private data is also delivered to the recipient in a peer-to-peer way before the transaction commit.

Implication. In all these transaction failure cases in the permissioned blockchains, the private data is prematurely disclosed to the recipient, but the transaction is not committed to the blockchain. That is, the recipient obtains the private data but the blockchain fails to correctly document the trail. The big data sharing protocol [5] does not handle such transaction failures, which will lead to improper chain of custody and owner unfairness.

C. Security Goals

To handle the transaction failure cases and enforce owner fairness, we propose a *transaction commit controlled release (CORE)* protocol, which ensures that private data m is obtained by a specific recipient only after TX_p is committed.

Security Assumption. We assume an asynchronous communication model, where the message delivery time is uncertain. Our protocol, unlike existing ones, does not rely on a deterministic message delivery bound for its execution. Transactions may fail due to message delivery delays and execution errors in a blockchain network. The data owner honestly transfers a *correct* data, since the data correctness can be verified by the recipient before the protocol execution as discussed in Sec. VII.

The ideal functionality F interacts with a private data owner O identified by a blockchain address id_O, a blockchain network B running an arbiter smart contract L, a private data recipient R, and a simulator Sim.

- **Propose Transaction** TX_p. On receiving $(send; TX_p(m; R))$ from O, where $TX_p(m; R)$ is a transaction for transferring private data m to R, leak $(send; R; id_O)$ to Sim.
- **Validate Transaction** TX_p. Send $TX_p(m; R)$ to blockchain network B for validating TX_p, and obtain an boolean indicator I_{tx} that indicates the validity of TX_p. Output I_{tx} and leak $TX_p(R)$ to Sim.
- **Reveal Private Data.** If TX_p is committed to Blockchain B as a valid transaction, i.e., I_{tx} = 1, reveal m only to the recipient R. Otherwise, withhold m.

Fig. 3. Ideal Functionality F of CORE

Security Definition. We define the security of CORE following the simulation-based formulation paradigm [21], designing a real-world protocol Π to achieve an ideal functionality F, i.e., the security goals. In the real world, parties who interact with Π may be corrupted by a probabilistic polynomial-time (P.P.T.) adversary A. In the ideal world, an ideal protocol interacts with honest parties and a P.P.T. simulator Sim. If the simulator Sim in the ideal world can simulate a view that is computationally indistinguishable from the view in the real world for A, it is said that Π securely realizes the security goals F.

As shown in Fig. 3, we define the ideal functionality F for CORE. On receiving a $(send; TX_p(m; R))$ message from owner O, send the transaction $TX_p(m; R)$ to blockchain network B. Only the event $(send; R; id_O)$ is revealed to Sim, and m is kept confidential; Then, B runs a consensus protocol to include TX_p into a block, and runs an arbiter smart contract L to validate the data in TX_p. A transaction validity indicator I_{tx} is output. The data m is kept confidential to Sim; Finally, if I_{tx} = 1, m is revealed only to the recipient R. If I_{tx} = 0, other parties including R cannot obtain the data m of owner O, even when TX_p is sent out but is delayed and fails.

Based on the ideal functionality F, the security of a real-world protocol Π of CORE is defined as follows.

Definition 1. (Security of Π**)** Let $IDEAL^L_{F,Sim}$ denote the execution of functionality F, and $REAL^L_{\Pi,A}$ denote the execution of protocol Π. Π is said to securely realize F if 9 a P.P.T. Sim, s.t. the following holds for 8 P.P.T. adversary A,

$$IDEAL^L_{F,Sim} \approx REAL^L_{\Pi,A}: \quad (1)$$

According to the functionalities in F, a real-world protocol Π that securely realizes F can rigorously guarantee that the private data m remains confidential in case of transaction failures, and is only obtained by a specific recipient when transaction TX_p is committed, thereby ensuring the owner fairness.

IV. Real-World Protocol of CORE

In this section, we introduce a real-world protocol Π that implements the ideal functionality F of CORE. We first present the basic idea and then introduce cryptographic preliminaries. Finally, we present the protocol Π in detail.

A. Basic Idea

We design a transaction commit-controlled release (CORE) protocol Π, in which the release of private data is controlled by a transaction commit event. CORE employs a group of n blockchain witnesses to attest transaction commit events, adopts bilinear pairing cryptography to control the timing of releasing private data to only an intended recipient while keeping the private data confidential from all witnesses, and utilizes threshold cryptography to tolerate $t - 1$ malicious witnesses.

As shown in Fig. 4, in *Steps 1-2*, an owner O first generates a random number r, uses bilinear pairing to create a symmetric key K as a function of the random number, the recipient's public key and the owner's blockchain address, and encrypts the private data m with the generated symmetric key K. The owner then creates a private data transfer transaction $TX_p^{core}(r; c; R)$ with r and the ciphertext c as its fields. At this point, the recipient R cannot derive the key K and cannot recover private data m even though R already gets c in the transaction. *Step 3*. The blockchain witnesses monitor the blockchain continuously. When a witness finds that transaction $TX_p^{core}(r; c; R)$ is committed, the witness signs the random number r and the owner's blockchain address in the transaction and publishes the signature as a *commit confirmation key (CK)*, which is publicly verifiable. If $TX_{\mathbf{p}}^{core}$ fails, a witness will not generate a signature on this transaction. *Step 4*. Once the recipient observes a number of t or more CKs, the recipient can use its private key and any t published CKs to derive the symmetric key K and thus recover the private data m from

ciphertT2419.275 Tm[(not)-354.004(generate)-353.994(a)-35350 0 1 4865 3997(Tm[c 642I(e)92799enerate)-353.9257.97nerifiable. If

orenthetransaction527T20om566T 0 0 to -34004990y trans01(data)] TJETBT89.1 9.20om566T 0 0 a2 trans4.004(4)] TJETBT97n9 6T20om566T 0 0 su4h543400499

CORE protocol Π runs over a blockchain network B with an arbiter smart contract L and n witnesses. Π interacts with a data owner O identified by id_O, and a data recipient R. The generator P of $\mathbb{G}_1$ is a public parameter.

Initiate Keys

- Witness W_i, $i = 1, 2, \ldots, n$ runs algorithm $(\mathtt{SK}_w^i, \mathtt{PK}_w^i) \leftarrow \mathtt{WitnessKeyGen}(\mathbb{G}_1, \mathbb{Z}_p)$ to derive its key pair.
- Recipient R runs algorithm $(\mathtt{SK}_u, \mathtt{PK}_u) \leftarrow \mathtt{UserKeyGen}(\mathtt{PK}_w^1, \mathtt{PK}_w^2, \ldots, \mathtt{PK}_w^n, \mathbb{Z}_p)$ to initiate its key pair.

Propose Transaction TX_p^{core}

- Owner O generates a transaction reference as $r \leftarrow \mathtt{ReferGen}(\lambda)$.
- Owner O encrypts private data m, and gets its ciphertext by running the algorithm $c \leftarrow \mathtt{Enc}(m, \mathtt{PK}_u, r, id_O)$. O puts c and r as parameters of transaction $TX_p^{core}(r, c, R)$. Then owner O sends a request $(send, TX_p^{core}(r, c, R))$ to Π.
- On receiving $(send, TX_p^{core}(r, c, R))$, send $TX_p^{core}(r, c, R)$ to the blockchain network B.

Validate Transaction TX_p^{core}

- B orders transaction TX_p^{core} following a consensus protocol. An arbiter smart contract L verifies data in the transaction and/or manages tokens. B outputs I_{tx} which indicates the validity of TX_p^{core}.
- Each witness W_i, $i = 1, 2, \ldots, n$ continually monitors and confirms committed transactions. Only if the transaction TX_p^{core} is committed, W_i publishes a commit confirmation key as $\mathtt{CK}_i \leftarrow \mathtt{CmtConfirm}(TX_p^{core}, \mathtt{SK}_w^i)$.

Reveal Private Data

- Recipient R obtains c from a valid transaction TX_p^{core}, and gets the published $\mathtt{CK}_i$ from W_i. When R obtains t or more commit confirmation keys, R recovers the private data by running algorithm $m \leftarrow \mathtt{Dec}(c, \mathtt{SK}_u, CK^{1,w}, CK^{2,w}, \ldots, CK^{t,w})$. With insufficient CKs, R cannot derive the private data.

Fig. 5. A Real-World Protocol Π of CORE

A_{i0} is a part of $\mathtt{PK}_w^i$. Please note that others such as an owner can verify whether the recipient's public key $\mathtt{PK}_u$ is derived based on the witnesses' public keys by checking if $\prod_{i=1}^{n} e(uP, A_{i0}) = e(P, u\sum_{i=1}^{n} A_{i0})$ holds. A correct $\mathtt{PK}_u$ ensures that the recipient has to obtain signatures, i.e., commit confirmation keys, of witnesses to perform decryption.

2) Propose Transaction TX_p^{core}: When a private data owner O, who is identified by a blockchain address id_O, wants to transfer the private data m to a recipient R through a transaction, the owner first derives a transaction reference r and runs algorithm $r \leftarrow$ **ReferGen**(λ). Reference r is generated by a random number generator $\mathtt{PRNG}(\cdot)$, and $r = \{0,1\}^\lambda$.

Then the owner encrypts the private data m using the transaction reference r, its identifier id_O and the public key $\mathtt{PK}_u$ of a specific recipient, and obtains the ciphertext c. The encryption algorithm $c \leftarrow$ **Enc**$(m, \mathtt{PK}_u, r, id_O)$ has four main steps. (i) The owner verifies the correctness of the recipient's public key $\mathtt{PK}_u$, as introduced previously. If it is correct, the encryption continues. (ii) The owner randomly selects $k \in \mathbb{Z}_p$, and calculates kP. (iii) The owner concatenates id_O after the reference r and gets $\sigma = r||id_O$, where $||$ denotes the concatenation of two strings. Then the owner derives a symmetric key as $K = e(ku\sum_{i=1}^{n} A_{i0}, H^1(\sigma))$; (iv) The owner uses the symmetric key K to encrypt private data m, and obtains the ciphertext $c = \langle kP, C_m \rangle$, where $C_m = m \oplus H^2(K)$. As a result, the decryption of such a ciphertext will require both the witnesses' signatures on $r||id_O$ and the recipient's private key u. Now, the owner can use r and ciphertext c as two parameters to create a transaction $TX_p^{core}(r, c, R)$, and trustingly broadcast this enforced transaction to the blockchain network. Please note that, the owner's address id_O is assigned by the blockchain system to the initiator address field of TX_p^{core} such as the *From* field in an Ethereum transaction.

Choice of transaction reference r. Transaction hash cannot perform as r since the owner needs r for encryption before a transaction is created. The $Timestamp$ field in a transaction cannot work as r since two transactions may have the same $Timestamp$. We use a strong random number generator $\mathtt{PRNG}(\cdot)$ to generate the transaction reference r which negligibly repeats itself. A random number as r is generic and applicable to mainstream blockchains such as Ethereum, Hyperledger Fabric and so on.

3) Validate Transaction TX_p^{core}: The blockchain network B follows a consensus protocol to bundle the transaction into a new block, and runs an arbiter smart contract L to verify data in TX_p^{core} and manage tokens. The time it takes to commit a transaction to the ledger is undeterministic, and the transaction may fail due to faults and vulnerabilities in the blockchain system as analyzed in Section III-B.

The witness W_i, $i = 1, 2, \ldots, n$ keeps monitoring the blockchain network, and periodically (one period is one block) confirms the commit of new transactions in a new block. After the private data transfer transaction $TX_p^{core}(r, c, R)$ is committed, witness W_i runs algorithm $\mathtt{CK}_i \leftarrow$ **CmtConfirm**$(\mathtt{TX}_p^{core}, \mathtt{SK}_w^i)$ to attest to the commit, regardless of how long the commit process takes. W_i parses the reference r and the transaction initiator (the owner) identifier id_O in the fields of committed TX_p^{core}, and gets $\sigma = r||id_O$. W_i uses its private key $\mathtt{SK}_w^i$ to sign σ, obtains a *commit confirmation key* as $\mathtt{CK}_i = s_i H^1(\sigma)$, and publishes the $\mathtt{CK}_i$ on any public bulletin board (or the internet). $\mathtt{CK}_i$ is publicly verifiable. Anyone can verify $\mathtt{CK}_i$ by checking if $e(s_iP, H^1(\sigma)) = e(P, \mathtt{CK}_i)$ holds. An honest witness does not sign a transaction that fails or has not passed the validation, e.g., a failed HTLC withdrawal transaction that has surpassed the time lock T.

4) Reveal Private Data: If TX_p^{core} is successfully committed and t (or more) witnesses publish $CK^{i,w}$, $i = 1,2,\ldots,t$ and $w \in \{1,2,\ldots,n\}$, the recipient can recover the private data by running the algorithm $m \leftarrow \textbf{Dec}(c, \mathtt{SK_u}, CK^{1,w}, CK^{2,w}, \ldots, CK^{t,w})$. $CK^{i,w} = (x_i, y_i)$ where $x_i = w$ and $y_i = \mathtt{CK}_w$, and $CK^{i,w}$ is from the w-th witness. Recipient R obtains the ciphertext c from a committed $TX_p^{core}(r, c, R)$ in the ledger, and obtains commit confirmation keys from a public bulletin board or directly from witnesses. The recipient first uses its private key $\mathtt{SK_u}$ and any t published commit confirmation keys $\{CK^{i,w}, i = 1,2,\ldots,t\}$ to recover the symmetric key, i.e.,

$$K' = e(kP, \sum_{i=1}^{t} y_i \prod_{j=1, j \neq i}^{t} \frac{x_j}{x_j - x_i})^{\mathsf{SK}_u}. \tag{2}$$

K' is actually equal to the owner generated symmetric key K. Then the recipient can obtain the private data by calculating $m = C_m \oplus H^2(K')$. With less than t commit confirmation keys, recipient R cannot recover the private data m.

Correctness Analysis. We now analyze the correctness of the algorithm Dec. Assume $f(x) = \sum_{i=1}^{n} f_i(x)$. Then the W_i's secret key $s_i = f(i)$ and $f(0) = \sum_{i=1}^{n} a_{i0}$. According to Lagrange Interpolation Theorem, $f(0)$ can be obtained by t points on polynomial $f(x)$ of degree $t-1$. We have

$$\begin{aligned} K' &= e(kP, H^1(\sigma) \sum_{i=1}^{t} f(x_i) \prod_{j=1, j \neq i}^{t} \frac{x_j}{x_j - x_i})^{u} \\ &= e(P, H^1(\sigma))^{kuf(0)}. \end{aligned} \tag{3}$$

Similarly, we have

$$K = e(ku \sum_{i=1}^{n} (a_{i0}P), H^1(\sigma)) = e(P, H^1(\sigma))^{ku \sum_{i=1}^{n} a_{i0}}. \tag{4}$$

Therefore, we have $K' = K$, and $C_m \oplus H^2(K') = m \oplus H^2(K) \oplus H^2(K') = m$. The decryption algorithm is correct.

According to equations (3) and (4), $\sigma = r || id_O$ in algorithms **Enc** and **CmtConfirm** ensures that the ciphertext c can only be decrypted through a transfer transaction $TX_p^{core}(r, c)$ initiated by the owner O with the identifier id_O. Please note that a blockchain system does not allow other entities than the owner O to use id_O in the initiator field of a transaction.

D. Witnesses Selection and Incentive

Several properties of CORE allow the blockchain community to construct witness services by majority-honest committees, to ensure the data security of the *special* type of private data transfer transactions with a profit motive. First, private data remains confidential to all witnesses, and CORE can tolerate a fraction of malicious witnesses. Second, commit confirmation keys from witnesses are publicly verifiable with witnesses' public keys and the reference and initiator address of a committed transaction, making it easy to audit witness behaviors. Third, the role of a witness is limited to signing the reference of a committed transaction. Its workload is minor as demonstrated in Section VI. In public blockchains like Ethereum, existing RPC service [33] providers such as *Infura* could further integrate the witness services to expand business and attract more users. Recipients could subscribe the witness service for querying the commit confirmation keys like subscribing RPC services for querying blockchain states. Organizations involved in a permissioned blockchain could perform as witnesses, such as hospitals and research institutions in a Fabric-based healthcare data sharing system [5].

V. Security Analysis

In this section, we first formally prove that the real-world protocol Π of CORE is secure, and then analyse that CORE can resist faults and failures in a blockchain system.

A. Security Proof

Theorem 1. *The real-world CORE protocol Π securely realizes the ideal functionality F and is secure under Definition 1, given secure cryptographic primitives and a maximum of $t-1$ malicious witnesses.*

Proof. We show that the ideal world and the real world are computationally indistinguishable for adversary A which runs P.P.T. algorithms. Adversary A can get the inputs and outputs of the corrupted parties. The owner is honest as discussed in Sec. III-C. The adversary can corrupt at most $t-1$ witnesses and/or the recipient. We consider private keys of honest parties are secure because of the DL assumption. To formally prove Theorem 1, we construct simulators for each possible corruption case and prove that Sim can simulate views where A cannot distinguish the real world from the ideal world.

Case 1. ***$t-1$ Malicious Witnesses and Malicious Recipient.*** There exists a P.P.T. simulator Sim_{WR} such that for adversary A that corrupts both the recipient and at most $t-1$ witnesses, it holds that the view of Π in the presence of adversary A is computationally distinguishable from the view in the ideal world with Sim_{WR}. Sim_{WR} works as follows.

1) Sim_{WR} invokes F and obtains the outputs of F including the transaction validity indicator I_{tx}, data m and id_O.
2) Sim_{WR} samples the random keys for witness W_i, $i = 1,2,\ldots,n$ by running algorithm ($\widetilde{\text{SKET}}$ Tm i T T itness T I

If A generates and sends out $t-1$ corrupted commit confirmation keys, Sim_{WR} emulates its behavior by sending $t-1$ corrupted commit confirmation keys to the recipient.

We now prove that A cannot distinguish the view in the ideal world from the real-world Π executions. Due to the DDH and DBDH assumptions, the simulated keys and ciphertext c^0 in the ideal world and the keys and ciphertext c in the real world are computationally indistinguishable for adversary A.

In case of $I_{tx} = 1$, A can recover m from the ciphertext in both the ideal and real worlds.

In case of $I_{tx} = 0$ and A does not generate commit confirmation keys, A will not obtain keys and cannot derive m in both ideal and real worlds.

In case of $I_{tx} = 0$ and A sends $t-1$ commit confirmation keys to the recipient, A cannot derive m in both ideal and real worlds. Recall that $H^1(\sigma) \in \mathbb{G}_1$, and assume $H^1(\sigma) = yP$. The key $K = e(P, H^1(\sigma))^{kuf(0)}$, where $f(0) = \sum_{i=1}^{n} a_{i0}$. $f(0)$ is unknown since $t-1$ witnesses cannot recover $f(0)$ or $f(0)P$ according to Lagrange interpolation theorem. It is computationally hard to find $e(P, P)^{ykf(0)}$ without knowing y, k and $f(0)$ according to the BDH assumption. Consequently, A cannot find $e(P, P)^{ykuf(0)}$ or recover m, even if A has $\widetilde{SK}_u$. $IDEAL^L_{F,\mathrm{Sim}} \approx REAL^L_{\Pi,A}$ holds.

Case 2. $t-1$ ***Malicious Witnesses.*** Adversary A corrupts at most $t-1$ witnesses and the recipient is honest. The simulation and the proof are similar to those in Case 1. A simulator Sim_W generates a transaction $\widetilde{TX}_p^{core}(r^0, c^0, R)$ and sends it to A. $IDEAL^L_{F,\mathrm{Sim}} \approx REAL^L_{\Pi,A}$ holds due to DBDH.

Specially, when $I_{tx} = 1$, A can get $t-1$ commit confirmation keys from corrupted witnesses, and may also get one or more valid commit confirmation keys from honest witnesses since they are publicly verifiable. Sim_W sends t or more $\widetilde{CK}^{i,w}$ to A. We prove that even A gets t or more $\widetilde{CK}^{i,w}$, A cannot obtain m in both the real and ideal worlds. $K = e(P, \sum_{i=1}^{t} y_i \prod_{j=1, j\neq i}^{t} \frac{x_j}{x_j - x_i})^{ku}$, where $y_i = \widetilde{\mathrm{CK}}_w = s_w H^1(\sigma)$. Assume $\sum_{i=1}^{t} y_i \prod_{j=1, j\neq i}^{t} \frac{x_j}{x_j - x_i} = xP$. Without knowing x, k and $\widetilde{SK}_u$, it is computationally hard to find $e(P, P)^{xku}$ given P, kP, xP and uP. That is, it is computationally hard for A to obtain symmetric key K or recover m, though A knows t or more commit confirmation keys.

Case 3. Malicious Recipient. Adversary A only corrupts the recipient, and witnesses are honest. The simulation and the proof are similar to those in Case 1 about simulating messages to the recipient. If $I_{tx} = 1$, a simulator Sim_R generates $\{\widetilde{CK}^{i,w}, i = 1, 2, \dots, t\}$ and $\widetilde{TX}^{core}$

TABLE II
Distribution of Witnesses and the Corresponding RPC Services in Ethereum

Witness Num	1	2	3	4
Location	New Jersey	California	London	Seoul
RPC Service	Infura	BlockPI	Alchemy	Ankr
Witness Num	5	6	7	
Location	Singapore	Sydney	Toronto	
RPC Service	Blast	OnFinality	Omnia	

Fig. 6. Performance of Six Algorithms

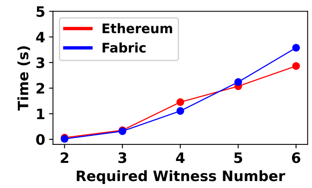

Fig. 7. Time of Getting Data vs. Number of Required Witnesses

witness server can be queried with a *HTTP Get* request with a transaction hash for the corresponding CK.

Hyperledger Fabric. We work on Hyperledger Fabric v2.3.3, and build a test network with 7 nodes on multiple *Vultr* cloud servers with Ubuntu 18.04 and 8GB memory. The block generation period is set as 10s. A witness is developed in *Golang* with a *CouchDB* database, and interacts with the test network with the Golang Fabric SDK. The Fabric witness queries the latest block every 5 seconds, then parses the block and derives and stores CKs. It also provides a *HTTP* service for users querying the published CKs.

B. Performance

We use the same test data set that contains private data of different sizes to evaluate the performance of CORE in different blockchains, and run each case 20 times.

Algorithms Performance. The boxplots in Fig. 6 shows the performance of the developed six algorithms of a (4*;* 7)-threshold CORE written in *Golang* on a computer running Ubuntu 18.04 with 32 GB memory. It takes less than 3.5 ms for each algorithm to perform their work efficiently.

Performance in Public Blockchain. In Ethereum's *Goerli* test network, we deploy a HTLC smart contract in *Solidity*. We apply a (4, 7)-threshold CORE to its withdrawal transaction, i.e., a private data transfer transaction. Fig. 8 shows that CORE incurs negligible transaction latency overhead. When a recipient attempts to obtain private data once the transaction is committed, CORE incurs some overhead due to time cost of collecting sufficient CKs from witnesses, although the overhead amount is relatively small. Fig. 9 illustrates that the withdrawal transactions with CORE incur more gas cost, because we deliver additional data such as the transaction reference in the transaction. Assuming the gas price is 50 *gwei*, we calculate the cost in terms of Ether. It can be observed that the monetary cost overhead is acceptable. Our HTLC contract and the related withdrawal transactions can be found via the contract address *0x4D46599A814bfd8fBE629F969a115F0104bcfb9C* through the *Goerli EtherScan* explorer.

Performance in Permissioned Blockchain. In the Hyperledger Fabric test network, we deploy an official smart contract example in *Golang* which involves private data transfer transactions. Fig. 10 shows that CORE has negligible overhead in the private transactions latency in Hyperledger Fabric. Please note that a transaction in Fabric does not have

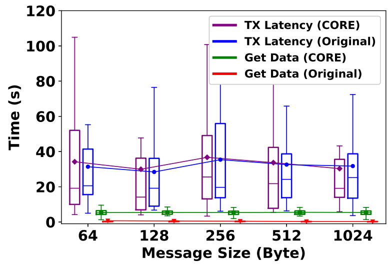

Fig. 8. Transaction Latency in Ethereum.

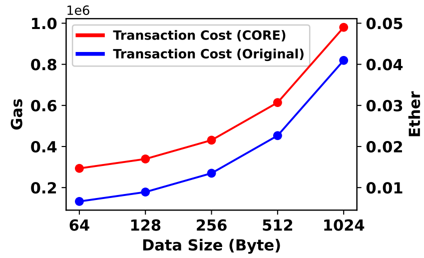

Fig. 9. Transaction Gas Cost in Ethereum.

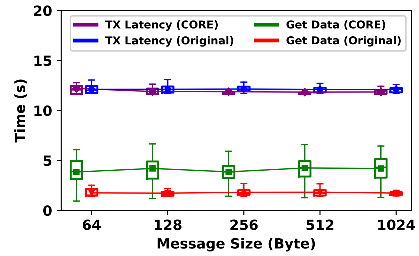

Fig. 10. Transaction Latency in Fabric.

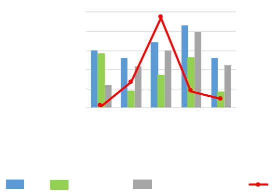

Fig. 11. HTLCs in Ethereum

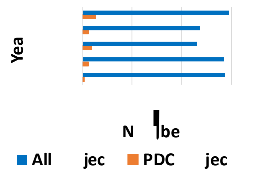

Fig. 12. Hyperledger Fabric Projects with private data

following techniques like zero knowledge proof (ZKP) used by existing protocols outlined in Section II. Prior to transferring the data, the data owner can utilize ZKP [2] to prove to the recipient that h is the hash of the ciphtertext c, as well as c is derived from the correct private data m using CORE, without disclosing either the private data or the ciphtertext. The proof can be completed off-chain and will not interfere with our CORE protocol. Subsequently, the recipient can use h to lock tokens and utilize an arbiter smart contract to validate weather the delivered data matches the hash h. A successfully committed CORE-enforced transfer transaction indicates that the owner has disclosed the correct c matching with hash h, allowing for the recovery of the correct m from c.

VIII. Related Work

We now compare CORE with related work that shares some similarities but cannot address the private data leak issues.

Timed Release Encryption (TRE) solves the problem of sending information into the future. Witness encryption [34] based TRE requires several hours for encryption and decryption [35]. A trusted time server based TRE [36] is efficient, but cannot tolerate a single-point failure. It can send a message to a future time like *11:59PM EDT, August 1, 2024*, but is not well-suited for the blockchain context because the commit time of a transaction is unpredictable.

Threshold Cryptosystem. In secret sharing [6], a secret is divided into multiple shares. A predetermined number of participants with shares can reconstruct the original secret. In a threshold cryptosystem [7], [8], the decryption key of a ciphertext is shared among n parties. Any t out of n parties can work together to recover the plaintext. With these typical constructions, private data cannot be kept invisible to witnesses as our CORE does. In addition, these constructions do not have an appropriate parameter to perform as the transaction introduced to monitor the blockchain network and generate commit confirmation keys for a committed transaction. A recipient has to obtain t out of n commit confirmation keys to recover the private data from the ciphertext transferred in transactions and can only derive the private data when the transfer transaction does not fail and is committed. In CORE, witnesses cannot derive the private data. Our extensive analysis and experiments validate the security and performance of CORE.

ACKNOWLEDGMENT

This research was supported in part by National Key R&D Program of China (No. 2023YFC3605804), National Natural Science Foundation of China (Nos. 62072103, 622322004), Jiangsu Provincial Key R&D Programs (Nos. BE2021729, BE2022680, BE2022065-5), HK RGC Collaborative Research Fund (No. C2004-12GF), HK RGC Research Impact Fund (No. R5034-18), US National Science Foundation Awards (Nos. 2325451, 1931871, 1915780), and Research Institute for Artificial Intelligence of Things, The Hong Kong Polytechnic University. Any opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] S. Wadhwa, J. Stoeter, F. Zhang, and K. Nayak, "He-htlc: Revisiting incentives in htlc," *Cryptology ePrint Archive*, 2022.

[2] Wiki, "Zero knowledge contingent payment," 2020. [Online]. Available: https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment

[3] Y. Li, C. Ye, Y. Hu, I. Morpheus, Y. Guo, C. Zhang, Y. Zhang, Z. Sun, Y. Lu, and H. Wang, "Zkcplus: Optimized fair-exchange protocol supporting practical and flexible data exchange," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[4] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[5] S. Wang, M. Yang, T. Ge, Y. Luo, and X. Fu, "Bbs: A blockchain big-data sharing system," in *ICC 2022-IEEE International Conference on Communications*. IEEE, 2022, pp. 1–6.

[6] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *Journal of Cryptology*, vol. 20, pp. 51–83, 2007.

[7] J. Baek and Y. Zheng, "Simple and efficient threshold cryptosystem from the gap diffie-hellman group," in *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*, vol. 3. IEEE, 2003, pp. 1491–1495.

[8] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," *Journal of Cryptology*, vol. 15, no. 2, 2002.

[9] V. Madathil, S. A. Thyagarajan, D. Vasilopoulos, L. Fournier, G. Malavolta, and P. Moreno-Sanchez, "Cryptographic oracle-based conditional payments," in *NDSS*, 2023.

[10] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the 13th EuroSys Conference*, 2018.

[11] R. Song, S. Gao, Y. Song, and B. Xiao, "Zkdet: A traceable and privacy-preserving data exchange scheme based on non-fungible token and zero-knowledge," in *2022 IEEE 42st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2022.

[12] F. Chen, J. Wang, C. Jiang, T. Xiang, and Y. Yang, "Blockchain based non-repudiable iot data trading: Simpler, faster, and cheaper," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2022.

[13] S. He, Y. Lu, Q. Tang, G. Wang, and C. Q. Wu, "Blockchain-based p2p content delivery with monetary incentivization and fairness guarantee," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, 2022.

[14] Y. Lu, Q. Tang, and G. Wang, "Zebralancer: Private and anonymous crowdsourcing system atop open blockchain," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2018.

[15] ——, "Dragoon: Private decentralized hits made practical," in *IEEE International Conference on Distributed Computing Systems*, 2020.

[16] M. Herlihy, "Atomic cross-chain swaps," in *Proceedings of the 2018 ACM symposium on principles of distributed computing*, 2018.

[17] R. van der Meyden, "On the specification and verification of atomic swap smart contracts," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2019, pp. 176–179.

[18] J.-Y. Zie, J.-C. Deneuville, J. Briffaut, and B. Nguyen, "Extending atomic cross-chain swaps," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2019, pp. 219–229.

[19] S. Wang, M. Yang, Y. Zhang, Y. Luo, T. Ge, X. Fu, and W. Zhao, "On private data collection of hyperledger fabric," in *IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021.

[20] Ethereum, "Ethereum mainnet for enterprise," 2021, [Online]. (Accessed 13 June 2021). [Online]. Available: https://ethereum.org/en/enterprise/

[21] Y. Lindell, "How to simulate it–a tutorial on the simulation proof technique," *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, pp. 277–346, 2017.

[22] L. Luu, R. Saha, I. Parameshwaran, P. Saxena, and A. Hobor, "On power splitting games in distributed computation: The case of bitcoin pooled mining," in *IEEE Computer Security Foundations Symposium*, 2015.

[23] Y. Kwon, D. Kim, Y. Son, E. Vasserman, and Y. Kim, "Be selfish and avoid dilemmas: Fork after withholding (faw) attacks on bitcoin," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 195–209.

[24] S. Gao, Z. Li, Z. Peng, and B. Xiao, "Power adjusting and bribery racing: Novel mining attacks in the bitcoin system," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 833–850.

[25] M. Saad, A. Anwar, S. Ravi, and D. Mohaisen, "Revisiting nakamoto consensus in asynchronous networks: A comprehensive analysis of bitcoin safety and chainquality," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[26] A. Lewis-Pye and T. Roughgarden, "How does blockchain security dictate blockchain implementation?" in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[27] Alchemy, "Ethereum transactions - pending, mined, dropped & replaced," 2023. [Online]. Available: https://docs.alchemy.com/docs/ethereum-transactions-pending-mined-dropped-replaced

[28] F. Winzer, B. Herd, and S. Faust, "Temporary censorship attacks in the presence of rational miners," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2019.

[29] T. Nadahalli, M. Khabbazian, and R. Wattenhofer, "Timelocked bribing," in *International Conference on Financial Cryptography and Data Security*. Springer, 2021, pp. 53–72.

[30] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 129–144.

[31] M. Tran, I. Choi, G. J. Moon, A. V. Vu, and M. S. Kang, "A stealthier partitioning attack against bitcoin peer-to-peer network," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 894–909.

[32] S. Wang, M. Yang, B. Pearson, T. Ge, X. Fu, and W. Zhao, "On security of proof-of-policy (pop) in the execute-order-validate blockchain paradigm," in *2022 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2022, pp. 317–325.

[33] K. Li, J. Chen, X. Liu, Y. R. Tang, X. Wang, and X. Luo, "As strong as its weakest link: How to break blockchain dapps at rpc service." in *NDSS*, 2021.

[34] S. Garg, C. Gentry, A. Sahai, and B. Waters, "Witness encryption and its applications," in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, 2013, pp. 467–476.

[35] G. Uberti, K. Luo, O. Cheng, and W. Goh, "Building usable witness encryption," *arXiv preprint arXiv:2112.04581*, 2021.

[36] A.-F. Chan and I. F. Blake, "Scalable, server-passive, user-anonymous timed release cryptography," in *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*. IEEE, 2005, pp. 504–513.

[37] D. Nunez, "Umbral: a threshold proxy re-encryption scheme," *NuCypher Inc and NICS Lab, University of Malaga, Spain*, 2018.

[38] P. Zhang, J. Wei, Y. Liu, and H. Liu, "Proxy re-encryption based fair trade protocol for digital goods transactions via smart contracts," *arXiv preprint arXiv:2306.01299*, 2023.

[39] Y. Xiao, N. Zhang, J. Li, W. Lou, and Y. T. Hou, "Privacyguard: Enforcing private data usage control with blockchain and attested off-chain contract execution," in *The 25th European Symposium on Research in Computer Security (ESORICS)*, 2020.