PoPETs Proceedings on Privacy Enhancing Technologies ; 2022 (3):711–731

Rujia Li[♮], Qin Wang[♮], Qi Wang*, David Galindo*, and Mark Ryan*

SoK: TEE-Assisted Confidential Smart Contract

Abstract: The blockchain-based smart contract lacks privacy, since the contract state and instruction code are exposed to the public. Combining smart-contract execution with Trusted Execution Environments provides an efficient solution, called *TEE-assisted smart contracts* (TCSC), for protecting the confidentiality of contract states. However, the combination approaches are varied, and a systematic study is absent. Newly released systems may fail to draw upon the experience learned from existing protocols, such as repeating known design mistakes or applying TEE technology in insecure ways. In this paper, we first investigate and categorize existing systems into two types: the *layer-one* solution and the *layer-two* solution. Then, we establish an analysis framework to capture their common aspects, covering desired properties (for contract services), threat models, and security considerations (for underlying systems). Based on our taxonomy, we identify their ideal functionalities, and uncover fundamental flaws and challenges in each specification's design. We believe that this work would provide a guide for the development of TEE-assisted smart contracts, as well as a framework to evaluate future TCSC systems.

Keywords: Confidential Smart Contract, Blockchain, Trusted Execution Environment (TEE)

DOI 10.56553/popets-2022-0093
Received 2021-11-30; revised 2022-03-15; accepted 2022-03-16.

1 Introduction

Smart contract was originally introduced by Szabo [1] and further developed by Ethereum [2] in the blockchain systems. The blockchain-based smart contracts [3–5] adopt Turing-complete scripting languages to achieve complicated functionalities [6] and execute the predefined logic through state transition replication over consensus algorithms to realize final consistency. Smart contracts enable unfamiliar and distributed participants to fairly exchange without trusted third parties, and are further used to establish a uniform approach for developing decentralized applications (DApps [7]). However, blockchain-based smart contract lacks *confidentiality*. The state information and the instruction code are completely transparent [8–10]. Any states with their changes are publicly accessible and all users' transaction data and contract variables are visible to external observers. Without privacy, building advanced DApps that rely on the user's sensitive data becomes a challenge [11–14]. For instance, smart contracts in Ethereum [2] cannot be directly applied to Vickrey auction [15, 16] or e-voting systems [17, 18], where the bid and vote require to be hidden from the public. Moreover, DApps without privacy protection might be prohibited by European Union because they go against the General Data Protection Regulation [19, 20]. Thus, the complete transparency of smart contracts constrains their wide adoption. Recently, researchers have explored many cryptographic solutions to solve these issues, including utilizing techniques of zero-knowledge proof (ZKP) [12, 21–25], homomorphic encryption (HE) [26] and secure multiparty computation (MPC) [27]. However, these approaches are merely applicable to applications requiring simple computations.

Moving complex computations into secure hardware can provide applications with privacy as well as good performance. The use of the trusted execution environments (TEEs) [28–32] becomes thus a general-purpose solution for confidential smart contracts. The TEE is a new feature provided by recent commodity processors. It has the ability to provide secure environments for running contract code in isolation while guaranteeing execution integrity and state confidentiality. For instance, Intel$^{®}$ Software Guard Extension (Intel$^{®}$ SGX) [33–36] allows a user to create a secure area called *enclave*. Afterwards, the user unitizes the remote attestation protocol to prove to remote parties that the applications are indeed running inside an enclave. Then, the enclave establishes a secure channel to communicate with remote hosts, where the messages are encrypted. In this

Rujia Li[♮]: Equal contribution. Southern University of Science and Technology & University of Birmingham, rxl635@student.bham.ac.uk

Qin Wang[♮]: Equal contribution. CSIRO Data61, qinwangtech@gmail.com

***Corresponding Author: Qi Wang:** Southern University of Science and Technology, wangqi@sustech.edu.cn

***Corresponding Author: David Galindo:** University of Birmingham, d.galindo@cs.bham.ac.uk

***Corresponding Author: Mark Ryan:** University of Birmingham, m.d.ryan@cs.bham.ac.uk

way, SGX runs trusted codes in an enclave and uses the CPU hardware to prevent attackers from seeing or tampering with sensitive data. Such a technique provides the high-level security for inside processes to resist attacks against outside software, even the most privileged instruction from the operating system. As a promising alternative, various smart contract platforms taking advantages of TEEs have been proposed, especially by companies working on consortium blockchain platforms, such as Alibaba CONFIDE [37], Visa's LucidiTEE [38] and China's official CHANG'AN Chain [39, 40].

Although various TCSC protocols have been proposed, newly released projects may fail to draw upon the experience learned from existing protocols, such as repeating known design mistakes or applying cryptography in insecure ways. For example, an absence of economic incentives will pose security risks and decrease the protocol's stability. However, the recent-proposed TCSC scheme Hybridchain [41] repeats similar pitfalls by simply combining the TEE with a permissioned blockchain network, omitting considerations on the miner's incentive mechanism. The repeating of pitfalls comes from twofold. Firstly, in-the-wild projects differ from one to another, and a relatively unique model is absent, which narrows the developers' vision. Meanwhile, a unified evaluation framework is missing, causing many security threats to be uncovered and resulting in considerable loss from applications underpinning the execution of confidential smart contracts. This paper aims to abstract a high-level framework to simply and clearly systematize knowledge on current TCSC schemes. We attempt to capture some commonalities among these projects regarding their features, properties, and potential security vulnerability. We believe that establishing evaluation criteria to measure features and identify problems and flaws of existing TCSC protocols will offer a good guide for industry communities and promote the DApps prosperity. Main contributions (a visualized guideline in Fig.2) are:

- We provide a systematization of existing TCSC systems driven from academic work and *in production* projects. Based on their operating mechanisms and ways of combination, we investigate and categorize a set of typical protocols into two main classifications: the *layer-one* solution and the *layer-two* solution.
- We establish a unified evaluation framework for confidential smart contract systems. We consider two parts: the smart contracts used as *service*s, and underlying supported blockchain *system*s. Accordingly, the framework covers three aspects: *desirable properties* for contract services, *threat model* and *security consideration* for underlying systems. Specifically, we discuss two different types of desirable properties: *typical properties* that inherit from traditional smart contracts and featured *privacy-related properties*. Then, we emphasize practical issues, pitfalls, and remedies in designing TEE-assisted blockchains from four aspects (*host*/*TEE*/*program* securities and *key management* services).
- We conduct a comparative analysis of existing protocols based on our evaluation framework. We discuss systems both from their *common designs* (system classification, threat model) and *distinguishing features* (designs, properties). The common designs show us the consistent idea when re-designing the system, while the distinguished features highlight the ingenuity of each system design that deviates from others (see Tab.3/Tab.4).
- We further give a comprehensive discussion of current designs and implementations, including a running example, comparisons between layer-one and layer-two systems from the perspectives of *security*, *efficiency* and *easy-adoption*, and common issues on *public verifiability.* Unfortunately, a mature design is still not ready for large-scale applications. We thereby point out *research challenges* in this field, wishing to give insights for communities on defining their models and discovering possible solutions.

The rest of the paper is organized as follows. Sec.2 gives a high-level introduction on how to operate a confidential smart contract inside TEEs. Sec.3 provides the systematization methodology (*system classification* and *evaluation framework*). *Layer-one* and *layer-two* systems are analysed in Sec.4 and Sec.5. Discussions are provided in Sec.6. Research challenges are summarised in Sec.7. Finally, Sec.8 gives concluding remarks. Supplementary details are stated in Appendix A-D.

2 A Lightning Tour

This section gives a high-level description and offers a running example to illustrate how a typical confidential smart contract operates.

2.1 Overview

Invocation. In current blockchain systems, once a contract is deployed successfully, the initial state and operational code are replicated to distributed nodes. The state transition must be based on an external message call that is represented as a transaction Tx sent from a user. The TEE-assisted confidential contract, as a special type of smart contract, inherits the state-triggering mechanism. The major difference between confidential contracts and original protocols lies in whether a transaction has to carry the ciphertext c_u encrypted by the TEE's public key.

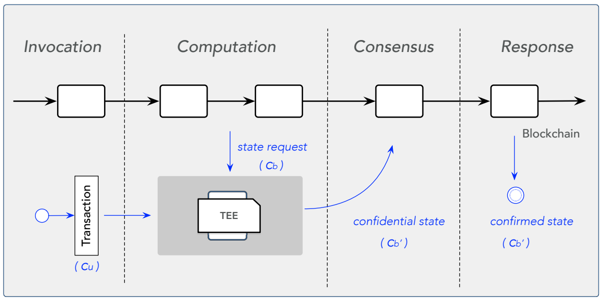

Fig. 1. TEE-assisted confidential smart contract workflow.

Computation. Once receiving an invocation request (Tx with an encrypted argument of c_u) from the user, a TEE decrypts the ciphertext c_u and loads the contract source code and current encrypted contract state c_b fetched from the blockchain. Then, the TEE decrypts the state c_b using a TEE service key, executes the contract logic, and outputs an execution result m_b. Afterwards, the TEE encrypts m_b with a specific user's public key and obtains the ciphertext c_b'. Next, the TEE sends c_b' to the blockchain network.

Consensus. After obtaining the encrypted state c_b' carried by Tx, the consensus algorithm starts to reach an agreement over distributed nodes. In particular, when a blockchain node receives a newly mined block, it will re-execute all transactions inside the block. When a majority of consensus nodes receive the same block and re-execute included transactions, the state c_b' with its carrier Tx is deemed to be confirmed and becomes immutable.

Response. The blockchain returns the final state c_b' and corresponding transaction Tx to the user, and this user decrypts the ciphertext c_b' with obtaining the final state. Here we emphasize that, even if the state c_b' is publicly accessible, only the user who owns the private key can obtain the final plaintext.

2.2 Running Example

From a bird's eye view, a TCSC can be used as an ideal contract-based *black box* [42] with secrecy and correctness. This idea has been adopted by several advanced security protocols [43, 44]. We provide a secret e-voting example borrowed from Oasislabs [45].

In this example, the number of voter's choices is not allowed to be revealed until the voting is finished. Meanwhile, the voter does not want other participants to know her choice. A high-level overview is: a voter calls the contract inside TEEs by sending a transaction with an encrypted argument c_u. Next, the TEE decrypts the argument c_u and decrypts the current encrypted state c_b using the service key (see Tab.5 and Fig. 5 in Appendix A). Afterward, TEE confidentially executes the voting logic and correspondingly returns m_b'. Then, the TEE encrypts m_b' as c_b', and sends c_b' to the blockchain. Eventually, the voter fetches the final result c_b' from blockchain and decrypts it with her private key to obtain the voting result *voteresult* (Tab.1).

Table 1. Data workflow of an e-voting protocol based on TEE-assisted confidential smart contracts.

Stage	Voter	TEE	Blockchain
Invocation	$votedata \rightarrow c_u$; $c_u \rightarrow$ Tx		c_b;
Computation		$c_u \rightarrow data$; $c_b \rightarrow m_b$; $data, m_b \rightarrow m_b'$; $m_b' \rightarrow c_b'$;	
Consensus			Tx $\rightarrow$ B; $c_b' \rightarrow$ B;
Response	$c_b' \rightarrow voteresult$		

A TCSC can be well qualified for the role of *decentralized vote manager* in an e-voting system [17, 46]. Once a contract-based manager is deployed successfully, the voting logic is loaded into a TEE and corresponding secret keys are privately generated and stored inside TEEs. The encrypted state is then confirmed by the blockchain nodes. This offers the e-voting protocol with *confidentiality*, *neutrality*, *auditability* and *accountability*. Firstly, the voter's input c_u is encrypted, and intermediate parameters (e.g., m_b) are privately processed through TEEs. External attackers cannot obtain the knowledge of sensitive information, and thus the confidentiality is achieved. Secondly, the predefined voting logic only occurs in the decentralized network when certain conditions are satisfied, bringing neutrality for the access control management. Thirdly, if a voter wants to vote for a candidate, she needs to in advance build a channel to the TEE and then send a transaction Tx to call the contract. Due to the protection of encrypted channels, transaction arguments are kept secret. Meanwhile, such invoking records in the form of transactions remain visible and will become immutable, ensuring the voting process accountable. Unfortunately, *verifiability*, as one of fundamental properties, performs not smooth in the context of encryption. Contracts that are executed inside TEEs make the execution procedures lack public verifiability. Only the nodes who install TEEs with correct corresponding keys can verify the correctness of contract executions. However, the metadata of the transaction Tx retains unencrypted, making it possible to verify the absence of double spending.

3 Systematization Methodology

To find common aspects (e.g., offered functionality, design model, adversary model), we extract recurring design patterns from publicly available publications and projects, focusing on systematization and evaluation of desirable properties (the main target of TCSC) and potential pitfalls of underlying systems. Our systematization methodology follows the idea in [47]: *classification* and *evaluation*. We first make a classification for current systems and then define a framework to evaluate them.

3.1 System Classification

We classify the existing systems into two main categories: *layer-one solution* (L_1) and *layer-two solution* (L_2). The layer-one solution executes the contract inside a TEE in the blockchain, requiring every blockchain node to equip a TEE. Instead, the layer-two solution decouples contract computations from the blockchain. It performs most of the smart contract computations off-chain. For a clear understanding, we make a comparison of the original blockchain (e.g., Ethereum), L_1 solution, L_2 solution. As in Tab.2, Ethereum runs smart contracts (in EVM) and consensus procedures in the same machine of distributed nodes. All the contract and transaction operations are publicly verifiable due to their total transparency. The layer-one solution performs such operations (contract execution and consensus) in the same machine, but contract operations are separate from consensus procedures. In contrast, the layer-two solution makes both of them operate independently. Contracts are executed outside the blockchain network, while the consensus happens inside each node.

Table 2. A comparison of Ethereum, L_1 and L_2 solution

	Ethereum	L_1 Solution	L_2 Solution
EVM and consensus in same machine	✓	✓	✗
EVM and consensus in same TEE	-	✗	✗
Contract execution publicly verifiable	✓	✗	✗
Contract execution peer verifiable	✓	✓	✗
Consensus procedure publicly verifiable	✓	✓	✓

3.2 Desirable Property

Ideally, moving smart contract executions into TEEs brings additional privacy and maintains original benefits of blockchain. Therefore, we identify desirable properties in two main categories: *privacy-preserving property* and *blockchain intrinsic feature*.

Privacy-Preserving Property. The property of confidentiality is the most distinguished feature in TCSC.

A1. Specification hidden. The source code of a smart contract is hidden during the deployment and the subsequent synchronization and execution.

A2. Input privacy. The inputs fed into a confidential smart contract are hidden from the public.

A3. Output privacy. The outputs returned from a confidential smart contract should be kept private.

A4. Procedure privacy. The execution procedure is hidden from unauthorized parties. An adversary cannot learn the operation knowledge inside a TEE.

A5. Address unlinkability. The address pseudonymity does not entail strong privacy guarantees [48, 49]. This property prevents an adversary to learn the address linkability by observing users' activities.

A6. Address anonymity. The contract caller's identity (a user who invokes a smart contract) is hidden from an anonymity set [24] (see Appendix B).

Blockchain Intrinsic Feature. TEE-assisted smart contracts inherit features given by original blockchain systems. We summarize these features as follows.

A7. Code immutability. Once a contract is successfully deployed, its source code cannot be altered.

A8. (Confidential) state consistency. Executions happening at a certain blockchain height will output the same result across different nodes.

A9. Contract interoperability. A smart contract can call another contract and be called by others.

A10. High availability. A smart contract is continuously accessible without the single point of failure.

A11. Decentralized execution. A smart contract runs over the decentralized network.

A12. Automatic execution. A smart contract can be automatically executed once conditions are satisfied.

A13. Gas mechanism. Operations running on the smart contract will be charged with gas fees [2].

A14. Explicit invocation. Each invocation will be formatted as a transaction and stored on blockchain.

A15. Public verifiability. The procedure of contract execution and result are publicly verifiable.

A16. Consensus verifiability. The consensus procedure on the (confidential) state is publicly verifiable.

3.3 System Evaluation

Essentially, all TCSC systems share the same principle: *a TEE will handle the data from users. After that, encrypted data flows from the TEE to blockchain.* The TEE plays a crucial role. Thus, this part defines a framework for evaluating underlying blockchain systems from four aspects: *TEE host*, *TEE security*, *TEE program*, and *TEE key management*. This framework aims to identify potential design flaws and pitfalls based on the threat model and data workflow.

Threat Model. Our threat model mainly captures three types of attackers, as follows.

T1. User adversary (active/passive). An attacker may control network between users and TEE host nodes.

T2. TEE adversary (active/passive). An adversary may control TEE hosts or control the network between TEE and blockchain platforms.

T3. Blockchain adversary (active/passive). An adversary may drop, modify and delay the blockchain messages. But the majority (or two-thirds) of the blockchain nodes are assumed to be honest.

Note that adversaries are not necessarily exclusive. In some cases, adversaries in different types may collude.

Security Considerations. This section defines four metrics regarding system security according to the data workflow: approaches to enhance the security of a TEE host, countermeasures to mitigate intrinsic TEE issues, measures to prevent program flaws or bugs inside TEEs, and solutions to clear up TEE key security dilemma.

TEE host security. A TEE and its interaction with the external environment (e.g., with users or the blockchain) are operated and controlled by a host (e.g., L_1 blockchain node). A malicious host may abort the executions of a TEE, delay and change inputs, or drop any ingoing or outgoing messages. The following metrics discuss approaches to improve the TEE host's security.

P1. Host punishment mechanism. Penalty mechanisms to reduce the risk of doing evil by a TEE host.

P2. Host incentive mechanism. Incentive mechanisms to promote a TEE host to behave honestly.

P3. Host fault tolerance. Solutions to make systems continually operate despite malfunctions or failures.

P4. Host authentication. Methods to check the identity and the capability of a TEE host.

TEE security. A TEE has some inevitable weaknesses. For example, a TEE is vulnerable to side-channel attacks [50, 51]. These innate weaknesses directly pose severe challenges to the design and implementation of TEE-assisted contract systems. This part defines the defence approaches against these threats.

P5. TEE attestation security. Methods to prevent TEE attestation service from being abnormally broken.

P6. TEE memory limitation. Methods to optimize the memory size to prevent confidential data overflow.

P7. TEE physical attacks. Approaches to prevent physical attacks, such as the Spectre vulnerability or the Meltdown vulnerability [52].

P8. TEE trusted timer. Approaches to provide a trusted timer when running a TEE.

TEE program security. Even if a TEE is secure as assumed, a program bug may destroy the contract's confidentiality in the real world. This part focuses on the measurements to prevent TEE programs from flaws.

P9. Workload measurement. The workload measurement approach to prevent an infinite loop attack.

P10. Flaws detection. Formal techniques used for the modelling and verification of the source code of smart contracts to reduce the vulnerabilities.

P11. User query restriction. The restriction on users' queries, aiming to avoid data leakage resulting from differential-privacy analysis [53].

P12. Blockchain data confirmation. Methods for a TEE to check whether input data from blockchain has been confirmed to prevent the rollback attack [54].

P13. TEE output conflicts. Methods to avoid multiple TEEs to produce a conflict result.

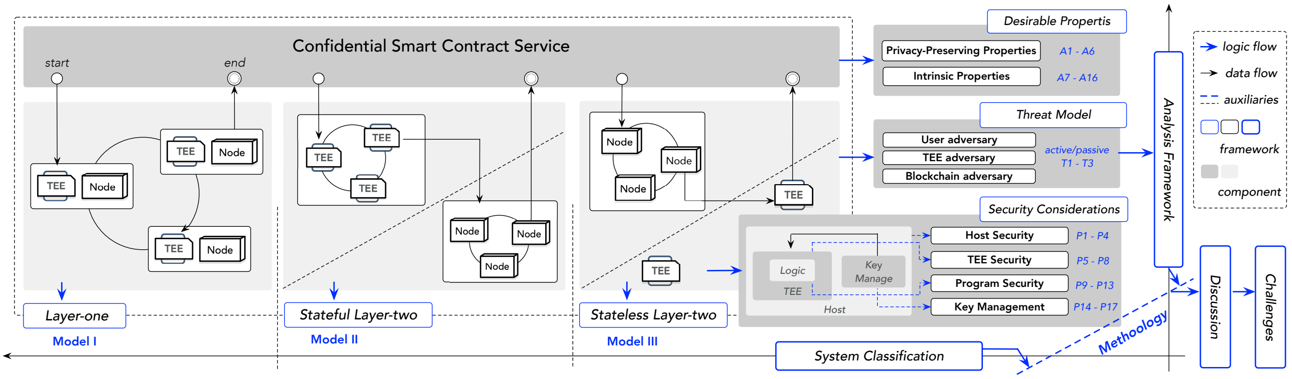

Fig. 2. Systematization methodology: We delineate current confidential smart contract systems along two principal axes. In the *horizontal* axes, we identify two types of TEE-assisted systems according to the ways of combination. In the *vertical* axes, we give our analysis framework in three aspects. The *property* corresponds to the confidential smart contract service, providing the functionalities to the end-users. The *threat model* and *security consideration* focus on their underlying systems that support upper-layer services. With these two axes as our research methodology, we further present related discussions and open challenges.

TEE key security. Various keys (cf. Appendix A) are involved in the contract execution, including TEE internal keys such as the attestation key and TEE service keys for state encryption/decryption. Since service keys directly affect the protection of contract states, the key security evaluation in this SoK focuses on the generation, exchange, and storage of the TEE service key.

P14. Distributed key protocol. The keys of confidential contracts are managed by a distributed protocol.

P15. Key rotation protocol. The TEE replaces an old key with a fresh key for future contract encryption.

P16. Independent contract key. Each contract is associated with a unique key, independent from the TEE.

P17. Independent TEE key. Each TEE has a unique key, and different contracts share the same key.

The *system classification* shows a general view of the TCSC systems. The *desirable property* focuses on evaluating contract service provided by a TEE-assisted blockchain system. The *threat model* describes the potential threats and system assumptions. The *security considerations* show the evaluating indicator for current TEE-assisted systems. In the following section 4.2 and 5.2, we attempt to answer the following questions: (i) What are the potential pitfalls in each security aspect; (ii) Do these pitfalls have significant security impacts; (iii) Do the designers/developers consider these pitfalls and accordingly come up with feasible remedies in their systems; (iv) What are the remedies and do they address above problems. Note that hundreds of TCSC systems have been proposed in both industry and academia. An exhaustive analysis is undesirable and infeasible. We only selected the projects that provide publicly accessible technical reports or academic papers.

4 Layer-One Solution

The layer-one approach enables blockchain nodes to run contracts in their isolated areas, as well as conducting the consensus (see Fig. 3). This approach combines the consensus procedure and state execution, either in terms of logically or physically. The reason why we call this method *layer-one* is that all executions are completed in the same layer of the blockchain network. The key to such an approach is to equip every blockchain node with a TEE. Indeed, this requires more integration efforts, but also comes with several advantages. The smart contract can implement stateful functionalities that receive arguments and update states instantly. In particular, a smart contract can directly access the ledger data stored in a local disk, greatly saving time often wasted in interactive network communications.

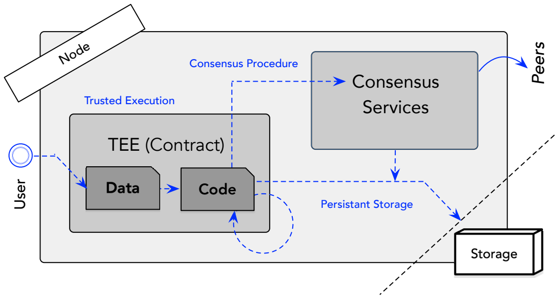

Fig. 3. Layer-one execution model

System model. In a layer-one execution model, the operation of ledger update (consensus) and state transition (contract execution) are coupled. Like Ethereum [2], smart contracts run inside blockchain nodes. Assume that a user plans to use the private contract; she only needs to upload data to the blockchain service and wait for results. The remaining procedures are completed by TEE-assisted distributed nodes. A TEE in these nodes acts as a *black box* for data processing and output targeted results without the data leakage. This approach greatly improves convenience for users due to its easy access and management. As illustrated in Fig.3, a generic data flow goes as follows: A contract creator deploys the code into blockchain. Then, a user sends the transaction with an encrypted argument to an arbitrary blockchain node. Her request is confidentially executed inside TEEs in this node and output encrypted state. Then, the consensus algorithm in this node broadcasts the encrypted results to peers. After the encrypted results are confirmed by other blockchain nodes, users fetch on-chain results and decrypt them for the plaintext.

4.1 Property Evaluation

Privacy-Preserving Property. This property indicates that contract states and the procedure of contract executions are hidden from the public. To achieve privacy, layer-one systems execute these confidential contracts inside TEEs in every distributed node. CCF [55], Fabric [56] and CONFIDE [37] follow this straightforward design where confidential contracts are loaded to the TEE of each consensus node, which encrypts both the inputs and outputs of contract states, together with their operating logic and predefined rules. Enigma[1] [57] introduces the secret network and allows users to submit their transactions together with encrypted data to miners. We also notice that current layer-one solutions only focus on internal procedures rather than the linkability and anonymity of addresses and transactions. This indicates that confidential smart contracts only protect the contents that have been loaded into TEEs, while the data that relates to external users is out of scope.

Blockchain Intrinsic Feature. The layer-one systems inherit most of the features empowered by blockchain. More precisely, the properties of *code immutability*, *high availability*, *explicit invocation*, *decentralized execution*, *automatic execution* and *consensus verifiability* remain the same because basic contract executions still rely on their underlying blockchain systems. Also, the property of (confidential) *state consistency* in Enigma [57], CCF [55] and Fabric [56] remains unchanged. The states and executions from these systems follow the procedures of online consensus processes. Then, the returned results from inside TEEs still require to be confirmed on-chain. This makes their actions effectively perform the same functions as a normal smart contract, except for that the contents of states are transmitted from plaintext to ciphertext. In contrast, the property of *contract interoperability* is lost since the contracts are executed in isolated TEEs. This isolation requires additional communications such as dispatching keys through the remote attestation service, bringing much complexity.

4.2 System Evaluation

The layer-one solution encapsulates TEE computations into blockchain nodes. Every node in the network has to take responsibility for conducting confidential executions and performing the consensus. The design to coordinate TEEs and consensus within the same physical space brings many distinguished features. We start the analysis from their threat model and then dive into each component of these systems.

Threat Model. Users in the layer-one approach are assumed to be unreliable. They may have mistakes unconsciously, like dropping messages or mis-sending transactions. Even worse, a malicious user can arbitrarily behave like faking messages, identities, or compromising other nodes. As for TEE hosts, an external attacker can monitor, eavesdrop or even compromise part of involved TEE hosts among these distributed nodes, but cannot block all traffic transmitted in communication channels. Subsequently, a TEE is supposed to work in a good condition: The attestation service is trusted, and the cryptographic primitives used inside TEEs are secure. Meanwhile, as for the blockchain network, the basic systems (ledgers) are assumed to be robust [58–60]. When running the consensus, the majority (might be two-third, depends on specific consensus algorithms) of nodes are assumed to be honest [61]. Also, forging smart contract codes or states will happen in honest blockchain nodes with a negligible possibility. Based on that, we analyse securities from four aspects.

TEE host security. Firstly, we focus on the security of TEE hosts, or equally, individual nodes that run TEEs.

[1] Enigma's secret network is made of a list of secret nodes equipped with TEE, which is a layer-one solution in the context of our definition. We also note that such a secret network can be used as a layer-two solution for Ethereum.

Unlike classical blockchain systems, there are no explicit incentive or punishment mechanisms in this solution. This is easy to understand: A node with malicious behaviors will be instantly moved out of the committee and replaced by a new honest participant. Meanwhile, due to the fact that CCF [55] and Enigma [57] rely on Tendermint (a BFT variant) consensus algorithm, they can tolerate at most one-third of TEE Byzantine nodes. But the sacrifice is the increased difficulty in synchronization, especially when every node has to establish a secure channel for communications of distributed TEEs. In layer-one systems, host authentication is necessary. The node who wants to join the committee has to obtain permission from communities by proving her TEE capability. For instance, CONFIDE [37] builds a mutual authenticated protocol (MAP) (supported by SGX remote attestation techniques [62]) among blockchain nodes. Any nodes joining in the network have to pass the authentication via MAP.

TEE security. Then, we analyse TEE-level securities. Attestation service is an essential part of TEE techniques. Systems in the layer-one solution still require such services for network connection and verification. Enigma [57], Fabric [56] and CCF [55] follow the original attestation mechanism with an implicit rule: The Intel Attestation Service (IAS) should be reliable. However, this cannot be guaranteed in the case of IAS being comprised. In contrast, CONFIDE [37] utilizes a customized Decentralized Attestation Service to provide the robust authentication. As for memory limitations, layer-one systems load contract executions and consensus algorithms into one TEE-embedded node, causing an increase in disk and memory usage of individual nodes. Once the usage of TEE memory runs over the predefined settings, a decrease in the performance is inevitable [34]. This may further cause an unpredictably severe result like system crash-down. Fortunately, Fabric [56] mitigates such issues by separating the operations into two types (execution and ordering) and delays the transaction-*ordering* procedures after state-*execution*. Among them, only the state-*execution* parts are processed inside TEEs. This decreases computation complexity and limits the memory usage to a suitable range. Physical attacks like the Spectre and Meltdown vulnerabilities [52] are intrinsic design pitfalls that may occur inside the TEE kernel. To our knowledge, no layer-one solutions mention them or provide the remedies.

TEE program security. Next, we focus on the program-level security. Issues like overburdening may frequently happen, especially when a malicious developer deploys a contract with infinite loop logic. Unlike using the gas mechanism in Ethereum [2], systems in the layer-one model constrain their running programs by the *time-out* mechanism. It sets a threshold, namely, a suitable range of time that allows processing contract operations. When exceeding the time-bound, the system will abort under-processing states and restart a new round. As for the flaw detection, no formal techniques or verification tools, based on our observation, have been applied to layer-one systems. This gap needs further exploration. Similar to the previous discussion, the properties of data verification (covering both user data authenticity and blockchain data confirmation) and output conflicts are guaranteed by their underlying consensus algorithms. Each time performing the consensus, these properties are automatically checked. For instance, Enigma [57] relies on trusted validators, who equip with TEEs to conduct the verification procedure. Such validators maintain both the privacy of executions inside TEEs and the consistency of states that connects to peers. Once conflicts occur, validators will quickly make decisions on a block and remove another conflicting block. Fabric [56] performs such a process inside TEEs among committee nodes and then submits the passed results to its abstract ordering service. This service prevents forks caused by conflicting states, as well as proving a fact that: All executed messages are valid and integral once reaching the consensus agreement. It should be noted that, successful consensus procedures can merely guarantee the integrity of transactions and states, rather than linkability and authenticity that relates to physical entities.

TEE key management. Lastly, we move to the aspect of TEE key management. In layer-one systems, the key management service takes over the task of creating and managing keys for activities like attestation, verification, encryption, etc. To achieve the key management service among distributed nodes, several types of designs have been proposed. CCF [55] relies on the public key infrastructure (PKI) for certificate issuance, management, and revocation. It creates key pairs and dispatches them to every participated TEE, where each TEE holder is authenticated by the certificate. Similarly, Fabric [56] adopts an admin peer to provision the specific decryption key to *chaincode enclave* during bootstrapping. Enigma [57] setups an independent key management component to reply to the requests for encryption. Such designs help to simplify complex management procedures, as well as providing distinguishable keys for each TEE. However, these independent key management services lead to centralization even they are maintained by a group of nodes in the committee. CONFIDE [37] mitigates this issue by proposing a decentralized key management protocol. Two types of keys are involved in this protocol: the *asymmetric private key* used to decrypt confidential transactions from clients and the *symmetric states root key* used for state encryption/decryption between the confidential engine and storage service.

4.3 Pros and Cons

The layer-one solution provides a highly integrated approach towards confidential smart contracts.

> This method (L_1) retains most blockchain features such as high availability, rollback attack resistance, decentralized execution, since the contract workflow, data structure, and usage model are consistent with existing systems.

The layer-one solution provides a consistent interface for users without changing the customer's habits transformed from non-TEE blockchain systems. A user can use the layer-one system by directly interacting with the blockchain interface, without considering cumbersome and complicated operations between the TEE and blockchain. However, the layer-one solution still confronts several common disadvantages.

Minimizing the size of Trusted Computing Base (TCB) contributes to the TEE security [63]. In particular, a small TCB has fewer errors and can reduce attack surfaces. However, complicated interactive operations for contract execution and consensus agreement in the L_1 solution greatly increase the size of TCB. Meanwhile, TEE products have limited secure memory. For example, in the current implementation of Intel SGX [35], the enclave page caches are constrained to 128 MB, and only 93 MB of those is available for applications, which limits the concurrent execution.

Furthermore, the layer-one solution lacks compatibility, which means being incompatible with existing blockchain systems. The solution integrates consensus procedure and contract execution into one blockchain node, requiring each node to equip a TEE. Nevertheless, this requirement is difficult to be fulfilled in a public blockchain while already in use (e.g., Ethereum [2]).

5 Layer-Two Solution

The layer-two solution is a straightforward approach that combines the TEE and blockchain to provide smart contracts with confidentiality while keeping scalability. In such systems, the operations of smart contracts are decoupled from their underlying blockchain systems. The smart contracts are executed in an independent layer outside blockchain systems.

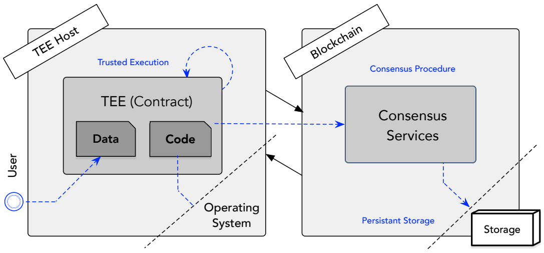

Fig. 4. Layer-two execution model

System model. In a general layer-two solution, the blockchain is used as a dispute resolution layer. The smart contract is executed outside the blockchain, making TEEs act as an agent between users and blockchain systems. Suppose that a user aims to use a private contract. She first needs to compile the original contract code, push binary codes to a TEE, and then upload execution results to the public ledger. As illustrated in Fig.4, we extract a generic data flow as follows. A user sends the encrypted input data to a TEE-powered node. Then, the TEE decrypts the input data and executes the contract. After that, the encrypted execution results are sent to the blockchain platform for verification and storage. Finally, the user fetches and decrypts the blockchain-confirmed results.

5.1 Property Evaluation

Privacy-Preserving Property. The *confidential execution* is an essential property. In layer-two systems, such as [64–66], the contract computations run inside Intel SGX enclaves, while TZ4Fabric [67] moves contract executions into ARM Trusted Zone. Since the contract state-transition process happens inside TEEs, any intermediate states remain invisible to the outside. Meanwhile, to achieve the full life-cycle security for a smart contract, the input sent to a TEE and the output returned from this TEE are also required to be encrypted. For example, in ShadowEth [65], PDOs [64], Phala [66] and Hybridchain [41], the contract invocation arguments are encrypted with the TEE public key. They can only be decrypted within the enclave. Also, before transferring execution results to the blockchain (or users), the intermediate (or final) states in an enclave are encrypted. Some variants also enhance the privacy-preserving properties from other aspects. In Phala [66], only authorized queries to the contract will be answered. The smart contract source codes in ShadowEth [65] are hidden during the procedures of deployment and synchronization. This further reduces the possibility of data leakage in subsequent contract executions. Considering a fixed address may expose the user who has invoked the contract, PDOs [64] also allows the user to use pseudonym addresses for submitting a transaction (including TEE outputs) to the blockchain.

Blockchain Intrinsic Feature. ShadowEth [65] and Taxa [68] introduce an external distributed service to manage the contracts, achieving the properties of *code immutability*, *high availability* and *decentralized execution*. Meanwhile, layer-two systems satisfy *state consistency* for reasons that the encrypted states of contracts in different blockchain nodes will eventually get consistent when reaching a successful agreement. Intuitively, the contracts deployed in layer-two systems should retain the features given by original blockchains. However, some fundamental properties are lost when using layer-two solutions. For example, most layer-two systems lose contract interoperability since each contract is executed in different machines. Among all the evaluated systems, only Phala [66] identifies this issue and proposes a command query responsibility segregation architecture to ensure certain interoperability. Also, public verifiability is a crucial property for the blockchain since it allows each contract invocation, and contract execution to be publicly verifiable. Unfortunately, contracts are executed in TEEs so that the outputs are encrypted. To check whether the TEE has executed contracts following loaded contract specifications is a non-trivial task.

5.2 System Evaluation

Threat Model. An attacker may control the network between users and TEE hosts. Meanwhile, TEEs are assumed to produce the correct results, and the smart contract inside TEEs will not deviate from its specification. The main difference compared with layer-one is that the adversary can observe the network activities between the TEE interface and blockchain nodes.

TEE host security. Several layer-two solutions adopt incentive or punishment mechanisms to encourage TEE hosts to provide a stable and secure environment for executing confidential contracts. For example, Fastkitten [71] and Erdstall [75, 76] propose *penalty* transactions, in which a host will be punished if its malicious behavior has been identified. In particular, if the TEE execution is aborted, the host will be charged according to previous deposits. In Taxa [68], every node can identify any faulty nodes with reliable proofs for executing further economic punishment. On another route, TEE hosts in Phala [66] will get paid by providing their computing resources to users. Similarly, the remuneration in ShadowEth [65] will be transferred to TEE hosts who execute private contracts. These mechanisms can effectively prevent malicious TEE hosts from an economic aspect. However, they are powerless against external threats. An adversary may directly terminate a TEE host at any time. Even worse, the TEE provides users with an open interface that is vulnerable to DoS [77] or single-point attack. To overcome such issues and achieve fault tolerance, different methods are proposed. Fastkitten provides low-level fault tolerance by periodically saving an encrypted snapshot of current states in enclaves. If the enclave fails, the TEE host can instantiate a new enclave and restart the computation starting from the encrypted snapshot. Similarly, Taxa [68] stores a session file for maintaining and recovering user's requests. However, a malicious attacker may directly terminate the TEE host, and Fastkitten does not tolerate such host failures. Another technical route is to maintain a secure network. ShadowEth maintains a group of TEE nodes to ensure consistency via a Paxos-like [78] algorithm. Taxa adopts TEE-enabled computing nodes powered by a PBFT-derived PoS [79] algorithm. Any node in the network has the same responsibility to privately execute smart contracts and transfer execution results to the blockchain. However, this brings additional authentication issues. A TEE host must be carefully authenticated to ensure her TEE capability when joining an external network.

Meanwhile, the systems PDOs [64], Phala [66], Ekiden [70] and COMMITEE [73] introduce an expendable and interchangeable solution. TEEs are stateless: any particular TEE can be easily replaced once it has clashed or finished its task. Unfortunately, these solutions are along with new challenges. Firstly, even if TEEs are changeable, detecting a compromised TEE is still difficult. For instance, PDOs can re-execute a method multiple times for the verification. Given the same input parameters to different TEEs, TEEs are be-

Table 3. Desired properties for current TEE-assisted confidential smart contracts

	Selected Examples	Privacy-Preserving Properties						Blockchain Intrinsic Benefits									
		Specification hidden	Procedure privacy	Input privacy	Output privacy	Address unlinkability	Address anonymity	Code immutable	State consistency	Contract interoperability	High availability.	Decentralized execution.	Automatic execution	Gas mechanism	Explicit invocation	Public verifiability	Consensus verifiability
Layer-one solution	2017, Enigma [57, 69]	○	●	●	●	○	○	↓	–	↓	–	–	–	↓	–	↓	–
	2018, Fabric [56]	○	●	●	●	○	○	–	–	↓	–	–	–	–	–	↓	–
	2019, CCF [55]	○	●	●	●	○	○	–	–	↓	–	–	–	–	–	↓	–
	2020, CONFIDE [37]	○	●	●	●	○	○	–	–	↓	–	–	–	↓	–	↓	–
Layer-two solution	2016, Hawk [21],	○	●	●	●	●	●	–	–	–	–	–	–	–	–	–	–
	2018, PDOs [64]	●	●	●	●	●	○	↓	–	↓	↓	–	–	–	–	↓	–
	2018, ShadowEth [65]	●	●	●	●	◐	○	–	–	–	–	–	–	–	–	↓	–
	2019, Phala [66]	○	●	●	●	○	○	–	–	–	↓	↓	–	–	–	↓	–
	2019, Ekiden [70]	○	●	●	●	○	○	↓	–	↓	↓	–	–	↓	–	↓	–
	2019, Fastkitten [71]	○	●	●	◐	○	○	–	–	↓	↓	↓	↓	–	↓	↓	–
	2019, Avalon [72]	○	●	●	●	○	○	–	–	–	↓	↓	↓	↓	–	↓	–
	2020, Hybridchain [41]	○	●	●	●	○	○	–	–	↓	↓	↓	↓	↓	–	↓	–
	2020, COMMITEE [73]	○	●	●	●	○	○	↓	–	↓	↓	↓	↓	↓	↓	↓	–
	2020, PrivacyGuard [74]	○	●	●	●	○	○	↓	–	↓	↓	↓	↓	↓	–	↓	–
	2020, TZ4Fabric [67]	○	●	●	●	○	○	–	–	↓	↓	↓	↓	↓	–	↓	–
	2020, Taxa [68]	○	●	●	●	○	○	↓	–	–	↓	–	–	↓	–	↓	–
	2021, Erdstall [75, 76]	○	●	●	●	○	○	↓	–	↓	↓	↓	↓	↓	↓	↓	–

● Support; ◐ Optionally support; ○ Did not support;
– Benefit unchanged; ↓ Benefit weakened; Layer-one solutions are in Yellow background.

lieved to work securely only if their outcomes match. Then, the outputs of enclaves are allowed to commit to the blockchain. COMMITEE adopts *master/backup* TEE host mechanism. If the master TEE host is proved to be malicious, a backup TEE host will continue to work without communications to the master TEE host. Nevertheless, this model increases the attack interface and makes the whole system vulnerable. Secondly, TEE hosts are stateless. That means, to ensure an exceptional execution is recoverable, any persistent state must be stored in the blockchain or a trusted third party (TTP). However, for a non-deterministic blockchain system such as Ethereum 2.0 [2], verifying whether an item has been stored on a blockchain is a non-trivial task. Also, storing data in TTPs may lead to single-point failures, which goes against the blockchain's real intention.

TEE security. A contract runs inside TEE, and heavily depends on remote attestation service. The SGX-supported blockchain systems including PDOs [64], Fastkitten [71], ShadowEth [65], Phala [66] and Ekiden [70] assume that Intel Attestation Service (IAS) is trusted. IAS can correctly and completely report whether a certain output with cryptographic material (*quote* [80]) is produced by SGX-enabled hardware. However, IAS might be compromised, posing a risk to these architectures. A compromised or hijacked remote attestation service may maliciously report an attestation with the wrong cryptographic material that does not belong to its corresponding TEE hardware, breaking the promised security. Meanwhile, a centralized service might be crashed, causing the leakage of private states. Unfortunately, none of layer-two schemes consider these risks in their designs or implementations.

As discussed, current TEE implementations have memory limitations for confidential executions. If the memory usage exceeds the threshold, it may confront significant performance and security issues [81]. Hybridchain [41] optimizes the storage by maintaining transaction records outside Intel SGX. Meanwhile, TZ4Fabric

[67] minimizes TCB by avoiding all the executions inside TEEs. However, these approaches increase the implementation complexity. A well-known fact is that a TEE is vulnerable to physical vulnerabilities [52]. Unfortunately, very few layer-two solutions provide remedial measures to reduce the risk of being attacked.

TEE program security. A poorly-written contract might deviate from designated functionalities and further leak the secret information. This part discusses potential pitfalls and remedies when deploying contracts.

In original smart contract systems, gas mechanism is a powerful tool to prevent *infinite loop* attacks [2]. Since the layer-two systems execute smart contract outside the blockchain, a similar mechanism must be considered. Fastkitten [71] and Hybridchain [41] protect against such attacks by using the *timeout* mechanism. Limitations are firstly defined on the maximum amount of execution steps that allow to perform inside a TEE per round. Then, TEE monitors smart contract operations. If the number of execution steps exceeds a predefined threshold, the enclave will terminate executions. ShadowEth [65] combines a timeout mechanism with a *remuneration* mechanism. Similar to the gas mechanism in Ethereum [2], TEE hosts can still gain remuneration even if a contract exits after timeout since they provide sufficient computing power. These mechanisms effectively protect against endless loops and denial-of-service (DoS) launched by external attackers.

As discussed, a TEE itself lacks self-awareness of input data, since it cannot distinguish which state is fresh. A lack of input data authentication makes the system vulnerable to the rollback attack [54, 82]. A malicious user may attempt to invoke the confidential contract many times to seek the leaked secret information. Authentication of the user's identity is helpful to prevent this attack. However, none layer-two solution provides these remedies for these potential pitfalls. On the other hand, the TEE input may come from a non-deterministic blockchain system [83, 84], in which deciding whether an input has been confirmed is tricky. Fastkitten [71] and COMMITEE [73] mitigate this issue by using a *checkpoint* mechanism. As for TEE output conflicts, Ekiden [70] uses a probabilistic proof-of-publication protocol to avoid the ambiguous input.

After the invocation of a private contract, the outputs returned from TEEs are uploaded on-chain for the final confirmation. But a malicious TEE host may send an exceptional result to the blockchain. Even worse, two hosts may publish different updates towards the same contract simultaneously. To prevent such malicious publications and to evade conflicts, PDOs [64] depends on Coordination and Commit Log (CCL) to manage the synchronization in the execution of interacting contracts and enables a contract owner to decide on selecting the enclave for contract executions, which effectively avoid conflicts. Phala [66] adopts an event sourcing command query responsibility segregation architecture to scale up and avoid conflicts, in which the write operations are recorded as events and read operations can be served by the current view of states. Again, these solutions contradict the property of decentralization. Ekiden [70] and ShadowEth [65] rely on the blockchain to resolve conflicts resulting from concurrency. In particular, ShadowEth [65] requires a worker to specify the version number with a timestamp when pushing data to the blockchain. Even miners accept different responses at first, they will eventually reach an agreement by comparing version number and the timestamp, with the help of consensus. Yet, such an approach is inefficient, especially in non-deterministic blockchains.

TEEs key management. PDOs [64] uses a key provisioning service to distribute private keys. The drawback is obvious: A compromised provisioning service could make the entire system fail. To increase the robustness of a private key, Ekiden [70] designs a distributed key generation (DKG) [85] protocol using the secret sharing scheme [86]. Even if one key manager is compromised, an adversary cannot obtain the entire key. However, this solution does not completely solve the key leakage issue. The final keys are assembled and replicated among all end-TEEs. If an adversary compromises an end-TEE, exposing all the contract state becomes a trivial task. The key rotation technology, adopted by Ekiden [70], Fastkitten [71], Phala [66] partially solves the above issue by providing a short-term key in every epoch. An adversary cannot corrupt a future or previous committed state, which minimizes the possibility of key exposure to attackers and further helps the layer-two system to achieve forward secrecy. Also, layer-two projects such as COMMITEE [73] mitigate these key issues by providing each TEE per secret key. Even if a certain TEE's private key were stolen, this only would affect the smart contract running on that compromised TEE. Furthermore, Phala Network [66], equips each contract with an asymmetric key called the *contract key*, which also enhances key security to a certain degree.

Table 4. Evaluation for current TEE-assisted confidential smart contract systems

Selected Examples	Host incentive mechanism	Host punishment mechanism	Host fault tolerance	Host authentication	TEE attestation security	TEE memory limitation	TEE physical attacks	TEE trusted timer	Workload Measurement	Flaws detection	User query restriction	Blockchain data confirmation	TEE output conflicts	Distributed key protocol	Key rotation protocol	Independent contract key	Independent TEE key
2017, Enigma [57, 69]	▲	▲	●	●	▲	▲	●	▲	●	▲	▲	●	●	▲	-	▲	⋆
2018, Fabric [56]	▲	▲	●	●	❍	▲	●	▲	●	▲	▲	●	●	▲	▲	▲	▲
2019, CCF [55]	▲	▲	●	●	▲	▲	●	▲	●	▲	▲	●	●	▲	●	▲	▲
2020, CONFIDE [37]	▲	▲	▲	●	●	▲	▲	▲	●	▲	▲	●	●	●	-	▲	▲
2016, Hawk [21]	▲	▲	-	-	-	-	▲	-	-	-	-	-	-	-	-	-	-
2018, PDOs [64]	▲	▲	●	●	▲	▲	▲	▲	▲	●	▲	▲	●	▲	▲	●	●
2018, ShadowEth [65]	●	▲	●	●	▲	▲	▲	▲	●	▲	●	●	●	▲	●	●	▲
2019, Phala [66]	●	▲	●	▲	●	▲	▲	▲	▲	▲	●	●	●	❍	●	●	▲
2019, Ekiden [70]	▲	▲	●	●	▲	▲	❍	●	▲	▲	❍	●	●	●	●	▲	▲
2019, Fastkitten [71]	▲	●	▲	●	▲	▲	▲	▲	●	▲	●	●	▲	❍	▲	▲	▲
2019, Avalon [72]	▲	▲	●	●	▲	▲	●	▲	▲	▲	▲	-	-	▲	▲	▲	●
2020, Hybridchain [41]	▲	▲	●	▲	▲	●	❍	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲
2020, COMMITEE [73]	▲	▲	●	●	▲	▲	▲	▲	▲	●	▲	●	●	▲	▲	▲	▲
2020, PrivacyGuard [74]	●	▲	▲	▲	▲	▲	▲	▲	▲	▲	▲	●	●	▲	▲	▲	▲
2020, TZ4Fabric [67]	▲	▲	▲	▲	-	●	▲	▲	▲	▲	●	▲	●	▲	▲	▲	●
2020, Taxa [68]	▲	●	●	▲	▲	▲	▲	▲	▲	▲	▲	▲	●	▲	▲	▲	▲
2021, Erdstall [75, 76]	▲	●	●	●	▲	▲	▲	▲	▲	▲	▲	▲	●	▲	▲	▲	●
	TEE Host Security				TEE Security				TEE Program Security					TEE Key Security			

● Considered this pitfall with offering remedy; ❍ Discussed this pitfall without offering remedy; ⋆ Without reference; ▲ Did not consider this pitfall; – The system is secure without this pitfall; Layer-one solutions are in Yellow background.

5.3 Pros and Cons

The layer-two solution decreases computational burden and avoids latency by decoupling the smart contract executions from consensus mechanisms. The solution merely puts the execution results on-chain rather than all processing states.

> In addition to bringing the privacy properties, the layer-two solution enables complex contract executions without slowing down the consensus process, reducing contract costs and improving performance and scalability.

Meanwhile, the layer-two solution does not require a dedicated public ledger, meaning that such a solution can smoothly integrate with existing public blockchain platforms. Unfortunately, this method also brings security and functionality challenges when delegating the task of contract management to an external TEE layer.

Firstly, the layer-two solution complexifies contract data management. The contracts that are deployed outside the blockchain require an external execution/storage party. A malicious storage maintainer may reject to provide the service, while a malicious host may abort TEE executions, terminate enclaves or delay/drop messages. Even an honest host might accidentally lose states in a power cycle. To solve the centralization issue and tolerate host failures, many countermeasures such as the TEE network, stateless TEEs and punishment mechanisms, are proposed. However, these solutions are not effortless, inevitably making the system complicated and hard to implement in practice.

Secondly, the layer-two solution increases the attack surface and thus becomes vulnerable to rollback attacks. There is a high probability that an adversary node can revert transactions where temporary forks, representing inconsistent blockchain views, are allowed in blockchain systems with probabilistic consensus (e.g., PoW). Since TEEs provide no guarantee on verification of input data; they cannot distinguish whether an input state is fresh or not. An attacker may offer stale states to resume a TEE's execution. This enables rollback attacks against randomized TEEs programs. Even worse, plugging up these loopholes needs much effort.

6 Discussion

This section compares layer-one and layer-two solutions, and discusses the hardware's options and impacts.

6.1 L1 and L2 Comparison

Which solution is more secure? Even if we have built clear security metrics based on threat models and give concise security analyses in the context of layered architectures, it is still inadequate for answering this question: *Which solution is more secure, layer-one solution or layer-two solution?* This is because system security is a multidimensional topic, and measuring all security aspects is impractical. The security flaws may happen in any phase in a system [87]. Despite some projects performing well in our evaluation, we cannot roughly say that they are more secure. As hybrid technologies, both layer-one and layer-two systems have unsatisfactory security vulnerabilities in existing systems, and they must be carefully treated when applying them to real applications. Frankly speaking, there is a long road to achieving such a practically secure and confidential system. Our aim is not to argue which solution is more secure. Instead, we focus on helping developers and communities to establish a security measurement and avoid potential pitfalls in designing TCSC.

Which solution is more efficient? The layer-one solutions require the contract to be confidentially executed in a distributed TEE network, which is time-consuming and hard to scale out. In contrast, layer-two systems only upload final calculated results from offline TEEs to online blockchains. Local TEE hosts can execute complicated computations with high scalability and short execution time. Assuming that the on-chain processing time remains stable, the overall performance gets improved by enabling parallel off-chain executions. Thus, from the view of performance and scalability, the layer-two solution is our recommendation.

Which solution is more adoptable? From the aforementioned discussion, we can observe that the layer-one and layer-two solutions fit different scenarios. The layer-one solution is more adoptable in consortium blockchain systems, while the layer-two solution well fits the existing public blockchain systems. Layer-one systems require each blockchain node to equip a TEE, which is difficult to be fulfilled in a public blockchain while already in use. In a consortium blockchain, the nodes are controllable and manageable, and the committee can require each node to equip a TEE when joining the network. On the flip side, the layer-two solution does not change the original blockchain trust assumption. Instead, it creates an independent layer for executing the smart contract, and thus allows developers to seamlessly integrate the TEE into existing public blockchains without significant modifications.

6.2 Hardware-Anchored TEE Options

Securing smart contracts with TEEs is challenging because we have to assume a strong attacker model, in which the attacker has physical possession of the hardware running the smart contract and can interfere with it in powerful ways. This part discusses the security impact of choosing different TEE architectures. In particular, we select *Intel SGX* [88], *Arm TrustZone* [89] and *dedicated chip* [90] as examples.

Intel SGX is a system allowing one to set up protected enclaves running on an Intel processor. Such enclaves are protected from malware running outside the enclave, including in the operating system. Enclaves can attest their software and computations using a signing key ultimately certified by Intel. Intel SGX has been marketed for desktop machines and servers alike; Microsoft Azure [91] is a commercial cloud offering that allows cloud customers to set up SGX enclaves in the cloud. Many attacks on SGX have been published in the eight years since its release. They may be categorised as side-channel attacks (such as [92]), fault attacks (e.g., [93, 94]) and software attacks ([95]). While some of these attacks can be solved by improvements of SGX, it is unclear that it will ever be possible to have a completely secure version, because the attack surface is large, in the case of smart contracts, one has to assume that attackers have physical possession of the hardware.

ARM TrustZone [89] is a technology widely used in mobile phones to protect secrets, such as the secrets used in banking apps. Its ubiquity makes it an attractive option. However, ARM TrustZone has been attacked even more than Intel SGX, and doesn't offer a suitable attestation framework. Future hardware-anchored security products from ARM may address this problem.

Dedicated chips such as the Open Titan [90] family of chips offer a better solution. Open Titan is an open-source design inspired by Google Titan, a chip used on Google servers and in Google mobile phones. The fact that the smart contract runs on a dedicated chip not shared with attacker code means that the attack surface is much smaller. Attestation frameworks exist for such chips, and the attestation keys can be rooted in a manufacturer's certificate. The kind of attacks mentioned for SGX become much harder to mount. Nevertheless, even dedicated chips may succumb to a dedicated and resourceful attacker. Researchers have succeeded in mounting attacks based on power side-channels and electromagnetic (EM) radiation side channels. Defences against such attacks include masking, which consists of randomly splitting every sensitive intermediate variable into multiple shares. Even if the adversary is able to learn a share of the secret via side-channel, it would need all of them in order to recover the secret. Fault attacks such as EM and voltage glitching are also possible, but again, there are known defences [96] at both a software and hardware level. Software defences include making secret-dependent computations twice (in general n times) and then comparing results before producing any output. Countermeasures in hardware involve having internal voltage monitoring circuitry, which makes sure that the input voltage remains within a safe operation range and resets the device otherwise.

7 Research Challenges

Side Channel Attack. Inevitably, all types of TEEs suffer from side-channel attacks. An attacker may observe untrusted resources to obtain the control flow and data access mode from the running hardware to infer sensitive information. Beyond the normal side-channel attack, an attacker can keep track of the changes in encrypted states recorded on the blockchain to extract secrets. The attacker carefully compares encrypted states before and after running a particular confidential transaction. Even if the attacker cannot directly learn about the plaintext, the changes of the encrypted state may lead to a valuable side-channel attack. Come back to the e-voting example, the changes of state c'_b indicates a specific sender or receiver's invocation, and ciphertext length reveals which method is being invoked given different arguments size. Meanwhile, the contract application binary interface (ABI) [2], and the contract path will be spied by an attacker, causing data leakage.

Key Management Dilemma. The private keys in TEE-assisted systems are extremely crucial but hard to manage. On the one hand, putting the application keys in a single TEE contributes to the key security. However, it also makes the system raise the risk of a single point of failure. On the other hand, sharing the private key among multiple TEEs offers practical availability but (as a sacrifice) increases key exfiltration risk. Meanwhile, key sharing technologies are too complicated to adopt and cannot completely solve the key issues. Suppose that an attacker steals the attestation key somehow. She might consequently generate the attestation materials to deceive the user with a fake fact: The contract has been executed. Even worse, if a root key stored in the tamper-resistant hardware (e.g., Memory Encryption Engine Key in SGX) is compromised, all key technologies for protecting application keys become useless.

Transparency Issues. Compared with cryptographic approaches backed by mathematics [22, 23, 27], the confidential smart contracts relied on TEEs are lack of transparency. On the one hand, contracts are executed inside TEEs, and the outputs are usually encrypted, which lacks public verifiability inherited from traditional blockchain systems. The attestation service can only guarantee that the encrypted outputs indeed come from a TEE. However, neither users nor the blockchain nodes can learn whether a TEE is compromised or executes contracts following the predefined specifications. Even if many TEEs can re-execute the same contract with the same setup (e.g., the same private key) to check outputs, this inevitably increases the key exfiltration risk in the face of a confidentiality breach. On the other hand, the precise architectures of chips are still unclear for some TEE products, such as Intel SGX [80]. TEE-assisted solutions force the user to put too much trust in the manufacturers of this hardware. Users even argue that Intel may have reduced the security of SGX to improve performance to cater for market demand [97]. Additionally, the attestation service used to prove that a program runs inside TEEs is *centralized* and *non-transparent*. A compromised provider has the ability to insert fake IDs, and further, steal the confidential state in smart contracts.

8 Concluding Remarks

The technologies on how to combine smart-contract execution with TEEs are mushrooming nowadays. The absence of systematic work confuses newcomers. In this paper, we provide the first SoK on TEE-assisted confidential smart contract systems. TEE technologies empower transparent smart contracts with confidentiality, greatly extending the scope of upper-layer applications. We summarize state-of-the-art solutions by proposing a unified framework covering aspects of design models, desired properties, and security considerations. Our analysis clarifies existing challenges and future directions for two mainstream architectures (layer-one and layer-two solutions). We believe that our evaluation and analysis will offer a good guide for communities, and greatly promote the prosperity of TCSC applications.

Acknowledgement

Rujia Li and Qi Wang are partially supported by the Shenzhen Fundamental Research Programs under Grant No.20200925154814002. We also gratefully acknowledge financial support from EPSRC under grants EP/V000454/1 (CAP-TEE: Capability Architectures for Trusted Execution); EP/S030867/1 (SIPP - Secure IoT Processor Platform with Remote Attestation); and EP/R012598/1 (User-Controlled Hardware Security. Anchors: Evaluation and Designs). Lastly, we thank Xinrui Zhang (SUSTech), Feng Liu (USTC), and anonymous reviewers for their valuable comments.

References

[1] Nick Szabo. Smart contracts: building blocks for digital markets. *EXTROPY: JTT,(16)*, 18(2), 1996.

[2] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *https://ethereum.github.io/yellowpaper/paper.pdf*, 2022.

[3] Kevin Delmolino et al. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In *FC*, pages 79–94. Springer, 2016.

[4] Hewa et al. Survey on blockchain based smart contracts: Technical aspects and future research. *IEEE Access*, 2021.

[5] Maher Alharby and Aad Van Moorsel. Blockchain-based smart contracts: A systematic mapping study. *arXiv preprint arXiv:1710.06372*, 2017.

[6] Marc Jansen et al. Do smart contract languages need to be turing complete? In *CBA*, pages 19–26. Springer, 2019.

[7] Siraj Raval. *Decentralized applications: harnessing Bitcoin's blockchain technology.* " O'Reilly Media, Inc.", 2016.

[8] Weiqin Zou et al. Smart contract development: Challenges and opportunities. *TSE*, 2019.

[9] Rui Zhang, Rui Xue, and Ling Liu. Security and privacy on blockchain. *CSUR*, 52(3):1–34, 2019.

[10] Steven Goldfeder. Private smart contracts. 2018.

[11] Samuel S., Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin Vechev. zkay: Specifying and enforcing data privacy in smart contracts. In *CCS*, pages 1759–1776, 2019.

[12] Karim Baghery. On the efficiency of privacy-preserving smart contract systems. In *AFRICACRYPT*, pages 118–136. Springer, 2019.

[13] A. Unterweger, F. Knirsch, et al. Lessons learned from implementing a privacy-preserving smart contract in ethereum. *NTMS*, pages 1–5, 2018.

[14] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In *CCS*, pages 270–282, 2016.

[15] Erik-Oliver Blass and Florian Kerschbaum. Borealis: Building block for sealed bid auctions on blockchains. In *AsiaCCS*, pages 558–571, 2020.

[16] Hisham S Galal and Amr M Youssef. Trustee: full privacy preserving vickrey auction on top of ethereum. In *FC*, pages 190–207. Springer, 2019.

[17] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Mueller, and Tomasz Truderung. Sok: Verifiability notions for e-voting protocols. In *SP*, pages 779–798. IEEE, 2016.

[18] Geetanjali Rathee et al. On the design and implementation of a blockchain enabled e-voting application within iot-oriented smart cities. *IEEE Access*, 9:34165–34176, 2021.

[19] General data protection regulation. https://gdpr-info.eu/. 2020.

[20] Paul Voigt et al. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10:3152676, 2017.

[21] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *SP*, pages 839–858. IEEE, 2016.

[22] Harry Kalodner et al. Arbitrum: Scalable, private smart contracts. In *USENIX Security*, pages 1353–1370, 2018.

[23] B. Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *SP*, pages 315–334. IEEE, 2018.

[24] Benedikt Bünz et al. Zether: Towards privacy in a smart contract world. In *FC*, pages 423–443. Springer, 2020.

[25] Yu Chen, Xuecheng Ma, Cong Tang, and Man Ho Au. Pgc: Decentralized confidential payment system with auditability. In *ESORICS*, pages 591–610. Springer, 2020.

[26] Ravital Solomon et al. smartfhe: Privacy-preserving smart contracts from fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2021:133, 2021.

[27] Guy Zyskind et al. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv:1506.03471*, 2015.

[28] Dayeol Lee, David Kohlbrenner, et al. Keystone: An open framework for architecting trusted execution environments. In *EuroSys*, pages 1–16, 2020.

[29] Jan-Erik Ekberg et al. Trusted execution environments on mobile devices. In *CCS*, pages 1497–1498, 2013.

[30] Seongmin Kim et al. Enhancing security and privacy of tor's ecosystem by using trusted execution environments. In *NSDI*, pages 145–161, 2017.

[31] David Kaplan, Jeremy Powell, and Tom Woller. Amd memory encryption. *White paper*, 2016.

[32] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. Sanctuary: Arming trustzone with user-space enclaves. In *NDSS*, 2019.

[33] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. *Hasp@ isca*, 10(1), 2013.

[34] ChongChong Zhao et al. On the performance of intel sgx. In *WISA*, pages 184–187. IEEE, 2016.

[35] Jinhua Cui et al. Dynamic binary translation for sgx enclaves. *arXiv preprint arXiv:2103.15289*, 2021.

[36] Rujia Li, Qin Wang, et al. An offline delegatable cryptocurrency system. *arXiv preprint arXiv:2103.12905*, 2021.

[37] Ying Yan, Changzheng Wei, et al. Confidentiality support over financial grade consortium blockchain. In *SIGMOD*, pages 2227–2240, 2020.

[38] Rohit Sinha et al. Luciditee: A tee-blockchain system for policy-compliant multiparty computation with fairness.

[39] Chinese chang'an chain enterprise blockchain joins digital yuan project, Mar 2021.

[40] Financials. Changan chain, the first independent and controllable blockchain technology system in china, was released today.

[41] Yong Wang et al. Hybridchain: A novel architecture for confidentiality-preserving and performant permissioned blockchain using trusted execution environment. *IEEE Access*, 8:190652–190662, 2020.

[42] Adam Young and Moti Yung. The dark side of "black-box" cryptography or: Should we trust capstone? In *CRYPTO*, pages 89–103. Springer, 1996.

[43] Rujia Li, David Galindo, and Qi Wang. Auditable credential anonymity revocation based on privacy-preserving smart contracts. In *CBT*, pages 355–371. Springer, 2019.

[44] Rujia Li, Qin Wang, et al. An accountable decryption system based on privacy-preserving smart contracts. In *ISC*, pages 372–390. Springer, 2020.

[45] Oasis lab. *https://github.com/oasislabs/secret-ballot/blob/master/contracts/SecretBallot.sol*.

[46] Véronique Cortier et al. Election verifiability for helios under weaker trust assumptions. In *ESORICS*, pages 327–344. Springer, 2014.

[47] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, H. Perl, I. Goldberg, and M. Smith. Sok: Secure messaging. *SP*, pages 232–249, 2015.

[48] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In *FC*, pages 34–51. Springer, 2013.

[49] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, et al. A fistful of bitcoins: characterizing payments among men with no names. In *IMC*, pages 127–140, 2013.

[50] Ferdinand Brasser et al. Software grand exposure:{SGX} cache attacks are practical. In *WOOT*, 2017.

[51] Yuanzhong Xu et al. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *SP*, pages 640–656. IEEE, 2015.

[52] Mark D Hill et al. On the spectre and meltdown processor security vulnerabilities. *IEEE Micro*, 39(2):9–19, 2019.

[53] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[54] Ivan Homoliak and Pawel Szalachowski. Aquareum: A centralized ledger enhanced with blockchain and trusted computing. *arXiv preprint arXiv:2005.13339*, 2020.

[55] Mark Russinovich et al. Ccf: A framework for building confidential verifiable replicated services. Technical Report MSR-TR-2019-16, Microsoft, April 2019.

[56] Marcus Brandenburger et al. Blockchain and trusted computing: Problems, pitfalls, and a solution for hyperledger fabric. *arXiv preprint arXiv:1805.08541*, 2018.

[57] Enigma – securing the decentralized web. *https://www.enigma.co/*.

[58] Juan Garay et al. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, pages 281–310. Springer, 2015.

[59] Juan Garay et al. The bitcoin backbone protocol with chains of variable difficulty. In *CRYPTO*, pages 291–323. Springer, 2017.

[60] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT*, pages 643–673. Springer, 2017.

[61] Juan Garay and Aggelos Kiayias. Sok: A consensus taxonomy in the blockchain era. In *RSA*, pages 284–318. Springer, 2020.

[62] Intel. Intel software guard extensions (intel sgx). *Accessible on https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html*, 2020.

[63] Robert Krahn, Donald Dragoti, Franz Gregor, et al. Teemon: A continuous performance monitoring framework for tees. In *Middleware*, pages 178–192, 2020.

[64] Mic Bowman et al. Private data objects: an overview. *arXiv preprint arXiv:1807.05686*, 2018.

[65] Rui Yuan et al. Shadoweth: Private smart contract on public blockchain. *JCST*, 33(3):542–556, 2018.

[66] Yin Hang, Zhou Shunfan, and Jiang Jun. Phala network: A confidential smart contract network based on polkadot. *https://files.phala.network/phala-paper.pdf*, 2019.

[67] Christina Müller, Marcus Brandenburger, et al. Tz4fabric: Executing smart contracts with arm trustzone. *arXiv preprint arXiv:2008.11601*, 2020.

[68] Taxa. Taxa network: a universal logic layer for blockchain. Website, 2021. https://taxa.network/.

[69] Enigma. The developer quickstart guide to enigma | by enigma project | enigma. *https://blog.enigma.co/the-developer-quickstart-guide-to-enigma-880c3fc4308*.

[70] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *EuroSP*, pages 185–200. IEEE, 2019.

[71] Poulami Das et al. Fastkitten: Practical smart contracts on bitcoin. In *USENIX Security*, pages 801–818, 2019.

[72] Hyperledger. Introducing hyperledger avalon. www.hyperledger.org/blog/2019/10/03/introducing-hyperledger-avalon, 2019. (Accessed on 04/19/2021).

[73] Andreas Erwig, S. Faust, et al. Commitee: An efficient and secure commit-chain protocol using tees. *IACR Cryptol. ePrint Arch.*, 2020:1486, 2020.

[74] Yang Xiao et al. Privacyguard: Enforcing private data usage control with blockchain and attested off-chain contract execution. In *ESORICS*, pages 610–629. Springer, 2020.

[75] Perun Network. Introducing erdstall: Scaling ethereum using trusted execution environments | by perun network | perunnetwork | medium.

[76] Erdstall. Technology – erdstall. https://erdstall.dev/technology/. (Accessed on 04/17/2021).

[77] Wentao Liu. Research on dos attack and detection programming. In *Third International Symposium on Intelligent Information Technology Application*, volume 1, pages 207–210. IEEE, 2009.

[78] Roberto De Prisco et al. Revisiting the paxos algorithm. *Theoretical Computer Science*, 243(1-2):35–91, 2000.

[79] Peter Gaži, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *SP*, pages 139–156. IEEE, 2019.

[80] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.

[81] Nico W., Pierre-Louis Aublin, and Rüdiger Kapitza. sgx-perf: A performance analysis tool for intel sgx enclaves. In *Middleware*, pages 201–213, 2018.

[82] R. Pries et al. A new replay attack against anonymous communication networks. *ICC*, pages 1578–1582, 2008.

[83] Marcus Brandenburger, Christian Cachin, Rüdiger Kapitza, and Alessandro Sorniotti. Trusted computing meets blockchain: Rollback attacks and a solution for hyperledger fabric. In *SRDS*, pages 324–32409. IEEE, 2019.

[84] Shenbin Zhang et al. A solution for the risk of non-deterministic transactions in hyperledger fabric. In *ICBC*, pages 253–261. IEEE, 2019.

[85] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT*, pages 295–310. Springer, 1999.

[86] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[87] Shari Pfleeger and Robert Cunningham. Why measuring security is hard. *IEEE SP*, 8(4):46–54, 2010.

[88] Intel. Introduction to intel® sgx sealing. Website, 2016. https://software.intel.com/content/www/us/en/develop/blogs/introduction-to-intel-sgx-sealing.html.

[89] Sandro Pinto and Nuno Santos. Demystifying arm trustzone: A comprehensive survey. *CSUR*, 51(6):1–36, 2019.

[90] Scott Johnson et al. Titan: enabling a transparent silicon root of trust for cloud. In *Hot Chips: A Symposium on High Performance Chips*, volume 194, 2018.

[91] Cynthia Dwork. Microsoft azure. 2021.

[92] Jo Van Bulck et al. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *USENIX Security*, pages 991–1008, 2018.

[93] Kit Murdock, David Oswald, Flavio D Garcia, et al. Plundervolt: Software-based fault injection attacks against intel sgx. In *SP*, pages 1466–1482. IEEE, 2020.

[94] Zitai Chen et al. Voltpillager: Hardware-based fault injection attacks against intel sgx enclaves using the svid voltage scaling interface. In *USENIX Security*, 2021.

[95] Jo Van Bulck, David Oswald, et al. A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes. In *CCS*, pages 1741–1758, 2019.

[96] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPOTO*, pages 513–525. Springer, 1997.

[97] Tu Dinh Ngoc, Bao Bui, et al. Everything you should know about intel sgx performance on virtualized systems. *POMACS*, 3(1):1–21, 2019.

[98] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[99] Tsz Hon Yuen, Shi-feng Sun, et al. Ringct 3.0 for blockchain confidential transaction: Shorter size and stronger security. In *FC*, pages 464–483. Springer, 2020.

[100] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2008.

[101] Ying Lan et al. Trustcross: Enabling confidential interoperability across blockchains using trusted hardware. *arXiv preprint arXiv:2103.13809*, 2021.

Appendix A. Key Management

A variety of different keys are used in the life cycle of TCSC. For simplicity, we use Intel SGX as the instance. We classify these keys into two types, namely, *service keys* (top half) and *SGX internal keys* (bottom half).

Service keys. The keys sk_{tx} and pk_m are used to sign a transaction and encrypt a message resulting from a TEE. Correspondingly, the keys pk_{tx} and sk_m are used to verify a signature and decrypt a ciphertext, respectively. Meanwhile, the TEE service key key_{tee} is used to encrypt the contract state, and the asymmetric TEE service key sk_{tee} is used to encrypt a voter's input. Since the key management technologies have significant impacts on these service keys, we emphasize them with the yellow background.

SGX internal keys. The MEE key is generated at boot, and is placed in special registers, and destroyed at system reset. The MEE key is used for memory encryption and decryption, which plays a crucial role in protecting the confidentiality and integrity of enclaves. At the same time, different enclaves in the same TEE platform share one function key, such as the report key and the attestation key [88].

Appendix B. Anonymity and Confidentiality

Anonymity refers to the privacy that relates to real entities, especially for users' identities. In a blockchain system, anonymity indicates that users' transaction activities will not expose any personal information about them. Alternatively, an attack cannot obtain the correct links between real users and their corresponding account/address that sends the transaction [98]. Bitcoin and Ethereum only provide a very early version of anonymity, using the pseudonym-based address mechanism to protect identities. However, this cannot guarantee anonymity because attackers can effortlessly map virtual addresses to physical entities through the relationship analysis.

Confidentiality in a blockchain system mainly refers to the privacy of data and contents recorded on-chain [9, 99]. Classic blockchain systems expose all transactions (includes amount information, addresses, amount, etc.) plainly where anyone can read and access. Sensitive information might unconsciously be leaked to malicious analyzers. For instance, ERC20 tokens in the Ethereum system do not provide confidentiality, since anyone can observe every amount's balance. Adversaries can keep tracing the accounts that have a huge amount of tokens and launch attacks such as using the phishing website or cheating through offline activities.

Appendix C. Background

Blockchain Technology. Blockchain, conceptualized by Nakamoto [100], was proposed as a distributed and append-only ledger, in which all committed transactions are stored in a chain of data records (named as blocks). According to the initial idea of Bitcoin [1], when the blockchain maintainers reach an agreement on the newest block, related transactions appearing in that time will be packaged in this block and further stored in a distributed network to maintain a continuously growing list. By providing a secure solution to distribute the information and allowing all participants to audit the shared records, blockchain obtains many key characteristics such as decentralization, auditability and non-repudiation, transparency, and non-equivocation.

Smart Contract Proposed by Szabo [1], the smart contract are widely applied in blockchain systems by Ethereum [2]. Blockchain-based smart contracts adopt Turing-complete scripting languages to achieve complicated functionalities [6] and execute thorough state transition/replication over consensus algorithms to realize final consistency. By the design, a blockchain-based smart contract includes multiple functions, methods, and a few parameters that can run on the blockchain when specific conditions or events are met and encompass business logic and transactions between two or more parties. To be specific, the source code of a contract forming as part of a transaction is first sent to the blockchain. Once the transaction is included in a new block and confirmed by the majority of the participants, the contract code becomes immutable and executable. When an external user invokes the contract, the state will be updated under the instruction of the preloaded source code. The neutrality of the execution environment among all blockchain nodes facilitates the same execution result of the program code. Smart contracts thus enable unfamiliar and distributed participants to fairly exchange without trusted third parties and present a uniform approach to improve applications across a wide range of industries.

Trusted Execution Environments. Trusted Execution Environment (TEE) [29] provides a protected processing area in the main processor that runs on a separation kernel to ensure confidentiality and integrity of inside data and computations. State-of-the-art implementations include Intel Software Guard Extensions (SGX) [80], ARM TrustZone [89], Keystone [28], *etc.* For a TEE, three main TEE features are highlighted, including *runtime isolation*, *sealing technologies* and *attestation technologies.* For simplicity, we use Intel SGX as an example to explain these features in the following paragraphs. It has to be mentioned that the Intel SGX design used in our paper can also be implemented on other trusted hardware platforms such as Keystone [28].

Runtime Isolation. The secure and isolated regions of memory are called *enclaves.* Sensitive data and intermediate computations run inside enclaves to provide protection against outside programs. Besides, all the runtime enclave memories are stored in Enclave Page Cache (EPC) [101] and encrypted by Memory Encryption Engine (MEE). These protective mechanisms enforced in SGX protect memories against the access of any process outside the enclave itself, including the operating system, hypervisors, etc.

Sealing. Sealing [88] is a process of loading enclave internal secret state to persistent storage. Roughly speaking, using the Sealing, the secrets can be encrypted and stored in the untrusted memory or disk. Further, it

Table 5. Key Types in Confidential Smart Contracts: The table shows a voting example achieved by Intel SGX and Ethereum.

Keys	Purpose	Remarks
Transaction signing key (sk_{tx}, vk_{tx})	signing a transaction	generated by a voter
A vote's message key (sk_m, pk_m)	encrypt a message that comes from a voter	generated by a voter
TEE service key key_{tee}	encrypt and decrypt a blockchain state	generated inside an enclave
TEE service key (sk_{tee}, pk_{tee})	decrypt a user's input/encrypt a TEE's output	generated inside an enclave
Memory Encryption Engine Key	encrypt (decrypt) the data before writing (reading) it to (from) RAM.	stored inside a CPU; different enclaves in the same TEE platform share the same MEE key.
Report key	generate a MAC tag for the measurement.	generated by EGETKEY instruction; different enclaves in the same TEE platform share one report key;
Attestation key	produce attestation signatures.	stored in tamper-resistant hardware; different enclaves in the same TEE platform share one attestation key;
Sealing key	migrate secrets between enclaves.	stored in tamper-resistant hardware; different enclaves in the same TEE platform may share the sealing key depending on key policies.

allows such encrypted secrets to be retrieved once the enclave is torn down (either due to the host's power or the application itself). Sealing is achieved by using a private seal key [80], which covers two types of identities: Enclave Identity and Signing Identity. Enclave Identity is represented by the value of *MRENCLAVE*, which is a cryptographic hash of the enclave measurement. Any operation inside an enclave that changes measurement will yield a different key. Thus, it restricts the permission to sealed data; only the corresponding enclave can access the sealed data. In contrast, Signing Identity is provided by an authority and represented by *MRSIGNER*. It provides the same sealing key for different enclaves, or even different versions of the same enclave. Therefore, Signing Identity can be used to share sensitive data between multiple enclaves produced by the same development firm.

Attestation. Attestation mechanism [33] is used to prove to a validator that an enclave has been correctly instantiated and when in that condition can then proceed to further establish a secure, authenticated connection for the data transmission. SGX provides two types of attestation: *local attestation* and *remote attestation*. In the former attestation, SGX facilitates the instructions to help an enclave to attest to another enclave on the same platform. In the latter one, SGX enables an enclave to prove a correct loading of code and data to another enclave that resides in a remote platform.

Appendix D. A TCSC-Based Voting Protocol

In this part, we provide a detailed description of TCSC-based voting system that utilizes the Intex SGX. The protocol mainly consists of two sub-procedures: *deployment stage* and *execution stage*.

Deployment Stage. In the deployment stage, all the operational code and the initial state are coded into a TCSC. This stage includes two steps.

a. Compile. Firstly, contract binary codes are compiled into enclave codes. Since an enclave has only a small quantity of trusted zones for application code and data (the protected memory is 128MB, and only 96MB is usable for an enclave in the current version of Intel SGX [88]), a contract has to determine the boundary of these zones and identify corresponding zones used for privacy-critical functionalities. In particular, the e-voting contract needs to define: *the scope of secret states*, *the scope of public states*, *the approach to access secret states* and *the approach to access external states.*

Enclave Definition Language (EDL) [80] defines trusted components, untrusted components, and corresponding interfaces between them, which takes charge of translation from contract code to enclave code. It provides two functionalities: Enclave Calls (ECALLs) and Outside Calls (OCALLs). ECALLs define the functions inside the enclave that are used to expose APIs for untrusted applications to call in. OCALLs specify untrusted functions outside the enclave where the enclave code is able to invoke. In our example, the total number of votes cast for a candidate cannot be revealed until the

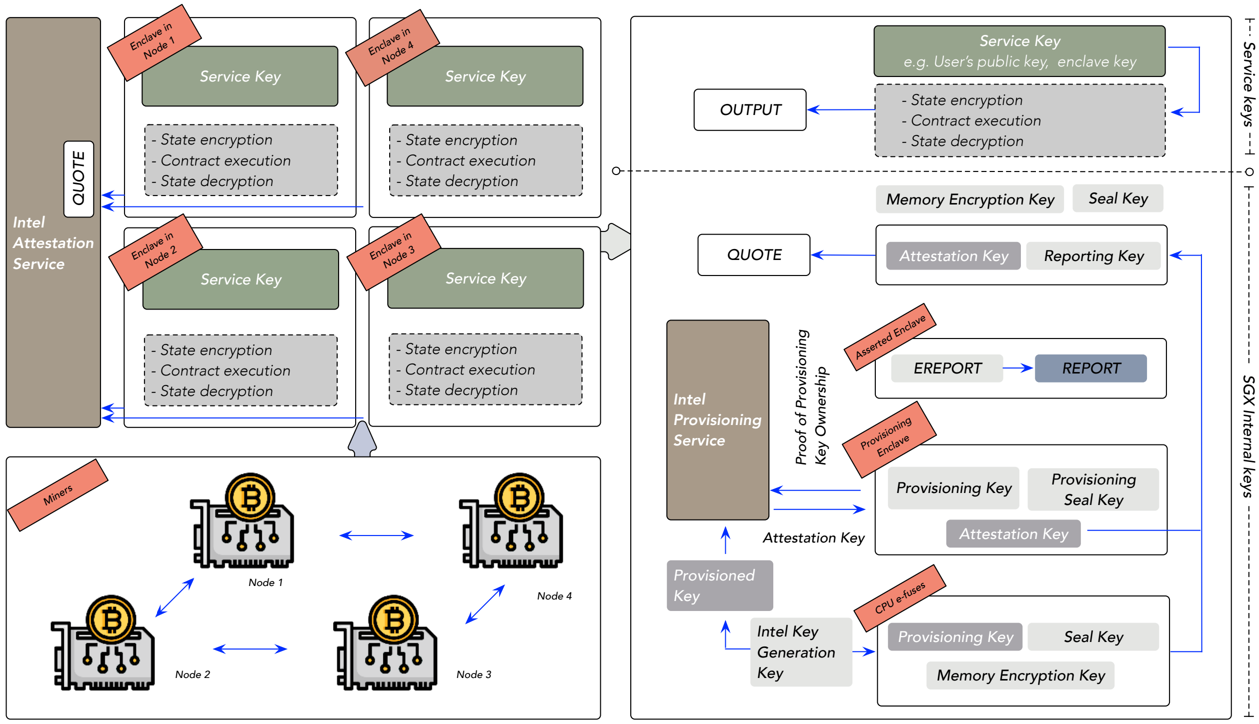

Fig. 5. Key Usage in TEE-assisted Confidential Blockchain System.

voting has ended. Thus, the total number of votes cast is defined at the access point ECALLs, and is thereby hidden from the public, and can only be revealed once the voting procedure has been completed.

b. Load. Afterwards, EDL files will load into an enclave, which is stored in the Enclave Page Cache (EPC). From a micro perspective, the first step is to call the ECREATE instruction for creating an enclave. This will allocate memory inside the Enclave Page Cache (EPC). Then, enclave code and data are added to pages in EPC by calling the EADD instruction. Finally, when the instruction EINIT completes successfully, an enclave's $INIT$ attributes become true, and the above instructions cannot be used any more. After a successful deployment, the initial state and operational code of this contract will be replicated among blockchain nodes. This means the e-voting logic cannot be changed. But, the state of functionalities can be transferred to parties who have been granted permission with a message-call [2].

Execution Stage. In the execution stage, voters call the deployed TCSC to finish the voting. Firstly, an enclave needs to fetch the current contract state from the blockchain. Then, the CPU executes the plaintext contract in the enclave mode. External attackers cannot obtain the knowledge of sensitive information since the Memory Encryption Engine (MEE) key never leaves TCB. A critical aspect of Intel SGX's functionality is that the code inside an enclave can access the particular enclave state by performing additional checks on memory semantics. Back to our example, confidential state (the encrypted number of votes cast for a candidate *voteresult*) will return only when the following four requirements are fulfilled: (1) The processor runs in enclave mode; (2) The requested page is part of the same enclave; (3) The page access is through the correct specific virtual address; (4) The code semantics successfully pass the check. In a word, the CPU is acting as a doorman in the TCSC, providing a hardware-based access control mechanism. After obtaining results from TEEs, the consensus algorithm starts to reach an agreement. To be specific, when a miner receives a newly mined block, he will re-execute all transactions inside the block to obtain the newly transferred state. Once enough blockchain miners receive the block and re-execute transactions, the voting results and the transactions triggering the contract execution will eventually reach the final agreement. When all the voting procedures have completed, the teller can fetch the final encrypted state and obtain the final voting result. Meanwhile, the transactions can be used as evidence to trace the voter's behavior.