IvyCross: A Privacy-Preserving and Concurrency Control Framework for Blockchain Interoperability

Ming Li, Jian Weng*, *Member, IEEE*, Yi Li, Yongdong Wu, Jiasi Weng, Dingcheng Li, Guowen Xu, Robert Deng, *Fellow, IEEE*

***Abstract*—Interoperability is a fundamental challenge for long-envisioned blockchain applications. A mainstream approach is to use Trusted Execution Environment (TEEs) to support interoperable off-chain execution. However, this incurs multiple TEEs configured with non-trivial storage capabilities running on fragile concurrent processing environments, rendering current strategies based on TEEs far from practical. The purpose of this paper is to fill this gap and design a practical interoperability mechanism with simplified TEEs as the underlying architecture. Specifically, we present IvyCross, a TEE-based framework that achieves low-cost, privacy-preserving, and race-free blockchain interoperability. Specifically, IvyCross allows running arbitrary smart contracts across heterogenous blockchains atop only two distributed TEE-powered hosts. We design an incentive scheme based on smart contracts to stimulate the honest behavior of two hosts, bypassing the requirement of the number of TEEs and large memory storage. We examine the conditions to guarantee the uniqueness of the Nash Equilibrium via Sequential Game Theory. Furthermore, a novel extended optimistic concurrency control protocol is designed to guarantee the correctness of concurrent execution of off-chain contracts. We formally prove the security of IvyCross in the Universal Composability framework and implement a proof-of-concept prototype atop Bitcoin, Ethereum, and FISCO BOCS. The experiments indicate that (i) IvyCross is able to support privacy-preserving and multiple-round smart contracts for cross-chain communication; (ii) IvyCross successfully decreases the off-chain costs on storage and communication of a TEE without using complex cryptographic primitives; and (iii) the total on-chain transaction fees in cross-chain communication are relatively low, within ranges of 0.2 USD $\sim$ 1 USD.**

***Index Terms*—Blockchain interoperability, privacy-preserving, smart contracts, TEE.**

I. Introduction

Over the last decade, blockchain technology has become a revolutionary technology to be employed in many fields such as the internet of thing [1], cloud computing [2], and decentralized finance [3], since it was coined in 2008 [4]. It is a new wave of innovations that numerous tech giants, such as Google and IBM, are investing large funds in research and applications. According to the recent statistics[1], hundreds of blockchain systems have been deployed, and the global market value of blockchain size has reached 4.9 billion in 2021.

M. Li, R. Deng are with the School of Computing and Information Systems, Singapore Management University, 178902, Singapore. Y. Li and G. Xu are the School of Computer Science and Engineering, Nanyang Technological University, Singapore. Y. Wu, J. Weng, and D. Li are with the College of Cyber Security, Jinan University, Guangzhou 510632, China. Jian Weng is the corresponding author. E-mail: cryptjweng@gmail.com.

[1]https://www.marketsandmarkets.com/Market-Reports/blockchain-technology-market-90100890.html

With the wide adoption of blockchain technology, people have recognized that blockchain is not a "*coin to rule them all*" solution, where all applications can be put in a single blockchain [5]. Interoperability[2] has become an essential requirement for blockchain-based decentralized applications. Since different blockchain systems have their own underlying constructions and security mechanisms, it is hard to design a general and efficient interoperable protocol. Many efforts have been made to achieve blockchain interoperability and guarantee specific functionalities such as privacy preservation [6], [7], [8], [9], [10], [11], [12], [13], [14], and smart contracts [15], [16], [17].

Basically, blockchain interoperability protocols can be divided into three categories: (i) Trusted Execution Environment-based (TEEs-based) approach, (ii) cryptography-based approach, and (iii) sidechain-based approach (see Section VIII for more details). This paper we focus on designing for TEEs-based approach, since its advantages on flexible compatibility, real-time efficiency, and confidentiality-preserving smart contracts compared with the other two approaches [12], [11], [13], [14].

Despite the successful applications in cross-chain scenario, the TEEs-based approach exists two issues with respect to cost and concurrent execution that have not been studied systematically yet, which limits its practical applications. On the one hand, current proposals utilize a pool of TEEs (i.e., multiple TEE-empowered hosts), incurring high communication cost and monetary cost for users since each host charges a substantial fee for providing the off-chain execution service. In addition, they require a TEE to store a large number of block headers (e.g., 8,064 block headers for each blockchain [12]) to verify the correctness of on-chain data. However, TEE is a memory-limited environment that would exceed the maximum storage capability when processing large-scale data in cross-chain communication.

On the other hand, previous works on concurrent execution of off-chain contracts use the serializable two-phase commit protocol, allowing all chosen transactions to be included and executed without blocking in the first phase, and then validating the results in the second phase [18], [19]. However, these works also do not consider the limited memory of TEE, where access on-chain data in concurrent contracts execution would exceed the memory storage of the TEE. Furthermore, if an off-chain contract uses on-chain states for contract execution,

[2]We will use the terms "interoperability" and "cross-chain" interchangeably in this paper.

potential concurrency conflicts such as dirty read may happen when on-chain states are revoked by a blockchain fork (see Section IV for more details).

In terms of the cost issue, the intuitive idea is to employ only one TEE-empowered host. However, this idea introduces collusion attacks between the host and participant, which causes economic loss to other participants. Theft events (or possible inside attacks[3]) in MtGox exchange have shown the weaknesses of this idea [12]. In addition, minimizing the memory requirement of the TEE while achieving concurrent contracts execution is particularly challenging, since they seems contradictory in reality. Specifically, concurrent contracts execution requires the TEE to access and process many data within the TEE. While removing the storing of numerous block headers is a mitigation solution. However, it can cause the TEE to use fake or outdated on-chain data for contract execution. Therefore, this work is motivated by the following question: *Is it possible to design a low-cost interoperability mechanism with simplified TEEs as the underlying architecture while preserving concurrency and privacy?*

This paper presents a positive answer to the above question. We propose **I**vy**C**ross, a novel framework that enables privacy-preserving smart contracts and concurrency control for blockchain interoperability using only two TEE-powered hosts, i.e., an execution host and a verification host. Specifically, the privacy of inputs and execution outputs can be preserved by encrypting them under the key pair generated by two TEEs. We employ a game theoretic mechanism for preventing the collusion between the two hosts by using the Time-lock Smart Contract (TSC). To avoid storing many block headers in TEE while preventing it from receiving error data, we require a cross-chain smart contract to run simultaneously on both hosts. The verification host can challenge the execution host in case of incorrect execution using the TSC. Furthermore, we design a non-blocking optimistic concurrency control protocol which uses the data-driven timestamp to address concurrency conflicts. As a result, **I**vy**C**ross enjoys three desirable features that previous proposals may not provide: (i) *Low economic overhead*. A two-TEE based cross-chain protocol is implemented, leading to lower communication and storage overhead. (ii) *Cross-chain privacy*. The confidentiality of execution states and user inputs, as well as the unlinkability of transactions in cross-chain smart contracts can be fully preserved. (iii) *Concurrent Correctness*. The concurrency correctness of TEE-based smart contracts execution can be ensured.

Our Contributions. Specifically, the major contributions of this work can be summarized as follows:

- **A low-cost framework for enabling privacy-preserving and concurrent cross-chain contracts execution**. Based on only two TEE-powered hosts, we design a novel framework called **I**vy**C**ross that can achieve multiple-round cross-chain smart contracts for blockchain interoperability while preserving privacy. We design the customized concurrency control protocol and utilize a smart contract-supporting blockchain to model a sequential game between two hosts, and thus **I**vy**C**ross can guarantee the correctness of off-chain contract execution at low costs. We emphasis that the game theory-based TEEs computation has wider applications beyond blockchain interoperability.
- **Formal Security Analysis**. We model the functionality of **I**vy**C**ross in the Universal Composability (UC) framework, and formally prove the security of our construction with respect to *privacy preservation* and *correct contract execution* (see online version [20]).
- **Implementation and Evaluation**. We implement a prototype of **I**vy**C**ross over three blockchains, including Bitcoin, Ethereum and FISCO BOCS, and evaluate the performance with three real-world use cases. Extensive experiment results on end-to-end performance and concurrency control demonstrate the practicality and efficiency of **I**vy**C**ross.

[3] https://www.coindesk.com/markets/2015/01/01/missing-mt-gox-bitcoins-likely-an-inside-job-say-japanese-police/.

TABLE I
The notations of explanation.

Notation	Explanation
λ	The security parameter.
k	The security parameter of a blockchain.
m, n	The amount of blockchain systems and participants.
$[m]$	The set of natural numbers $(0, 1, ..., m)$.
$\mathcal{P}$	The participants.
$\mathcal{A}$	The adversary.
$\mathcal{R}, \mathcal{V}$	The execution host and verification host.
$\mathcal{B}_j, \widetilde{\mathcal{B}}$	A blockchain system j ($j \in [m]$), and a smart contract-supporting blockchain system.
(pk, sk)	The public key and secret key pair of a participant.
(mpk, msk)	The public key and secret key pair of an enclave.
$datagram$	The data to be stored in the distributed data storage.
T_0, T_1	The enclave T_0 and T_1 run in $\mathcal{R}$ and $\mathcal{V}$, respectively.
tx, bh	A transaction and block hight in blockchain $\widetilde{\mathcal{B}}$.
L_0, L_1, L_2, L_3, L_4	The time of deposits to be set in blockchains, where $L_0 < L_1 < L_2 < L_3 < L_4$.
ω, C	The amount of blocks to be stored in a TEE, e.g., $\omega = 6$ in Bitcoin and $C = 8064$ in [12].
cid	A unique identifier of a cross-chain smart contract.
$H(\cdot)$	A cryptographic hash function.
$getBK(\mathcal{B}_j, w_0, w_1)$	The function to retrieve a succession of blocks starting from a block containing a state w_0 and ending with a block containing a state w_1 from $\mathcal{B}_j$.
$\mathcal{KA}.(KGen, Retrieve)$	The key generation and retrieve algorithm in key agreement.
$\mathcal{AE}.(KGen, Enc, Dec)$	The key generation, encryption, and decryption algorithm in an asymmetric encryption scheme.
$\sum.(KGen, Sign, Verify)$	The key generation, signing, and verification algorithm in digital signature.

II. Preliminaries and Solution Overview

In this section, we present the background of blockchain and cross-chain communication (CCC) and TEE. The notations used throughout this paper are listed in Table I.

Cross-chain Communication. Blockchain is essentially a transparent, immutable distributed ledger that is maintained by many decentralized networks of peers (i.e., blockchain nodes). One can not tamper with a transaction once it has been recorded in the blockchain, and can check the historical details of a transaction, facilitating tracking the digital assets in a trustworthy way. With the rapid deployment of

blockchains, many cross-chain communication (CCC) protocols have emerged to resolve the limitation of blockchain scalability [5]. They allow two relatively dependent blockchains (e.g., Bitcoin and Ethereum) to interact with each other, e.g., cryptocurrency exchange [21] and information transitions [15].

In particular, a CCC protocol is said to be correct if it satisfies *effectiveness, atomicity* and *timeliness* [5]. Taking cryptocurrency exchange as an example, Alice intends to exchange ETH with Bob who has Ethereum using her Bitcoin. This process takes four steps: 1) Setup, 2) (pre-)commit on Bitcoin, 3) verify, and 4a) commit on Ethereum or 4b) abort [5]. The first step is to set initial parameters between Alice and Bob, including time constraints, and the amount of coin. In the second step, Alice posts a commitment transaction that sends Bitcoin to Bob. After confirming the correctness and validation of the commitment transaction in the third step, Bob can post a commitment transaction that transfers an equivalent of ETH to Alice.

Trusted Execution Environment. Trusted Execution Environment (TEE) is one of the key building blocks of **I**vy**C**ross. It builds an isolated environment that enables hardware-based protections on user-level code and data to provide confidentiality and integrity. Currently, several hardware-based protection technologies have been designed, such as Intel SGX [22], [23], ARM TrustZone, and AMD Secure Processor [24]. Here, we adopt the Intel SGX in the implementation of **I**vy**C**ross. Specifically, Intel SGX creates a sandbox environment called *enclave* in which a program can be executed without interference. When an enclave interacts with other enclaves, Intel SGX introduces *attestation* to allow users to trust each other. It is essentially a proving process that enables a user to trust: 1) his program is running in an enclave, 2) the enclave is up to date. There are two types of attestation, *local attestation* and *remote attestation*. Intel SGX provides Intel Attestation Service (IAS) to help users to validate a remote attestation. We recommend readers to review [23] for more details.

A. Protocol Overview

Ivy**C**ross is essentially built as a general-purpose framework that interacts with heterogeneous blockchains and achieves secure and correct off-chain computation with receiving, generating, and verifying transactions using the TEE. Specifically, Ethereum is adopted as the underlying blockchain $\widetilde{\mathcal{B}}$ for supporting on-chain smart contract between execution host $\mathcal{R}$ and verification host $\mathcal{V}$. The TSC contract, called CrossChainGame, is designed for $\mathcal{R}$ and $\mathcal{V}$ to play a *challenge-and-response* game. The basic design of this game is to allow host $\mathcal{V}$ (as a challenger) to retrieve the deposit of $\mathcal{R}$ (as a responder) by providing valid proofs, if $\mathcal{V}$ finds the incorrect execution of $\mathcal{R}$, e.g., $\mathcal{R}$ accepting invalid deposit transactions for contract execution. The CrossChainGame contract serves as a *Judge* to execute the punishment.

To understand our protocol more easily, we use a simple lottery contract to show the phases of **I**vy**C**ross. Specifically, the lottery contract interacts with three participants $\mathcal{P} := \{p_1, p_2, p_3\}$ who lie in Bitcoin, Ethereum, and FISCO BOCS (FB), respectively. The purpose of this contract is to choose a winner, which is determined by calculating a number based on each participant's encrypted input. **I**vy**C**ross protocol consists of four basic phases: the initialization phase, the execution phase, the challenge-and-response phase, and the finalization phase.

- **Initialization**. In this phase, all of the involved parties generate their key pairs and make deposits. After that, one participant (e.g., p_1) can deploy the lottery contract into hosts' enclaves by using the $install$ operation of Intel SGX. A key pair is generated for the contract in the enclaves, and the public key is published. At this point, all participants can make deposits to enclaves' addresses $(pk_{btc}, pk_{eth}, pk_{fb})$ and send deposit transactions to $\mathcal{R}$.
- **Execution**. Participants can send inputs to the contract to trigger a state transition in this phase. The input is encrypted using the public key of the contract, and decrypted for calculation in the enclaves. If there is no dispute, i.e., two hosts and participants behave honestly in this phase, the contract goes to the finalization phase. Otherwise, it enters the next challenge-and-response phase.
- **Challenge-and-response**. This phase happens is conducted between $\mathcal{V}$ and $\mathcal{R}$ within a time of several blocks. For example, if $\mathcal{V}$ detects that $\mathcal{R}$ accepts an invalid deposit transaction for contract execution, it complains to the CrossChainGame contract by sending a challenge transaction and waits for a response. If the challenge succeeds, i.e., $\mathcal{V}$ proves that the deposit is valid, the states in cross-chain contract will rollback and the challenger gets a reward from the CrossChainGame contract. If $\mathcal{R}$ does not receive a response in due time, $\mathcal{V}$ can upload the evidence to $\widetilde{\mathcal{B}}$ and terminate this contract execution by complaining to the participants.
- **Finalization**. After the execution, the protocol returns a final output and assigns the deposits according to the predefined policy. For the lottery contract, $\mathcal{R}$ instructs its enclave to generate two transactions that transfer participants' deposits to the winner's address.

III. IvyCross Models

In this section, we present the proposed **I**vy**C**ross framework by illustrating the system model and security model.

A. System Model

The system model of **I**vy**C**ross involves four parties (cf. Figure 1): the participants, the execution host, the verification host and the blockchain nodes.

- **The participants**, identified by $\mathcal{P} = \{p_1, ..., p_n\}$, refer to a set of parties who invoke cross-chain interaction to achieve cryptocurrency exchange or state transition across heterogeneous blockchains.
- **The execution host**, identified by $\mathcal{R}$, also called a CCC service provider, refers to the party who is equipped with an attested secure processor (identified by T_0) to run general-purpose cross-chain smart contracts.
- **The verification host**, identified by $\mathcal{V}$, refers to the party who also has an attested processor (identified by T_1)

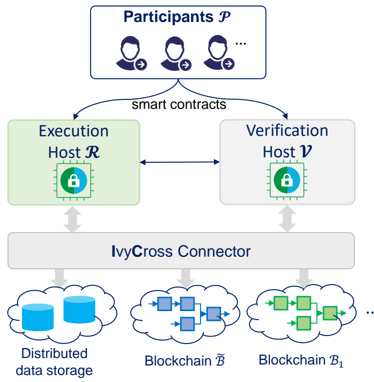


Fig. 1. The system model of **I**vy**C**ross.

and is responsible for monitoring the execution of $\mathcal{R}$. It verifies the correctness of $\mathcal{R}$'s cross-chain execution and punishes it when detecting any dishonest behavior.
- **The blockchain nodes**, refer to parties who maintain the underlying blockchains. They take the responsibility of recording transactions and responding upon receiving a request.

As shown in Figure 1, the construction of **I**vy**C**ross is composed of three components: an **I**vy**C**ross connector and two hosts equipped with an Intel SGX. The **I**vy**C**ross connector acts as a gateway to enable **I**vy**C**ross to interact with different blockchains and a distributed data storage (DDS) to store intermediary data [25], where DDS can be a public bulletin board, e.g., IPFS or S3. Two hosts $\mathcal{R}$ and $\mathcal{V}$ could be the existing centralized cryptocurrency exchanges (e.g., LocalBitcoins and Coinbase) that their roles can be interchangeable. To defend against such a SPoF, we can build a decentralized ecosystem, where many cryptocurrency exchanges can take part in providing cross-chain services for profits (see Sect. VI).

Specifically, running a cross-chain contract requires two hosts to deposit in a smart contract-supporting blockchain (identified by $\widetilde{\mathcal{B}}$) previously, e.g., Ethereum. After that, the participants can register[4] in **I**vy**C**ross and one of them deploys a smart contract using the SGX *install* instruction. All participants need to deposit in their blockchains and send the deposit transaction to two enclaves. Afterward, it goes to the contract execution phase where two hosts execute the cross-chain contract simultaneously. The result is sent to participants with a remote attestation which can validate the output. According to the final results of the contract, these deposits are assigned or withdrawn by the participants.

B. Security Model

Security Threats. We assume that a probability polynomial time (ppt) adversary $\mathcal{A}$ who can corrupt a subset of parties, which means the internal states and operations of the corrupted parties are controlled by $\mathcal{A}$. Here, we consider a *static corruption* where $\mathcal{A}$ controls parties before the contract execution starts. Specifically, we consider the following security threats when blockchain and TEEs technologies are used in a cross-chain setting. First, the execution host $\mathcal{R}$ may behave dishonestly in an attempt to (i) collude with the verification host $\mathcal{V}$ or certain participant to maximize its profits, e.g., accepting an invalid deposit transaction, (ii) discard participants' inputs, and (iii) replay old stored or incorrect state to the TEE (i.e., rollback attacks that the latest data is replaced with an incorrect copy [26]). Host $\mathcal{V}$ may also behave dishonestly by challenging $\mathcal{R}$ with an incorrect on-chain state. Both $\mathcal{R}$ and $\mathcal{V}$ are curious about participants' sensitive inputs. With respect to participants, some of them might collude with $\mathcal{R}$ or $\mathcal{V}$ to maximize their profits in a smart contract, and attempt to obtain others' private inputs. Besides, one of the two hosts might connect to a corrupt blockchain node that has been compromised by the *eclipse attack* [27], causing a host to retrieve fake evidence.

Security Assumptions. The security of **I**vy**C**ross depends on the security of TEEs and blockchains, thus we make the following assumptions with respect to the blockchain and TEE technology. Without loss of generality, $\mathcal{A}$ can not break the fundamental security of blockchain. Communication between the participants, TEEs and blockchain nodes are established through a secure channel (TLS) and information is synchronous between honest parties.

In terms of TEEs, we assume that the confidentiality of programs can be guaranteed in the TEE. An adversary $\mathcal{A}$ can not access the attestation private key. We note that *side-channel* attacks which could leak private information do exist and is a real threat to TEEs, while we consider that it is an orthogonal problem to mitigate such attacks, such as introducing *key committee* as in [13], and thus outside of scope for our design. Besides, T_0 and T_1 are deployed in different platforms but are connected through a network. A single host creating multiple enclave instances is not allowed and can be detected by our design.

Meanwhile, the two hosts are not necessarily to be fully honest, in light of the recent attacks on TEEs [28], [26]. We assume that one of two hosts can be compromised by $\mathcal{A}$, but they can not be compromised simultaneously. All parties are considered as rational and incentive-driven participants that they will not participate in a cross-chain contract if their profits are negative. In addition, to prevent denial-of-service (DoS) attacks [29], e.g., a never-halting program is loaded into the TEEs, we assume that the number of execution rounds and execution time in each round are predefined in advance. Besides, a cross-chain contract deliberately uses an old state within a specific block as input is not considered here.

C. Design Goals

Ivy**C**ross aims to achieve the general-purpose smart contracts for blockchain interoperability. Here, we summarize the design goals of **I**vy**C**ross as follows:
- **Confidentiality**: The privacy of intermediary states and parties' inputs are preserved during contract execution.
- **Correctness**: Given authenticated blockchain evidence, the execution of cross-chain contracts can be guaranteed with correctness and concurrency control.

[4]Here, we do not discuss participants' identity management which has been extensively studied by [2], [13].

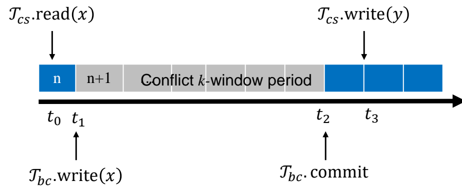


Fig. 2. The illustration of concurrency conflict in cross-chain transactions. $\mathcal{T}_{cs}$.read is sent by a party in **I**vy**C**ross, while $\mathcal{T}_{bc}$.write is sent by a user in a source blockchain.

- **Interoperability**: **I**vy**C**ross can be applied to currently existing blockchains seamlessly, e.g., Bitcoin and Ethereum, achieving state transition and token exchange across heterogenous blockchains.
- **Security against several attacks**: **I**vy**C**ross is able to defend against collusion attacks, and rollback attacks.

IV. Design Challenges

In this section, we discuss the technical challenges that arise when introducing two TEE-powered hosts to enable privacy-preserving smart contracts for cross-chain applications.

A. Collusion Attacks and Rollback Attacks

The biggest challenge of relying on only two TEE-based hosts is the collusion attack. There is no restriction for participants to detect such attacks if both hosts coordinate and output incorrect results. Worse still, the hosts may collude with a certain participant to gain profits from a smart contract application, e.g., to let him win a cross-chain lottery game. For instance, dishonest participants may collude with the host to deceive the TEE into accepting an invalid deposit transaction, causing honest participants to lose their coins. Our core insight is that collusion always occurs when participants find it is more profitable by colluding with others than behaving honestly. Towards this end, economic incentives would be an effective method to thwart collusion attacks. We employ the sequential game theory based on a Turing-complete smart contract to resolve this challenge (see Sect. 7).

On the other hand, cross-chain communication involves a complex network structure where heterogeneous blockchain networks exist. As a consequence, host $\mathcal{V}$ or $\mathcal{R}$ (and its connected blockchain nodes) can be compromised by an adversary $\mathcal{A}$, making the TEE suffer from potential threats not only from its running host but also from external blockchain nodes. More concretely, a compromised host can launch rollback attacks by replacing the latest data with an older (or incorrect) copy [26]. Current research works resort to using a cluster of hosts (or a trusted authority) to tackle this challenge, e.g., ROTE [28], while this method suffers from high communication and computation costs. We aim to achieve the same security goal with lower costs (and better security than a trusted authority).

B. Cross-chain Concurrency Issue

The second challenge lies in the concurrency issue in the off-chain TEEs. Note that a typical contract is essentially converted as several *read* and *write* operations. For a read operation, it is *read-only* call that will not update states, while a write operation, it is potentially a *change-state* procedure that will change the value. The concurrency issue occurs when a cross-chain contract reads data, while another on-chain transaction writes that data concurrently.

More concretely, the concurrency issue is caused because of the *delayed confirmation* of blockchain. Recall that a transaction written on a blockchain might be revoked because of forks. Here, we define a period named *conflict k-window* period, during which the blockchain state cannot reach final consistency, where k refers to the number of blocks that a transaction has been confirmed. During this period, the dirty read issue might exist if a server retrieves a state for its contract execution, while this state is not confirmed finally on the blockchain because of forks. Therefore, we call a blockchain transaction "commit" after passing the conflict k-window period.

To understand how concurrency conflicts occur, let us consider an example as shown in Figure 2. It involves two types of concurrent transactions: $\mathcal{T}_{cs}$ and $\mathcal{T}_{bc}$, and two on-chain states x and y, where $\mathcal{T}_{cs}$ refers to an off-chain transaction executed in the TEE, and $\mathcal{T}_{bc}$ refers to an on-chain transaction executed in a blockchain node. We consider that two transactions $\mathcal{T}_{cs}$ and $\mathcal{T}_{bc}$ run as the following sequence of executions:

1) $\mathcal{T}_{cs}.\mathsf{read}(x)$
2) $\mathcal{T}_{bc}.\mathsf{write}(x)$
3) $\mathcal{T}_{bc}.\mathsf{commit}$
4) $\mathcal{T}_{cs}.\mathsf{write}(y)$

According to the serial order, $\mathcal{T}_{cs}$ can interleavedly run with $\mathcal{T}_{bc}$ without violating the serializability. However, if $\mathcal{T}_{cs}$ commits after $\mathcal{T}_{bc}$, while x has been modified by $\mathcal{T}_{bc}$, this can cause the concurrency conflict. Different commit orders of $\mathcal{T}_{cs}$ and $\mathcal{T}_{bc}$ would affect the correctness of cross-chain contracts. The key challenge lies that $\mathcal{T}_{cs}$ is executed outside of the blockchain and the commit operation of $\mathcal{T}_{cs}$ and $\mathcal{T}_{bc}$ are controlled by different entities. Thus, it is necessary to design a concurrent protocol to manage the serial execution of on-chain and off-chain transactions for cross-chain communication.

C. Limited storage

The third challenge lies in the TEE itself. To keep synchronous with blockchain and verify on-chain states, an off-chain TEE needs to load a number of block headers (∼5MB for 8,064 block headers) to its environment in current research works [13], [29], [12]. However, there are several shortcomings of this approach. First, a TEE is a memory-limited environment. In particular, the reserved memory for Intel SGX application is limited to a total of 128 MB, and only 93MB is available in reality [30], [31]. Since thousands of blockchain systems exist currently, using this approach would cause TEE to exceed the memory capacity. One may suggest to leverage external storage to mitigate this issue, but this approach might suffer from rollback attacks as mentioned in IV-A. Second, a TEE is required to update the stored data periodically, thus incurring high computation and communication cost. The adverse impact of the limited memory of TEE would

be amplified when it is used in multiple-chain applications. Therefore, it remains challenging to guarantee the verifiability of on-chain data owing to the limited memory of TEEs.

V. IvyCross Protocol

In this section, we begin with modeling a cross-chain smart contract, and then formally specify the concrete design and the concurrency control protocol. Afterward, we elaborate a game theory-based method for enhancing the security of **I**vy**C**ross.

A. Modeling Cross-chain Smart Contracts

A privacy-preserving cross-chain execution is modeled as multiple-round interactions among the following parties: a set of participants $\mathcal{P} = \{p_1, ..., p_n\}$, and several involved blockchains $\{\mathcal{B}_j\}_{j\in[m]} := \{\mathcal{B}_1, ..., \mathcal{B}_m\}$. Formally, an SGX program is recognized as a finite state machine that is modeled as a general-purpose *smart contract*:

$$\mathsf{Contract}_{ccc} := (cid, \mathsf{st}, ents, ops, dep, \{\mathcal{B}_j\}_{j\in[m]}), \quad (1)$$

where cid is a unique identifier of the contract, st is the states of the contract, $ents$ refers to a set of entities including *account*, *contract* and *object* involved in the contract. ops refers to the operations over these entities, e.g., cryptocurrency exchange. dep refers to the dependence among operations, e.g., some operations should be executed before or after another. The lifecycle of $\mathsf{Contract}_{ccc}$ can be denoted as a state machine, where the states contain the following states {Unknown, Init, Inited, Comp, Comped, Closed}. A key pair $(\widetilde{pk_{cid}}, \widetilde{sk_{cid}})$ is generated for the contract using $\mathcal{KA}.KGen(1^\lambda)$. The TEE publishes $\widetilde{pk_{cid}}$ and keeps $\widetilde{sk_{cid}}$ private.

In particular, an interaction (i.e., a cross-chain transaction $\mathcal{T}_{cs}$) between a participant p_i and $\mathsf{Contract}_{ccc}$ is modeled as the following 5-tuple:

$$\mathcal{T}_{cs} := (cid, ct, w, ti, \sigma_{p_i}), \quad (2)$$

where $ct = \mathcal{AE}.Enc(\widetilde{pk_{cid}}, inps)$ is a ciphertext of the participant's inputs $inps$. w refers to a set of blockchain evidences. ti is the timestamp of the transaction, σ_{p_i} is a signature of p_i which is used to identify the identity. More precisely, w contains some transactions and the corresponding block information, i.e., $w := (\mathcal{B}_j, \mathcal{T}_{bc}, mkR, diff, hd, bh)$, where $\mathcal{T}_{bc}$ is the latest on-chain transaction which is related with w, mkR is the Merkle path for authenticating $\mathcal{T}_{bc}$, $diff$ is the difficulty level, hd is the block header and bh is the corresponding block height.

In addition, an interaction between a participant and the TEE can be modeled as the following state transition:

$$\mathsf{TEEContract}(cid, st_\iota; r_\iota) \xrightarrow{\mathcal{T}_{cs}} (oupts_\iota, st_{\iota+1}), \quad (3)$$

where st_ι refers to the previous state which is securely stored in the IPFS, r_ι refers to the randomness of the round ι. The TEE triggers a state transition by taking $\mathcal{T}_{cs}$ as an input, and outputs $oupts_\iota$ and generates a new state $st_{\iota+1}$. Specially, we follow the output method of an attestation as in Ekiden [13] where an output $oupts_\iota$ of an SGX program contains a pair of parts $(oupt_{\iota_1}, oupt_{\iota_2})$. A hash of the second output $H(oupt_{\iota_2})$ and an input $h_{inps} = H(ct_\iota, w_\iota)$ of the program are included in $oupt_{\iota_1}$. Only $oupt_{\iota_1}$ is signed by the TEE in an attestation, i.e., $\sigma_T := \sum.Sign(msk_T, oupt_{\iota_1})$.

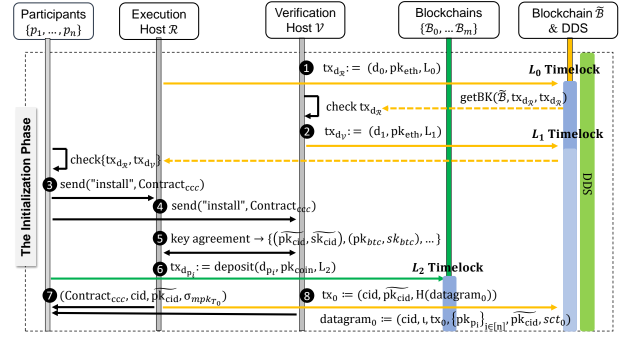
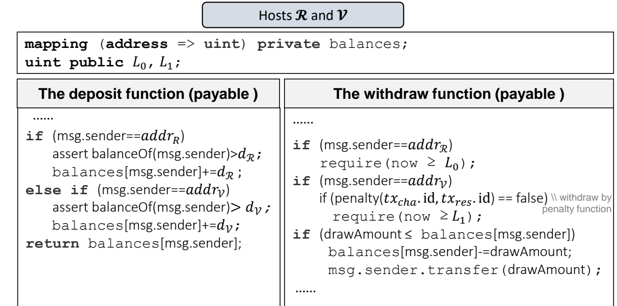


Fig. 3. The initialization phase of Prot_{ccc}. Solid dark arrows indicate interactions among TEEs and participants. Yellow dashed arrows denote interactions among participants, TEEs and $\widetilde{\mathcal{B}}$, green dashed arrows denote interactions between participants, TEEs and $\{\mathcal{B}_j\}_{j\in[m]}$. Blue bands indicate the time interval of the deposits.


Hosts $\mathcal{R}$ and $\mathcal{V}$

```
mapping (address => uint) private balances;
uint public L0, L1;
```

The deposit function (payable)

```
......
if (msg.sender==addrR)
   assert balanceOf(msg.sender)>dR;
   balances[msg.sender]+=dR;
else if (msg.sender==addrV)
   assert balanceOf(msg.sender)> dV;
   balances[msg.sender]+=dV;
return balances[msg.sender];
```

The withdraw function (payable)

```
......
if (msg.sender==addrR)
   require(now ≥ L0);
if (msg.sender==addrV)
   if (penalty(tx_cha.id, tx_res.id) == false) \\ withdraw by penalty function
   require(now ≥L1);
if (drawAmount ≤ balances[msg.sender])
   balances[msg.sender]-=drawAmount;
   msg.sender.transfer(drawAmount);
......
```

Fig. 4. The illustration of the deposits of two hosts.

B. Protocol Specification

1) The Initialization Phase: We now specify the formal details of the **I**vy**C**ross protocol. In this phase (cf. Figure 3), T_0 and T_1 initialize the key pairs (mpk_{T_0}, msk_{T_0}) and (mpk_{T_1}, msk_{T_1}), respectively. They attest the public keys (mpk_{T_0}, mpk_{T_1}) by Intel IAS and establish a secure communication channel. For simplicity, we treat the blockchain $\widetilde{\mathcal{B}}$ and IPFS as a whole, where it stores encrypted data (identified by $datagram$) in IPFS and a corresponding hash value (i.e., $H(datagram)$) in $\widetilde{\mathcal{B}}$.

Afterwards, two hosts $\mathcal{R}$ and $\mathcal{V}$ individually generate several addresses with the security parameter λ for making deposits in $\widetilde{\mathcal{B}}$ and accepting rewards from different blockchains. Their deposits are locked in the *time-lock* contract $\mathsf{CrossChainGame}$ that can only be redeemed after L_0 and L_1 blocks, respectively (Steps ❶-❷). During this timeframe, the enclaves can spend the locked deposit by providing a proof. In case of dishonest behaviors, a penalty function $penalty(\cdot)$ is designed to allow $\mathcal{V}$ to withdraw a certain amount of coins from $\mathcal{R}$'s deposit, if $\mathcal{V}$ finds an evidence that can prove $\mathcal{R}$ is cheating (cf. Figure 4). The inputs of $penalty(\cdot)$ mainly contain two transactions $(tx_{cha},\ tx_{res})$ that are generated in the challenge-and-response phase, which will be described in detail in the third phase.

At this point, a set of participants $\mathcal{P} = \{p_1, ..., p_n\}$ can execute a cross-chain smart contract. Each participant generates a key pair (pk_{p_i}, sk_{p_i}) and registers the public key in **IvyCross** for identity verification. A certain participant p_i $(i \in [n])$ can deploy a smart contract $\mathsf{Contract}_{ccc}$ with an initial state st_0, the public keys of participants to T_0 and T_1 by invoking the "install" instruction (Steps ❸-❹). Specially, st_0 contains several blocks $\{bk_{cp_1}, ..., bk_{cp_{m_0}}\}$ (including the difficulty level) of the involved blockchains $\{\mathcal{B}_j\}_{j \in [m_0]}$ as the checkpoints of the contract, each blockchain $\mathcal{B}_j$ loads only a few of the latest blocks, which is to prevent malicious hosts from using incorrect blockchain to deceive the enclave.

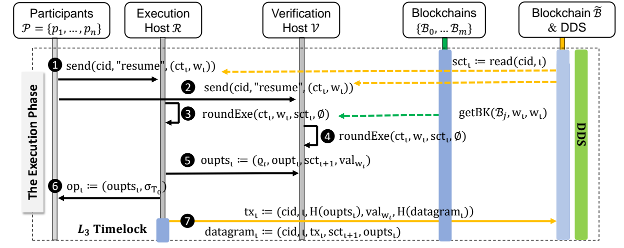


Fig. 5. The execution phase of Prot_{ccc}.

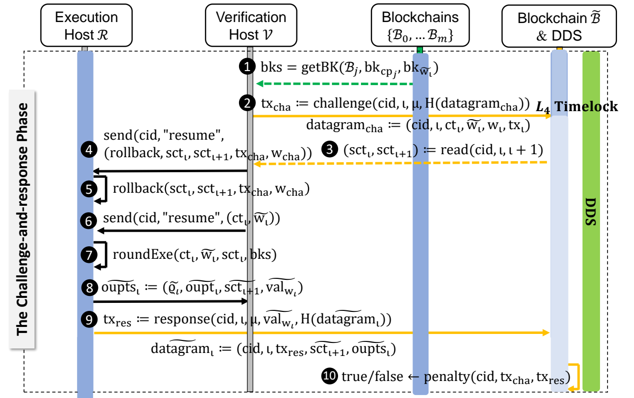


Fig. 6. The challenge-and-response phase of Prot_{ccc}.

Two enclaves T_0 and T_1 run $\mathcal{KA}.KGen(1^\lambda)$ to generate a fresh key pair $(\widetilde{pk_{cid}}, \widetilde{sk_{cid}})$ for preserving the privacy of contract interaction, and a set of key pairs for accepting participants' deposits, e.g., (pk_{btc}, sk_{btc}) is generated for Bitcoin (Step ❺). To prevent dishonest participants from participating in different contracts using the same deposit, we require the enclaves to generate the deposit addresses for each smart contract. Participants need to make deposits in their corresponding blockchains before entering the next phase (Step ❻). These deposits are locked in the corresponding addresses $(pk_{btc}, pk_{eth}, ...)$ of $\mathsf{Contract}_{ccc}$, and can only be spent by the enclaves within L_2 blocks. After the locked time, participants can redeem the deposit if it is not spent by the enclave.

With the successful deployment of $\mathsf{Contract}_{ccc}$, two hosts respond with a contract identifier cid and initial parameters (Step ❼). $\mathcal{R}$ sends $(cid, \{pk_{p_i}\}_{i \in [n]}, \widetilde{pk_{cid}}, datagram_0)$ to $\widetilde{\mathcal{B}}$ and waits for the confirmation (Step ❽), where $sct_0 = \mathcal{AE}.Enc(\widetilde{pk_{cid}}, st_0)$ in $datagram_0$ is an initial encrypted state of the contract. Meanwhile, a response $\{\mathsf{Contract}_{ccc}, cid, \widetilde{pk_{cid}}, \sigma_{mpk_{T_0}}\}$ is sent to the participants, where $\sigma_{mpk_{T_0}}$ is a signature of T_0.

2) The Execution Phase: In this phase, the contract $\mathsf{Contract}_{ccc}$ is triggered with a state transition upon receiving an input from a participant (cf. Figure 5). Concretely, in round ι $(\iota \in [\ell])$, a participant p_i gets the public key $\widetilde{pk_{cid}}$ and encrypts an input $ct_\iota = \mathcal{AE}.Enc(\widetilde{pk_{cid}}, inps_\iota)$. Then, p_i sends an input (cid, ct_ι, w_ι) to T_0 (and T_1) by invoking the "resume" instruction $(resume, cid, (ct_\iota, w_\iota))$ (Steps ❶-❷). $\mathcal{R}$ reads a previous state of the contract sct_ι from $\widetilde{\mathcal{B}}$. The input is sent with a signature of p_i so that the enclave T_0 can verify its validation. w_ι is a blockchain evidence that both $\mathcal{R}$ and $\mathcal{V}$ can check whether it is a correct on-chain state. Here, we use $roundExe(ct_\iota, w_\iota, sct_\iota, bks)$ to represent the execution of contract function (Steps ❸-❹), in which the fourth parameter denotes a succession of blocks related with w_ι. If bks is not empty, $\mathsf{Contract}_{ccc}$ will use it to check the correctness of w_ι. Otherwise, i.e., $bks = \emptyset$, it only uses w_ι for contract execution. By setting this parameter, it allows $\mathcal{V}$ to send a challenge by providing a succession of correct blocks in case of incorrect execution.

After the execution, the contract $\mathsf{Contract}_{ccc}$ outputs $oupts_\iota := (\varrho, sct_{\iota+1}, oupt_\iota, \mathsf{val}_w)$ (Step ❺), where $\varrho := (h_{inps}, h_{st}, h_{stsf}, h_{oupt}, \sigma_{mpk_{T_0}})$ refers to a set of hash values on the input $h_{inps} = H(ct_\iota, w_\iota)$, a previous state $h_{st} = H(st_\iota)$, a state transition $h_{stsf} = H(st_\iota || st_{\iota+1})$, an output $h_{oupt} = H(oupt_\iota)$. $sct_{\iota+1} = \mathcal{AE}.Enc(\widetilde{pk_{cid}}, st_{\iota+1})$ is a new contract state which can be used for the rollback of states in the third phase (if necessary). If it is required by the contract, $oupt_\iota$ can be an encrypted output of the function using pk_{p_i}. $\mathsf{val}_w \in (0, 1)$ is a validation result of w. T_0 sends the output to T_1 with an attestation $attes_\iota := (oupts_\iota, \sigma_{mpk_{T_0}})$ (Step ❻). Meanwhile, a digest of the output $(cid, \iota, H(oupts_\iota), H(datagram_\iota), \mathsf{val}_w)$ is sent to $\widetilde{\mathcal{B}}$ (Step ❼) using the Pedersen commitment scheme [32]. To enforce the atomic delivery of $attes_\iota$ and $(cid, \iota, H(oupts_\iota), H(datagram_\iota), \mathsf{val}_w)$ between $\mathcal{V}$ and $\widetilde{\mathcal{B}}$, we can adopt the protocol *proof of publication* as in [13].

In particular, we design a *data-lock* in the contract $\mathsf{CrossChainGame}$ for an on-chain state sct_ι that it can only be treated as stable and useful (identified by FINAL) after a predefined L_3 blocks. Before that, it is an unstable data (identified by $\mathsf{UNCONFIRM}$). Therefore, when an enclave retrieves a previous state of the cross-chain contract in a new round, it first validates the status in the previous round. By doing so, it can guarantee the correctness of states reading between two hosts and $\widetilde{\mathcal{B}}$, no matter whether there will be a state rollback case.

Providing that all involved parties behave honestly, $\mathcal{R}$ runs the next round for the contract $\mathsf{Contract}_{ccc}$ after L_3 blocks, and goes to the finalization phase after ℓ rounds.

3) The Challenge-and-response Phase: The above contract execution is a normal case that two hosts and participants are honest, so the interactions are minimal. While if any party behaves dishonestly, the protocol Prot_{ccc} enters into the challenge-and-response phase (cf. Figure 6). Note that the enclave T_0 has attested the outputs in $\widetilde{\mathcal{B}}$ that $\mathcal{V}$ can check the

correctness of the execution based on this information. If $\mathcal{V}$ detects that the execution outputs from T_1 is different from T_0, it might represent that $\mathcal{R}$ accepts an invalid input w_ι for contract execution, e.g., accepting a fake deposit transaction. At this point, $\mathcal{V}$ can launch a *challenge-and-response* game which is modelled as an interaction between $\mathcal{V}$ and $\mathcal{R}$ to ask for a response within a time of L_4 blocks.

Specifically, $\mathcal{V}$ sends a challenge transaction tx_{cha} to $\widetilde{\mathcal{B}}$ and requires a response transaction tx_{res} from $\mathcal{R}$. $\mathcal{V}$ first reads a succession of blocks bks from $\mathcal{B}_j$, where $bks = getBK(\mathcal{B}_j, bk_{cp_j}, bk_{\widetilde{w_\iota}})$ refers to the blocks from the checkpoint bk_{cp_j} to a latest block $bk_{\widetilde{w_\iota}}$ which contains a correct $\widetilde{w_\iota}$ (Step ❶). If w_ι is not an on-chain data in the longest blockchain, bks can read from the checkpoint to the latest block. Then, $\mathcal{V}$ generates the transaction tx_{cha} which contains the previous input ct_ι, a hash value $H(bks)$, and a correct blockchain evidence $\widetilde{w_\iota}$ which is the correct on-chain data related with $inps$ in $\mathcal{B}_j$ (Step ❷). In particular, $\mathcal{V}$ transfers a little number of coins μ to the balance of $\mathcal{R}$ in tx_{cha} and requires $\mathcal{R}$ to return μ back to $\mathcal{V}$'s balance. To conduct the challenge, $\mathcal{V}$ reads two previous encrypted states sct_ι and $sct_{\iota+1}$ from $\widetilde{\mathcal{B}}$ (Step ❸) and makes a $resume$ call with an input $(rollback, sct_\iota, sct_{\iota+1}, tx_{cha}, w_{cha})$ to trigger a rollback operation from $sct_{\iota+1}$ back to sct_ι in T_0 (Step ❹), where w_{cha} refers to the blockchain evidence of tx_{cha}.

Upon receiving a rollback resume, $\mathcal{R}$ needs to respond to a transaction tx_{res} within L_4 blocks. Specifically, $\mathcal{R}$ first checks the signature of this call, and then sends a $resume$ call to T_0 with the inputs $(cid, sct_\iota, sct_{\iota+1})$ to trigger the state rollback from $sct_{\iota+1}$ back to sct_ι (Step ❺). T_0 also checks the signature of rollback instruction and then accomplishes the rollback with outputting an attestation to claim the success of this call. Note that only T_1 can be authorized to resume a rollback operation by providing tx_{cha}. After that, $\mathcal{V}$ makes a $resume$ call to T_0 with an input $(cid, ct_\iota, \widetilde{w_\iota}, st_\iota, bks)$ (Step ❻). $\mathcal{R}$ completes the execution upon receiving the call and outputs an attestation to T_1 (Step ❼-❽). Based on the output attestation, $\widetilde{\mathcal{R} \text{ generates the}}$ response transaction $tx_{res} := (cid, \iota, \mu, \widetilde{\mathsf{val}_{\widetilde{w_\iota}}, H(datagram_\iota)})$ (Step ❾). Then, the *Judge* contract $\mathsf{CrossChainGame}$ takes (tx_{cha}, tx_{res}) as inputs to execute the comparison and determines the amount of coins to be punished (Step ❿). Specifically, the process of verification can be extended to support more complicated comparison operations within T_0 or T_1, i.e., the comparison after the decryption of $oupts_\iota$.

If $\mathcal{V}$ succeeds to use a correct $\widetilde{w_\iota}$ to trigger a different state transition, it proves a wrong execution of $\mathcal{R}$ and γ_0 of $\mathcal{R}$'s deposit is sent to $\mathcal{R}$ as a reward. Otherwise, $\mathcal{V}$ should compensate $\mathcal{R}$ with γ_1 of $\mathcal{V}$'s deposit for the cost of transaction fee, where $0 < \gamma_0, \gamma_1 \leq 1$. For example, if a malicious participant provides a deposit transaction in w_ι which has been consumed by another transaction (i.e., a consumed UTXO) in Bitcoin, $\mathcal{V}$ can provide the evidences $\widetilde{w_\iota}$ that contain the transaction consumed w_ι (i.e., a created UTXO), the difficulty level, and a succession of blocks which start from the block containing w_ι to the block containing $\widetilde{w_\iota}$. These uploaded blocks bks can be deleted from the storage of T_0 after the execution.

In particular, if $\mathcal{R}$ does not respond with tx_{res} in due time, $\mathcal{V}$ can prove this dishonest behavior by providing the blockchain evidence to $\mathsf{CrossChainGame}$, and notice all participants to abort the execution of this contract. Inspired by [29], [13], the *challenge-and-response* game can also be used to prevent malicious behaviors: (i) $\mathcal{R}$ refrains participants' inputs, (ii) a participant does not send inputs to $\mathcal{R}$ on time or (iii) sends an invalid on-chain data.

4) The Finalization Phase: After $\ell-1$ rounds of execution, T_0 outputs a final result $oupts_{fin}$, and assigns the deposits to the address of the participants according to the outputs of $\widetilde{\mathsf{Contract}_{ccc}}$. Specifically, it generates a set of transactions $\{\widetilde{tx_{d_1}}, ..., \widetilde{tx_{d_n}}\}$ which send the deposits back to the corresponding addresses using the secret keys $(sk_{btc}, sk_{eth}, ...)$. The input of a transaction $\widetilde{tx_{d_i}}$ contains the deposit transaction tx_{d_i} generated by p_i. If there exists any dishonest behavior of a participant, the enclave will assign his deposit to other participants as punishment. This punishment policy is predefined in the contract by the participant. $\mathcal{V}$ is responsible for monitoring the finalization of the contract to ensure that these transactions are generated and sent to the blockchains successfully. Providing there is no contract to be executed anymore, and the lock time is up, $\mathcal{R}$ and $\mathcal{V}$ can redeem their deposits from $\widetilde{\mathcal{B}}$.

C. Concurrent Control and Validation

To guarantee concurrent correctness of cross-chain contracts execution, we address the concurrency conflict issue by introducing the Time Traveling Optimistic Concurrency Control, called TicToc [33]. TicToc utilizes the data-driven timestamp mechanism that assigns read and write timestamps for each data item. By doing so, a valid commit timestamp of a transaction can be obtained according to these timestamps. Compared with previous scenarios that TicToc is adopted, the main difference in cross-chain is that no central manager can lock a write transaction from the blockchain. Hence, we optimize the principle to determine whether a cross-chain transaction should be aborted. In particular, we assume that two hosts are aware of the write timestamp of involved on-chain data through the **I**vy**C**ross connector. They do not know the privacy of these data since data can be encrypted before being published to the blockchain.

Specifically, a cross-chain transaction $\mathcal{T}_{cs}$ can consist of two or three phases: *a read phase, a validation phase*, and a (possible) *write phase*. The hosts maintain two sets for each $\mathcal{T}_{cs}$: (i) a read set $\mathcal{T}_{cs}.\mathsf{RS}$ and a write set $\mathcal{T}_{cs}.\mathsf{WS}$[5]. The read (or write) set can be denoted as: $\{idx, \upsilon, wts, rts\}$, where idx is the pointer of this tuple in the storage of **I**vy**C**ross, υ is the value, wts refers to the timestamp that the latest committed transaction wrote to υ, rts refers to the timestamp that the latest transaction read υ. In terms of the write set of $\mathcal{T}_{cs}$, it involves two types of committed transactions: on-chain transactions and cross-chain transactions, i.e., $\mathcal{T}.\mathsf{WS} = \mathcal{T}_{cs}.\mathsf{WS} \cup \mathcal{T}_{bc}.\mathsf{WS}$.

[5]Since the processing of $\mathcal{T}_{bc}$ is serialized on-chain, and thus we do not consider its read or write set.

Algorithm 1: $\mathsf{Prot}_{con}(idx, \mathcal{T}_{cs}.\mathsf{RS}, \mathcal{T}.\mathsf{WS}, chainID, \delta, addr)$

1: **Read Phase**:
2: Inputs: Read Set $\mathcal{T}_{cs}.\mathsf{RS}$, tuple idx.
3: $en = \mathcal{T}_{cs}.\mathsf{RS}.get_new_entry()$.
4: $en.idx = idx$.
5: $\{en.v = s.v, en.wts = s.wts, en.rts = s.rts\}$
6: Outputs: en
7: **Validate Phase**:
8: Inputs: Read Set $\mathcal{T}_{cs}.\mathsf{RS}$, Write Set $\mathcal{T}.\mathsf{WS}$.
9: **for** ws in $\mathcal{T}.\mathsf{WS}$:
10: ▷ *Only lock cross-chain write set in DDS of* ***I****vy****C****ross.*
11: **if** $ws.idx \in \mathcal{T}_{cs}.\mathsf{WS}$
12: $lock(ws.idx)$
13: ▷ *calculate the commit timestamp*
14: **for** se in $\mathcal{T}_{cs}.\mathsf{RS} \bigcup \mathcal{T}.\mathsf{WS}$:
15: **if** se in $\mathcal{T}.\mathsf{WS}$ then
16: $com_{ts} = max(com_{ts}, se.idx.rts + 1)$
17: **else**
18: $com_{ts} = max(com_{ts}, se.idx.wts)$
19: **end**
20: **end**
21: **for** en in $\mathcal{T}_{cs}.\mathsf{RS}$ **do**:
22: **if** $en.rts < com_{ts}$ **then**:
23: ▷ *Read the maximum write timestamp from* $\mathcal{T}.\mathsf{WS}$.
24: $wts_0 = get_max_wts(\mathcal{T}.\mathsf{WS}, en.idx))$
25: ▷δ *is a minimum block number that determines if a transaction has concurrency conflict.*
26: **if** $en.wts \neq r.idx.wts$ **or** $((en.idx.rts \leq com_{ts})$ **and** $\mid en.rts - wts_0) \mid \leq \delta$ **and** $isLocked(en.idx))$
27: abort();
28: **else**
29: $en.tuple.rts = max(com_{ts}, en.idx.rts)$;
30: **end**
31: **end**
32: **end**
33: **Write Phase**:
34: Inputs: Write Set $\mathcal{T}_{cs}.\mathsf{WS}$, commit timestamp com_{ts}, blockchain ID $chainID$, address $addr$
35: **for** w in $\mathcal{T}_{cs}.\mathsf{WS}$ **do**:
36: $sendTransaction(w.idx.v, chainID, addr)$
37: $w.wts = w.rts = com_{ts}$
38: $unlock(w.idx)$
39: **end**

Specifically, when a cross-chain transaction $\mathcal{T}_{cs}$ reads a data from the blockchain, a version of transaction $\mathcal{T}_{cs}$ can be committed (i.e., enter the *write* phase) only when the following conditions hold:

$$\begin{aligned} &\exists com_{ts}, \\ &(\forall \eta \in \{versions\ read\ by\ \mathcal{T}_{cs}\},\ \eta.wts \leq com_{ts} \leq \eta.rts) \\ &\wedge (\forall \eta \in \{versions\ written\ by\ \mathcal{T}_{cs}\},\ \eta.rts \leq com_{ts}), \end{aligned} \tag{4}$$

where wts and rts refer to the previous write and read timestamp, com_{ts} refers to the commit timestamp of $\mathcal{T}_{cs}$. Equation 4 illustrates that a version read by $\mathcal{T}_{cs}$ is valid, only when its commit timestamp is between the values of wts and rts. A version write by $\mathcal{T}_{cs}$is valid, only when its commit timestamp is greater than rts.

Following [33], we demonstrate the concurrency control as shown in Algorithm 1. The first *read* phase shows the procedure for accessing a tuple. The value v and the timestamp wts and rts are read atomically to ensure the consistency of value and timestamp. The second phase is to compute the commit timestamp com_{ts} of a $\mathcal{T}_{cs}$ using the read and write set. As for the validation of $\mathcal{T}_{cs}.\mathsf{RS}$, the commit timestamp com_{ts} is calculated upon the principles of Equation 4. Note that if a tuple entry's rts is less than com_{ts}, then it is possible that another transaction (e.g., $\mathcal{T}.\mathsf{write}$) has modified the value between rts and com_{ts}. In this case, the cross-chain transaction $\mathcal{T}_{cs}$ has to be aborted and re-executed in the next round. Furthermore, if the local read timestamp rts is different with the latest one (i.e., $r.idx.rts$), and a blockchain transaction $\mathcal{T}_{bc}$ has modified $r.idx.v$ between rts and com_{ts}, which means a certain $\mathcal{T}_{bc}$ might be executed concurrently within this time. In our design, we follow a basic principle that $\mathcal{T}_{cs}.\mathsf{read}$ should be stagger with $\mathcal{T}_{bc}.\mathsf{write}$ if the absolute time difference between them is less than a setting value δ (see Line 25), where δ is set according to the block confirmation time of the blockchain. Otherwise, it can go to the final *write* step. In this step, the entries in $\mathcal{T}_{cs}.\mathsf{WS}$ will be written to the blockchain, and updated in the DDS of **I**vy**C**ross.

D. Incentive Mechanism using Sequential Game Theory

To prevent collusion attacks between $\mathcal{R}$, $\mathcal{V}$ and participants, we leverage economic means to incentivize them to behave honestly. Owing to the sequential behaviors between $\mathcal{R}$, $\mathcal{V}$ and participants, we model their interactions as a sequential game in which one player determines his action before the other players choose theirs [34]. Specifically, we consider a time horizon of ℓ rounds. In each round $\iota \in [\ell]$, a reward provided by participants is shared by $\mathcal{R}$ and $\mathcal{V}$. Assume that the total payment provided by participants for $\mathcal{R}$ and $\mathcal{V}$ is M (excluding the profits from collusion with certain participants), where each participant pays for M/n. The payment sharing ratio is α (where $\alpha < 1$) determined by the negotiation between two hosts. Then the average payment for each round denotes as $r(\iota) = M/\ell$, and the total amount of payments for $\mathcal{R}$ and $\mathcal{V}$ is denoted as $\alpha \cdot M$ and $(1-\alpha) \cdot M$, respectively.

For simplicity, the cost of a host in one round is considered as a function $c(\iota)$, regardless of an honest or dishonest behavior.[6] Without loss of generality, the reward payment for one round contract execution is larger than the costs of two hosts. The cost of contract execution is more expensive than the cost of verification, i.e., $c_{\mathcal{R}}(\iota) > c_{\mathcal{V}}(\iota)$. The transaction fee is considered as a function $\mathbf{f}(\iota) = \{f(1), \ldots, f(\ell)\}$, where $f(\iota)$ is determined by the number of transactions and the size of a transaction in round ι. Besides, if a party behaves dishonestly, this behavior can be detected and punished by a penalty

[6]It can be extended to take behavior-dependent (e.g., honest or dishonest) into consideration.

function P, which transfers the deposit of the dishonest party (i.e., a host or a participant) to the address of the others.

Following the game theory methodology [34], we formalize the essential concepts for modeling a sequential game, including the strategy and utility function, and further utilize the back reduction method to solve for a sequential Nash Equilibrium (NE) in this game. Specifically, the strategy profile of a party is assumed as a binary set $s = \{H, D\}$, where "H" stands for a party behaving honestly: it follows the protocol without collusion, data falsification (or accepting invalid inputs for a host). On the contrary, "D" stands for a party behaving dishonestly, i.e., not complying with the normal protocol. Given a reward payment function $\mathbf{r}(\cdot)$, a cost function $\mathbf{c}(\cdot)$, a transaction fee function $\mathbf{f}(\cdot)$ and a penalty function $p(\cdot)$. The total utility $U_\theta(S)$ of a host can be defined as follows:

$$U_\theta(S) = \sum_{\iota=1}^{\ell} (\mathbf{r}_{s_\theta}(\iota) - \mathbf{f}(\iota) - \mathbf{c}(\iota)) + P_{s_\theta}, \quad (5)$$

where s_θ refers to the strategy of a host $\theta \in \{\mathcal{R}, \mathcal{V}\}$, and P_{s_θ} refers to the reward payment.

The total utility of a participant p_i is determined by the design of the contract. We assume that the profit for participant p_i in a contract is denoted as R_{p_i}. For instance, if p_i wins a cross-chain lottery game, he can get a contract reward, otherwise, he just pays a transaction fee for the contract execution. We use R_{max} to denote the maximum profit of p_i, i.e., $0 \leq R_{p_i} \leq R_{max}$. Assume that the collusion happens between a malicious participant (identified by p_i^*) and two hosts, we can observe that only both hosts collude with p_i^*, p_i^* can successfully deceive other participants in **I**vy**C**ross. In this situation, we assume that the assignment of the collusion profits for $\mathcal{R}$ and $\mathcal{V}$ are $\beta_1 \cdot R_{max}, \beta_2 \cdot R_{max}$, then the profit obtained by p_i^* is $(1 - \beta_1 - \beta_2) \cdot R_{max}$.

The utility of $\mathcal{R}$, $\mathcal{V}$ and participant p_i is shown in Figure 7. Each nonterminal node in the game tree is owned by a host, and each terminal node assign a profit vector for a participant and two hosts, i.e., $u(s) = (u_{p_i}(s), u_{\mathcal{R}}(s), u_{\mathcal{V}}(s))$. Specifically, to solve the optimal strategy of the sequential game, we first describe the definition of sequential NE as follows:

Definition 1 (Sequential Nash Equilibrium)**.** *A strategy profile $(s^*_{p_i}, s^*_{\mathcal{R}}, s^*_{\mathcal{V}})$ is denoted as a sequential Nash Equilibrium (NE) in the three parties sequential game, if no party $\widetilde{\theta} \in \{\mathcal{R}, \mathcal{V}, p_i\}$ can improve his profit by altering the strategy $s^*_{\widetilde{\theta}}$ with the other party's strategy $\boldsymbol{S}^*_{s_{\widetilde{\theta}}}$:*

$$\begin{aligned} &u_{p_i}(s^*_{p_i}, s^*_{\mathcal{R}}, s^*_{\mathcal{V}}) \geq u_{p_i}(s_{p_i}, s^*_{\mathcal{R}}, s^*_{\mathcal{V}}),\ for\ each\ s_{p_i} \in \boldsymbol{S}^*,\ and \\ &u_{\mathcal{R}}(s^*_{p_i}, s^*_{\mathcal{R}}, s^*_{\mathcal{V}}) \geq u_{\mathcal{R}}(s^*_{p_i}, s_{\mathcal{R}}, s^*_{\mathcal{V}}),\ for\ each\ s_{\mathcal{R}} \in \boldsymbol{S}^*,\ and \\ &u_{\mathcal{V}}(s^*_{p_i}, s^*_{\mathcal{R}}, s^*_{\mathcal{V}}) \geq u_{\mathcal{V}}(s^*_{p_i}, s^*_{\mathcal{R}}, s_{\mathcal{V}}),\ for\ each\ s_{\mathcal{V}} \in \boldsymbol{S}^*. \end{aligned} \quad (6)$$

In other words, a strategy profile $(s^*_{p_i}, s^*_{\mathcal{R}}, s^*_{\mathcal{V}})$ is a NE if no party in the defined game can improve his expected utility by altering the current strategy s^* (and vice versa).

Next, we leverage back reduction to analyze and solve for a NE in a sequential game. The back reduction method proceeds from the end of a problem to decide a sequence of choices. First, we consider the participant as an honest party ($s_{p_i} = s_{p_i 1} = H$) and analyze the utilities of the two hosts when $\mathcal{R}$ and $\mathcal{V}$ behave honestly and dishonestly, respectively (cf. Figure 7). We found that $s_{\mathcal{R}} = s_{\mathcal{R}2} = s_{\mathcal{V}} = s_{\mathcal{V}4} = H$, which means the contract execution is correct without the complaint of $\mathcal{V}$. This is because when $s_{p_i} = H$, the utility of $\mathcal{R}$ in $\{v_1, v_2\}$ where $s_{\mathcal{R}} = H$ is greater or equal to $\{v_3, v_4\}$ where $s_{\mathcal{R}} = D$, regardless of the strategy of $\mathcal{V}$. That is, if $\mathcal{R}$ accepts an invalid w_ι for contract execution, the dishonest choice of $\mathcal{R}$ can be detected by $\mathcal{V}$ that $\mathcal{R}$ would be punished with $P_{\mathcal{R}}$. Of course, if $\mathcal{V}$ fails in a challenge, he needs to pay for the transaction fees and the computation resources.

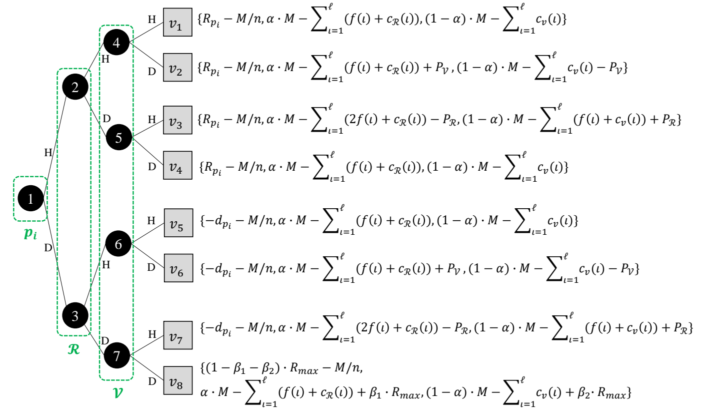


Fig. 7. The sequential game tree for $\mathcal{R}$, $\mathcal{V}$ and p_i. Each vertex denotes a choice for a party. The arrow of the vertex denotes a possible strategy $\{H, D\}$ for that party. The profits of three parties are specified at the terminal nodes, e.g., $v_1 = \{u_{p_i}, u_{\mathcal{R}}, u_{\mathcal{V}}\}$.

Similarly, if the participant is a dishonest party ($s_{p_i} = s_{p_i 1} = D$), we can observe that the strategy of $\mathcal{R}$ and $\mathcal{V}$ are determined by the amount of collusion profits. More concretely, if $P_{\mathcal{V}} \geq \beta_1 \cdot R_{max}$, then the utility of $\mathcal{R}$ in $\{v_5, v_6\}$ is greater or equal to $\{v_7, v_8\}$, i.e., $s_{\mathcal{R}} = s_{\mathcal{R}3} = H$. Consequently, $\mathcal{V}$ will behave honestly because $u_{\mathcal{V}}(D, H, H) > u_{\mathcal{V}}(D, H, D)$. We know that the party who moves first has the first-mover advantage. However, as long as the amount of $\mathcal{V}$'s deposit is larger than the collusion profits, both hosts will choose to behave honestly, making the profit of p_i^* negative.

Therefore, according to the back reduction, we obtain $(s_{p_i}, s_{\mathcal{R}}, s_{\mathcal{V}}) = (H, H, H)$ *or* (D, H, H). It can be found easily that $(s_{p_i}, s_{\mathcal{R}}, s_{\mathcal{V}}) = (s_{p_i 1}, s_{\mathcal{R}2}, s_{\mathcal{V}4}) = (H, H, H)$, because $u_{p_i}(H, H, H) = R_{p_i} - M/n > u_{p_i}(D, H, H) = -d_{p_i} - M/n$. Therefore, the NE of the sequential game in **I**vy**C**ross is found at $(s^*_{p_i}, s^*_{\mathcal{R}}, s^*_{\mathcal{V}}) = (H, H, H)$, i.e., the path $1 \rightarrow 2 \rightarrow 4 \rightarrow v_1$ in Fig. 7. Under this strategy, neither p_i, $\mathcal{R}$ nor $\mathcal{V}$ can improve their profit by altering the strategy. Regardless of the strategy of p_i and $\mathcal{R}$, $\mathcal{V}$ will behave honestly to avoid penalty and expand his profit. Therefore, the conclusion of the sequential game is given in Theorem 1 as follows:

Theorem 1. *In the sequential game between p_i, $\mathcal{R}$ and $\mathcal{V}$, there exists a strategy $\boldsymbol{S}^* = (s^*_{p_i}, s^*_{\mathcal{R}}, s^*_{\mathcal{V}}) = (H, H, H)$ that satisfies NE, where three parties choose the strategy of H if they are rational.*

TABLE II
State-of-the-art works in cross-chain communication. $\phi(\cdot)$ refers to the size of *proof* for proving an on-chain data. Below, m refers to the number of blockchains, C and ω refer to the blocks number to be stored in a TEE, $\omega \gg C$, for instance, $\omega = 8064$ in Bitcoin and $C = 6$ in Tesseract [12].

Approach	Attacks Defense		Privacy preservation	Minimal data for $\phi(\cdot)$ $\#Block$	Concurrency Control	Smart contacts for blockchain Interoperability
	Rollback attack	Collusion attack				
Ekiden [13]	○	$--$	●	$\mathcal{O}(m\omega)$	◐	◐
Fastkitten [29]	○	$--$	◐	$\mathcal{O}(m\omega)$	◐	◐
Tesseract [12]	◐	●	●	$\mathcal{O}(m\omega)$	◐	◐
HyperService [15]	$--$	●	○	$--$	◐	●
A^2L [9]	$--$	●	●	$--$	◐	○
IvyCross	●	●	●	$\mathcal{O}(mC)$	●	●

$--$ refers to the approach does not involve for different system model; ○ refers to the approach does not consider or solve; ◐ refers to the approach needs to be improved to achieve the property; ● refers to the approach can solve.

VI. Security Analysis and Discussion

A. Security analysis

In this section, we begin by specifying the definition of the UC Security. Due to space limits, the proof using the UC-framework is detailed in the appendix of the online version [20]. Then, we analyze the security of our proposed protocol Prot_{ccc} under the security model.

Definition 2 (UC Security of Prot_{ccc})**.** *Let λ be a security parameter, assume that $\mathcal{G}_{att}$'s attestation scheme and digital signature are unforgeable and $\mathcal{H}$ be second preimage resistant, $\mathcal{L}_{\mathcal{B}}$ be a global functionality of blockchain, $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ be an ideal contract functionality, and Prot_{ccc} be a protocol in the $(\mathcal{G}_{att}, \mathcal{H}, \mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}, \mathcal{L}_{\mathcal{B}})$-hybrid world. Then, Prot_{ccc} is said to UC-realize $\mathcal{F}_{ccc}$ in the $(\mathcal{G}_{att}, \mathcal{H}, \mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}, \mathcal{L}_{\mathcal{B}})$-hybrid world if for any ppt adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$, such that for all PPT environment $\mathcal{E}$ and $inps \in \{0,1\}^*$, the following formula holds:*

$$\begin{aligned} &\forall \mathcal{E},\ REAL_{\mathsf{Prot}_{ccc},\mathcal{A},\mathcal{E}}^{\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att},\mathcal{H},\mathcal{L}_{\mathcal{B}}}(\lambda, inps) \approx \\ &IDEAL_{\mathcal{S},\mathcal{E}}^{\mathcal{F}_{ccc},\mathcal{G}_{att},\mathcal{H},\mathcal{L}_{\mathcal{B}}}(\lambda, inps) \end{aligned} \tag{7}$$

Theorem 2. *There exists a privacy-preserving CCC protocol Prot_{ccc}, which UC-realize the ideal functionality $\mathcal{F}_{ccc}$ in the $\mathsf{CrossChainGame}$ contract $(\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}, \mathcal{G}_{att})$-hybrid world.*

Proof. In the appendix of the online version [20], we formally prove Theorem 2 in the hybrid world, where we elaborate the ideal functionalities and formal statements in the hybrid world. □

Confidentiality. It is straightforward that the confidentiality property can be achieved using TEE in a cross-chain scenario. The inputs of participants and the outputs of TEEs are encrypted using the public keys and an adversary $\mathcal{A}$ can not break the basic security of public encryption. All public keys are published in DDS which could be recognized as a secure certificate authority (CA), and intermediary states are stored with persistence and immutability in $\widetilde{\mathcal{B}}$. Furthermore, inspired by [35], the unlinkability between participants is preserved because the off-chain TEE can play the role of a *mixer* to unlink their relationship.

Security against collusion attacks. The critical challenge of **I**vy**C**ross is the collusion attacks among participants and the two hosts. To overcome these, we first require all parties to deposit in the corresponding blockchains. Then, during the execution phase, the execution host $\mathcal{V}$ monitors the behaviors of $\mathcal{R}$. If $\mathcal{R}$ behaves dishonestly, $\mathcal{V}$ can launch a challenge-and-response request in the CrossChainGame contract. In **I**vy**C**ross, both hosts cannot forge a blockchain evidence to deceive the enclave into accepting it. Based on this premise, we analyzed that when the amount of $\mathcal{V}$'s deposit is greater than the collusion profit, there exists an optimal strategy $\mathbf{S}^* = (s^*_{p_i}, s^*_{\mathcal{R}}, s^*_{\mathcal{V}}) = (H, H, H)$ that satisfies NE in the sequential game. Namely, collusion between certain participants and two hosts is not a dominant strategy for neither of them. Regardless of the strategy of others, each party will behave honestly to avoid penalty and make profit.

Security against unreliable hosts. An unreliable host can cause the incorrect contract execution in the enclave, including rescheduling the order of the transactions arbitrarily and replaying the old states to the enclave. To address this issue, during the execution phase, we require participants to send inputs (with a timestamp of ti) to both hosts simultaneously. Thus, if $\mathcal{R}$ schedules the execution order of transactions wrongly, it can be detected by $\mathcal{V}$ by checking the consistency of state transitions with $\mathcal{R}$'s execution attestation. $\mathcal{V}$ can challenge $\mathcal{R}$ in the third phase and the deposits of $\mathcal{R}$ will be sent to $\mathcal{V}$. Similarly, if one host launches rollback attack to trigger incorrect state transition, yet the state of each round is recorded in DDS and $\widetilde{\mathcal{B}}$, allowing others to check the consistency and correctness of the old state. As a matter of fact, the design of CrossChainGame and the sequential game theory-based incentive mechanism can be used to thwart this attack. Furthermore, we believe that such incentive mechanism could also impel two hosts (as cross-chain service providers) to invest enough resources to secure their systems, such as making more outgoing connections to prevent eclipse attacks which may cause the network to exhaust the connection bandwidth.

Protection against External Adversaries. $\mathcal{R}$ and $\mathcal{V}$ are assumed not to be compromised by external adversaries simultaneously, which is a strong assumption. The two hosts architecture may increasingly become the target of external adversaries. To defend against such a SPoF, we can build a de-

centralized ecosystem, where many cryptocurrency exchanges can take part in providing cross-chain services for profits and only two exchanges are required for each contract execution. We can design hosts selection based on randomness measures that could reduce the possibility of collusion with participants and hosts[7]. Alternatively, we can mark these exchanges with a reputation by using the off-the-shelf reputation mechanisms. Participants can select a reliable host based on the reputation. In addition, we consider that the external front-running attacks can also be restrained through the TEE-based smart contracts in cryptocurrency exchange. Similar to Tesseract [12], **I**vy**C**ross can prevent the adversary $\mathcal{A}$ from front-running others by inspecting the entire communication.

B. Comparison with State-of-the-arts

As shown in the Table II, we compare **I**vy**C**ross with the state-of-the-art works [15], [12], [13], [29], [9]. Enforcing blockchain interoperability by designing privacy-preserving, attacks-defending smart contracts, is still an open challenge. The off-chain approaches, such as Ekiden [13], Fastkitten [29], and Tesseract [12] using TEE can achieve privacy-preserving smart contracts. However, these approaches do not focus on solving the high costs of communication and computation of TEEs. More concretely, they all require each TEE to store several blocks. The minimal data for proving an on-chain data using these approaches is $\phi(m\omega)$ which is much larger than $\phi(mC)$ in **I**vy**C**ross. In addition, most of these approaches [11], [12], [13] resort to a number of TEEs for achieving interoperability and correctness, which introduces high communication costs between TEEs. Besides, HyperService [15], and A^2L [9] are cryptographic primitives-based approaches to achieve blockchain interoperability, while these approaches are not so efficient compared with **I**vy**C**ross. Nonetheless, we recognize that the advantages of existing approaches, e.g., the front-running attacks and race condition defense as in [12], can be combined into our design for achieving privacy-preserving cross-chain smart contracts in a more secure and resource-saving way.

In particular, we have to point out that **I**vy**C**ross does not have high robustness compared with [13], [12], the limitation here is that two hosts might become the target of the adversary, causing the single point of failure. Besides, our concurrency control protocol is a mitigation solution for executing off-chain contracts in TEEs. It is essential to design a more general protocol to solve the concurrency issue when facing a large number of concurrent transactions.

VII. Implementation and Evaluation

The proof-of-concept implementation of **I**vy**C**ross is implemented using the Solidity, Java, and C++. We implemented three use cases to demonstrate its effectiveness and practicality.

[7]We leave the defense against external adversaries and the choice of reliable hosts as future works.

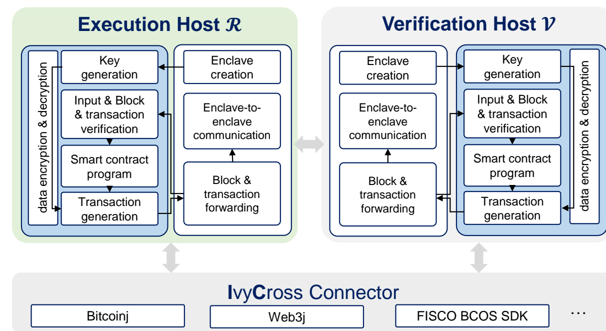


Fig. 8. The implementation of **I**vy**C**ross.

A. Implementation Design

In the implementation of **I**vy**C**ross (see Figure 8), each host consists of two parts: (i) one part is executed in memory-isolated SGX enclave, which is responsible for the private computation including key generation, data (e.g., inputs, transactions) encryption and decryption, and transaction generation. (ii) The other part is run outside of the SGX enclave. It is in charge of creating an enclave instance, communicating with another host and forwarding data to the internal enclave.

To avoid the complex development of Intel SGX based on SGX SDK, we leverage a light-weight library OS, named Occlum [36], to build the Intel SGX environment. Occlum allows user applications to run on an attested SGX enclave with little or no modifications to the programs, and supports different programming languages for deploying a TEE-based program, such as Go, C and Java. Here, we build two Occlum OS in different servers (Ubuntu18.04.4LTS, "Intel Xeon(R) CPU E5-2683 V3" @ 2.00GHz, Gtx 1080TI GPU, 256GB of DRAM) to simulate the two hosts $\mathcal{R}$ and $\mathcal{V}$.

Further, to validate the practicality of **I**vy**C**ross, we deploy three testnets on local servers (Ubuntu 16.04 xenial, "x86_64 Linux 4.15.0-142-generic" @ GeForce GTX 1080 Ti GPU, 2534MiB/31977MiB of RAM) for Bitcoin, Ethereum and FB. We design a virtual coin in FB and construct three use cases across three blockchain systems. Note that almost all of the blockchains have implemented a client library to allow interacting with them. Specifically, to ensure the randomness of the key generation and reduce the possibility that an adversary learns about the secret keys, we let two hosts $\mathcal{R}$ and $\mathcal{V}$ to generate a key pair by combining the internal hardware-based randomness sgx_read_rand(), extra OS's randomness, and a set of latest blocks hash from different blockchains. Besides, we leverage the OpenSSL toolkit for encryption and digital signature.

B. Applications

We highlight three concrete use cases to show the practicality and efficiency of **I**vy**C**ross.

Lottery. As mentioned earlier (see Sect. II-A), the cross-chain lottery contract determines one winner based on a collaboratively generated number. We only require participants to send a number to the enclave without using an inefficient commit-and-reveal scheme.

Auction. A second-price sealed-bid auction contract is implemented where participants submit bids without knowing others'. The contract determines the winner by checking who bids the highest price, and requires the winner to pay for the second-highest price. The seller generates a set of public addresses in all different blockchains for accepting the payment of the winner, and other participants can redeem their deposits back. Note that the privacy of the bidding price can be preserved, which provides a fairer way for achieving decentralized auction across blockchains.

Housing Loan. Housing loan requires a participant to provide documents to show that he has the capability to pay for the loan. These documents can be obtained from different domains. We implement a cross-chain contract HousingLoan that fetches values from different blockchains. In our implementation, we assume the inputs of participants, e.g., salary and deposits, are privately stored in Ethereum and FISCO BOCS. The amount of the money that a participant could loan is calculated based on the private data. This case illustrates that **IvyCross** can be adopted in permissioned blockchains for preserving the privacy of on-chain data.

C. Evaluation Results

To evaluate the performance of **IvyCross**, we conducted the experiments for 10 times with 1200 clients for the above three use cases, where each case has identical clients on Bitcoin, Ethereum, FB testnets, respectively.

Scalability. We recognize that the overhead caused by remote attestation and state persistency on $\widetilde{\mathcal{B}}$ affects the scalability of **IvyCross** significantly. Inspired by Ekiden [13], we use the off-chain transactions batching and multithreading contracts execution to improve the scalability of **IvyCross**. More concretely, during the contract execution, we compress multiple off-chain transitions into a single on-chain transaction, instead of sending each state checkpoint to $\widetilde{\mathcal{B}}$ separately. In fact, the off-chain transactions batching does not compromise the security of our protocol, as long as the outputs of contracts are kept secret in the enclave. Similarly, the enclave T_0 is able to cache the states in the enclave, and send the last state transition to IAS for attestation. Once the execution of a contract is completed, $\mathcal{R}$ sends the last state with the authenticated attestation to the $\mathcal{V}$. This can significantly reduce the communication latency caused by remote attestation.

Specifically, each client sent 6 transactions (in the initialization and execution phase) to the hosts in total. We start the timer when the participant sends a request and end when a contract outputs the final result. We disregard the first and last 10% of transactions and evaluate the stable performance. Multiple threads are created in the enclave to process the participants' requests. As shown in Figure 9 (a), **IvyCross** achieves high performance that the peak throughput of the lottery, auction, and housing loan contract can up to 5960txn/sec, 5942txn/sec, 5883txn/sec, respectively. The throughput of three contracts changes with the increasing amount of participants, where the consensus time is set as 15s (cf. Figure 9 (b)).

End-to-end Execution Latency. In each experiment, we first evaluate the end-to-end latency of a normal situation that all

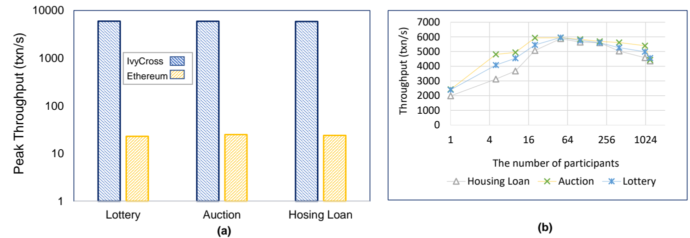


Fig. 9. (a) The peak throughput of three cross-chain contracts. Three contracts are similar because the differences between the contract logic are unobvious. (b) The throughput of three contracts with the increasing amount of participants.

parties behave honestly, and then evaluate the case that $\mathcal{R}$ behaves dishonestly (cf. Table III). Specifically, the latency comes from three aspects: (i) The remote attestation using the IAS between two hosts $\mathcal{R}$ and $\mathcal{V}$. For each execution round, an output from $\mathcal{R}$ needs to be attested before being sent to $\mathcal{V}$, which takes $2.923s$ on average. (ii) The latency caused by the challenge-and-response phase in which an execution result needs to wait for L_3 blocks to allow $\mathcal{V}$ to send a challenge in case of dispute. (iii) The latency of on-chain transaction confirmation. The on-chain transactions takes on average $14.254s$ for a state persistency on $\widetilde{\mathcal{B}}$.

More concretely, in the initialization phase, we analyze the latency on key generation, contract deployment, and the deposit of two hosts and participants in blockchain. The average time of contract deployment in the TEE takes about 2.013s which has a relatively high proportion in Occlum, while it executes only once for each contract. The performance of deposit, including the transaction generation and confirmation, takes 0.519s, 15.111s, 1.083s in local Bitcoin, Ethereum and FB testnets on average, which depends on the difficulty level setting. A low difficulty level is set in our configurations for saving resources. In the execution phase, the time consumption mainly refers to the computation which includes the process of signature verification, data decryption, and result computation in the enclave. Note that these three applications are one-round interaction contracts. It takes about 0.499ms, 0.532ms, 0.604ms on average in this enclave. If we add the time cost of one time attestation (about $2.923s$), this phase is still comparably efficient for secure computation. The finalization phase mainly contains the process of payment transaction generation in enclave and confirmation in blockchain. The time consumption depends on the types of transaction and the consensus time. In the housing loan case, all outputs generate an Ethereum transaction and take the longest time of 15.796s on average. Note that the time consumption in challenge-and-response takes 32.290s on average, which is a bit high, but this phase only happens when there exist malicious parties.

Transaction fees. We implement the contract CrossChainGame using *solidity* in local Ethereum network and analyze the transaction fees in the lottery case. The transaction fee of deposit for each participant takes $0.05 \sim 0.82$ USD in Bitcoin (cf. Table IV). As for an Ethereum transaction, the cost is calculated according to the gas consumed in a specific function. The gas prices are denoted in Gwei, where

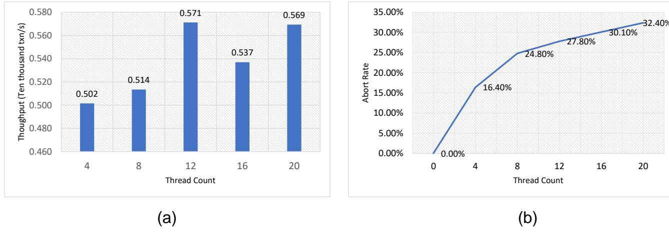


Fig. 10. The scalability and abort rate of our concurrency control algorithm with 10 data items.

$1\ Gwei = 1 \times 10^{-9}\ ETH$. The price of $1\ ETH$ is equal to $\$1,864$ at the time of writing (August 2021). Each participant needs to pay 0.05 USD for a deposit in Ethereum on average. The costs of the transaction fee in each execution round ranges from 26,801 Gwei to 65,341 Gwei, i.e., $\$0.0499 \sim \$0.1217$. To reduce the transaction fee, in the execution phase, we can compress many state update transactions into a single transaction. The total costs can be controlled in two rounds, i.e., $0.0499 * n + 0.1207$, where n refers to the number of participants. The challenge-and-response transactions tx_{cha} and tx_{res} require 27,361 Gwei to 27,896 Gwei, i.e., $0.051 \sim 0.052$ USD in transaction fees.

Concurrency Control. To evaluate the designed cross-chain concurrency control algorithm, we utilize the transaction statistics including transaction throughput and abort rate, where abort rate refer to the total number of aborts divided by the total number of transactions. We use the HousingLoan cross-chain contract that retrieves data from the blockchain and other blockchain transactions can update the on-chain data. We prepare 1000 transactions which include 500 cross-chain transactions and 500 source blockchain transactions.

Each cross-chain transaction contains several *read* and *write* operations, while a blockchain transaction only contains *write* operation. We execute these transactions on 10 data items and design different threads to run them. As shown in Fig.VII-C, we can see that with the increasing of the thread count, the abort rate increases obviously, which is due to that reason that the *write* transactions execute concurrently with the *read* transactions. These concurrent transactions should be abort if their time difference is less than or equal to δ. We further set different δ and find that the abort rate decreases with the increasing of δ, which is reasonable due to the lower probability of transactions concurrency.

Existing well-known cross-chain platforms, such as HyperService [15] and Polkadot [17], inherently publish the data on chain, and thus can not accomplish the above use cases in a privacy-preserving way. To summarize, **I**vy**C**ross can implement them with privacy-preserving, and is highly efficient by moving the computation to the off-chain. The cost of transaction fee is comparatively low due to the reduced size of transactions.

TABLE III
The performance of **I**vy**C**ross execution in different phases. All time are in seconds. We present the statistics including the mean (average), and the proportion (%) taken by different phases.

CCC Phases	Lottery		Auction		Housing Loan	
	Mean	%	Mean	%	Mean	%
Initialization	22.097	33.43	21.451	34.27	22.743	30.36
Execution	2.831	4.28	2.916	4.66	2.97	3.96
Challenge-and-response	32.265	48.81	31.193	49.84	33.413	44.60
Finalization	8.913	13.48	7.029	11.23	15.796	21.08
Total	66.106		62.589		74.922	

TABLE IV
The number of off-chain and on-chain transactions (# txs), and estimated transaction fees by **I**vy**C**ross. It calculates the fees (USD) using the data from BINANCE retrieved on June. 18, 2021

Lottery Contract across Bitcoin, Ethereum, and FISCO BOCS.							
CCC Phases	Off-chain		On-chain				
	# txs	Size (bytes)	# txs	Size (bytes)			Fees (USD)
				ETH	BTC	FB	
Initialization	22	2,643∼ 18,551	13	720∼ 721	166∼ 2,486	908∼ 1,012	0.05∼ 0.82
Execution	20	1,077∼ 1,090	10	311∼ 318	0	0	0.051∼ 0.051
Challenge-and-response	2	1,098∼ 23,315	2	310∼ 326	0	0	0.051∼ 0.052
Finalization	0	0	11	1,023∼ 1,031	309∼ 310	916∼ 1,108	0.05∼ 0.1

VIII. Related Work

In this section, we review the state-of-the-art schemes which are related with blockchain interoperability.

A. Blockchain Interoperability

Blockchain interoperability attracts extensive attention both in academic and industrial communities is a hot topic with the rapid adoption of blockchain. In general, the blockchain interoperability technologies can be mainly classified into three categories: (i)cryptography-based approach, (ii) sidechains-based approach, and (iii) TEEs-based approach.

Cryptography-based approach. The cryptography-based approach mainly refers to off-chain payment/state channels which utilize cryptographic primitives to achieve interoperability [6], [7], [8], [10]. The atomic swap, proposed by Herlihy [21], is a prior approach that can achieve atomic coin exchange based on hash-lock time contracts in public blockchain, e.g., Bitcoin and Ethereum. It enables the ability that users can exchange their coin with others across heterogenous blockchains. Deshpande [37] proposed a privacy-preserving cross-chain atomic swap. It formally defined the notions of privacy in atomic swap and introduced the primitive Atomic Release of Secrets (ARS) to achieve privacy-preserving. Carsten *et al* proposed P2EDX to realize privacy-preserving cryptocurrency exchange using multi-Party computation [6]. Thyagarajan *et al* use scriptless scripts to implement interoperability without relying on Hash Time Lock Contracts (HTLC) [8]. Their scheme is compatible with different blockchain systems by introducing the notion of Lockable Signatures. Tairi *et al* proposed an elegant scheme named A^2L, an Anonymous Atomic Locks to realize unlinkablity and interoperability in payment channel hubs by leveraging adapter signatures and randomizable puzzles [7].

Most of cryptography-based approaches focus on atomic decentralized cryptocurrencies exchange between different blockchain systems in an all-or-nothing manner, e.g., exchanging Bitcoin to Ethereum. As we know, cryptocurrencies exchange is only one of the functionality of blockchain interoperability. When faced with complex computation across blockchains, e.g., supporting cross-chain smart contracts, current cryptography-based approaches might come with an expensive computation due to the need for heavy cryptographic techniques, e.g., secure multi-party computation.

Sidechains-based approach. Sidechains-based approach realizes blockchain interoperability by introducing an intermediary blockchain which is essentially a "blockchain of blockchains" [15], [38], [16], [17]. One of the typical research works is HyperService [15]. The core design is the Network Status Blockchain (NSB) which supports programmability by introducing customized smart contracts across heterogeneous blockchains. However, this approach lacks privacy and is vulnerable to front-running attacks for exposing data in the intermediary blockchain [12]. Sidechains have their consensus protocol and rely on a set of parties to maintain the security of the underlying sidechain. While these proposals have inherently scalability issues, and some of them lack privacy protection for exposing data in the intermediary blockchain [15]. Polkadot [17] and Cosmos [16] are two typical sidechains-based approaches that aim to provide interoperability framework for blockchain. Currently, there are still in the process of development and testing.

TEEs-based approach. Recent many efforts to realize blockchain interoperability focus on using trusted hardware, e.g., Intel SGX, to improve the functionality and security of cross-chain. More concretely, Bentov *et al* proposed a TEEs-based solution to realize *real-time* cross-chain cryptocurrency exchange named Tesseract [12]. Tesseract is able to address the front-running issue. Cheng designed a platform called Ekiden which is for privacy-preserving smart contracts [13]. In addition, Teechain [14] constructed a layer-two payment network that can execute off-chain transactions across blockchains. These proposals have laid foundations for the TEEs-based cross-chain paradigm. Compared with these, **I**vy**C**ross considers a more practical solution to reduce the communication and computation overhead. Adaption of cryptography-based primitives, e.g., zero-knowledge proof and homomorphic encryption, to **I**vy**C**ross for secure multi-computation is a direction for our future work.

B. Concurrency control in Blockchain Interoperability

Concurrency control is the management of contention for controlling serializable schedules in Database Management Systems (DBMS) [39], [40]. Current concurrency control protocols usually use timestamp, i.e., timestamp ordering, to address conflicting operations. They can be divided into three categories: (i) Two-phase Locking (2PL) [39], (ii) Deterministic Concurrency Control (DCC) [40], and (iii) Optimistic Concurrency Control (OCC) [33]. 2PL utilizes *locks* to guarantee serializability. In the DCC protocol, transactions are first ordered by a (centralized or distributed) sequencing layer with a simple and deterministic locking protocol. The OCC protocol allows transactions to be executed without locking at first, and then a manager will examine if conflicts have taken place. It has a better performance than the other two protocols when concurrency does not emerge frequently. Wang [41] extended the two-phase commit protocol to accomplish arbitrary transactions without a central component. Zhao [42] discussed the ACID properties and proposed two distributed commit protocols for distributed cross-chain transactions. However, a coordinator is required exists to schedule transactions across blockchains. This paper adopts the Time Traveling OCC (TicToc) protocol which supports data-driven timestamp management [33]. Compared with TicToc, we extend it to enable non-blocking validation in the computation of commit timestamp.

IX. Conclusion

In this work, we presented **I**vy**C**ross, a privacy-preserving blockchain interoperability framework that supports smart contracts across multiple blockchains. **I**vy**C**ross achieves interoperability by employing two TEE-powered hosts while without requiring several TEEs to store a large number of blocks as in the existing approaches in the literature. The overhead of communication and on-chain costs can be significantly reduced. The interaction of the two TEE-powered hosts is modelled as a game based on sequential game theory, making collusion a less favorable choice for the rational hosts. We believe that this game theory based interaction model can find applications beyond blockchain interoperability. We formally analyzed the security of **I**vy**C**ross in the UC framework, and demonstrated its feasibility and efficiency via three real-world use cases.

References

[1] H.-N. Dai, Z. Zheng, and Y. Zhang, "Blockchain for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8076–8094, 2019.

[2] K. Gai, J. Guo, L. Zhu, and S. Yu, "Blockchain meets cloud computing: a survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2009–2030, 2020.

[3] P. Tolmach, Y. Li, S.-W. Lin, and Y. Liu, "Formal analysis of composable DeFi protocols," in *Proceedings of the 1st Workshop on Decentralized Finance (DeFi)*, Mar. 2021.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[5] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt, "Sok: communication across distributed ledgers." 2019.

[6] C. Baum, B. David, and T. K. Frederiksen, "P2dex: privacy-preserving decentralized cryptocurrency exchange," in *International Conference on Applied Cryptography and Network Security*. Springer, 2021, pp. 163–194.

[7] E. Tairi, P. Moreno-Sanchez, and M. Maffei, "A 2 l: Anonymous atomic locks for scalability in payment channel hubs," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1834–1851.

[8] S. A. K. Thyagarajan and G. Malavolta, "Lockable signatures for blockchains: Scriptless scripts for all signatures," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 937–954.

[9] E. Tairi, P. Moreno-Sanchez, and M. Maffei, "A2l: Anonymous atomic locks for scalability in payment channel hubs," Cryptology ePrint Archive, Report 2019/589, Tech. Rep., 2019.

[10] S. A. Thyagarajan, G. Malavolta, and P. Moreno-Sánchez, "Universal atomic swaps: Secure exchange of coins across all blockchains," *Cryptology ePrint Archive*, 2021.

[11] Y. Yan, C. Wei, X. Guo, X. Lu, X. Zheng, Q. Liu, C. Zhou, X. Song, B. Zhao, H. Zhang *et al.*, "Confidentiality support over financial grade consortium blockchain," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 2227–2240.

[12] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1521–1538.
[13] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 185–200.
[14] J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. Pietzuch, "Teechain: a secure payment network with asynchronous blockchain access," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 63–79.
[15] Z. Liu, Y. Xiang, J. Shi, P. Gao, H. Wang, X. Xiao, B. Wen, and Y.-C. Hu, "Hyperservice: Interoperability and programmability across heterogeneous blockchains," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 549–566.
[16] J. Kwon and E. Buchman, "Cosmos whitepaper," 2019.
[17] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," *White Paper*, 2016.
[18] T. Dickerson, P. Gazzillo, M. Herlihy, and E. Koskinen, "Adding concurrency to smart contracts," *Distributed Computing*, vol. 33, no. 3, pp. 209–225, 2020.
[19] P. S. Anjana, S. Kumari, S. Peri, S. Rathor, and A. Somani, "An efficient framework for optimistic concurrent execution of smart contracts," in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2019, pp. 83–92.
[20] Y. L. Y. W. J. W. D. L. R. D. Ming Li, Jian Weng, "Ivycross: A trustworthy and privacy-preserving framework for blockchain interoperability," 2021, https://ia.cr/2021/1244.
[21] M. Herlihy, "Atomic cross-chain swaps," in *Proceedings of the 2018 ACM symposium on principles of distributed computing*, 2018, pp. 245–254.
[22] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 86, pp. 1–118, 2016.
[23] R. Pass, E. Shi, and F. Tramer, "Formal abstractions for attested execution secure processors," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017, pp. 260–289.
[24] F. Zhang and H. Zhang, "Sok: A study of using hardware-assisted isolated execution environments for security," in *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, 2016, pp. 1–8.
[25] J. Benet, "Ipfs-content addressed, versioned, p2p file system (draft 3)," *arXiv preprint arXiv:1407.3561*, 2014.
[26] G. Kaptchuk, M. Green, and I. Miers, "Giving state to the stateless: Augmenting trustworthy computation with ledgers." in *NDSS*, 2019.
[27] K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn mining: Generalizing selfish mining and combining with an eclipse attack," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 305–320.
[28] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun, "{ROTE}: Rollback protection for trusted execution," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 1289–1306.
[29] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A.-R. Sadeghi, "Fastkitten: Practical smart contracts on bitcoin," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 801–818.
[30] T. Dinh Ngoc, B. Bui, S. Bitchebe, A. Tchana, V. Schiavoni, P. Felber, and D. Hagimont, "Everything you should know about intel sgx performance on virtualized systems," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 1, pp. 1–21, 2019.
[31] M. Fang, Z. Zhang, C. Jin, and A. Zhou, "High-performance smart contracts concurrent execution for permissioned blockchain using sgx," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 1907–1912.
[32] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual international cryptology conference*. Springer, 1991, pp. 129–140.
[33] X. Yu, A. Pavlo, D. Sanchez, and S. Devadas, "Tictoc: Time traveling optimistic concurrency control," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 1629–1642.
[34] J. R. Kale and T. H. Noe, "Risky debt maturity choice in a sequential game equilibrium," *Journal of Financial Research*, vol. 13, no. 2, pp. 155–166, 1990.
[35] M. Tran, L. Luu, M. S. Kang, I. Bentov, and P. Saxena, "Obscuro: A bitcoin mixer using trusted execution environments," in *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018, pp. 692–701.
[36] Y. Shen, H. Tian, Y. Chen, K. Chen, R. Wang, Y. Xu, Y. Xia, and S. Yan, "Occlum: Secure and efficient multitasking inside a single enclave of intel sgx," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 955–970.
[37] A. Deshpande and M. Herlihy, "Privacy-preserving cross-chain atomic swaps," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 540–549.
[38] P. Robinson, R. Ramesh, and S. Johnson, "Atomic crosschain transactions for ethereum private sidechains," *Blockchain: Research and Applications*, p. 100030, 2021.
[39] A. Thomasian, "Two-phase locking performance and its thrashing behavior," *ACM Transactions on Database Systems (TODS)*, vol. 18, no. 4, pp. 579–625, 1993.
[40] A. Thomson and D. J. Abadi, "The case for determinism in database systems," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 70–80, 2010.
[41] X. Wang, O. T. Tawose, F. Yan, and D. Zhao, "Distributed nonblocking commit protocols for many-party cross-blockchain transactions," *arXiv preprint arXiv:2001.01174*, 2020.
[42] D. Zhao, "Cross-blockchain transactions," in *Conference on Innovative Data Systems Research (CIDR)*, 2020.
[43] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 136–145.
[44] 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, pp. 967–984.

APPENDIX

A. Supplementary Formalism

To analyze the security of the **I**vy**C**ross protocol Prot_{ccc}, we leverage the *Universal Composability* (UC) framework which can simplify the process of protocol analysis[43]. Specifically, we first present the high level description of the UC framework, and then model the ideal functionalities of blockchain $\mathcal{L}$ and TEE $\mathcal{G}_{att}$ that capture their core operations under the UC model, and then illustrate the formal protocol Prot_{ccc} of **I**vy**C**ross and the security analysis in the hybrid world. After that, we present the ideal functionality of $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$ of **I**vy**C**ross. To formalize the security of Prot_{ccc}, we illustrate definition 2 and specify the ideal functionalities, and then give the construction of Simulator $\mathcal{S}$.

B. Brief Description of the UC Model

The UC framework is one of the most widely used methods that are adopted to analyze the security of cryptographic protocols. It allows people to analyze their designed protocol isolatedly by using the *universal composition operation*. Such an operation enables us to construct a protocol based on cryptographic building blocks while still proving its security. The UC framework is essentially a simulation-based method that compares the execution of the designed protocol π under the *read world* with an idealized protocol under the *ideal world*. Specifically, in the real world, the execution of the designed protocol π is modeled as tuples $\{\mathcal{E}, \mathcal{A}, p_1, ..., p_n\}$, where $\mathcal{E}$ refers to the environment, $p_1, ..., p_n$ refer to the participants of π. Each participant may execute different modules of π. The real world is related with the execution of the real protocol

which here can be represented as $REAL_{\mathsf{Prot}_{ccc},\mathcal{A},\mathcal{E}}(\lambda, x)$. λ is the security parameter and x is the inputs of $\mathcal{E}$. On the other hand, the ideal world which is related with the execution of the ideal protocol can be presented as $IDEAL_{\mathcal{F}_{cce},\mathcal{S},\mathcal{E}}(\lambda, x)$, where $\mathcal{S}$ refers to the simulator $\mathcal{S}$ (called as "ideal adversary") that simulates the behaviors of the adversary in the real world. Specifically, if for any adversary $\mathcal{A}$, there exists a simulator in the ideal world that $\mathcal{E}$ can not distinguish whether it is interacting with the read world π and $\mathcal{A}$ or the ideal world $\mathcal{F}_\pi$ and $\mathcal{S}$, then a designed protocol π is said to UC-*realize* an ideal functionality $\mathcal{F}_\pi$.

C. UC Model for the Blockchains and TEEs

Due to space limits, we put a thorough security proof in a **public anonymous website** https://sites.google.com/view/ivycross/. It presents the global functionalities of blockchain $\mathcal{L}_{\mathcal{B}}$, TEE $\mathcal{G}_{att}$, and the construction and the validation of the simulator $\mathcal{S}$.

The random oracle ideal functionality $\mathcal{H}$. The security properties are provided in the common random oracle model which assumes that a hash function exists that no one can predict output random values. We utilize a random oracle ideal functionality $\mathcal{H}$ that responds to uniformly random numbers r. A global set R is used to store the query q and the return r, i.e., $(q, r) \in R$.

The global functionality of blockchain. In the cross-chain scenario, people have the requirements to exchange coins with others who lie in different (heterogeneous) blockchains. To support the security analysis, we utilize the existing model of [44] to model the basic properties of a global functionality $\mathcal{L}_{\mathcal{B}}$ for a blockchain. The initial state of $\mathcal{B}$ is public that participant $\{p_i\}_{i\in(1,\ldots,n)}$ have a balance $\{c_i\}_{i\in(1,\ldots,n)} \in \mathbb{N}$. A partial function F is defined that maps an identifier sid to an amount of coins locked in blockchain, e.g., use OP_CHECKSEQUENCEVERIFY in Bitcoin to restrict an execution of a script. The balance of participant p_i can be updated via the instruction *update*, *freeze* and *unfreeze* from the environment $\mathcal{E}$. In addition, each blockchain has a persistent storage Storage where a state (e.g., OP_RETURN in Bitcoin) and transactions can be stored. Storage is public that any party can read and write states on it, including the adversary $\mathcal{A}$.

The global functionality of TEE $\mathcal{G}_{att}$. We adopt the ideal functionality $\mathcal{G}_{att}$ introduced in [23]. A TEE is modelled as a basic attestation abstraction (cf Figure 12). The basic model of SGX execution can be depicted as two phases: 1) the initialization phase and 2) the enclave operations phase. In the first phase, $\mathcal{G}_{att}$ is initialized with a security parameter λ and parameterized with a registry $regN$ which refers two secure hosts equipped with an attested secure processor (e.g., an Intel SGX). In our design, we consider a static registry that contains an execution TEE T_0 and a verification TEE T_1. The second phase refers to the enclave operations on a host $\mathcal{N} \in (\mathcal{R}, \mathcal{V})$ registered in $regN$. $\mathcal{N}$ first creates an enclave instance and installs a program $pram$ which is to be executed into the secure environment, and then triggers it to execute with valid inputs in the attested execution processor.

The ideal functionality $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ for the CrossChainGame contract. The CrossChainGame contract is modelled as an ideal functionality $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ (cf. Figure 11). $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ maintains a global state machine s_{ccg} := INIED, ACTIVE, EXECG, FINAL which refers to the initialized, active, executing and finalized state. Specifically, the CrossChainGame contract proceeds in three phases. In the first phase, it receives input from a registered host $\mathcal{N} \in (\mathcal{R}, \mathcal{V})$. Both two hosts are required to make a deposit on $\widetilde{\mathcal{B}}$ in this phase.

In the execution phase, the contract stores an output which is sent by a global $\mathcal{G}_{att}$, $\mathcal{G}_{att0}$ into $\mathcal{L}_{\widetilde{\mathcal{B}}}$ upon receiving a write input, and responds an on-chain data upon receiving a read input. Specially, upon receiving a challenge input from the verification host $\mathcal{V}$, it runs a comparison inside the penalty function and responds to the output of this challenge to $\mathcal{V}$. According to the pre-defined penalty setting, $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ will update the balance of two hosts. If the challenge succeeds, $\mathcal{V}$ gets a challenge reward from $\mathcal{R}$. Otherwise, $\mathcal{R}$ is compensated by $\mathcal{V}$ for the cost of the transaction fee. In the finalization phase, two hosts can send an input $(finalize, sid)$ to finalize the protocol. Specially, it requires that $\mathcal{R}$ is able to redeem own deposit after $\mathcal{V}$ does.

Formal protocol of Prot_{ccc}. Given the global functionalities $\{\mathcal{L}_{\mathcal{B}_1}, \ldots, \mathcal{L}_{\mathcal{B}_m}\}$, $\mathcal{G}_{ccg}^{\mathcal{L}_{\widetilde{\mathcal{B}}}}$, $\mathcal{G}_{att0}$ and $\mathcal{G}_{att1}$, the formal protocol of Prot_{ccc} in the hybrid world can be shown in Figure 14. The main description of Prot_{ccc} has been given in Section V (cf. Figure 3, 5, and 6).

D. The ideal Functionality for cross-chain execution

As shown in Fig.13, the ideal functionality $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ describes an interaction between a set of participants $\{p_i\}_{i\in[n]}$ and the attested execution processors $\mathcal{G}_{att0}$ and $\mathcal{G}_{att1}$, a set of global functionalities of blockchains $\mathcal{L}_{\mathcal{B}_{[m]}}$. $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ interacts with a set of global functionalities to manage the balances of participants and two hosts in different blockchains. If there exists any dispute, $\mathcal{V}$ can challenge $\mathcal{R}$ through a predicate function $penalty(\cdot)$ in $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$. Specially, a global contract storage Set is used to store the deployed cross-chain contract. We assume that Set can preserve all of the old states of a contract, thus we conduct a rollback operation in case of a request.

Specifically, the lifecycle of a cross-chain smart contract (within the TEE) can be depicted as a state machine which contains six states s := {UNKN, INIT, DEPD, EXEC, EXED, FINA}, where s = UNKN refers to an initializing state that two hosts are under negotiation, s = INIT refers that two hosts have made deposits that can accept cross-chain contract deployment. Then s = DEPD denotes that a certain participant has deployed a contract into the secure attested processors. s = {EXEC, EXED, FINA} refer that a cross-chain contract lies in the state of executing, executed and finalized. Note that s = FINA denotes the participants have been assigned coins according to the output of execution.

The ideal functionality $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ consists of three phases: the initialization phase, round execution phase, and the

The ideal functionality $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ interacts as with the global functionality of $\mathcal{L}_{\widetilde{\mathcal{B}}}$, two hosts $(\mathcal{R}, \mathcal{V})$ who have the balance $(c_{k_0}, c_{k_1}) \in \mathbb{N}$, and two attested secure processors $\mathcal{G}_{att0}$, $\mathcal{G}_{att1}$. It stores the address of two hosts $pk_{\mathcal{R}}$ and $pk_{\mathcal{V}}$, and a global storage $\mathsf{Storage} := \emptyset$. Two penalty ratio γ_0 and γ_1 are set for hosts in case of dishonestly behavior.

Initialization

$\mathcal{R}$ **Deposit:** Upon receiving input $(init, sid, \mathcal{R}, c_{k_0})$ from $\mathcal{R}$ with $c_{k_0} \in \mathbb{N}$, send $(freeze, sid, \mathcal{R}, c_{k_0})$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$. If $\mathcal{L}_{\widetilde{\mathcal{B}}}$ responds with $(frozen, sid, \mathcal{R}, c_{k_0})$, then set $s_{ccg} = INITED$ and output $(accept, sid, \mathcal{R})$.

$\mathcal{V}$ **Deposit:** Upon receiving input $(active, sid, \mathcal{V}, c_{k_1})$ from $\mathcal{V}$ with $c_{k_1} \in \mathbb{N}$ when $s_{ccg} = INITED$, check if $pk_{\mathcal{R}}$ has frozen in $\mathcal{L}_{\widetilde{\mathcal{B}}}$. If yes, send $(freeze, sid, \mathcal{V}, c_{k_1})$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$ and $\mathcal{L}_{\widetilde{\mathcal{B}}}$ responds with $(frozen, sid, \mathcal{V}, c_{k_1})$. Set $s_{ccg} = ACTIVE$ and output $(accept, sid, \mathcal{V})$.

Execution

Write States: Upon receiving input $(write, sid, cid, inps)$ from a host $\mathcal{N} \in (\mathcal{R}, \mathcal{V})$ when $s_{ccg} = ACTIVE$. If $\mathsf{Storage}[sid]$ not found, set $\mathsf{Storage}[sid] = \perp$. Send $(write, sid, (cid, inps))$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$. If $\mathcal{L}_{\widetilde{\mathcal{B}}}$ responds with $(receipt, sid)$ then set $s_{ccg} = EXECG$ and output $(recorded, sid)$.

Read States: Upon receiving input $(read, sid, cid)$ from a host $\mathcal{N} \in (\mathcal{R}, \mathcal{V})$ when $s_{ccg} = EXECG$. Send $(read, sid)$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$. Output $(receipt, \mathsf{Storage}[sid][cid])$, or $\perp$ if not found.

Challenge Outputs: Upon receiving input $(challenge, sid, cid, tx_{cha}, tx_{res})$ from $\mathcal{V}$ when $s_{ccg} = EXECG$.

- If $penanlty(tx_{cha}, tx_{res}) = true$, send $(challenge, sid, cid, true)$ to $\mathcal{G}_{att0}$ and $\mathcal{G}_{att1}$, and send $(update, \mathcal{V}, c_{k_1} + \gamma_0 \cdot c_{k_0})$ and $(update, \mathcal{R}, (1-\gamma_0) \cdot c_{k_0})$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$. Output $(challenge, sid, cid, true)$.
- Otherwise, send $(challenge, sid, cid, false)$ to $\mathcal{G}_{att0}$ and $\mathcal{G}_{att1}$, and send $(update, \mathcal{V}, (1-\gamma_1) \cdot c_{k_1})$ and $(update, \mathcal{R}, c_{k_0} + \gamma_1 \cdot c_{k_1})$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$. Output $(challenge, sid, cid, false)$. Then proceed to the next round.

Finalization

$\mathcal{V}$ **Finalize:** Upon receiving input $(finalize, sid, \mathcal{V})$ from $\mathcal{V}$, check if all states of the global storage $\mathsf{Storage}[sid]$ are $FINA$. If yes, send message $(unfreeze, sid, c_{k_1}, \mathcal{V})$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$. Then output $(finalized, sid, \mathcal{V})$ to $\mathcal{V}$ and terminate.

$\mathcal{R}$ **Finalize:** Upon receiving input $(finalize, sid, \mathcal{V})$ from $\mathcal{R}$, check if all states in the global storage $\mathsf{Storage}[sid]$ are $FINA$, and $\mathcal{V}$ has received $(finalized, sid)$. If yes, send message $(unfreeze, sid, c_{k_0}, \mathcal{R})$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$. Then output $(finalized, sid, \mathcal{R})$ to $\mathcal{R}$.

Fig. 11. The ideal functionality $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ for the CrossChainGame contract.

The global functionality $\mathcal{G}_{att}$ is initialized with a security parameter λ, and running with a host $\mathcal{H}$, and a set of registry $regH$ that are equipped with a TEE. It executes upon receiving the following queries:

Initialization

$(mpk, msk) = \mathsf{TEE}.GenKey(1^\lambda)$, $T = \emptyset$.

Read Public Key: Upon receive$(getPK, _)$ from $\mathcal{H} \in regH$, send mpk to $\mathcal{H}$.

Enclave Operations

Install Enclave: Upon receive$(install, sid, pram)$ from an host $\mathcal{H} \in regH$. If $\mathcal{H}$ is honest, assert $idx = sid$ and generate a randomness $eid \in \{0,1\}^\lambda$, store $T[eid, \mathcal{H}] := (idx, pram, 0)$, and then send eid to $\mathcal{H}$.

Resume Enclave: Upon receive$(resume, eid, inps)$ from an host $\mathcal{H} \in regH$. Let $(idx, pram, mem) := T[ssid, \mathcal{H}]$, abort if not found. Let $(oupts, mem) := pram(inps, mem)$, update $T[ssid, \mathcal{H}] := [idx, pram, mem]$. Let $\sigma := \mathsf{TEE}.DS(Sign, idx, eid, pram, oupt, msk)$, and then send $(oupt, \sigma)$ to $\mathcal{H}$.

Fig. 12. The global functionality $\mathcal{G}_{att}$ modeling an TEE.

finalization phase. During the initialization phase, the functionality $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ requires inputs from two hosts who need to send the deposit transactions into $\mathcal{G}_{att0}$ and $\mathcal{G}_{att1}$ in order for proceeding to the next round. After that, a participant p_i could deploy a cross-chain contract $\mathsf{Contract}_{ccc}$ to the two secure environments. If the deployment succeeds, the functionality $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ can accept participants' deposit inputs. It goes to the execution phase if receiving n deposits.

In the execution phase, the functionality $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ accepts inputs from participants to trigger states transition of the $\mathsf{Contract}_{ccc}$. The correctness of states is guaranteed by the global $\mathcal{L}_{\widetilde{\mathcal{B}}}$ which stores the states on blockchain $\widetilde{\mathcal{B}}$ and maintains the state-lock mechanism in smart contracts. Lastly, the functionality $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ goes to the finalization phase and sends the deposits back to the corresponding participants based on the results of the smart contract, e.g., the lottery smart contract.

In order to prove that Prot_{ccc} UC-realizes $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ in the hybrid world, we need to construct an ideal adversary $\mathcal{S}$ with dummy parities (i.e., participants $\mathcal{P}$, two TEE-powered hosts $\mathcal{R}$ and $\mathcal{V}$) to show that the ideal world is indistinguishable from the hybrid world. In our setting, Prot_{ccc} can interact with two global functionalities $\mathcal{G}_{att0}$ and $\mathcal{G}_{att1}$, and a set of global blockchains $\{\mathcal{L}_{\mathcal{B}_j}\}_{j\in[m]}$, and a special global blockchain $\mathcal{L}_{\widetilde{\mathcal{B}}}$, a *Judge* smart contract $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$. Each cross-chain contract has a persist public space for storing all of the states. The contract $\mathsf{Contract}_{ccc}$ is considered as running in a black-box that an encrypted input is sent into it with responding an encrypted output.

In the ideal world, all parties can be considered as dummy parties that they only relay the input and output of $\mathcal{E}$ to $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$. Specifically, $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ allows an ideal adversary $\mathcal{S}$ to know a leakage information using a function $len(\cdot)$ which can obtain the length of the encryption. In addition, we follow the public *delayed output* terminology that a message is first sent to the adversary $\mathcal{A}$ before be sent to a party. Prot_{ccc} follows the standard corruption model in UC

The ideal functionality $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ interacts with the ideal adversary $\mathcal{S}$, a set of global blockchain functionalities $\{\mathcal{L}_{\mathcal{B}_j}\}_{j\in[m]}$, a global blockchain functionality $\mathcal{L}_{\widetilde{\mathcal{B}}}$, an execution host $\mathcal{R}$ with an ideal functionality $\mathcal{G}_{att0}$, a verification host $\mathcal{V}$ with an ideal functionality $\mathcal{G}_{att1}$, and a set of participants $\{p_i\}_{i\in[n]}$ who have the balances $\{c_i\}_{i\in[n]} \in \mathbb{N}$. $\mathsf{Set}(\cdot)$ is a contract storage.

Initialization Phase

(Host Deposit): Upon receiving input $(init, sid, \mathcal{N}, c_k)$ with $c_k \in \mathbb{N}$ from $\mathcal{N} \in (\mathcal{R}, \mathcal{V})$, leak $(init, sid, \mathcal{N}, c_k)$ to the simulator $\mathcal{S}$ and send $(accept, sid, \mathcal{N})$ to $\mathcal{N}$. Set $s = \mathsf{INIT}$ and initialize $\mathsf{Set} := \emptyset$. Then go to the next round.

(Deploy Contract): Upon receiving input $(deploy, sid, \mathsf{Contract}_{ccc}))$ from p_i $(i \in [n])$ when $s = \mathsf{INIT}$, generate a contract identifier $cid \leftarrow \{0,1\}^{\lambda}$, leak $(deploy, sid, cid, p_i, \mathsf{Contract}_{ccc})$ to $\mathcal{S}$ and store $\mathsf{Set}[cid] = \mathsf{Contract}_{ccc}$. Send $(write, sid, cid)$ to the $\mathcal{L}_{\widetilde{\mathcal{B}}}$. If $\mathcal{L}_{\widetilde{\mathcal{B}}}$ responds with $(receipt, sid)$ set $s = \mathsf{DEPD}$.

(Participant Deposit): Upon receiving input $(deposit, sid, cid, c_i)$ with $c_i \in \mathbb{N}$ from p_i $(i \in [n])$ when $s = \mathsf{DEPD}$, leak $(deposit, sid, cid, p_i, c_i)$ to $\mathcal{S}$ and send $(freeze, cid, p_i, c_i)$ to the blockchain $\mathcal{L}_{\mathcal{B}_j}$[a]. If successful, $\mathcal{L}_{\mathcal{B}_j}$ responds with $(frozen, cid, p_i, c_i)$. If received with n responses, set $s = \mathsf{EXEC}$ and then go to the execution phase.

Execution Phase

(Contract Request): Upon receiving input $(execute, sid, cid, inps_\iota)$ from p_i for $i \in [n]$ when $s = \mathsf{EXEC}$, $\mathsf{Contract}_{ccc} = \mathsf{Set}[cid]$, abort if not found. Read w_ι from $\mathcal{L}_{\mathcal{B}_j}$ and st_ι from $\mathcal{L}_{\widetilde{\mathcal{B}}}$. Leak $(execute, sid, p_i, len(inps), w_\iota)$ to $\mathcal{S}$. Compute $ct_\iota = \mathcal{AE}.Enc(inps_\iota, \widetilde{pk_{cid}})$, Then, send $(resume, sid, (cid, (ct_\iota, st_\iota, w_\iota)))$ to $\mathcal{G}_{att0}$ and respond with $(sid, (cid, oupts_\iota), \sigma_{mpk_{T_0}})$, leak $(cid, len(oupts_\iota), st_\iota, val_{w_\iota})$ to $\mathcal{S}$, and update $\mathsf{Set}[cid]$ with the latest state. Then, send $(write, sid, (cid, \iota, H(oupts_\iota), st_{\iota+1}, \mathsf{val}_{w_\iota}))$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$.

(Contract Execute): Upon receiving input $(execute, sid, cid, ct_\iota, w_\iota, sct_\iota)$, send $(resume, sid, (cid, ct_\iota, w_\iota, sct_\iota))$ to $\mathcal{G}_{att0}$ and $\mathcal{G}_{att1}$, and receives the responses $(sid, oupts_\iota, \sigma_{mpk_{T_0}})$ and $(sid, oupts'_\iota, \sigma_{mpk_{T_1}})$. Send $(write, sid, (cid, \iota, H(oupts_\iota), H(datagram_\iota), \mathsf{val}_{w_\iota}))$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$ and receives a response $(receipt, sid)$. If $\iota = \ell - 1$, then set $s = \mathsf{EXED}$ and go to the finalization phase.

(Contract Challenge): Upon receiving input $(challenge, sid, (cid, ct_\iota, \widetilde{w_\iota}, sct_\iota, sct_{\iota+1}))$ from $\mathcal{V}$, and update $\mathsf{Set}[cid]$ to the last state. Send $(resume, sid, (cid, (ct_\iota, sct_\iota, \widetilde{w_\iota})))$ to $\mathcal{G}_{att0}$ and receives a response $(sid, (cid, \widetilde{oupts_\iota}), \widetilde{\sigma_{mpk_{T_0}}})$. Leak $(cid, len(\widetilde{oupts_\iota}), sct'_\iota, val'_{w_\iota})$ to $\mathcal{S}$, and update $\mathsf{Set}[cid]$ with the latest state.

- If $penanlty(tx_{cha}, tx_{res}) = true$, send $(update, \mathcal{V}, c_{k_1} + \gamma_0 \cdot c_{k_0})$ and $(update, \mathcal{R}, (1-\gamma_0) \cdot c_{k_0})$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$ and output $(challenge, sid, cid, true)$.
- Otherwise, send $(update, \mathcal{V}, (1-\gamma_1) \cdot c_{k_1})$ and $(update, \mathcal{R}, c_{k_0} + \gamma_1 \cdot c_{k_1})$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$ and output $(challenge, sid, cid, false)$.

Finalization Phase

(Participant Finalize): Upon receiving input$(finalize, sid, cid, p_i)$ from p_i when $s = \mathsf{EXED}$, send $(unfreeze, sid, p_i, c_i)$ to the blockchain $\mathcal{L}_{\mathcal{B}_j}$ and receive a response $(frozen, sid, p_i, c_i)$. If receive with n responses, then set $set = \mathsf{FINA}$.

(Host Finalize): Upon receiving input$(finalize, sid, \mathcal{N})$ where $\mathcal{N} \in (\mathcal{R}, \mathcal{V})$ when $s = \mathsf{EXED}$, send $(finalize, sid, \mathcal{N})$ to $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ and receive a response $(finalized, sid, \mathcal{N})$.

[a] Assume that p_i is a user in the blockchain $\mathcal{B}_j$.

Fig. 13. Ideal functionality $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ for modeling cross-chain contract execution.

framework. Based on our security assumption, $\mathcal{E}$ can corrupt any of the participants or TEE hosts, while it can not corrupt both two hosts and all participants simultaneously. Once a party is corrupted, the secret input of the party is known by $\mathcal{E}$.

E. Construction of Simulator $\mathcal{S}$

Specifically, the simulator $\mathcal{S}$ proceeds as an ideal adversary that interacts with $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ and $\mathcal{E}$ in the ideal world. If an input is sent from an honest party, then $\mathcal{S}$ emulates as a real world "network traffic" to relay information which obtained from $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ to $\mathcal{E}$. On the other hand, if an input is sent from a dishonest party (also called as a corrupted party), then $\mathcal{S}$ interacts with these dishonest parties and extracts the inputs by communicating with $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$. In particular, we consider four construction of $\mathcal{S}$ under four hybrid world H_1, H_2, H_3, H_4: 1) all involving participants and two hosts are honest, 2) a certain participant behaves dishonestly (identified by p_i^*), 3) $\mathcal{R}$ behaves dishonestly (identified by $\mathcal{R}^*$) and 4) $\mathcal{V}$ behaves dishonestly (identified by $\mathcal{V}^*$).

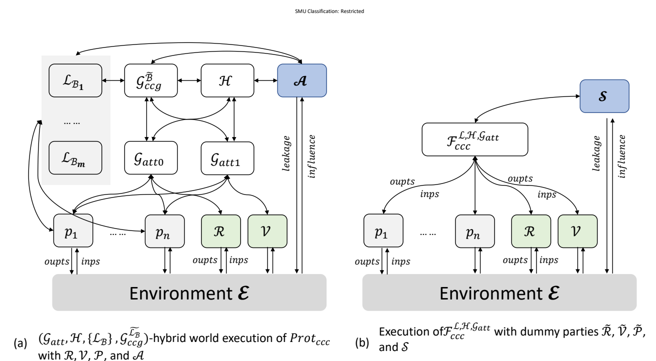


Fig. 15. Setup of a simulation with honest parties.

1) Simulation without corrupted parties: The setup of the simulation with honest parties is shown in Figure 15. In this case, all the involved parties, including the participants

The formal protocol Prot_{ccc} describes the behavior of a set of honest participants $\mathcal{P} = \{p_1, ..., p_n\}$ who have the balance of $\{c_i\}_{i\in[n]} \in \mathbb{N}$, two honest TEE-powered hosts $\mathcal{R}$, $\mathcal{V}$ who have the balance of $(c_{k_0}, c_{k_1}) \in \mathbb{N}$, and the corresponding global functionalities $\mathcal{G}_{att0}$ and $\mathcal{G}_{att1}$, and a set of global blockchain functionalities $\{\mathcal{L}_{\mathcal{B}_j}\}_{j\in[m]}$.

Initialization Phase

$\mathcal{R}$: Upon receiving input $(init, sid, \mathcal{R}, c_{k_0})$ in the deposit round, $\mathcal{R}$ sends $(init, sid, \mathcal{R}, c_{k_0})$ to $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ and $(init, _)$ to $\mathcal{G}_{att0}$, and $\mathcal{L}_{\widetilde{\mathcal{B}}}$ responds with $(accept, sid, \mathcal{R})$.

$\mathcal{V}$: Upon receiving input $(init, sid, \mathcal{V}, c_{k_1})$, $\mathcal{V}$ checks if $\mathcal{R}$ deposits in $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ and sends $(read, sid)$ to $\mathcal{V}_{\widetilde{\mathcal{B}}}$ and receives a response $(receipt, \mathsf{Storage(sid)})$. If it is found in $\mathcal{V}_{\widetilde{\mathcal{B}}}$, $\mathcal{V}$ sends $(init, sid, \mathcal{V}, c_{k_1})$ to $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ and $(init, _)$ to $\mathcal{G}_{att1}$. $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ responds with $(accept, sid, \mathcal{V})$. Then, $\mathcal{V}$ instructs $\mathcal{G}_{att1}$ to establish a secure communication channel with $\mathcal{G}_{att0}$ through a remote attestation.

$\mathcal{P}$: Upon receiving input $(deploy, sid, \mathsf{Contract}_{ccc})$ from environment $\mathcal{E}$, p_i checks if two hosts $\mathcal{R}$ and $\mathcal{V}$ have deposited in $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$. If yes, p_i sends $(install, sid, \mathsf{Contract}_{ccc})$ to the secure processors $\mathcal{G}_{att0}$ and $\mathcal{G}_{att1}$, and receives a response $(installed, cid)$. Then, $\mathcal{G}_{att0}$ sends $(write, sid, (cid, DEPLOD))$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$, and receives a response $(receipt, cid)$.

$\mathcal{P}$: Upon receiving input $(deposit, sid, cid, c_i)$, p_i checks if the state of $\mathsf{Contract}_{ccc}$ by sending $(read, cid)$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$, and receives a response $(receipt, \mathsf{Storage}[cid])$. If the initial state of $\mathsf{Storage}[cid]$ is confirmed, p_i sends $(freeze, sid, p_i, c_i)$ to $\mathcal{L}_{\mathcal{B}_j}$. $\mathcal{L}_{\mathcal{B}_j}$ responds with a response $(frozen, sid, p_i, c_i)$. p_i sends the response both to $\mathcal{G}_{att0}$ and $\mathcal{G}_{att1}$. If both $\mathcal{G}_{att0}$ and $\mathcal{G}_{att1}$ have received n distinct deposit responses from participants, then $\mathcal{G}_{att0}$ sends $(write, sid, (cid, DEPOSID))$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$ and goes to the execution phase.

Execution Phase

$\mathcal{P}$: Upon receiving input $(execute, sid, cid, inps_\iota)$ from environment $\mathcal{E}$, p_i computes $ct_\iota = \mathcal{AE}.Enc(\widetilde{pk_{cid}}, inps_\iota)$, and reads w_ι from $\mathcal{L}_{\mathcal{B}_j}$ and st_ι from $\mathcal{L}_{\widetilde{\mathcal{B}}}$, respectively. Then, p_i sends $(resume, sid, (cid, p_i, ct_\iota, w_\iota, st_\iota))$ both to $\mathcal{R}$ and $\mathcal{V}$, and receives a response $(sid, (cid, oupts_\iota), \sigma_{mpk_{T_0}})$ from $\mathcal{R}$. p_i verifies $\sum.Verify(mpk_{T_0}, oupts_\iota, \sigma_{mpk_{T_0}})$, and proceeds to the next round if succeeds.

$\mathcal{R}$: Upon receiving input $(execute, sid, cid, ct_\iota, w_\iota, sct_\iota)$ from p_i where $i \in [n]$, $\mathcal{R}$ instructs $\mathcal{G}_{att0}$ to send $(verify, sid, w_\iota)$ to $\mathcal{L}_{\mathcal{B}_j}$ and responds with (sid, v). If v is false, aborts. Otherwise, $\mathcal{R}$ sends $(resume, sid, (cid, ct_\iota, w_\iota, sct_\iota))$ to the $\mathcal{G}_{att0}$ and receives a response $(sid, oupts_\iota, \sigma_{mpk_{T_0}})$. After that, $\mathcal{G}_{att0}$ sends $(write, sid, (cid, \iota, H(oupts_\iota), H(datagram_\iota), \mathsf{val}_{w_\iota}))$ to $\mathcal{L}_{\widetilde{\mathcal{B}}}$, and $\mathcal{L}_{\widetilde{\mathcal{B}}}$ responds with $(receipt, sid)$. Meanwhile, $\mathcal{R}$ sends $(sid, (cid, oupts_\iota), \sigma_{mpk_{T_0}})$ to p_i and $(cid, \iota, oupts_\iota)$ to $\mathcal{V}$. If $\iota = \ell$, $\mathcal{R}$ instructs $\mathcal{G}_{att0}$ to send $(write, sid, (cid, FINA))$ and goes to the finalization phase.

$\mathcal{V}$: Upon receiving input $(execute, cid, sid, ct_\iota, w_\iota, sct_\iota)$ from p_i where $i \in [n]$, $\mathcal{V}$ sends $(verify, cid, w_\iota)$ to $\mathcal{L}_{\mathcal{B}_j}$ and receives a response (cid, v_0). If v_0 is true, $\mathcal{V}$ sends $(resume, sid, (cid, ct_\iota, w_\iota, sct_\iota))$ to the $\mathcal{G}_{att1}$ and responds with $(sid, oupts'_\iota, \sigma_{mpk_{T_1}})$. $\mathcal{V}$ compares the response with $\mathcal{R}$'s output. If there is difference between them, $\mathcal{V}$ sends $(challenge, sid, cid, tx_{cha}, tx_{res})$ to $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$, and $(challenge, cid, (ct_\iota, \widetilde{w_\iota}, sct_\iota, sct_{\iota+1}))$ to $\mathcal{R}$, respectively. $\mathcal{V}$ waits for a response within L_4 blocks.

$\mathcal{R}$: Upon receiving input $(challenge, sid, (cid, ct_\iota, \widetilde{w_\iota}, sct_\iota, sct_{\iota+1}))$ from $\mathcal{V}$, $\mathcal{R}$ sends $(resume, sid, (rollback, cid, sct_\iota, sct_{\iota+1}))$ to $\mathcal{G}_{att0}$ and receives a response $(sid, (cid, \widetilde{rollback, succeed}), \sigma_{mpk_{T_0}})$. Then, $\mathcal{R}$ sends $(resume, sid, (cid, ct_\iota, \widetilde{w_\iota}, sct_\iota))$ to $\mathcal{G}_{att0}$ and receives a response with $(sid, (cid, \widetilde{oupts_\iota}), \widetilde{\sigma_{mpk_{T_0}}})$. $\mathcal{R}$ sends $(challenge, sid, cid, tx_{cha}, tx_{res})$ to $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$, and $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ responds with $(challenge, sid, cid, v_1)$. $\mathcal{R}$ proceeds to the next round after L_4 blocks.

Finalization Phase

$\mathcal{P}$: A participant p_i checks if the contract $\mathsf{Contract}_{ccc}$ is $FINA$ and the timeframe of the deposit exceeds L_3 blocks. If yes, p_i sends $(resume, sid, (cid, finalize, p_i))$ to $\mathcal{G}_{att0}$, and receives a response $(sid, oupts_{fin}, \sigma_{mpk_{T_0}})$. Then p_i sends a redeem transaction (included in $oupts_{fin}$) to $\mathcal{B}_j$ and gets the deposit back.

$\mathcal{V}$: The verification host $\mathcal{V}$ checks if all states of contract in $\mathcal{L}_{\widetilde{\mathcal{B}}}$ is $FINA$. If yes, then sends $(finalize, sid, \mathcal{V})$ to $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$. $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ responds with $(accept, sid, \mathcal{V})$.

$\mathcal{R}$: The execution host $\mathcal{R}$ checks if $\mathcal{V}$ has sent *finalize* instruction to $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ and all states of contract in $\mathcal{L}_{\widetilde{\mathcal{B}}}$ is $FINA$. If yes, $\mathcal{R}$ can send $(finalize, sid, \mathcal{R})$ to $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$. $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$ responds with $(accept, sid, \mathcal{R})$.

Fig. 14. Formal protocol Prot_{ccc} description for honest participants and two hosts in the hybrid world.

$\mathcal{P} = \{p_1, ..., p_n\}$ and two hosts $\mathcal{R}$ and $\mathcal{V}$, behave honestly to accomplish a cross-chain smart contract, which is a special case that the simulation in this case is straight forward. The simulator $\mathcal{S}$ only generates and forwards the messages to $\mathcal{E}$ which are generated under the execution of Prot_{ccc}.

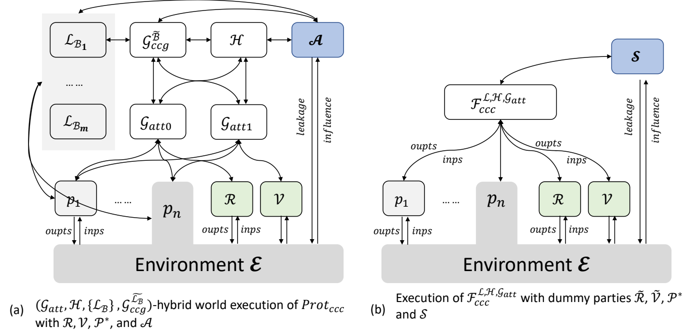
Claim 1. *There exists a simulator $\mathcal{S}^{honest}$ with no corrupted parties, that the execution of Prot_{ccc} in the $(\mathcal{G}_{att}, \mathcal{H}, \mathcal{L}_{\mathcal{B}}, \mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}})$-hybrid world, for any PPT adversary $\mathcal{A}$, is computationally indistinguishable from the execution of $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ with the simulator $\mathcal{S}$ in the ideal world.*

Proof. The proceeding of the simulation is defined as follows:

Firstly, two hosts $\mathcal{R}$ and $\mathcal{V}$ are honest that they follow the ideal protocol $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ to conduct their behaviors. The simulator $\mathcal{S}$ can obtain $(init, sid, \mathcal{N}, c_k)$ from $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ where $\mathcal{N} \in (\mathcal{R}, \mathcal{V})$, and emulate an execution of "init" call of Prot_{ccc} in the initialization phase. Then, a honest participant p_i who is in charge of deploying contract is honest, $\mathcal{S}$ can obtain $(deploy, sid, p_i, \mathsf{Contract}_{ccc}))$ from $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ and emulate the "deploy" call of Prot_{ccc}. Afterwards, similar with the host deposit, $\mathcal{S}$ can emulate the participant deposit on behalf of an honest participant. Secondly, during the execution of the contract, if a honest participant p_i who is given an input $(sid, cid, p_i, inps_\iota)$ to trigger the

state transition of $\mathsf{Contract}_{ccc}$. $\mathcal{S}$ can extract $len(inps_\iota)$ from $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$, and compute $ct'_\iota = \mathcal{AE}.Enc(\widetilde{pk_{cid}}, \vec{0})$. Then $\mathcal{S}$ sends the "read" queries to $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$ to obtain the dummy state and blockchain evidence (sct_ι, w_ι). Then, on behalf of p_i, $\mathcal{S}$ emulates a "resume" instruction $(resume, sid, (cid, ct'_\iota, sct_\iota, w_\iota))$ to $\mathcal{G}_{att0}$ and $\mathcal{G}_{att1}$, respectively. Upon receiving $(sid, (cid, oupts_\iota), \sigma_{T_0})$, $\mathcal{S}$ from $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$, $\mathcal{S}$ computes a ciphertext $oupt'_\iota$ where $oupt'_\iota = \mathcal{AE}.Enc(mpk_{T_0}, 0^{len(oupt_\iota)})$, and emulates a message $(sid, (cid, oupts'_\iota), \sigma_{T_0})$ from $\mathcal{G}_{att1}$ to p_i. In the finalization phase, $\mathcal{S}$ can emulate honest parties to interact with $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$, and output $(finalized, sid, \mathcal{N})$, $(frozen, sid, p_i, c_i)$.

It can be seen that running the simulator $\mathcal{S}$ with honest parties in the $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$ ideal world is indistinguishable from the $(\mathcal{G}_{att}, \mathcal{L}_{\mathcal{B}}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{L}_{\tilde{\mathcal{B}}}})$-hybrid world, unless the environment $\mathcal{E}$ can decrypt $oupt'_\iota$ to check that $oupt'_\iota \neq oupt_\iota$. However, this can only happen if $\mathcal{E}$ can break the security of $\mathcal{G}_{att0}$ or $\mathcal{G}_{att1}$ to obtain sk_{cid}, which is impossible as depicted in our security assumption.



(a) $(\mathcal{G}_{att}, \mathcal{H}, \{\mathcal{L}_{\mathcal{B}}\}, \mathcal{G}_{ccg}^{\mathcal{L}_{\tilde{\mathcal{B}}}})$-hybrid world execution of $Prot_{ccc}$ with $\mathcal{R}, \mathcal{V}, \mathcal{P}^*$, and $\mathcal{A}$ (b) Execution of $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{att}}$ with dummy parties $\tilde{\mathcal{R}}, \tilde{\mathcal{V}}, \mathcal{P}^*$, and $\mathcal{S}$

Fig. 16. Setup of a simulation with a corrupted participant p_n^* and honest $\mathcal{R}$, $\mathcal{V}$.

2) Simulation with a corrupted participant: The setup of the simulation with a corrupt participant p_n^* is shown in Figure 16. In this case, two hosts $\mathcal{R}$ and $\mathcal{V}$ still behave honestly to accomplish a cross-chain smart contract, while the private input $inps_\iota$ of the corrupted p_n^* can be learned by the environment $\mathcal{E}$ and the simulator $\mathcal{S}$. In this case, $\mathcal{S}$ can emulate the message between p_n^* and $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$ with guaranteeing the correctness of the contract states.

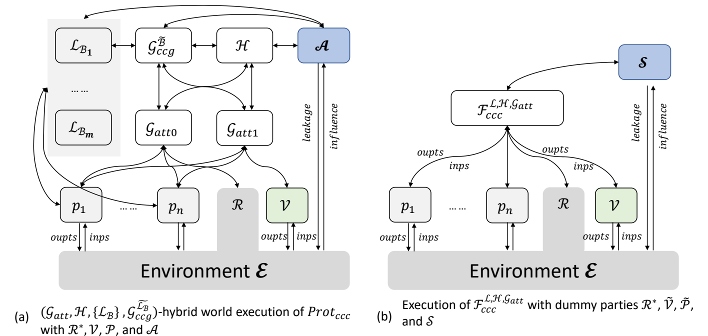
Claim 2. *There exists a simulator $\mathcal{S}^{\mathcal{P}}$ with a corrupted participant, that the execution of Prot_{ccc} in the $(\mathcal{G}_{att}, \mathcal{H}, \mathcal{L}_{\mathcal{B}}, \mathcal{G}_{ccg}^{\tilde{\mathcal{B}}})$-hybrid world, for any PPT adversary $\mathcal{A}$, is computationally indistinguishable from the execution of $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$ with the simulator $\mathcal{S}$ in the ideal world.*

Proof. The proceeding of the simulation with a corrupted participant is defined as follows:

Firstly, two honest hosts $\mathcal{R}$ and $\mathcal{V}$ follow the ideal protocol $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$ to accomplish the deposits as the above description. In the contract deployment, if p_n^* is responsible for deploying the contract $\mathsf{Contract}_{ccc}$, $\mathcal{S}$ can extract the input $\mathsf{Contract}_{ccc}$ from $\mathcal{E}$ and sends $(deploy, sid, p_i, \mathsf{Contract}_{ccc})$ on behalf of p_n^*. Meanwhile, $\mathcal{S}$ instructs $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$ to emulate the output $(deployed, sid, cid)$ from $\mathcal{G}_{att}$ to other participants $\mathcal{P}_{-p_n^*}$. Here, the behaviors of other participants $\mathcal{P}_{-p_n^*}$ are considered as honest to follow the protocol Prot_{ccc}. Afterwards, $\mathcal{S}$ can emulate the participant deposit on behalf of an honest participant upon receiving input from $\mathcal{E}$.

During the execution of the contract, if the dishonest participant p_n^* who is given an input $(sid, cid, p_n^*, inps_\iota^*)$. $\mathcal{S}$ can extract $len(inps_\iota)$ from $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$, and compute $ct'_\iota = \mathcal{AE}.Enc(\widetilde{pk_{cid}}, \vec{0})$. Then $\mathcal{S}$ sends the "read" queries to $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$ to obtain the dummy state and blockchain evidence (sct_ι, w_ι). Then, on behalf of p_i, $\mathcal{S}$ emulates a "resume" instruction $(resume, sid, (cid, ct'_\iota, sct_\iota, w_\iota))$ to $\mathcal{G}_{att0}$ and $\mathcal{G}_{att1}$, respectively. Upon receiving $(sid, (cid, oupts_\iota), \sigma_{T_0})$, $\mathcal{S}$ from $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$, $\mathcal{S}$ computes a ciphertext $oupt'_\iota$ where $oupt'_\iota = \mathcal{AE}.Enc(mpk_{T_0}, 0^{len(oupt_\iota)})$, and emulates a message $(sid, (cid, oupts'_\iota), \sigma_{T_0})$ from $\mathcal{G}_{att1}$ to p_i. In the finalization phase, $\mathcal{S}$ can emulate honest parties to interact with $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$, and output $(finalized, sid, \mathcal{N})$, $(frozen, sid, p_i, c_i)$.

It can be seen that running the simulator $\mathcal{S}$ with honest parties in the $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$ ideal world is indistinguishable from the $(\mathcal{G}_{att}, \mathcal{L}_{\mathcal{B}}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{L}_{\tilde{\mathcal{B}}}})$-hybrid world, unless the environment $\mathcal{E}$ can decrypt $oupt'_\iota$ to check that $oupt'_\iota \neq oupt_\iota$. However, this can only happen if $\mathcal{E}$ can break the security of $\mathcal{G}_{att0}$ or $\mathcal{G}_{att1}$ to obtain sk_{cid}, which is impossible as depicted in our security assumption. In addition, the states stored in DDS are also encrypted under the public key of $\widetilde{pk_{cid}}$, and thus are also computationally indistinguishable.



(a) $(\mathcal{G}_{att}, \mathcal{H}, \{\mathcal{L}_{\mathcal{B}}\}, \mathcal{G}_{ccg}^{\mathcal{L}_{\tilde{\mathcal{B}}}})$-hybrid world execution of $Prot_{ccc}$ with $\mathcal{R}^*, \mathcal{V}, \mathcal{P}$, and $\mathcal{A}$ (b) Execution of $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{att}}$ with dummy parties $\mathcal{R}^*, \tilde{\mathcal{V}}, \tilde{\mathcal{P}}$, and $\mathcal{S}$

Fig. 17. Setup of a simulation with a corrupted execution host $\mathcal{R}$ and honest $\mathcal{P}$, $\mathcal{V}$.

3) Simulation with a corrupted execution host $\mathcal{R}$: The setup of the simulation with a corrupt execution host $\mathcal{R}^*$ is shown in Figure 17. In this case, the participants $\mathcal{P}$ and the verification host $\mathcal{V}$ are honest. The simulator $\mathcal{S}^{\mathcal{R}}$ needs to simulate the instructions of Prot_{ccc} and all outputs of the corrupted host $\mathcal{R}^*$ towards $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$ and $\mathcal{E}$. In particular, the corrupted host $\mathcal{R}^*$ can disrupt or terminate the execution of T_0 at any point. $\mathcal{S}^{\mathcal{R}}$ only utilizes these inputs or instructions which are sent from $\mathcal{E}$ to simulate the execution of Prot_{ccc} by interacting with $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}},\mathcal{G}_{att}}$ on behalf of $\mathcal{R}^*$. If $\mathcal{E}$ instructs to terminate the

emulated simulation, $\mathcal{S}^{\mathcal{R}}$ also needs to instruct $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ to abort.

Claim 3. *There exists a simulator $\mathcal{S}^{\mathcal{R}}$ with a corrupted execution host $\mathcal{R}^*$, that the execution of Prot_{ccc} in the $(\mathcal{G}_{att}, \mathcal{H}, \mathcal{L}_{\mathcal{B}}, \mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}})$-hybrid world, for any PPT adversary $\mathcal{A}$, is computationally indistinguishable from the execution of $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ with the simulator $\mathcal{S}^{\mathcal{R}}$ in the ideal world.*

Proof. The proceeding of the simulation with a corrupted execution host is defined as follows:

Firstly, in the initialization phase, upon receiving an input $(init, sid, \mathcal{R}, c_{k_0})$ from $\mathcal{R}^*$, $\mathcal{S}^{\mathcal{R}}$ simulates the executions of the global functionality $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$by sending $(frozen, sid, \mathcal{R}^*, c_{k_0})$ to the $\mathcal{R}^*$. If $\mathcal{R}^*$ does not send $(init)$ message, $\mathcal{S}^{\mathcal{R}}$ aborts the simulation. Then, honest verification host $\mathcal{V}$ and the participant $\mathcal{P}$ follow the ideal protocol $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ to accomplish the deposits as the above description.

In the execution phase, when $\mathcal{S}^{\mathcal{R}}$ receives $(execute, sid, cid, ct_\iota, w_\iota, sct_\iota)$ from $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$, it means that the participants have deployed a contract $\mathsf{Contract}_{ccc}$ in $\mathcal{G}_{att0}$ and $\mathcal{G}_{att1}$. To simulate that the contract execution was performed by $\mathcal{G}_{att0}$ (with a valid attestation of T_0), $\mathcal{S}^{\mathcal{R}}$ sends the input to $\mathcal{G}_{att0}$. As analyzed in the sequential game, $\mathcal{S}^{\mathcal{R}}$ also verifies if the input w_ι is correct. If not, $\mathcal{S}^{\mathcal{R}}$ aborts the simulation. Otherwise, $\mathcal{S}^{\mathcal{R}}$ interacts with $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ and emulates $\mathcal{R}^*$ to output $(sid, (cid, oupts_\iota), \sigma_{mpk_{T_0}})$.

It can be seen that running the simulator $\mathcal{S}^{\mathcal{R}}$ with a corrupted execution host $\mathcal{R}^*$ in the $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ ideal world is indistinguishable from the $(\mathcal{G}_{att}, \mathcal{L}_{\mathcal{B}}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{L}_{\widetilde{\mathcal{B}}}})$-hybrid world. The output of Prot_{ccc} is computationally indistinguishable under the public encryption. Therefore, $\mathcal{R}^*$ does not know the inputs or the outputs of the execution. $\mathcal{S}^{\mathcal{R}}$ only needs to follow the instructions of $\mathcal{E}$ to run the simulation.

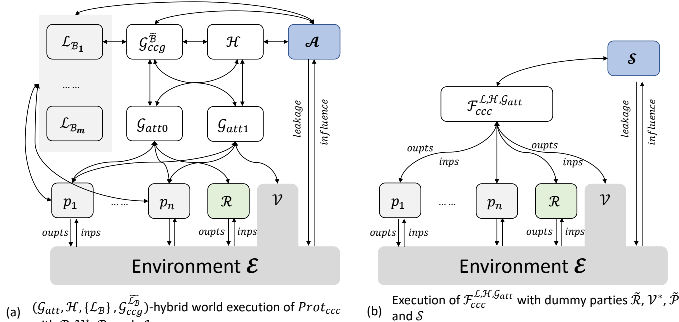


Fig. 18. Setup of a simulation with a corrupted verification host $\mathcal{V}$ and honest $\mathcal{P}$, $\mathcal{R}$.

4) Simulation with a corrupted verification host $\mathcal{V}$: The setup of the simulation with a corrupt verification host $\mathcal{V}^*$ is shown in Figure 18. In this case, the participants $\mathcal{P}$ and the execution host $\mathcal{R}$ are honest. The corrupted $\mathcal{V}^*$ can run the challenge instruction even though there is not wrong execution of $\mathcal{R}$. The simulator $\mathcal{S}^{\mathcal{V}}$ in this case needs to simulate the instructions of Prot_{ccc} and all outputs of the corrupted host $\mathcal{V}^*$ towards $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ and $\mathcal{E}$.

Claim 4. *There exists a simulator $\mathcal{S}^{\mathcal{P}}$ with a corrupted verification host $\mathcal{V}$, that the execution of Prot_{ccc} in the $(\mathcal{G}_{att}, \mathcal{H}, \mathcal{L}_{\mathcal{B}}, \mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}})$-hybrid world, for any PPT adversary $\mathcal{A}$, is computationally indistinguishable from the execution of $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ with the simulator $\mathcal{S}$ in the ideal world.*

Proof. The proceeding of the simulation with a corrupted verifcaition host is defined as follows:

Firstly, in the initialization phase, the execution host $\mathcal{R}$ first makes a deposit in $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$. Then, upon receiving an input $(init, sid, \mathcal{V}, c_{k_1})$ from $\mathcal{V}^*$, $\mathcal{S}^{\mathcal{V}}$ simulates the executions of the global functionality $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$by sending $(frozen, sid, \mathcal{V}^*, c_{k_1})$ to the $\mathcal{V}^*$. If $\mathcal{V}^*$ does not send $(init)$ message, $\mathcal{S}^{\mathcal{V}}$ aborts the simulation. Afterwards, the participant $\mathcal{P}$ follow the ideal protocol $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ to accomplish the deposits and contract deployment as the above description.

In the execution phase, when $\mathcal{S}^{\mathcal{V}}$ receives $(execute, sid, cid, ct_\iota, w_\iota, sct_\iota)$ from $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$, it sends the input to $\mathcal{G}_{att1}$ and receives a response with $(sid, (cid, oupts_\iota), \sigma_{mpk_{T_0}})$. No matter the verification result of the result, $\mathcal{E}$ can instruct $\mathcal{V}^*$ to send a challenge instruction to $\mathcal{R}$. Specifically, upon receiving an input $(challenge, cid, (ct_\iota, \widetilde{w_\iota}, sct_\iota, sct_{\iota+1}))$ from $\mathcal{E}$, $\mathcal{S}^{\mathcal{V}}$ sends it to $\mathcal{R}$ and waits for a response. If $\mathcal{R}$ does not respond in due time, $\mathcal{S}^{\mathcal{V}}$ notices the participants to abort on behalf of $\mathcal{V}^*$. Meanwhile, it emulates $\mathcal{G}_{att1}$ to respond with $(challenge, sid, cid, tx_{cha}, tx_{res})$, and instructs $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ to send it to $\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}}$.

It can be seen that running the simulator $\mathcal{S}^{\mathcal{V}}$ with a corrupted execution host $\mathcal{V}^*$ in the $\mathcal{F}_{ccc}^{\mathcal{L},\mathcal{H},\mathcal{G}_{ccg}^{\widetilde{\mathcal{B}}},\mathcal{G}_{att}}$ ideal world is indistinguishable from the $(\mathcal{G}_{att}, \mathcal{L}_{\mathcal{B}}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{L}_{\widetilde{\mathcal{B}}}})$-hybrid world. The output of Prot_{ccc} is computationally indistinguishable under the public encryption, and $\mathcal{V}^*$ does not know the inputs or the outputs of the execution. $\mathcal{S}^{\mathcal{V}}$ only emulates $\mathcal{V}^*$ to send a challenge instruction with a public blockchain evidence $\widetilde{w_\iota}$ by following the instructions of $\mathcal{E}$.