

IvyCross: A Trustworthy and Privacy-preserving Framework for Blockchain Interoperability

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

Abstract—Blockchain interoperability is essential for the long-envisioned cross-chain decentralized applications. Existing hardware-based approaches enable privacy-preserving off-chain execution and high scalability. However, they demand several Trusted Execution Environments (TEEs) and large storage on the storage-limited TEEs. This paper presents a TEE-based privacy-preserving blockchain interoperability framework, calls as IvyCross, which decreases the requirement of TEE numbers and TEE’s storage sizes by enforcing honest behaviors of TEE hosts with economic incentives. Specifically, IvyCross runs privacy-preserving cross-chain smart contracts atop two distributed TEE-powered hosts, and utilizes a sequential game between rational hosts to guarantee the correctness of contracts execution. IvyCross enables arbitrarily complex smart contracts execution across heterogeneous blockchains at low costs. We formally prove the security of IvyCross in the Universal Composability framework. We also implement a prototype of IvyCross 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 on-chain transaction fees in cross-chain communication are relatively low.

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

I. INTRODUCTION

In the last decade, blockchain technology has been employed in many fields such as internet of thing [1], cloud computing [2], and Decentralized finance [3], since it was coined in 2008 [4]. It becomes a revolutionary technology that numerous tech giants, such as IBM and Google, are investing large funds in research and applications. According to the recent statistics¹, the global market value of blockchain size has reached \$1,590.9 million in 2018, and is expected to grow at a compound annual growth rate 69.4% from 2019 to 2025. With remarkable benefits brought by blockchain technology, it can be expected that a more trustworthy digital society will emerge in the near future.

With the advent of numerous blockchains having been deployed independently, people have recognized that blockchain is not a “*coin to rule them all*” solution that all applications can be put in a single blockchain [5]. The scalability challenge and

innovation constraints have limited its application. A number of protocols have been proposed to enable interoperability across heterogeneous blockchains, aiming to resolve the limitation of scalability, bootstrap feature extensions and cryptocurrency exchanges [6], [7], [8], [9], [10], [11], [12]. Most of the protocols focus on atomic cryptocurrency exchange between distinct blockchains in an all-or-nothing manner [6].

Unfortunately, existing centralized or decentralized protocols for achieving blockchain interoperability suffer from several security problems. Concretely, the *centralized* protocols suffer from the weaknesses of single point of failure (SPoF), such as the theft events in MtGox and Bitfinex exchange [12], while the *decentralized* protocols like HyperService [7] and EtherDelta [10] are lack of privacy protection and vulnerable to front-running attacks [12]. Although many protocols have been proposed to address the privacy issue in a single blockchain [12], [11], [13], they can not be adopted directly in cross-chain² communication (CCC) [5] for two primary reasons: (i) they are designed for a specific blockchain (e.g., ZeroCash [14] or Monero [15]) that can not be integrated into other blockchains with different consensus mechanisms. (ii) They might be inefficient in the face of complex interactions due to the use of heavy cryptographic techniques, e.g., zero-knowledge proof or secure multiparty computation (SMC).

An alternative and comparably efficient approach is to resort to Trusted Execution Environments (TEEs), but previous proposals have yet truly taken off. TEEs enable an isolated environment that guarantees the confidentiality and integrity of the loaded code and data, to support off-chain confidentiality-preserving contract execution [12], [13], [16], [17], [11]. However, existing proposals such as Ekiden [13] and Teechain [16] utilize a pool of TEEs, which suffers from several drawbacks: (i) *high monetary cost*, as all involved TEEs may charge a substantial fee for the off-chain execution; (ii) *high storage demand*, since each TEE is required to store a large number of block headers (e.g., 8,064 block headers [12]) to verify the correctness of on-chain data, and dealing with multiple chains worsen the situation; (iii) *heavy communication complexity and latency*, because several TEE nodes need to interact with each other for key management and data decryption.

Motivated by the observations above, we propose IvyCross, a novel framework that leverages only two TEE-powered hosts to enable privacy-preserving smart contracts for blockchain interoperability. IvyCross starts by using TEEs as an off-chain intermediary, following a widely adopted approach [13],

M. Li, R. Deng are with the School of Computing and Information Systems, Singapore Management University, 178902, Singapore. Y. Li is 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.grandviewresearch.com/industry-analysis/blockchain-technology-market>

²We will use the terms “cross-chain” and “blockchain interoperability” interchangeably in this paper.

[17], [12]. Yet, it differs from the previous “TEE-and-blockchain-combination” works that utilize multiple TEEs to run on different hosts. This work enables trustworthy and privacy-preserving cross-chain communication across different blockchains using only two TEE-powered hosts, i.e., an execution host and a verification host. In addition, to reduce the storage cost of TEEs, we avoid storing a large number (e.g., 8,064) of block headers in each TEE, but storing only a small fraction, e.g., $k = 6$ blocks in Bitcoin. A cross-chain smart contract runs simultaneously on both hosts, and the verification host can challenge the execution host in case of incorrect execution. Meanwhile, the two hosts are not necessarily to be fully honest, in light of the recent attacks on TEEs [18], [19]. Therefore, we employ a game theoretic mechanism for preventing the collusion between the two hosts, and further enforce their honest behaviors. As a result, IvyCross enjoys three desirable features that previous proposals may not provide: (i) *Low economic overhead*. A two-TEE based CCC 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) *Correct execution*. The correctness of TEE-based smart contracts execution is ensured, e.g., atomicity guarantees. Besides, the security of IvyCross are formally analyzed using the Universal Composability (UC) framework. We emphasize that the present framework can be leveraged to improve scalability of existing blockchains owing to the enabling of convenient and secure communication capability.

Our Contributions. In this work, we present an interoperable, trustworthy, and privacy-preserving smart contracts execution framework for cross-chain applications. Specifically, the major contributions of this paper can be summarized as follows:

- **A framework for enabling privacy-preserving cross-chain smart contracts.** Based on two TEE-powered hosts, we design a novel framework called IvyCross that can achieve multiple-round cross-chain smart contracts for blockchain interoperability while preserving privacy. We formalize the process of a cross-chain smart contract and enumerate the design challenges (as depicted in Section V), and propose general-purpose methods to address them.
- **Correct off-chain execution (i.e., practical smart contracts) for blockchain interoperability.** IvyCross allows to design an off-chain execution across heterogeneous blockchains using only two TEE-powered hosts. Since we utilize a smart contract-supporting blockchain to model a sequential game between two hosts, IvyCross can guarantee the correctness of off-chain execution at low costs. The game theory-based TEEs computation has wider applications beyond blockchain interoperability.
- **Formal Security Analysis.** We model IvyCross in the UC framework, and formally prove the security of our construction with respect to *privacy preservation* and *trustworthy contract execution*.
- **Implementation and Evaluation.** We implement a prototype of IvyCross over three blockchains, including

Bitcoin, Ethereum and FISCO BOCS, and evaluate the performance with three real-world use cases. Extensive experiment results demonstrate the practicality and efficiency of IvyCross.

II. RELATED WORK

In this section, we review the state-of-the-art schemes which are related with blockchain interoperability. We will discuss the differences between them and IvyCross in section VII-B.

Blockchain Interoperability Technologies. Blockchain interoperability is a hot topic with the rapid adoption of blockchain both in academic and industrial communities. In general, it can be mainly classified into two categories: asset exchange and information (or state) exchange. The atomic swap, proposed by Herlihy [6], 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 heterogeneous blockchains. Cross-chain smart contract, proposed by Liu *et al* [7], is the first system that supports smart contract applications across blockchains. However, it does not preserve the privacy for state transition in cross-chain up to now. Polkadot [8] and Cosmos [20] are two typical projects that aim to provide interoperability framework for blockchain. They construct the fully decentralized “blockchain of blockchains” to realize cross-chain communication. Currently, there are still in the process of development and testing.

Privacy-preservation for Blockchain Interoperability. Privacy preservation is a necessity in cross-chain applications. Recently, Deshpande [21] 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. P2DEX, a privacy preserving decentralized exchange, was designed by Baum [9] that utilized multi-party computation (MPC) to enable asset exchange across different public blockchains. On the other hand, there have recently been some efforts on using trusted hardware, e.g., TEE, to improve the functionality and security of cross-chain. A prominent example is Tesseract [12], which utilizes an Intel SGX to support real-time cross-chain cryptocurrency trades against front-running attack. Yan [11] also utilized the TEE as a component to support confidentiality. Teechain [16] constructed a layer-two payment network that can execute off-chain transactions across blockchains. Compared with these, IvyCross considers a more economical solution to reduce the communication and computation overhead. Adaption of cryptography-based primitives, e.g., zero-knowledge proof and homomorphic encryption, to IvyCross for secure multi-computation is a direction for our future work.

III. PRELIMINARIES

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

Blockchain and Cross-chain Communication (CCC). A *blockchain* is essentially a transparent, immutable distributed

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 $(0, 1, \dots, m)$.
\mathcal{P}	The participants.
\mathcal{A}	The adversary.
\mathcal{R}, \mathcal{V}	The execution host and verification host.
$\mathcal{B}_j, \tilde{\mathcal{B}}$	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.
<i>datagram</i>	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 height in blockchain $\tilde{\mathcal{B}}$.
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].
<i>cid</i>	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 .
$\mathcal{K.A.}(KGen, Retrieve)$	The key generation and retrieve algorithm in key agreement.
$\mathcal{A.E.}(KGen, Enc, Dec)$	The key generation, encryption, and decryption algorithm in an asymmetric encryption scheme.
$\Sigma.(KGen, Sign, Verify)$	The key generation, signing, and verification algorithm in digital signature.

ledger that is maintained by many decentralized networks of peers (i.e., blockchain nodes). The blockchain nodes are responsible for verifying the validation of transactions with a consensus protocol, e.g., *proof-of-work* (PoW) [4] and *proof-of-stake* (PoS) [1], [2]. With such a distributed ledger, transactions can be recorded and managed without relying on a central party, reducing the risk of SPoF. Moreover, no participant could tamper with a transaction once it has been recorded in blockchain, and each participant can check the historical details of a transaction, facilitating tracking the digital assets in a trustworthy way. Further, smart contracts are introduced into blockchain for supporting the Turing-complete rules of a real-world application. It defines the specific conditions and is executed automatically in the blockchain upon valid inputs.

A ‘‘coin to rule them all’’ is unlikely due to the heterogeneous nature of the blockchain ecosystem. Thus, with the rapid progress of blockchains, the CCC technologies have emerged to resolve the limitation of blockchain scalability[5] and allow two relatively dependent blockchains (e.g., Bitcoin and Ethereum) to interact with each other, e.g., cryptocurrency exchange [6] and information transitions [7]. In particular, a CCC is said to be correct if it satisfies *effectiveness*, *atomicity* and *timeliness* [5]. Taking cryptocurrency exchange as an example, a user Alice intends to exchange ETH with an Ethereum user Bob using her Bitcoin. It takes four steps to accomplish this: 1) Setup, 2) (pre-)commit on Bitcoin, 3) verify and 4a) commit on Ethereum or 4b) abort [5]. The first step is to set the initial parameters between Alice and Bob, including time constraints, and the transferred coin amount. In the second step, Alice writes a commitment transaction

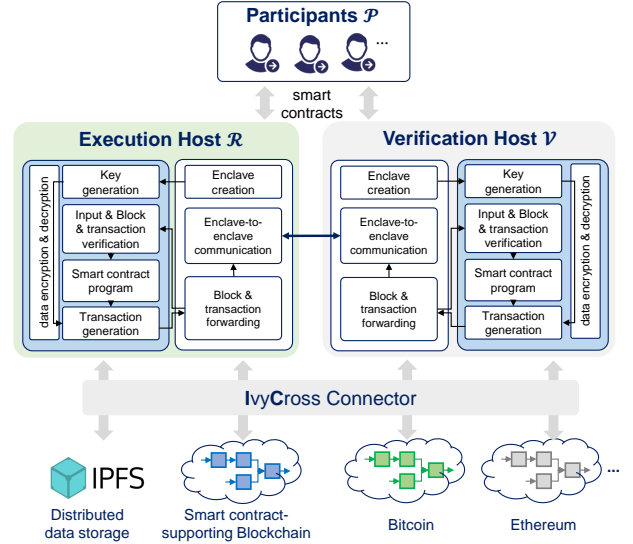


Fig. 1. The system model of IvyCross.

which sends an amount of Bitcoin to Bob on Bitcoin. Then, once Bob has confirmed the correctness and validation of the commitment transaction in the third step, he can write a commitment transaction that transfers an equivalent of ETH to Alice on Ethereum.

Trusted Execution Environment. Trusted Execution Environment (TEE) is one of the key building blocks of IvyCross. It builds an isolated environment that enables hardware-based protections on user-level code and data to provide confidentiality and integrity. Presently, a series of hardware-based protection technologies have been designed to support TEE, such as Intel SGX [22], ARM TrustZone and AMD Secure Processor [23]. We use Intel SGX in the implementation of IvyCross. Specifically, Intel SGX creates a sandbox environment called *enclave* in which a program can be executed without interference. When an enclave needs to interact with other enclaves, Intel SGX introduces *attestation* that allows users to trust each other. It is essentially a proving process that enables a user to trust: 1) his program is running in a trusted enclave which can use a secret securely, 2) the enclave is up to date. There are two types of attestation, *local attestation* and *remote attestation*, which can support a proof from the local and remote platforms, respectively. In terms of remote attestation, Intel SGX provides Intel Attestation Service (IAS) that helps to verify an attestation with a signature. We recommend readers to review [23] for more information about Intel SGX.

IV. IVY CROSS MODELS

In this section, we formalize the proposed IvyCross framework by illustrating the system model and security model.

A. System Model

IvyCross is a framework for achieving privacy-preserving smart contracts for blockchain interoperability across heterogeneous blockchains by leveraging the power of TEE. The IvyCross system model involves four parties (cf. Figure 1):

the participants, the execution host, the verification host and the blockchain nodes.

- **The participants**, identified by $\mathcal{P} = \{p_1, \dots, p_n\}$, refer to the users who invoke cross-chain interaction to achieve cryptocurrency exchange or state transition across 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) 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. We do not differentiate whether a blockchain node belongs to a UTXO model or an account-based model. They just take the responsibility of recording transactions and responding upon receiving a call.

As shown in Figure 1, the IvyCross framework is composed of three components: an IvyCross connector and two hosts equipped with an Intel SGX. The IvyCross connector acts as a gateway to enable IvyCross to interact with different blockchains and a distributed data storage (e.g., IPFS [24]). IvyCross utilizes IPFS as a public bulletin board to store intermediary data. 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. Each host consists of two parts: (i) the trusted part is executed in memory-isolated SGX enclave. It is responsible for the private computation including key generation, data (e.g., inputs, transactions) encryption and decryption, and transaction generation. (ii) The untrusted 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. Two enclaves created in different hosts communicate using remote attestation and execute a cross-chain contract simultaneously.

Specifically, to run a cross-chain contract, two hosts are required primarily to deposit in a smart contract-supporting blockchain (identified by $\tilde{\mathcal{B}}$), e.g., Ethereum. After that, the participants register³ in IvyCross and one of them deploys a smart contract by using the SGX *install* instruction. All participants need to deposit in their blockchains and send the deposit transaction to the enclaves. Then, it goes to the contract execution phase. An output is sent to participants through remote attestation which can prove the correctness of contract execution. According to the final results of the contract, these deposits are assigned or withdrawn by the participants.

B. Security Model

Security Threats. We consider a probability polynomial time (ppt) adversary \mathcal{A} that can corrupt a subset of parties, which means that the internal states and operations of the corrupted

parties are controlled by \mathcal{A} . Besides, 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, or (iii) replay old stored or incorrect state to the TEE (i.e., rollback attacks). The host \mathcal{V} may also behave dishonestly that challenges \mathcal{R} by providing on-chain state which are from an incorrect blockchain. Besides, both \mathcal{R} and \mathcal{V} are curious about participants' sensitive inputs and attempt to obtain the plaintexts. With respect to participants, some of them might collude with \mathcal{R} or \mathcal{V} to maximize their profits in a smart contract (e.g., to let him win a lottery game across blockchains), and attempt to obtain others' private inputs. Besides, one of the two hosts might connect to a certain blockchain node which is compromised by the *eclipse attack* [25], causing a TEE host to get a fake or overdue evidence.

Security Assumptions. The security of IvyCross depends on the security of TEEs and blockchains, thus we make the following assumptions with respect to the TEE and blockchain technology. Specifically, a TEE can guarantee the confidentiality of programs in execution. 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 realistic 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. Two hosts are considered as rational and incentive-driven service providers that aim to make profits in the process. In addition, to prevent denial-of-service (DoS) attacks [17], 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 with the agreement of participants and hosts. Besides, a cross-chain contract deliberately uses an old state in a specific block as input is not considered here.

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.

C. Design Goals

IvyCross aims to achieve the general purpose smart contracts for blockchain interoperability. Here, we summarize the security goals of IvyCross as follows:

- **Privacy-preserving cross-chain communication:** the contract state and honest participants' inputs are kept secret from all other participants.
- **Correct contract execution:** smart contracts executed in two TEE-powered hosts can be guaranteed with correct state transitions.

³Here, we do not discuss participants' identity management which has been extensively studied by [2], [13].

- **Interoperability:** IvyCross can be seamlessly integrated into currently existing blockchains to achieve information and cryptocurrency exchange, which is not limited to any specific blockchains.
- **Security against attacks:** IvyCross is able to defend against collusion attacks, eclipse attacks, and rollback attacks.

V. DESIGN CHALLENGES

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

A. Collusion

The biggest challenge of relying on only two TEE-based hosts is that they might collude with each other. There is not mechanism for contract participants to detect such collusion attacks, if both hosts coordinate and output incorrect contract execution results. Worse still, the hosts may collude with a certain participant to gain profits from a smart contract application, e.g., a cross-chain lottery game. More concretely, when each participant is required to make a deposit when they join the game, malicious participants can collude with the hosts to gain profits. For instance, they may provide an invalid deposit transaction (from an incorrect blockchain) to deceive the TEE into accepting it. Our key insight here is that collusion always occurs when participants find it is more profitable by colluding with each other than behaving honestly, and therefore economic incentives can be used to thwart collusion attacks. Towards this end, we employ game theory based on a Turing-complete smart contract to resolve this challenge.

B. Attacks on Hosts

Although TEEs have been employed to provide an attested secure environment for program execution, they still have to interact with the external environment through a host OS, which may be compromised. Therefore, a TEE-powered host is still prone to a series of network attacks. More precisely, the host might be compromised by an adversary and schedule the order of the transactions in the memory arbitrarily. Besides, the cross-chain scenario employs a complex network structure that involves multiple blockchains, where blockchain nodes can be compromised (e.g., by the eclipse attack [25]), making a TEE suffer from potential threats not only from its running host, but also from external blockchain nodes. On the other hand, a program is usually executed in a TEE for several rounds. Due to unpredictable factors, the host might be powered off and thus lose previous states of the running programs. If this happens, then a malicious host can launch rollback attacks by replacing the latest data with an older (or incorrect) encrypted copy [19]. Current research works resort to using a cluster of hosts (or a trusted-authority) to tackle this challenge, e.g., ROTE [18], and therefore suffer from high communication costs. Our aim is to achieve the same security goal with lower costs (and better security than a trusted-authority).

C. Limited storage

A cross-chain smart contract is a multiple-round interaction among many participants that relies on data from different blockchains, the *correctness* of these data are vital for the contract execution in a TEE. Existing works address this issue by introducing several TEEs and loading numerous block headers ($\sim 5\text{MB}$ for 8,064 block headers) to each TEE [13], [17], [12]. However, there are several shortcomings of this approach. First, a TEE is a storage-limited environment. In particular, the reserved memory for an Intel SGX application is limited to a total of 128 MB currently [26]. Thus, as many blockchains get involved into CCC, this approach will cause TEE to exceed the storage capacity. One may suggest to leverage external storage to mitigate this issue, but it might suffer from rollback attacks as mentioned in Section V-B. Second, a TEE is required to keep being connected to update the stored data periodically, thus incurring high computation and communication cost. Therefore, it remains challenging to guarantee the correctness of on-chain data owing to the limitations of TEEs.

VI. IVY CROSS PROTOCOL

In this section, we begin with modelling a cross-chain smart contract, and then introduce an overview of IvyCross. Afterwards, we formally specify the concrete design, and elaborate a game theory-based method for enhancing the security of IvyCross.

A. Modelling Cross-chain Smart Contracts

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

$$\text{Contract}_{ccc} := (cid, st, ents, ops, dep, \{\mathcal{B}_j\}_{j \in [m_0]}), \quad (1)$$

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. $\{\mathcal{B}_j\}_{j \in [m_0]}$ is the set of involved blockchain systems, where $m_0 \leq m$. A key pair $(\widetilde{pk_{cid}}, \widetilde{sk_{cid}})$ stored within both T_0 and T_1 is generated for the contract using $\mathcal{K.A.KGen}(1^\lambda)$. The enclave publishes $\widetilde{pk_{cid}}$ and keeps $\widetilde{sk_{cid}}$ private. Specifically, an interaction (i.e., a transaction) between a participant p_i and Contract_{ccc} is modeled as the following 5-tuple:

$$tx_{cce} := (cid, ct, w, ti, \sigma_{p_i}), \quad (2)$$

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 that contain some transactions and the corresponding block information. For instance, $w := (\mathcal{B}_j, tx, mkR, diff, hd, bh)$, where tx is the latest transaction involved w , mkR is the

Merkle path for authenticating tx , $diff$ is the difficulty level, hd is the block header and bh is the corresponding block height. ti is the timestamp of the transaction, σ_{p_i} is a signature of p_i which is used to identify the identity. Specifically, an interaction between a participant and the TEE can be modelled as the following state transition:

$$TEEContract(cid, st_l; r_l) \xrightarrow{tx_{ccc}} (oupts_l, st_{l+1}), \quad (3)$$

where st_l refers to the previous state which is securely stored in the IPFS, r_l refers to the randomness of the round l ($l \in [\ell]$). An enclave triggers a state transition by taking tx_{ccc} as an input, and outputs $oupts_l$ and generates a new state st_{l+1} . Specially, we follow the output method of an attestation as in Ekiden [13] where an output $oupts_l$ of an SGX program contains a pair of parts ($oupt_{l_1}, oupt_{l_2}$). A hash of the second output $H(oupt_{l_2})$ and an input $h_{inps} = H(ct_l, w_l)$ of the program are included in $oupt_{l_1}$. Only $oupt_{l_1}$ is signed by the TEE in an attestation, i.e., $\sigma_T := \sum .Sign(msk_T, oupt_{l_1})$.

B. Protocol Overview

IvyCross is essentially built as a general-purpose framework that interacts with heterogeneous blockchains and achieves secure off-chain computation with receiving, generating, and verifying transactions using the TEE. Specifically, we leverage several useful tools such as the OpenSSL toolkit for encryption and signature, Occlum OS for running an SGX program [27]. Ethereum is adopted as the underlying blockchain $\tilde{\mathcal{B}}$ for supporting on-chain smart contract between two hosts. The CrossChainGame contract (serves as a *Judger*) is designed for \mathcal{R} and \mathcal{V} to play a *challenge-and-response* game. One host can retrieve another's deposit in CrossChainGame by providing valid proofs in case of any inappropriate behavior [17]. Specifically, the main protocol of IvyCross $Prot_{ccc}$ consists of four basic phases: the initialization phase, the execution phase, the challenge-and-response phase, and the finalization phase. Here, we present a high-level overview of the four phases as follows:

- **Initialization.** In this phase, all of the involved parties (including \mathcal{P} , \mathcal{R} and \mathcal{V}) generate their key pairs and make deposits. After that, a registered participant can deploy a smart contract (i.e., an enclave program) into the enclaves T_0 and T_1 by using the *install* operation of Intel SGX. The rest of participants can check the correctness of the contract deployment through output attestation. If it is correct, they may make deposits in order to participate in the contract. Note that two enclaves cooperatively generates a different set of public addresses to accept participants' deposits for different contracts, which is to prevent dishonest participants from participating in different contracts using the same deposit.
- **Execution.** Participants can send inputs to the contract to trigger a state transition in this phase. The output of each round is encrypted and stored in IPFS with its hash value recorded in $\tilde{\mathcal{B}}$ for state persistence. If there is no dispute so that two hosts and participants are honest in this phase, the contract goes to the finalization phase. Otherwise, it enters the next challenge-and-response phase.

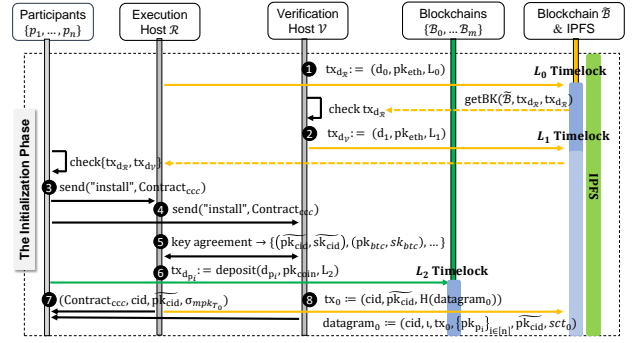


Fig. 2. The initialization phase of $Prot_{ccc}$. Solid dark arrows indicate interactions among TEEs and participants. Yellow dashed arrows denote interactions among participants, TEEs and $\tilde{\mathcal{B}}$, green dashed arrows denote interactions between participants, TEEs and $\{\mathcal{B}_j\}_{j \in [m]}$. Blue bands indicate the time interval of the deposits.

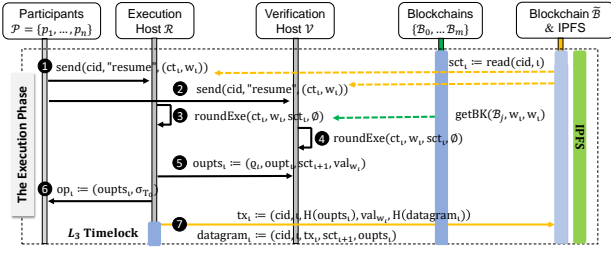
- **Challenge-and-response.** This phase happens when one of the parties behaves dishonestly, e.g., \mathcal{R} colludes with a participant. It is conducted between \mathcal{V} (as a challenger) and \mathcal{R} (as a responder) within a time of several blocks. More precisely, if \mathcal{V} detects that \mathcal{R} , for example, accepts an invalid blockchain state for contract execution, it complains to the CrossChainGame contract by sending a challenge transaction and waits for a response. If the challenge succeeds, the states in cross-chain contract will rollback and the challenger gets a reward from the contract. If \mathcal{R} does not receive a response in due time, \mathcal{V} can upload the evidence to $\tilde{\mathcal{B}}$ and terminate this contract execution by complaining to the participants.
- **Finalization.** In this phase, the protocol returns a final output and assigns the deposits according to the predefined policy.

C. The Initialization Phase

We now specify the formal details of the IvyCross protocol. In this phase (cf. Figure 2), 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 $\tilde{\mathcal{B}}$ 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 $\tilde{\mathcal{B}}$.

Afterwards, two hosts \mathcal{R} and \mathcal{V} individually generate several addresses with the security parameter λ for making deposits in $\tilde{\mathcal{B}}$ and accepting rewards from different blockchains. Their deposits are locked in the *time-lock* contract CrossChainGame that can only be redeemed after L_0 and L_1 blocks, respectively (Steps 1-2). 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. 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 $\mathcal{P} = \{p_1, \dots, p_n\}$ can execute a cross-chain smart contract. Each participant generates a key pair (pk_{p_i}, sk_{p_i}) and registers the public key

Fig. 3. The execution phase of Prot_{CCC} .

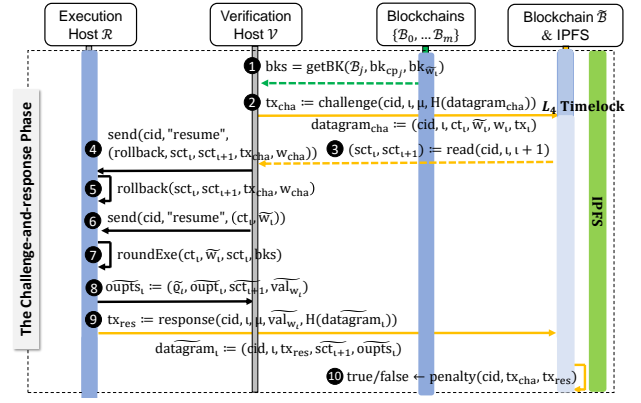
in IvyCross for identity verification. A certain participant p_i ($i \in [n]$) can deploy a smart contract $\text{Contract}_{\text{CCC}}$ with an initial state st_0 , the public keys of participants to T_0 and T_1 by invoking the “install” instruction (Steps ③–④). Specially, st_0 contains several blocks $\{bk_{cp_1}, \dots, bk_{cp_{m_0}}\}$ (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.

Two enclaves T_0 and T_1 run $\mathcal{K.A.KGen}(1^\lambda)$ to generate a fresh key pair (pk_{cid}, 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 ⑤). To prevent a participant from loading a same deposit transaction into different smart contracts, we require the enclaves to generate the deposit addresses for each smart contract, and all involved public keys are published in IPFS. Participants need to make deposits in their corresponding blockchains before entering the next phase (Step ⑥). These deposits are locked in the corresponding addresses $(pk_{btc}, pk_{eth}, \dots)$ of $\text{Contract}_{\text{CCC}}$, 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 $\text{Contract}_{\text{CCC}}$, two hosts respond with a contract identifier cid and initial parameters (Step ⑦). \mathcal{R} sends $(cid, \{pk_{p_i}\}_{i \in [n]}, pk_{cid}, datagram_0)$ to \mathcal{B} and waits for the confirmation (Step ⑧), where $sct_0 = \mathcal{AE.Enc}(pk_{cid}, st_0)$ in $datagram_0$ is an initial encrypted state of the contract. Meanwhile, a response $\{\text{Contract}_{\text{CCC}}, cid, pk_{cid}, \sigma_{mpk_{T_0}}\}$ is sent to the participants, where $\sigma_{mpk_{T_0}}$ is a signature of T_0 .

D. The Execution Phase

In this phase, the contract $\text{Contract}_{\text{CCC}}$ is triggered with a state transition upon receiving an input from a participant (cf. Figure 3). Concretely, in round l ($l \in [\ell]$), a participant p_i gets the public key pk_{cid} and encrypts an input $ct_l = \mathcal{AE.Enc}(pk_{cid}, inps_l)$. Then, p_i sends an input (cid, ct_l, w_l) to T_0 (and T_1) by invoking the “resume” instruction ($resume, cid, (ct_l, w_l)$) (Steps ①–②). \mathcal{R} reads a previous state of the contract sct_l from \mathcal{B} . The input is sent with a signature of p_i so that the enclave T_0 can verify its validation. w_l is a blockchain evidence that both \mathcal{R} and \mathcal{V} can check whether it is a correct on-chain state. Here, we use $roundExe(ct_l, w_l, sct_l, bks)$ to represent the execution of contract function (Steps ③–④), in which the fourth parameter

Fig. 4. The challenge-and-response phase of Prot_{CCC} .

denotes a succession of blocks related with w_l . If bks is not empty, $\text{Contract}_{\text{CCC}}$ will use it to check the correctness of w_l . Otherwise, i.e., $bks = \emptyset$, it only uses w_l 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 $\text{Contract}_{\text{CCC}}$ outputs $oupts_l := (q, sct_{l+1}, oupt_l, val_w)$ (Step ⑤), where $q := (h_{inps}, h_{st}, h_{stsf}, h_{oupt}, \sigma_{mpk_{T_0}})$ refers to a set of hash values on the input $h_{inps} = H(ct_l, w_l)$, a previous state $h_{st} = H(st_l)$, a state transition $h_{stsf} = H(st_l || st_{l+1})$, an output $h_{oupt} = H(oupt_l)$. $sct_{l+1} = \mathcal{AE.Enc}(pk_{cid}, st_{l+1})$ 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, $oupt_l$ can be an encrypted output of the function using pk_{p_i} . $val_w \in (0, 1)$ is a validation result of w . T_0 sends the output to T_1 with an attestation $attes_l := (oupts_l, \sigma_{mpk_{T_0}})$ (Step ⑥). Meanwhile, a digest of the output $(cid, l, H(oupts_l), H(datagram_l), val_w)$ is sent to \mathcal{B} (Step ⑦). To enforce the atomic delivery of $attes_l$ and $(cid, l, H(oupts_l), H(datagram_l), val_w)$ between \mathcal{V} and \mathcal{B} , we can adopt the protocol *proof of publication* as in [13].

In particular, we design a *data-lock* in the contract CrossChainGame for an on-chain state sct_l 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 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 \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 $\text{Contract}_{\text{CCC}}$ after L_3 blocks, and goes to the finalization phase after ℓ rounds.

E. 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 4).

Note that the enclave T_0 has attested the outputs in $\tilde{\mathcal{B}}$ 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_i 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 $\tilde{\mathcal{B}}$ and requires a response transaction tx_{res} from \mathcal{R} . \mathcal{V} first reads a succession of blocks bks from \mathcal{B}_j , where $bks = \text{getBK}(\mathcal{B}_j, bk_{cp_j}, bk_{\tilde{w}_i})$ refers to the blocks from the checkpoint bk_{cp_j} to a latest block $bk_{\tilde{w}_i}$ which contains a correct \tilde{w}_i (Step ❶). If w_i 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_i , a hash value $H(bks)$, and a correct blockchain evidence \tilde{w}_i 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_i and sct_{i+1} from $\tilde{\mathcal{B}}$ (Step ❸) and makes a *resume* call with an input ($rollback, sct_i, sct_{i+1}, tx_{cha}, w_{cha}$) to trigger a rollback operation from sct_{i+1} back to sct_i 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_i, sct_{i+1}) to trigger the state rollback from sct_{i+1} back to sct_i (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 ($cid, ct_i, \tilde{w}_i, st_i, bks$) (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, \mathcal{R} generates the response transaction $tx_{res} := (cid, \iota, \mu, \text{val}_{\tilde{w}_i}, H(\text{datagram}_i))$ (Step ❾). Then, the *Judger* contract **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 $oupts_i$.

If \mathcal{V} succeeds to use a correct \tilde{w}_i 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_i which has been consumed by another transaction (i.e., a consumed UTXO) in Bitcoin, \mathcal{V} can provide the evidences \tilde{w}_i that contain the transaction consumed w_i (i.e., a created UTXO), the difficulty level, and a succession of blocks which start from the block containing w_i to the block containing \tilde{w}_i . 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 **CrossChainGame**, and notice all participants to abort the execution of this contract. Inspired by [17], [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.

F. 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 **Contract_{ccc}**. Specifically, it generates a set of transactions $\{\tilde{tx}_{d_1}, \dots, \tilde{tx}_{d_n}\}$ which send the deposits back to the corresponding addresses using the secret keys ($sk_{btc}, sk_{eth}, \dots$). The input of a transaction \tilde{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 $\tilde{\mathcal{B}}$.

G. Incentive Mechanism using Sequential Game Theory

To prevent collusion attacks between \mathcal{R} , \mathcal{V} and certain participants, we leverage economic means to incentivize them to behave honestly. As mentioned earlier, these parties are assumed as risk-averse, namely, they will not take part in a cross-chain contract if their profits are negative. Here, 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 determine theirs [28]. The optimal strategy of one player depends on the behaviors of others.

During the analysis of the sequential game, 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 $\alpha < 1$) which is 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.⁴ Without loss of generality, the reward payment for one round contract execution is larger than the costs of two

⁴It can be extended to take behavior-dependent (e.g., honest or dishonest) into consideration.

hosts, and the cost of contract execution is more expensive than the cost of verification, i.e., $c_{\mathcal{R}}(\ell) > c_{\mathcal{V}}(\ell)$. Transaction fee is considered as a function $\mathbf{f}(\ell) = \{f(1), \dots, f(\ell)\}$, where $f(\ell)$ is determined by the number of transactions and the size of a transaction in round ℓ . Besides, if a party behaves dishonestly, this behavior can be detected and punished by a penalty function P which transfers the deposit of the dishonest party (i.e., a host or a participant) to the address of the others.

Then, by following the methodology of game theory, 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:

$$U_{\theta}(S) = \sum_{\ell=1}^{\ell} (\mathbf{r}_{s_{\theta}}(\ell) - \mathbf{f}(\ell) - \mathbf{c}(\ell)) + P_{s_{\theta}}, \quad (4)$$

where s_{θ} refers to the strategy of a host $\theta \in \{\mathcal{R}, \mathcal{V}\}$, and $P_{s_{\theta}}$ 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 for 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 IvyCross. 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 5. 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 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 $\tilde{\theta} \in \{p_i, \mathcal{R}, \mathcal{V}\}$ can improve his profit by altering the strategy $s_{\tilde{\theta}}^*$ with the other party's strategy $\mathbf{S}_{s_{\tilde{\theta}}^*}$:

$$\begin{aligned} u_{p_i}(s_{p_i}^*, s_{\mathcal{R}}^*, s_{\mathcal{V}}^*) &\geq u_{p_i}(s_{p_i}, s_{\mathcal{R}}^*, s_{\mathcal{V}}^*), \text{ for each } s_{p_i} \in S^*, \text{ and} \\ u_{\mathcal{R}}(s_{p_i}^*, s_{\mathcal{R}}^*, s_{\mathcal{V}}^*) &\geq u_{\mathcal{R}}(s_{p_i}^*, s_{\mathcal{R}}, s_{\mathcal{V}}^*), \text{ for each } s_{\mathcal{R}} \in S^*, \text{ and} \\ u_{\mathcal{V}}(s_{p_i}^*, s_{\mathcal{R}}^*, s_{\mathcal{V}}^*) &\geq u_{\mathcal{V}}(s_{p_i}^*, s_{\mathcal{R}}^*, s_{\mathcal{V}}), \text{ for each } s_{\mathcal{V}} \in S^*. \end{aligned} \quad (5)$$

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

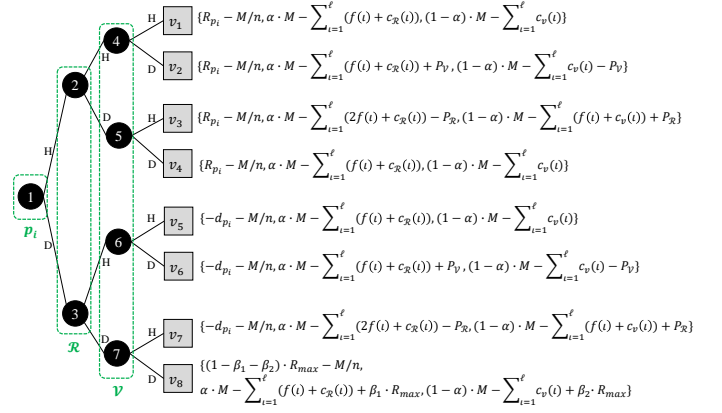


Fig. 5. The sequential game tree for \mathcal{R} , \mathcal{V} and p_i . Each vertex denotes a choice for a party. The arrow of the vertex denotes a possible strategy $\{H, D\}$ for that party. The profits of three parties are specified at the terminal nodes, e.g., $v_1 = \{u_{p_i}, u_{\mathcal{R}}, u_{\mathcal{V}}\}$.

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 5). 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_i 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.

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 a first mover advantage. However, as long as the amount of \mathcal{V} 's deposit is large 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 IvyCross 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. 5. 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 $\mathbf{S}^* = (s_{p_i}^*, s_{\mathcal{R}}^*, s_{\mathcal{V}}^*) = (H, H, H)$ that satisfies NE, where three parties choose the strategy of H if they are rational.

VII. 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 [29]. Then, we analyze the security of Prot_{ccc} under our 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, $\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}}$ be an ideal contract functionality, and Prot_{ccc} be a protocol in the $(\mathcal{G}_{att}, \mathcal{H}, \mathcal{G}_{ccg}^{\tilde{\mathcal{B}}}, \mathcal{L}_{\mathcal{B}})$ -hybrid world. Then, Prot_{ccc} is said to UC-realize \mathcal{F}_{ccc} in the $(\mathcal{G}_{att}, \mathcal{H}, \mathcal{G}_{ccg}^{\tilde{\mathcal{B}}}, \mathcal{L}_{\mathcal{B}})$ -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:*

$$\forall \mathcal{E}, \text{REAL}_{\text{Prot}_{ccc}, \mathcal{A}, \mathcal{E}}^{\mathcal{G}_{ccg}^{\tilde{\mathcal{B}}}, \mathcal{G}_{att}, \mathcal{H}, \mathcal{L}_{\mathcal{B}}}(\lambda, inps) \approx \text{IDEAL}_{\mathcal{S}, \mathcal{E}}^{\mathcal{F}_{ccc}, \mathcal{G}_{att}, \mathcal{H}, \mathcal{L}_{\mathcal{B}}}(\lambda, inps) \quad (6)$$

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

Proof. In the appendix of the online version [29], we formally prove Theorem 2 in the hybrid world, where we elaborate the ideal functionalities and formal statements in the hybrid world. \square

Privacy preservation of cross-chain smart contracts. It is straightforward that privacy preservation 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 the distributed IPFS which could be recognized as a secure certificate authority (CA), and intermediary states are stored with persistence and immutability in $\tilde{\mathcal{B}}$, and therefore the man-in-the-middle attacks can be prevented. Besides, 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 IvyCross 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 IvyCross, 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 the IPFS and $\tilde{\mathcal{B}}$, 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 decentralized 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 hosts⁵. 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], IvyCross 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 IvyCross with the state-of-the-art works [13], [17], [12], [7], [30]. 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 [17], and Tesseract [12] using TEE can achieve privacy-preserving smart contracts. However, these approaches can

⁵We leave the defense against external adversaries and the choice of reliable hosts as future works.

TABLE II

State-of-the-art works in cross-chain communication. $\phi(\cdot)$ refers to the *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, $\omega = 6$ in Bitcoin and $C = 8064$ in Tesseract [12].

Approach	Attacks Defense			Privacy preservation	Minimal data for $\phi(\cdot)$ #Block	Smart contracts for blockchain Interoperability
	Eclipse attack	Rollback attack	Front running attack			
Ekiden [13]	○	●	○	●	$\mathcal{O}(m\omega)$	○
FastKitten [17]	○	○	○	●	$\mathcal{O}(m\omega)$	○
Tesseract [12]	●	○	●	●	$\mathcal{O}(m\omega)$	○
HyperService [7]	●	--	○	○	--	●
A2L [30]	●	--	○	●	--	○
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.

not be directly utilized for supporting smart contracts in the cross-chain scenario due to their specific design and additional limitations. More concretely, they all require each TEE to store a number of 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 IvyCross. In addition, most of these approaches [11], [12], [13] resort to a number of TEEs for achieving interoperability and correctness, which results in high communication cost. HyperService [7] and A2L [30] are cryptographic primitives-based approaches to achieve blockchain interoperability, while they are not efficient compared with IvyCross. 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. Besides, we have to point out that IvyCross does not have highly robust compared with [13], [12], the limitation here is that two hosts might become the target of the adversary, causing the single point of failure. Thus, it is necessary to design a secure mechanism (as mentioned earlier) to protect two hosts.

VIII. IMPLEMENTATION AND EVALUATION

The prototype of IvyCross is implemented in about 5.4k lines of Solidity, Go, and Java. The source code will be released upon paper acceptance.

A. Experimental Setup

In the implementation of IvyCross, to avoid the complex development of Intel SGX based on SGX SDK, we leverage a light-weight library OS, named Occlum [27], 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 IvyCross, 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 FISCO BOCS (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.

B. Applications

We highlight three concrete use cases to show the practicality and efficiency of IvyCross.

Lottery. A cross-chain lottery contract is implemented atop three blockchains where each participant makes a deposit in the corresponding blockchain. They join in the lottery contract by uploading a deposit transaction and a number. The lottery contract determines one winner based on a collaboratively generated randomness. The winner will obtain all of the deposits. More precisely, the enclave T_0 generates a set of transactions that transfer the deposits from $(pk_{btc}, pk_{eth}, pk_{fb})$ to the winner's address. 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 1000 times with 99 clients for the above three

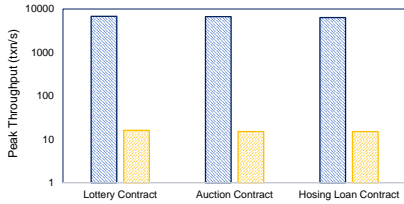


Fig. 6. The peak throughput across three cross-chain contracts. The peak throughput of these three contracts are similar because the differences between the contract codes is unobvious.

use cases, where each case has identical clients on Bitcoin, Ethereum, FB testnets, respectively.

Scalability. Specifically, we recognize that the overhead caused by remote attestation and state persistency on \tilde{B} affects the scalability of IvyCross significantly. Inspired by Ekiden [13], we use the off-chain transactions batching to improve the scalability of IvyCross. More concretely, during the contract execution, we compress multiple off-chain transactions into a single on-chain transaction, instead of sending each state checkpoint to \tilde{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.

In our experiments, 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 6, the peak throughput of three contracts can up to 6783txns/sec, 6667txns/sec, 6536txns/sec, respectively. Compared with Ethereum, IvyCross can achieve 2x orders of magnitude higher performance.

End-to-end Execution Latency. In each experiment, we first evaluate the end-to-end latency of a normal situation that all 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 \tilde{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 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 Cross-ChainGame 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 ~ 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 1 Gwei = 1×10^{-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 ~ \$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 27361 Gwei to 27896 Gwei, i.e., 0.051 ~ 0.052 USD in transaction fees.

Existing well-known cross-chain platforms, such as Hyper-Service [7] and Polkadot [8], inherently publish the data on chain, and thus can not accomplish the above use cases in a privacy-preserving way. In contrast, IvyCross 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 (with the help of IPFS).

IX. CONCLUSION

In this paper, we presented IvyCross, a privacy-preserving blockchain interoperability framework that supports smart contracts across multiple blockchains. IvyCross 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

TABLE III

The performance of IvyCross execution in different phases. All time are in seconds. We conducted 100 repeated runs, and 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 IvyCross. It calculates the fees (USD) using the data from BINANCE retrieved on June. 18, 2021

CCC Phases	Lottery						Fees (USD)
	Off-chain		# txs	On-chain			
	# txs	Size (bytes)		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

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 IvyCross 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] M. Herlihy, "Atomic cross-chain swaps," in *Proceedings of the 2018 ACM symposium on principles of distributed computing*, 2018, pp. 245–254.
- [7] 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.
- [8] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," *White Paper*, 2016.
- [9] C. Baum, B. David, and T. K. Frederiksen, "P2dex: Privacy-preserving decentralized cryptocurrency exchange."
- [10] "Dtherdelta-decentralized token exchange," "https://etherdelta.com/#PPT-ETH", [Online].
- [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] E. B. Saxon, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 459–474.
- [15] S.-F. Sun, M. H. Au, J. K. Liu, and T. H. Yuen, "Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero," in *European Symposium on Research in Computer Security*. Springer, 2017, pp. 456–474.
- [16] 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.
- [17] 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.
- [18] S. Matetic, M. Ahmed, K. Kostianen, A. Dhar, D. Sommer, A. Gervais, S. Faust, and S. Capkun, "{ROTE}: Rollback protection for trusted execution," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 1289–1306.
- [19] G. Kaptchuk, M. Green, and I. Miers, "Giving state to the stateless: Augmenting trustworthy computation with ledgers," in *NDSS*, 2019.
- [20] J. Kwon and E. Buchman, "Cosmos whitepaper," 2019.
- [21] 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.
- [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] J. Benet, "Ipfes-content addressed, versioned, p2p file system (draft 3)," *arXiv preprint arXiv:1407.3561*, 2014.
- [25] 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.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] 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>.
- [30] E. Tairi, P. Moreno-Sanchez, and M. Maffei, "A2I: Anonymous atomic locks for scalability in payment channel hubs," *Cryptology ePrint Archive*, Report 2019/589, Tech. Rep., 2019.
- [31] 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.
- [32] 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 IvyCross protocol Prot_{ccc} , we leverage the *Universal Composability* (UC) framework which can simplify the process of protocol analysis[31]. 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 IvyCross and the security analysis in the hybrid world. After that, we present the ideal functionality of $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}, \mathcal{G}_{att}}$ of IvyCross. 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 $\{\mathcal{E}, \mathcal{A}, p_1, \dots, p_n\}$, where \mathcal{E} refers to the environment, p_1, \dots, 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 $REAL_{\text{Prot}_{ccc}, \mathcal{A}, \mathcal{E}}(\lambda, x)$. 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 $IDEAL_{\mathcal{F}_{ccc}, \mathcal{S}, \mathcal{E}}(\lambda, x)$, 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}_{π} and \mathcal{S} , then a designed protocol π is said to UC-realize an ideal functionality \mathcal{F}_{π} .

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 [32] 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 $\{p_i\}_{i \in (1, \dots, n)}$ have a balance $\{c_i\}_{i \in (1, \dots, n)} \in \mathbb{N}$. 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 8). 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 $\mathcal{G}_{ccg}^{\mathcal{B}}$ for the CrossChainGame contract. The `CrossChainGame` contract is modelled as an ideal functionality $\mathcal{G}_{ccg}^{\mathcal{B}}$ (cf. Figure 7). $\mathcal{G}_{ccg}^{\mathcal{B}}$ maintains a global state machine $s_{ccg} := \text{INIED, ACTIVE, EXECG, FINAL}$ 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 \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 $\mathcal{L}_{\mathcal{B}}$ 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, $\mathcal{G}_{ccg}^{\mathcal{B}}$ 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 $\{\mathcal{L}_{\mathcal{B}_1}, \dots, \mathcal{L}_{\mathcal{B}_m}\}$, $\mathcal{G}_{ccg}^{\mathcal{L}_{\mathcal{B}}}$, \mathcal{G}_{att0} and \mathcal{G}_{att1} , the formal protocol of Prot_{ccc} in the hybrid world can be shown in Figure 10. The

The ideal functionality $\mathcal{G}_{ccg}^{\bar{\mathcal{B}}}$ interacts as with the global functionality of $\mathcal{L}_{\bar{\mathcal{B}}}$, 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 $\text{Storage} := \emptyset$. 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 $\mathcal{L}_{\bar{\mathcal{B}}}$. If $\mathcal{L}_{\bar{\mathcal{B}}}$ 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 $\mathcal{L}_{\bar{\mathcal{B}}}$. If yes, send $(freeze, sid, \mathcal{V}, c_{k_1})$ to $\mathcal{L}_{\bar{\mathcal{B}}}$ and $\mathcal{L}_{\bar{\mathcal{B}}}$ 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 $\text{Storage}[sid]$ not found, set $\text{Storage}[sid] = \perp$. Send $(write, sid, (cid, inps))$ to $\mathcal{L}_{\bar{\mathcal{B}}}$. If $\mathcal{L}_{\bar{\mathcal{B}}}$ 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 $\mathcal{L}_{\bar{\mathcal{B}}}$. Output $(receipt, \text{Storage}[sid][cid])$, 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 $penalty(tx_{cha}, tx_{res}) = true$, 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 $\mathcal{L}_{\bar{\mathcal{B}}}$. 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 $\mathcal{L}_{\bar{\mathcal{B}}}$. 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 $\text{Storage}[sid]$ are *FINA*. If yes, send message $(unfreeze, sid, c_{k_1}, \mathcal{V})$ to $\mathcal{L}_{\bar{\mathcal{B}}}$. Then output $(finalized, sid, \mathcal{V})$ to \mathcal{V} and terminate.

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

Fig. 7. The ideal functionality $\mathcal{G}_{ccg}^{\bar{\mathcal{B}}}$ 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

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

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 $\sigma := \text{TEE.DS}(\text{Sign}, idx, eid, pram, oupt, msk)$, and then send $(oupt, \sigma)$ to \mathcal{H} .

Fig. 8. The global functionality \mathcal{G}_{att} modeling an TEE.

main description of Prot_{cc} has been given in Section VI (cf. Figure 2, 3, and 4).

D. The ideal Functionality for cross-chain execution

As shown in Fig.9, the ideal functionality $\mathcal{F}_{cc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\bar{\mathcal{B}}}, \mathcal{G}_{att}}$ 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 $\mathcal{L}_{\mathcal{B}[m]}$. $\mathcal{F}_{cc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\bar{\mathcal{B}}}, \mathcal{G}_{att}}$

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 $\mathcal{G}_{ccg}^{\bar{\mathcal{B}}}$. 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 $s := \{\text{UNKN}, \text{INIT}, \text{DEPD}, \text{EXEC}, \text{EXED}, \text{FINA}\}$, where $s = \text{UNKN}$ refers to an initializing state that two hosts are under negotiation, $s = \text{INIT}$ refers that two hosts have made deposits that can accept cross-chain contract deployment. Then $s = \text{DEPD}$ denotes that a certain participant has deployed a contract into the secure attested processors. $s = \{\text{EXEC}, \text{EXED}, \text{FINA}\}$ refer that a cross-chain contract lies in the state of executing, executed and finalized. Note that $s = \text{FINA}$ denotes the participants have been assigned coins according to the output of execution.

The ideal functionality $\mathcal{F}_{cc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\bar{\mathcal{B}}}, \mathcal{G}_{att}}$ consists of three phases: the initialization phase, round execution phase, and the finalization phase. During the initialization phase, the functionality $\mathcal{F}_{cc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\bar{\mathcal{B}}}, \mathcal{G}_{att}}$ 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 Contract_{cc} to the two secure environments. If the deployment succeeds, the functionality $\mathcal{F}_{cc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\bar{\mathcal{B}}}, \mathcal{G}_{att}}$ can accept participants' deposit

The ideal functionality $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ 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 $\mathcal{L}_{\mathcal{B}}$, 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 $\{c_i\}_{i \in [n]} \in \mathbb{N}$. $\text{Set}(\cdot)$ 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 $s = \text{INIT}$ and initialize $\text{Set} := \emptyset$. Then go to the next round.

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

(Participant Deposit): Upon receiving input $(deposit, sid, cid, c_i)$ with $c_i \in \mathbb{N}$ from p_i ($i \in [n]$) when $s = \text{DEPD}$, 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 $s = \text{EXEC}$ and then go to the execution phase.

Execution Phase

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

(Contract Execute): Upon receiving input $(execute, sid, cid, ct_i, w_i, sct_i)$, send $(resume, sid, (cid, ct_i, w_i, sct_i))$ to \mathcal{G}_{att0} and \mathcal{G}_{att1} , and receives the responses $(sid, oupts_i, \sigma_{mpk_{T_0}})$ and $(sid, oupts'_i, \sigma_{mpk_{T_1}})$. Send $(write, sid, (cid, \iota, H(oupts_i), H(datagram_i), val_{w_i}))$ to $\mathcal{L}_{\mathcal{B}}$ and receives a response $(receipt, sid)$. If $\iota = \ell - 1$, then set $s = \text{EXED}$ and go to the finalization phase.

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

- If $penalty(tx_{cha}, tx_{res}) = true$, 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 $\mathcal{L}_{\mathcal{B}}$ 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 $\mathcal{L}_{\mathcal{B}}$ and output $(challenge, sid, cid, false)$.

Finalization Phase

(Participant Finalize): Upon receiving input $(finalize, sid, cid, p_i)$ from p_i when $s = \text{EXED}$, 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 $set = \text{FINA}$.

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

^aAssume that p_i is a user in the blockchain \mathcal{B}_j .

Fig. 9. Ideal functionality $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ for modeling cross-chain contract execution.

inputs. It goes to the execution phase if receiving n deposits.

In the execution phase, the functionality $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ accepts inputs from participants to trigger states transition of the Contract_{ccc} . The correctness of states is guaranteed by the global $\mathcal{L}_{\mathcal{B}}$ which stores the states on blockchain \mathcal{B} and maintains the state-lock mechanism in smart contracts. Lastly, the functionality $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ 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 $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ in the hybrid world, we need to construct an ideal adversary \mathcal{S} with dummy parties (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 $\mathcal{L}_{\mathcal{B}}$, a *Judger* smart contract $\mathcal{G}_{ccg}^{\mathcal{B}}$. Each cross-chain contract has a persist public space for storing all of the states.

The contract 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 $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$. Specifically, $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ 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 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 $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ and \mathcal{E} in the ideal world. If

The formal protocol Prot_{ccc} describes the behavior of a set of honest participants $\mathcal{P} = \{p_1, \dots, p_n\}$ 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 $\{\mathcal{L}_{B_j}\}_{j \in [m]}$.

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 $\mathcal{G}_{ccg}^{\mathcal{B}}$ and $(init, _)$ to \mathcal{G}_{att0} , and $\mathcal{L}_{\mathcal{B}}$ 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 $\mathcal{G}_{ccg}^{\mathcal{B}}$ and sends $(read, sid)$ to $\mathcal{V}_{\mathcal{B}}$ and receives a response $(receipt, \text{Storage}(sid))$. If it is found in $\mathcal{V}_{\mathcal{B}}$, \mathcal{V} sends $(init, sid, \mathcal{V}, c_{k_1})$ to $\mathcal{G}_{ccg}^{\mathcal{B}}$ and $(init, _)$ to \mathcal{G}_{att1} . $\mathcal{G}_{ccg}^{\mathcal{B}}$ 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 $(deploy, sid, \text{Contract}_{ccc})$ from environment \mathcal{E} , p_i checks if two hosts \mathcal{R} and \mathcal{V} have deposited in $\mathcal{G}_{ccg}^{\mathcal{B}}$. If yes, p_i sends $(install, sid, \text{Contract}_{ccc})$ 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 $\mathcal{L}_{\mathcal{B}}$, and receives a response $(receipt, cid)$.

\mathcal{P} : Upon receiving input $(deposit, sid, cid, c_i)$, p_i checks if the state of Contract_{ccc} by sending $(read, cid)$ to $\mathcal{L}_{\mathcal{B}}$, and receives a response $(receipt, \text{Storage}[cid])$. If the initial state of $\text{Storage}[cid]$ is confirmed, p_i sends $(freeze, sid, p_i, c_i)$ to \mathcal{L}_{B_j} . \mathcal{L}_{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 $\mathcal{L}_{\mathcal{B}}$ and goes to the execution phase.

Execution Phase

\mathcal{P} : Upon receiving input $(execute, sid, cid, inps_i)$ from environment \mathcal{E} , p_i computes $ct_i = \mathcal{AE}.Enc(pk_{cid}, inps_i)$, and reads w_i from \mathcal{L}_{B_j} and st_i from $\mathcal{L}_{\mathcal{B}}$, respectively. Then, p_i sends $(resume, sid, (cid, p_i, ct_i, w_i, st_i))$ both to \mathcal{R} and \mathcal{V} , and receives a response $(sid, (cid, oupts_i), \sigma_{mpk_{T_0}})$ from \mathcal{R} . p_i verifies $\sum. Verify(mpk_{T_0}, oupts_i, \sigma_{mpk_{T_0}})$, and proceeds to the next round if succeeds.

\mathcal{R} : Upon receiving input $(execute, sid, cid, ct_i, w_i, st_i)$ from p_i where $i \in [n]$, \mathcal{R} instructs \mathcal{G}_{att0} to send $(verify, sid, w_i)$ to \mathcal{L}_{B_j} and responds with (sid, v) . If v is false, aborts. Otherwise, \mathcal{R} sends $(resume, sid, (cid, ct_i, w_i, st_i))$ to the \mathcal{G}_{att0} and receives a response $(sid, oupts_i, \sigma_{mpk_{T_0}})$. After that, \mathcal{G}_{att0} sends $(write, sid, (cid, \iota, H(oupts_i), H(datagram_i), val_{w_i}))$ to $\mathcal{L}_{\mathcal{B}}$, and $\mathcal{L}_{\mathcal{B}}$ responds with $(receipt, sid)$. Meanwhile, \mathcal{R} sends $(sid, (cid, oupts_i), \sigma_{mpk_{T_0}})$ to p_i and $(cid, \iota, oupts_i)$ 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 $(execute, cid, sid, ct_i, w_i, st_i)$ from p_i where $i \in [n]$, \mathcal{V} sends $(verify, cid, w_i)$ to \mathcal{L}_{B_j} and receives a response (cid, v_0) . If v_0 is true, \mathcal{V} sends $(resume, sid, (cid, ct_i, w_i, st_i))$ to the \mathcal{G}_{att1} and responds with $(sid, oupts'_i, \sigma_{mpk_{T_1}})$. \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 $\mathcal{G}_{ccg}^{\mathcal{B}}$, and $(challenge, cid, (ct_i, \tilde{w}_i, st_i, st_{i+1}))$ to \mathcal{R} , respectively. \mathcal{V} waits for a response within L_4 blocks.

\mathcal{R} : Upon receiving input $(challenge, sid, (cid, ct_i, \tilde{w}_i, st_i, st_{i+1}))$ from \mathcal{V} , \mathcal{R} sends $(resume, sid, (rollback, cid, st_i, st_{i+1}))$ to \mathcal{G}_{att0} and receives a response $(sid, (cid, rollback, succeed), \sigma_{mpk_{T_0}})$. Then, \mathcal{R} sends $(resume, sid, (cid, ct_i, \tilde{w}_i, st_i))$ to \mathcal{G}_{att0} and receives a response with $(sid, (cid, oupts_i), \sigma_{mpk_{T_0}})$. \mathcal{R} sends $(challenge, sid, cid, tx_{cha}, tx_{res})$ to $\mathcal{G}_{ccg}^{\mathcal{B}}$, and $\mathcal{G}_{ccg}^{\mathcal{B}}$ 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 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 $\mathcal{L}_{\mathcal{B}}$ is $FINA$. If yes, then sends $(finalize, sid, \mathcal{V})$ to $\mathcal{G}_{ccg}^{\mathcal{B}}$. $\mathcal{G}_{ccg}^{\mathcal{B}}$ responds with $(accept, sid, \mathcal{V})$.

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

Fig. 10. Formal protocol Prot_{ccc} description for honest participants and two hosts in the hybrid world.

an input is sent from an honest party, then \mathcal{S} emulates as a real world “network traffic” to relay information which obtained from $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ 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 $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$. 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}^*).

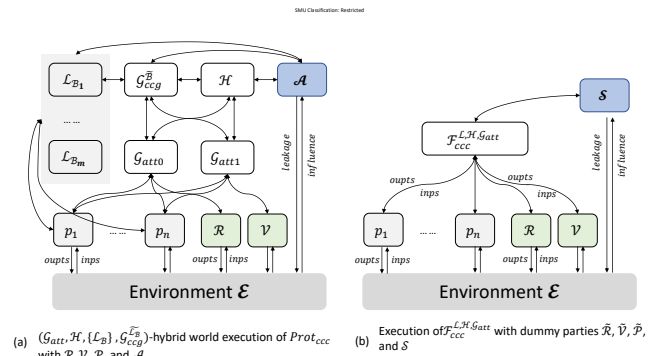


Fig. 11. Setup of a simulation with honest parties.

1) *Simulation without corrupted parties:* The setup of the simulation with honest parties is shown in Figure 11. In this case, all the involved parties, including the participants $\mathcal{P} = \{p_1, \dots, p_n\}$ 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 $\mathcal{S}^{\text{honest}}$ with no corrupted parties, that the execution of Prot_{ccc} in the $(\mathcal{G}_{\text{att}}, \mathcal{H}, \mathcal{L}_{\mathcal{B}}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}})$ -hybrid world, for any PPT adversary \mathcal{A} , is computationally indistinguishable from the execution of $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$ 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 $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$ to conduct their behaviors. The simulator \mathcal{S} can obtain $(\text{init}, \text{sid}, \mathcal{N}, c_k)$ from $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$ 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 $(\text{deploy}, \text{sid}, p_i, \text{Contract}_{\text{ccc}})$ from $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$ 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 $(\text{sid}, \text{cid}, p_i, \text{inps}_i)$ to trigger the state transition of $\text{Contract}_{\text{ccc}}$. \mathcal{S} can extract $\text{len}(\text{inps}_i)$ from $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$, and compute $ct'_i = \mathcal{A}\mathcal{E}.\text{Enc}(\widetilde{pk}_{\text{cid}}, \vec{0})$. Then \mathcal{S} sends the “read” queries to $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$ to obtain the dummy state and blockchain evidence (sct_i, w_i) . Then, on behalf of p_i , \mathcal{S} emulates a “resume” instruction $(\text{resume}, \text{sid}, (\text{cid}, ct'_i, \text{sct}_i, w_i))$ to $\mathcal{G}_{\text{att}0}$ and $\mathcal{G}_{\text{att}1}$, respectively. Upon receiving $(\text{sid}, (\text{cid}, \text{oupts}_i), \sigma_{T_0})$, \mathcal{S} from $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$, \mathcal{S} computes a ciphertext oupt'_i where $\text{oupt}'_i = \mathcal{A}\mathcal{E}.\text{Enc}(\text{mpk}_{T_0}, 0^{\text{len}(\text{oupt}_i)})$, and emulates a message $(\text{sid}, (\text{cid}, \text{oupts}'_i), \sigma_{T_0})$ from $\mathcal{G}_{\text{att}1}$ to p_i . In the finalization phase, \mathcal{S} can emulate honest parties to interact with $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$, and output $(\text{finalized}, \text{sid}, \mathcal{N})$, $(\text{frozen}, \text{sid}, p_i, c_i)$.

It can be seen that running the simulator \mathcal{S} with honest parties in the $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$ ideal world is indistinguishable from the $(\mathcal{G}_{\text{att}}, \mathcal{L}_{\mathcal{B}}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}})$ -hybrid world, unless the environment \mathcal{E} can decrypt oupt'_i to check that $\text{oupt}'_i \neq \text{oupt}_i$. However, this can only happen if \mathcal{E} can break the security of $\mathcal{G}_{\text{att}0}$ or $\mathcal{G}_{\text{att}1}$ to obtain sk_{cid} , which is impossible as depicted in our security assumption.

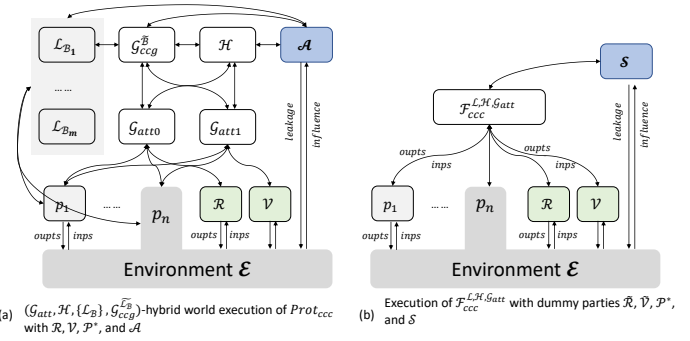


Fig. 12. Setup of a simulation with a corrupted participant p_n^* and honest \mathcal{R}, \mathcal{V} .

2) *Simulation with a corrupted participant:* The setup of the simulation with a corrupt participant p_n^* is shown in Figure 12. 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_i 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 $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$ 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 $(\mathcal{G}_{\text{att}}, \mathcal{H}, \mathcal{L}_{\mathcal{B}}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}})$ -hybrid world, for any PPT adversary \mathcal{A} , is computationally indistinguishable from the execution of $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$ 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 $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$ to accomplish the deposits as the above description. In the contract deployment, if p_n^* is responsible for deploying the contract $\text{Contract}_{\text{ccc}}$, \mathcal{S} can extract the input $\text{Contract}_{\text{ccc}}$ from \mathcal{E} and sends $(\text{deploy}, \text{sid}, p_i, \text{Contract}_{\text{ccc}})$ on behalf of p_n^* . Meanwhile, \mathcal{S} instructs $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$ to emulate the output $(\text{deployed}, \text{sid}, \text{cid})$ 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 $(\text{sid}, \text{cid}, p_n^*, \text{inps}_i^*)$. \mathcal{S} can extract $\text{len}(\text{inps}_i)$ from $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$, and compute $ct'_i = \mathcal{A}\mathcal{E}.\text{Enc}(\widetilde{pk}_{\text{cid}}, \vec{0})$. Then \mathcal{S} sends the “read” queries to $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$ to obtain the dummy state and blockchain evidence (sct_i, w_i) . Then, on behalf of p_i , \mathcal{S} emulates a “resume” instruction $(\text{resume}, \text{sid}, (\text{cid}, ct'_i, \text{sct}_i, w_i))$ to $\mathcal{G}_{\text{att}0}$ and $\mathcal{G}_{\text{att}1}$, respectively. Upon receiving $(\text{sid}, (\text{cid}, \text{oupts}_i), \sigma_{T_0})$, \mathcal{S} from $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$, \mathcal{S} computes a ciphertext oupt'_i where $\text{oupt}'_i = \mathcal{A}\mathcal{E}.\text{Enc}(\text{mpk}_{T_0}, 0^{\text{len}(\text{oupt}_i)})$, and emulates a message $(\text{sid}, (\text{cid}, \text{oupts}'_i), \sigma_{T_0})$ from $\mathcal{G}_{\text{att}1}$ to p_i . In the finalization phase, \mathcal{S} can emulate honest parties to interact with $\mathcal{F}_{\text{ccc}}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{\text{ccg}}^{\mathcal{B}}, \mathcal{G}_{\text{att}}}$, and output $(\text{finalized}, \text{sid}, \mathcal{N})$, $(\text{frozen}, \text{sid}, p_i, c_i)$.

It can be seen that running the simulator \mathcal{S} with honest parties in the $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ ideal world is indistinguishable from the $(\mathcal{G}_{att}, \mathcal{L}_{\mathcal{B}}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}})$ -hybrid world, unless the environment \mathcal{E} can decrypt $oupt'_i$ to check that $oupt'_i \neq oupt_i$. 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 the IPFS are also encrypted under the public key of pk_{cid} , and thus are also computationally indistinguishable.

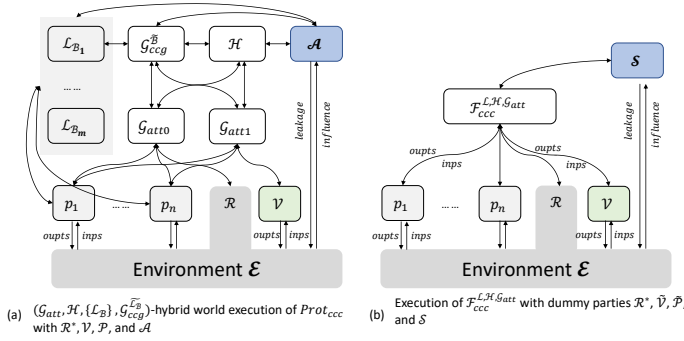


Fig. 13. Setup of a simulation with a corrupted execution host \mathcal{R} and honest \mathcal{P}, \mathcal{V} .

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 13. 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 $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ 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 $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ on behalf of \mathcal{R}^* . If \mathcal{E} instructs to terminate the emulated simulation, $\mathcal{S}^{\mathcal{R}}$ also needs to instruct $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ 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 $(\mathcal{G}_{att}, \mathcal{H}, \mathcal{L}_{\mathcal{B}}, \mathcal{G}_{ccg}^{\mathcal{B}})$ -hybrid world, for any PPT adversary \mathcal{A} , is computationally indistinguishable from the execution of $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ 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 $\mathcal{G}_{ccg}^{\mathcal{B}}$ 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 $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ to accomplish the deposits as the above description.

In the execution phase, when $\mathcal{S}^{\mathcal{R}}$ receives $(execute, sid, cid, ct_i, w_i, sct_i)$ from $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$, it means that the participants have deployed a contract $Contract_{ccc}$ 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_i is correct. If not, $\mathcal{S}^{\mathcal{R}}$ aborts the simulation. Otherwise, $\mathcal{S}^{\mathcal{R}}$ interacts with $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ and emulates \mathcal{R}^* to output $(sid, (cid, oupts_i), \sigma_{mpk_{T_0}})$.

It can be seen that running the simulator $\mathcal{S}^{\mathcal{R}}$ with a corrupted execution host \mathcal{R}^* in the $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ ideal world is indistinguishable from the $(\mathcal{G}_{att}, \mathcal{L}_{\mathcal{B}}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}})$ -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.

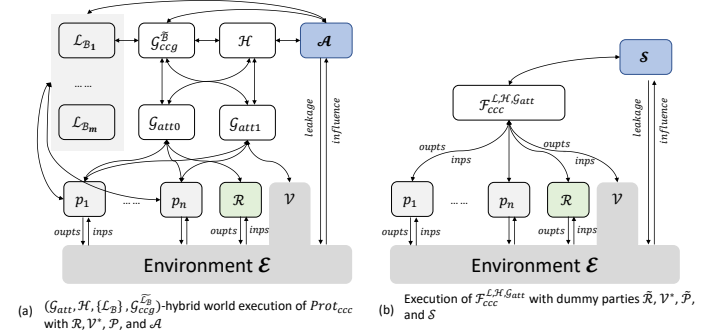


Fig. 14. Setup of a simulation with a corrupted verification host \mathcal{V} and honest \mathcal{P}, \mathcal{R} .

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 14. 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 $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ 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 $(\mathcal{G}_{att}, \mathcal{H}, \mathcal{L}_{\mathcal{B}}, \mathcal{G}_{ccg}^{\mathcal{B}})$ -hybrid world, for any PPT adversary \mathcal{A} , is computationally indistinguishable from the execution of $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ with the simulator \mathcal{S} in the ideal world.*

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

Firstly, in the initialization phase, the execution host \mathcal{R} first makes a deposit in $\mathcal{G}_{ccg}^{\mathcal{B}}$. 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 $\mathcal{G}_{ccg}^{\mathcal{B}}$ 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 $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ to accomplish the deposits and contract deployment as the above description.

In the execution phase, when $\mathcal{S}^{\mathcal{V}}$ receives $(execute, sid, cid, ct_i, w_i, sct_i)$ from $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$, it sends the input to \mathcal{G}_{att1} and receives a response with $(sid, (cid, oupts_i), \sigma_{mpk_{T_0}})$. 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 $(challenge, cid, (ct_i, \tilde{w}_i, sct_i, sct_{i+1}))$ 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 $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ to send it to $\mathcal{G}_{ccg}^{\mathcal{B}}$.

It can be seen that running the simulator $\mathcal{S}^{\mathcal{V}}$ with a corrupted execution host \mathcal{V}^* in the $\mathcal{F}_{ccc}^{\mathcal{L}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{B}}, \mathcal{G}_{att}}$ ideal world is indistinguishable from the $(\mathcal{G}_{att}, \mathcal{L}_{\mathcal{B}}, \mathcal{H}, \mathcal{G}_{ccg}^{\mathcal{L}, \mathcal{B}})$ -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}_i by following the instructions of \mathcal{E} .