High-Confidence Computing 4 (2024) 100183

Contents lists available at ScienceDirect

High-Confidence Computing

journal homepage: www.sciencedirect.com/journal/high-confidence-computing

Review

SoK: Privacy-preserving smart contract

Huayi Qi, Minghui Xu *, Dongxiao Yu, Xiuzhen Cheng [1]

School of Computer Science and Technology, Shandong University, Qingdao 266237, China

ARTICLE INFO

Article history:
Received 11 August 2023
Revised 13 October 2023
Accepted 6 November 2023
Available online 23 November 2023

Keywords:
Privacy
Smart contract
Zero-knowledge proof
Trusted execution environment
Blockchain

ABSTRACT

The privacy concern in smart contract applications continues to grow, leading to the proposal of various schemes aimed at developing comprehensive and universally applicable privacy-preserving smart contract (PPSC) schemes. However, the existing research in this area is fragmented and lacks a comprehensive system overview. This paper aims to bridge the existing research gap on PPSC schemes by systematizing previous studies in this field. The primary focus is on two categories: PPSC schemes based on cryptographic tools like zero-knowledge proofs, as well as schemes based on trusted execution environments. In doing so, we aim to provide a condensed summary of the different approaches taken in constructing PPSC schemes. Additionally, we also offer a comparative analysis of these approaches, highlighting the similarities and differences between them. Furthermore, we shed light on the challenges that developers face when designing and implementing PPSC schemes. Finally, we delve into potential future directions for improving and advancing these schemes, discussing possible avenues for further research and development.

1. Introduction

Smart contracts are programs designed to be correctly executed by a network of blockchain miners who reach a consensus. However, by the nature of blockchain, the execution process of smart contracts is publicly visible, which means anyone is able to inspect all data. Hence, many privacy-preserving smart contract applications in various areas have been proposed to prevent privacy disclosure, such as healthcare [1,2], data marketplace [3,4], energy [5,6], machine learning [7], and e-commerce [8]. These applications utilize commitments or ciphertexts to replace a portion of on-chain data in order to maintain confidentiality. However, these privacy-preserving solutions are not application-agnostic, i.e., specific to each application and cannot be easily applied to other schemes. Whenever a scheme necessitates a privacy-preserving smart contract solution, developers must either rely on fundamental cryptographic tools such as zero-knowledge proofs, encryption schemes, and hash algorithms to construct their own solutions, which demands a high level of expertise.

To address this issue, efforts have been made to develop comprehensive and universally privacy-preserving smart contract (PPSC) schemes. These schemes are not limited to specific applications but provide a solution that can benefit a wide range of scenarios, aiming at enabling developers to effortlessly create PPSC applications by leveraging compiler toolchains, similar to how regular smart contracts are developed. The objective of this paper is to systematize existing research on PPSC schemes, which we categorize into two types: crypto-based PPSC schemes and TEE-based PPSC schemes, depending on the techniques they utilize. These schemes have varying designs and face both shared and unique challenges.

Crypto-based PPSC schemes. These schemes utilize cryptographic tools such as non-interactive zero-knowledge proof (NIZK), secure multi-party computation (MPC), and homomorphic encryption (HE). To handle private data, most of these schemes either directly adopt and extend Zerocash's DAP scheme [9], where the commitments of private data are stored on the blockchain and can only be used once, or they become variants of DAP scheme to store the ciphertext of private data on the blockchain. Users keep their encryption keys or private data off the chain. The main challenge lies in enabling efficient smart contract execution that involves private data from multiple users. Additionally, these schemes also encounter challenges in terms of NIZK efficiency, NIZK expressiveness, function privacy, and inter-contract exchanging data.

TEE-based PPSC schemes. By introducing hardware manufacturers as trusted parties, these schemes utilize trusted execution environments (TEEs), especially Intel SGX. A key feature called remote attestation provide proofs to convince other parties of the trustworthiness and integrity of the executing environment, which eliminates the need for NIZK proofs. TEEs address

* Corresponding author.
E-mail address: mhxu@sdu.edu.cn (M. Xu).

[1] Given her role as Editor-in-Chief of this journal, Xiuzhen Cheng had no involvement in the peer-review of this article and had no access to information regarding its peer-review. Full responsibility for the editorial process for this article was delegated to Dr. Qin Hu.

https://doi.org/10.1016/j.hcc.2023.100183

the efficiency issues faced by crypto-based schemes, particularly in multi-user scenarios. Nevertheless, challenges persist in TEE-based PPSC schemes, especially regarding the proper design of key distribution and short-term key derivation to manage private data.

Structure of the paper. The reminder of this paper is organized as follows. Section 2 provides an overview of PPSC schemes in a general model, along with an introduction to the necessary cryptographic and hardware techniques required for constructing such a scheme. Sections 3 and 4 present a comprehensive overview of crypto-based PPSC schemes and TEE-based PPSC schemes respectively. In Section 5, we highlight the challenges in implementing PPSC schemes, and in Section 6, we discuss ways to further enhance these schemes. Finally, we conclude the paper in Section 7.

2. Preliminaries

This section presents an overview of privacy-preserving smart contract (PPSC) schemes in a general model and introduces the essential cryptographic and hardware techniques required for constructing a PPSC scheme.

2.1. Blockchain and smart contract

A blockchain is a decentralized state machine, maintained by a group of miners running a secure consensus protocol. Upon receiving a transaction message from a user, miners validate the transaction and apply state transition to update the ledger state. For blockchain systems without smart contract support, i.e., focusing on processing and recording financial transactions, such as BitCoin [10] and Litecoin, the state is a global ledger containing all users' currency values in either UTXO model or account model, and therefore the state transition performed by miners is transferring currency values from one user to another. State and transactions are public visible, which brings the privacy need: hide currency values and transaction routes. Later, Sasson et al. proposed Zerocash [9], a privacy-preserving cryptocurrency protocol that enable users to convert their public-visible balances into *coins* which support confidential transactions. Zerocash is based on non-interactive zero-knowledge proofs (NIZK), which are discussed in Section 2.4 and have a significant impact on subsequent privacy-preserving systems.

To support extensive applications based on blockchain technology, smart contracts were proposed by Ethereum [11]. A smart contract is a user-defined program deployed on chain, consisting of program codes and a contract state. Miners execute the codes when receiving a transaction that contains parameters to update the contract state. Nevertheless, the task of implementing a smart contract scheme that ensures privacy is significantly more challenging. Migrating Zerocash to a smart contract blockchain system is not a straightforward task. The operations in smart contracts are more intricate compared to simple integer calculations. In order to preserve privacy in smart contracts, various schemes have been proposed. These schemes rely on trusted hardware or NIZK proofs with other auxiliary cryptographic techniques. Before delving into the details of these schemes, it is important to establish a model for privacy-preserving smart contracts, providing readers with a comprehensive understanding.

2.2. Privacy-preserving smart contract: The system model

Despite the fact that the majority of privacy-preserving smart contract (PPSC) schemes rely on Zerocash's coin model, we adopt Kachina's definition [12] in order to modelize more PPSC schemes, specifically a proportion of TEE-based ones. In a smart contract instance deployed on chain, the contract state consists of a public state σ, and, for each user u, a private state ρ_u. The public state, along with either the hash or the ciphertext of each user's private state, is stored on the blockchain. Users maintain their respective private states off chain if the hash is stored on chain.

We classify state transitions as single-party transitions and multi-party transitions. In a single-party case, either a user u or a TEE executor evaluates the transition function Δ against the current state pair (σ, ρ_u), resulting in a new state pair (σ', ρ'_u) with a proof π. Upon receiving the proof, miners updates the public state σ and the hash/ciphertext of user u's private state ρ_u. Note that other users' private states cannot be updated in a single-party transition. On the other hand, the transition function Δ in a multi-party transition updates multiple users' state pair with a single proof π. However, implementing multi-party transition in crypto-based schemes is a complex task, resulting in a lack of efficient implementation. A proportion of crypto-based schemes have no support for multi-party transitions, which limits the applications. We define a privacy-preserving smart contract scheme as Definition 2.1.

Definition 2.1. A privacy-preserving smart contract (PPSC) scheme is a tuple of PPT algorithms (Setup, CreateAccount, Transit, UpdateState) defined as follows:

(1) Setup: Takes as input a security parameter λ and outputs system public parameter pp.
(2) CreateAccount: Takes as input pp. Generates and outputs a key pair $(\mathsf{sk}_u, \mathsf{pk}_u)$ and address addr_u. Executed by user u.
(3) Transit: Takes as input pp, transaction tx (containing addresses of calling users $u_1, u_2, \ldots, u_n$, transition function Δ, public parameter x_{pub} and private parameter x_{priv}), state pair (σ, ρ_{u_i}) for each user u_i. Outputs a proof π and new state pair (σ', ρ'_{u_i}) for each user u_i. Execute by a transition executor, which can be either a secure environment established by users $u_1, u_2, \ldots, u_n$ or a TEE node, depending on the specific PPSC scheme. Proper mechanisms are designed to restrict the transition executor from leaking private data or tampering the result.
(4) UpdateState: Takes as input pp, transaction tx, new public state σ', new private state hash/ciphertext h_i for each user u_i, and the proof π. Output 1 if the new state is accepted on chain, and 0 otherwise. Executed by miners.

2.3. A strawman example: PPSC-based silent auction

To better illustrate a privacy-preserving smart contract scheme, we present a strawman example of a silent auction. A silent auction is a type of auction where participants place bids on items, but their bids are kept secret. Unlike a traditional auction where bids are made openly and publicly, in a silent auction, bids are written on a piece of paper which is not public visible. At the end of the auction, the highest listed bidder wins the item. To implement this example using PPSC, the public contract state σ contains the addresses of each participant. Each private state ρ_u contains the bid of participant u. The single-party state transition Δ_{bid} takes the participant's address u as the public parameter and also takes the bid bid as the private parameter. A transition executor first checks whether the participant has not written the bid. Then, he/she update the public contract state σ and the participant's private state rho_u with a proof π, which are then broadcast to miners who will eventually validate the proof and update the contract state. At the end of the silent auction, the transaction executor execute a multi-party transition Δ_{reveal} to reveals the result by writing the highest bidder with his/her bid to public state σ.

Implementing a PPSC scheme that can effectively support the provided example presents various challenges. In the case of a crypto-based PPSC scheme, the participant u typically acts as the transaction executor. It is crucial to ensure that the executor only has access to his/her own bid, while preventing any unauthorized access to other participants' bids through a well-designed algorithm. Moreover, to support determining the highest bidder at the end of the silent auction, i.e., the multi-party transition Δ_{reveal}, one option is to trust the transaction executor for not leaking information and grant him/her access to all private data (e.g., Hawk [13]). Alternatively, cryptographic techniques such as MPC (e.g., zkHawk [14]) and FHE (e.g., Pesca [15]) can be utilized to compute the result without revealing the plaintext. Nonetheless, the efficiency of these cryptographic techniques is closely tied to the specific computation process. In some cases, utilizing these techniques may lead to inefficiencies in CPU time and memory usage. There may even be situations where implementing such a scheme becomes impractical or unfeasible. Consequently, developing a crypto-based PPSC scheme that can effectively support a wide range of computational tasks requires significant effort and presents numerous challenges. On the other hand, TEE-based PPSC schemes also face challenges. In this case, the participant with a TEE-capable machine acts as the transaction executor. However, there is a risk of side-channel attacks, where the executor could potentially obtain access to other participants' bids and exploit this information to manipulate the silent auction by increasing their own bid dishonestly.

2.4. Non-interactive zero-knowledge proof

Zero-knowledge proof is a cryptographic technique used to allow one party, called a prover, to prove the validity of a statement about public input $\mathbf{x}$ and private input $\mathbf{w}$ that $f(\mathbf{x}; \mathbf{w}) = \text{true}$ to a verifier, without revealing any knowledge or information about the secret input $\mathbf{w}$. Non-interactive zero-knowledge proof (NIZK) is a zero-knowledge proof scheme where multiple rounds of communication are not involved. A prover can construct the proof without communicating with verifiers. Moreover, this means an NIZK proof can be publicly verified, and therefore is widely adopted in blockchain schemes where miners are usually verifiers.

Zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) are highly popular NIZK constructions, widely utilized in various applications, supporting any statement that can be expressed as an arithmetic circuit. An arithmetic circuit is consist of wires and gates, where a wire contains a field element in $\mathbb{F}_p$, and a gate connects multiple wires to represent an addition constraint $a + b = c$ or a multiplication constraint $a \cdot b = c$. Each gate corresponds to an element in $\mathbf{x}$ or $\mathbf{w}$. The proof can only be successfully generated when the prover fills gates with values such that all constraints are met. Although zk-SNARKs are expressive and support various computation task, they take heavy resources. For example, all inputs in $\mathbf{x}$ or $\mathbf{w}$ are immutable, which means it is quite costly in both CPU time and RAM consumption to support conditional or loop structure compared with a traditional computer.

A trusted setup is needed in most zero-knowledge proof schemes in order to determine the initial parameters and conditions under which the proof will be conducted. This allows the verifiers involved in the proof to trust the integrity and validity of the proof itself. Some NIZK schemes, such as [16–18], do not require a trusted setup, while some schemes such as [19–22], only require a universal setup for once (universal SRS). However, these schemes are not efficient compared with other NIZK schemes which require a trusted setup for each circuit (circuit-specific SRS), such as [23–25]. This means, every time implementing the state transition Δ in a smart contract, it requires a trusted setup procedure.

2.5. Secure multi-party computation

Secure multi-party computation (MPC), is a cryptographic technique that allows multiple parties to jointly compute a function $y_1, y_2, y_3, \ldots, y_m = f(x_1, x_2, x_3, \ldots, x_m)$ over their inputs x_i owned by party i, without revealing their individual inputs to each other. MPC typically requires communication between the participating parties. In order to jointly compute the function, the parties need to interact and exchange messages during the computation process. Similar to zk-SNARKs, circuits are fundamental building blocks used to represent the calculation f. Both arithmetic and boolean circuits are widely adopted in MPC. Public auditable multi-party computation (PA-MPC) [26] allowing multiple parties jointly evaluate the function and provide a proof to convince verifiers the correctness of the computed result without requiring any knowledge of the participants' private inputs. It requires additional computational overhead and communication complexity.

2.6. Homomorphic encryption

A homomorphic encryption scheme is an encryption algorithm such that $\mathsf{Enc}_k(m_1) \circ \mathsf{Enc}_k(m_1) = \mathsf{Enc}_k(m_1 \circ m_2)$ holds for every pair of m_1 and m_2, where $\circ$ stands for one or two operations on ciphertext. For partial homomorphic encryption (PHE) schemes, the additively homomorphic property $\mathsf{Enc}_k(m_1) + \mathsf{Enc}_k(m_1) = \mathsf{Enc}_k(m_1 + m_2)$ holds. For fully homomorphic encryption (FHE) scheme, apart from the additively homomorphic property in PHE, the multiplicative homomorphic property $\mathsf{Enc}_k(m_1) \cdot \mathsf{Enc}_k(m_1) = \mathsf{Enc}_k(m_1 \cdot m_2)$ also holds. However, FHE is inefficient in practice. Similar to MPC, arithmetic and boolean circuits are utilized to represent the computation function.

2.7. Trusted execution environment

A trusted execution environment (TEE) is a secure and isolated operating environment that is separate from the main operating system on a computing device. It ensures that critical processes and sensitive data are protected from unauthorized access and tampering. One of the most important feature in TEE is called remote attestation. Remote attestation refers to the process of ensuring the integrity and authenticity of the TEE and the applications running within it. The remote attestation process plays a crucial role in establishing trust in TEE environments, ensuring that the TEE is secure and has not been tampered with, by providing a proof, which is also known as a *quote*.

Intel Software Guard Extensions, or Intel SGX, is the most widely deployed TEE implementation. It allows developers to create secure enclaves in which selected portions of their code and data can be executed in a protected and isolated environment. This enclave is isolated from the regular OS, providing a trusted and controlled execution environment where certain functions and applications can run securely. Note that, the remote attestation service in Intel SGX requires an Internet connection to Intel's online provisioning servers. An additional aspect that requires attention is that Intel SGX has a limited capacity of protected memory, 128 MB, until they recently lifted the restriction.[2] A proportion of PPSC-based TEE schemes [27,28] suffer from the memory restriction.

[2] Intel recently increased the maximum capacity of protected memory (Processor Reserved Memory Range Registers, PRMRR) from 128 MB to 512 GB per processor when releasing 3rd Gen Intel Xeon processors. https://www.intel.com/content/www/us/en/support/articles/000089550/software/intel-security-products.html

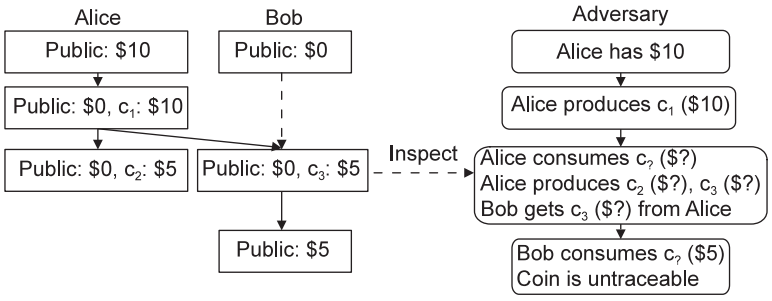

Fig. 1. Example on Zerocash's decentralized anonymous payment scheme.

3. Crypto-based PPSC

In this section, we present a comprehensive overview of crypto-based PPSC schemes. We begin by introducing Zerocash and its DAP scheme, which is a fundamental concept. Next, we describe the general idea to construct PPSC schemes from DAP scheme. Then, we summarize different approach to construct a PPSC scheme that supports multi-party state transition, which is remaining a challenge. Next, we outline PPSC schemes that focus on functional privacy. In conclusion, we provide a concise summary of crypto-based PPSC schemes.

3.1. Decentralized anonymous payment

Zerocash [9] is a decentralized digital currency that provides enhanced privacy and anonymity for its users. They proposed *decentralized anonymous payment (DAP) scheme*. The majority of crypto-based PPSC schemes are based on DAP scheme or its variants. We brief the most significant concept, a *coin*.

Coins. A coin contains verified private data. Each data in a coin can only be used once, by a procedure that consumes old coins and produces new coins. NIZK is applied in these procedures to verify to miners that the private data in coins meets the data restrictions and that these coins are validly consumed and produced. We give the definition in Definition 3.1.

Definition 3.1. A coin is a data object c to which the following attributes are assigned:

(1) A *coin value* $v(c)$. The private data, which is verified when producing the coin. Kept private; can also be temporarily public visible when a coin is produced without old coins.
(2) A *coin serial number* $\mathsf{sn}(c)$. A unique string to prevent double spending. Revealed after the coin is consumed.
(3) A *coin address*. $\mathsf{addr}_{\mathsf{pk}}(c)$. Identifies the owner. Always public visible.
(4) A *coin commitment*. $\mathsf{cm}(c)$. Hash of the coin. When the coin is produced, the commitment is public visible, so that it can be saved on the chain's Merkle tree. However, when the coin is consumed, the commitment is no longer public visible. The owner proves the serial number corresponds to a commitment on chain, without explicitly specifying the exact commitment.

We illustrate an example in Fig. 1 to demonstrate how coins provide privacy and anonymity for users. Initially, Alice holds \$10 public balance. In the first step, she first convert her balance to a coin c_1, publishing $\mathsf{cm}(c_1)$ with $v(c_1) = 10$. Miners add the commitment $\mathsf{cm}(c_1)$ to the Merkle tree, updating the Merkle root. Note that, in this step, everyone knows Alice owns a coin with plaintext commitment and value. However, this does not matter, as when the coin is consumed, both the commitment and value are kept hidden, and therefore no one would be able to trace this coin anymore. Next, Alice secretly transfers \$5 to Bob, by consuming her coin c_1 and producing two coins, c_2 to Alice and c_3 to Bob. She provides an NIZK proof such that $v(c_1) = v(c_2) + v(c_3)$, i.e., the sum of coin values are not changed, without exposing specific values. In this NIZK proof, it also exposes the old coin's serial number $\mathsf{sn}(c_1)$, as well as new coins' commitments $\mathsf{cm}(c_2)$, $\mathsf{cm}(c_3)$. Since the serial number has never be used before, after validating the NIZK proof, miners accept this transaction, adding these two commitments to the Merkle tree. Note that, in this step, it is publicly known that Alice has consumed a coin, but since she owns a lot of coins, no one knows which specific coin is consumed in this transaction, i.e., the coin c_1 is untraceable. Lastly, Bob withdraws his coin by consuming c_3 without producing new coins, with an NIZK proof to reveal $\mathsf{sn}(c_3)$ while exposing $v(c_3)$ as a public output. Financial censorship against c_3 would be almost impossible.

3.2. From Zerocash to PPSC: Single-party case

Building a PPSC scheme from Zerocash's DAP scheme requires extending the coin value from integers to arbitrary private data. Also, instead of simply proving the sum of the numbers remains unchanged, an NIZK proof validates how private data is correctly computed during producing new coins. Fig. 2 demonstrates the zero-knowledge proof extension to build a PPSC scheme. In Fig. 2(a), the on-chain storage for coin commitments remains unchanged, which continue to ensure uniqueness of commitments and prevent double spending. The procedure for producing a coin, as shown in Fig. 2(b), replaces the currency value with arbitrary private data. Afterwards, during the process of consuming old coins and generating new ones, the calculations no longer involve solely integers; instead, a function is employed which generates outputs based on inputs. The private inputs are derived from old coins, while the newly generated private outputs become the new coins. To verify the accuracy of the computation, as well as the proper generation of commitments and serial numbers, a corresponding non-interactive zero-knowledge (NIZK) proof is utilized, as shown in Fig. 2(c).

Challenges are involved after extending Zerocash to PPSC. NIZK schemes, especially zk-SNARKs, support validating a relationship that can be expressed as an arithmetic circuit, as previously introduced in Section 2.4. Here brings the first challenge. Real-world smart contracts are implemented in high-level expressive languages like Solidity. Developers need to have a comprehensive knowledge of cryptographic tools, particularly implementation details in zero-knowledge proofs, which can be a hurdle or obstacle. To migrate this issue, PPSC schemes provide their compilers to migrate regular smart contract applications into PPSC ones, such as Hawk [13], zkay [29], ZeeStar [30], and Zapper [31]. Specifically, in zkay, the compiler is utilized to manage secret variables in smart contract. A smart contract usually contains structures such as unbounded state and loops, which

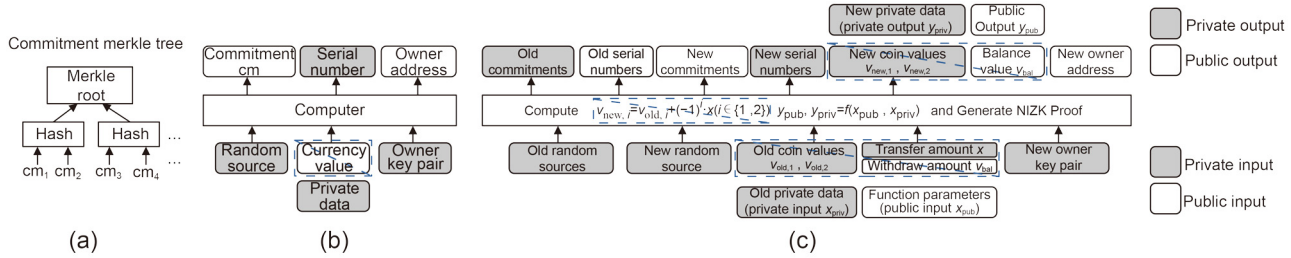

Fig. 2. Extend Zerocash's DAP scheme to privacy-preserving smart contract. (a) On-chain storage. (b) Produce a coin. (c) Consume and produce coins.

requires unrolling technique, significantly increasing the circuit size. In this scheme, each NIZK proof is applied to update a single secret variable, to overcome the inefficiency of large arithmetic circuits. Different from the original DPC scheme that user holds private data in coins, in this scheme, encrypted values are stored on chain, and users provide proofs to execute a smart contract function and update these encrypted values. This can be considered as a variant of DPC scheme.

We continue to take the strawman example in Section 2.3 where users secretly place bids on items. In this example, a bid is implemented as a coin. A user place a bid by producing a coin while providing an NIZK proof such that the bid is less or equal to the user's deposit. Different from Zerocash, although this coin is produced without consuming old coins, the coin value, i.e., user's bid, is kept secret. At the end of the silent auction, there should be another NIZK proof that consuming all coins and revealing the winner. Since multiple user's secret data is involved, we will continue the example in the Section 3.3.

3.3. Foreign data computation support: Multi-party case

The strawman example presented in Section 3.2 raises a crucial question: How can we maintain data privacy when generating an NIZK proof that involves private data from multiple users? In Zerocash, transactions are exclusively between two users. When Alice transacts money to Bob by consuming an old coin from each and producing a new coin for each, both Alice and Bob know all the private data during the transaction, i.e., the coin values of all the four coins. While this is not a problem in Zerocash due to the ability to mix coins, it becomes an issue when extending Zerocash into a general PPSC implementation. This challenge is more demanding. Therefore, not all crypto-based PPSC schemes support the multi-party case, i.e., handle foreign data computation. There are two approaches to overcome the challenge and support multi-party case, manager-based and homomorphic encryption.

3.3.1. Manager-based

To support arbitrary calculation of foreign data, Hawk [13] introduced a semi-trusted party called the *manager*. The manager is accessible for users' coin value, and is trusted not to disclose these private data. However, the manager is not trusted for not affecting the correct execution of the contract. In other words, the manager is considered as a dishonest-but-uncurious adversary. NIZK proofs are capable of preventing the manager from a dishonest calculation, as modifying the new coin's value resulting in an NIZK verification failure.

We finish the strawman example presented in Section 3.2. After the conclusion of the silent auction, the auctioneer takes on the role of the manager. Participants in the auction submit their bid values to the manager. The manager then performs calculations and generates an NIZK proof, which reveals the highest bid and its respective bidder, while ensuring the confidentiality of all other bids. This NIZK proof serves as a safeguard against any potential dishonesty or attempts by the auctioneer to favor a specific winner or manipulate bid amounts for a higher gain.

Nevertheless, there are situations where it becomes challenging to rely on a party to keep private data confidential. One such example is medical data statistics, where it is crucial that no individual party has access to personal sensitive information. Enigma [32] involves public-auditable secure multi-party computation (PA-MPC) [26] to perform confidential computation on secret data from multiple users. Similarly, in Eagle [33], PA-MPC is utilized to support MPC computation among private data holders and servers on chain. However, Ren et al. [34] pointed out that applying PA-MPC to construct multi-party state transition is still a challenge, as miners do not trust any MPC participants, and therefore the participants cannot only multi-sign messages as a proof. In other words, these works focus on performing a confidential computation and get a valid result only among these MPC participants, without convincing miners that the result is valid. Therefore, constructing a solid PPSC scheme remains a challenge even if MPC or PA-MPC is involved.

Different from these works, zkHawk [14] and V-zkHawk (also known as HawkNess) [35] aim to eliminate the need for trust in the manager by employing MPC and NIZK. The idea is to perform the whole procedure to generate NIZK proof inside MPC. However, it is important to note that these approaches are primarily theoretical, as the generation of an NIZK proof within an MPC framework is extremely inefficient.

3.3.2. Homomorphic encryption

Another approach to implement multi-party state transition is homomorphic encryption.

Zeestar [30] utilized additively homomorphic encryption to generate the NIZK proof without the need for exposing secret data to an NIZK prover. For example, an NIZK proof for a financial transaction validates $v(c_A^{(1)}) + v(c_B^{(1)}) = v(c_A^{(2)}) + v(c_B^{(2)})$ when Alice has a transaction with Bob. In Zerocash, all these four private values are exposed to both Alice and Bob. To get rid of such a disclosure, Alice encrypts $v(c_A^{(1)})$ and $v(c_A^{(2)})$ by Bob's public key pk_B. In this way, Bob is capable of convincing miners $v(c_A^{(1)}) + v(c_B^{(1)}) = v(c_A^{(2)}) + v(c_B^{(2)})$ by proving $\mathsf{Enc}_{\mathsf{pk}_B}(v(c_A^{(1)})) + \mathsf{Enc}_{\mathsf{pk}_B}(v(c_B^{(1)})) = \mathsf{Enc}_{\mathsf{pk}_B}(v(c_A^{(2)})) + \mathsf{Enc}_{\mathsf{pk}_B}(v(c_B^{(2)}))$ since additively homomorphic encryption holds. Besides, implementing additively homomorphic encryption in NIZK is efficient in practice.

Obviously, applying additively homomorphic encryption is not sufficient to support arbitrary multi-party state transition, for example, an NIZK proof that requires a multiplication of secret values from two users. For the same reason as in zkHawk [14] and V-zkHawk [35] in Section 3.3.1, implementing fully homomorphic encryption in NIZK is not practical at all. smartFHE [36] proposed a PPSC scheme with fully homomorphic encryption. Different from the majority PPSC schemes that work on DAP scheme,

i.e., produces and consuming coins by NIZK proof, in smartFHE, miners perform computation over encrypted data. In this way, NIZK proof is only needed for proving well-formedness of the private data. However, implementing multi-party state transition requires multi-key FHE, which is currently not practical.

Pesca [15] also adopted fully homomorphic encryption such that miners perform computation. However, the key difference is that all users' secret data is encrypted via a global public key, where all consensus nodes holds a Shamir secret sharing of global private key. This approach is known as distributed key generation (DKG). Consequently, there is no requirement for multi-key FHE, which means Pesca is more practical compared with smartFHE. It is important to note, however, that users must place trust in the consensus nodes' integrity and their inability to collude. In the event of collusion, the private data of all users would be vulnerable to exposure. This diverges from other crypto-based PPSC schemes that do not necessitate such an assumption. Typically, such a trust assumption is mostly suitable for adoption in TEE-based schemes, of which we will delve into the specifics in Section 4.2.

In summary, the task of building a crypto-based PPSC scheme with multi-party state transition support continues to be a demanding challenge. Previous attempts to solve this problem involved integrating with secure multi-party computation, homomorphic encryption, and making strong assumptions. Further efforts are needed to make significant progress in this area.

3.4. Function privacy

The primary focus of PPSC schemes is to ensure data privacy by safeguarding user data, where an NIZK proof directly proving the computation in private data, exposing the corresponding transition function. However, situations exist where users do not wish to expose *which* transition function they called. Bowe et al. named it *function privacy* [37]. For example, a lottery application contains two transition functions named `participate` – buying a lottery ticket, and `redeem` – collecting the price money. Evidently, a lottery winner never wishes `redeem` to be public visible, while a PPSC scheme without function privacy exposes the fact that he/she wins the lottery, although the specific price value remains confidential. This knowledge poses a danger to the winner's personal security.

Zexe achieved function privacy, where the transition function remains undisclosed in the NIZK proof [37]. Within Zexe, the NIZK proof functions as a universal function that interprets instructions derived from the transition function and the prover's private inputs. This ensures that adversaries remain unaware of the specific details of the function. Nevertheless, protecting function privacy presents new difficulties. Since transition function contains arbitrary instructions, a malicious function can produce faked coins, or faked private data. For instance, counterfeit currency in financial applications. One possible attempt in Zexe adds a new attribute to coins in Definition 3.1, secretly identifying the function that creates the coin. When consuming coins, an NIZK proof checks whether these coins are made by trusted functions or not. However, this attempt brings out the interoperability issue, as contracts only trust functions published by their own publisher.

Zexe proposed records nano-kernel (RNK) where *coins* in Definition 3.1 are extended to *records* by adding two attributes, namely *birth predicate* and *death predicate*. A predicate is a boolean function that defines a policy, which is ensured by NIZK proofs. When a record is produced, the birth predicate is satisfied. Similarly, the death predicate must hold when the record is consumed. This design allows a record to interact with other records when its predicates decide to accept. Take the conditional exchange example in Zexe. Alice withes to exchange a digital asset id_1 with id_2. She creates a record containing id_1, with a death predicate such that, any transaction consuming the record must create another record containing id_2, whose death predicate only allows Alice to cliam id_2. The example above shows how user-defined assets are achieved. More features like inter-contract calls can be realized by properly defining records and their predicates.

Xiong et al. highlighted that the original Zexe utilizes a non-universal NIZK scheme, necessitating a trusted setup for each application. To address this drawback in practical scenarios, they introduced Veri-Zexe [38], which incorporates a universal NIZK scheme that only necessitates a single trusted setup. However, employing such an NIZK scheme results in notably poorer performance. In Veri-Zexe, strategic designs are implemented to avoid compromising performance. Steffen et al. emphasized that while Zexe has several vulnerabilities and shortcomings, it was the sole PPSC scheme that successfully withstands deanonymization attacks. In order to overcome the limitations of Zexe, they introduced Zapper [31], achieving identity privacy by hiding accessed objects through their oblivious Merkle tree construction.

We provide a summarized overview of crypto-based PPSC schemes in Table 1.

4. TEE-based PPSC

This section focuses on TEE-based PPSC schemes. We begin by discussing the practical integration of TEE with the blockchain. Next, we introduce per-contract private state management and explore function privacy. Then, we delve into a scheme that emphasizes interactive multi-round computation. To conclude, we provide a summary of TEE-based PPSC schemes.

4.1. Integrate TEE with chains

TEE has significant efficiency advantage over cryptographic techniques, while brings a trust assumption for hardware manufacturers. TEE integration within a privacy-focused smart contract structure was highlighted by Hawk [13] as a viable solution. As we previously mentioned in Section 3.3, a tough and unresolved challenge in achieving practical crypto-based PPSC is efficiently enabling multi-party state transitions without requiring additional trust assumptions, while keeping computation costs reasonable. To bypass this challenge, Hawk introduced a semi-trusted party called the *manager*, assuming the manager will never leak users' private data. However, this assumption of trustworthiness is not feasible. Therefore, Hawk suggested that the manager role could be filled by employing trusted computing hardware, specifically a TEE-based manager. By integrating TEE implementations like Intel SGX, the off-chain computation can be efficiently performed within a confidential SGX enclave, remaining hidden from any untrusted software or users, thereby preventing data disclosure.

Nevertheless, as Hawk primarily focuses on cryptography, they have not provided thorough details for integrating TEE. Brandenburger et al. [27] have addressed this aspect in their Hyperledger Fabric extension: In Intel SGX, the *remote attestation* procedure generates a quote as proof of the integrity and authenticity of an SGX enclave and its current state. Verifiers can use this quote to ensure the trustworthiness of the enclave. The quote is then sent to the Intel Attestation Service for verification using the *enhanced privacy ID (EPID)* group signature scheme.

Remote attestation serves two purposes in a TEE-enabled PPSC. Firstly, it facilitates the secure transmission of private data from users/key holders to the enclave. This is achieved by establishing trust between each user/key holder and the enclave through remote attestation. Once trust is established, a secure communication channel can be set up as a TLS-like connection,

Table 1
Overview of Crypto-based PPSC schemes.

References	NIZK implementation	Deployable on EVM chains	Multi-party transition	Trust assumptions[a]	Auxiliary cryptographic techniques	Function privacy	Identity privacy
Hawk [13]	BCTV14 [23]	Modified	Yes	Semi-trusted manager[b]	No	No	No
ZkHawk [14]	No	No	Yes[c]	–	MPC and[d]	No	No
V-zkHawk [35]	No	No	Yes[c]	–	MPC and[e]	No	No
Zexe [37]	GM17 [25]	No	No	–	No	Yes	No
Veri-Zexe [38]	Plonk [21]	No	No	Universal NIZK setup	No	Yes	No
Zapper [31]	GM17 [25]	No	No	Single NIZK setup[f]	No	No[g]	Yes
Zkay [29]	GM17 [25]	Yes	No	–	No	No	No
ZeeStar [30]	Groth16 [24]	Yes	Addition only	–	PHE	No	No
SmartFHE [36]	DPLS19 [39] and Bulletproofs [40]	No	Yes[c]	–	FHE	No	No
Pesca [15]	No	No	Yes	Byzantine assumptions[h]	FHE	No	No
Kachina [12]	No[i]	No	No	–	No	No	No

[a] All schemes need NIZK setup as a trust assumption. If omitted in table, non-universal NIZK setup is needed per application.
[b] Hawk trusts managers for privacy (but not for correctness). If the manager is TEE-implemented, Hawk can also be considered TEE-based.
[c] Yes on theory. Inefficient to be implemented.
[d] Homomorphic commitments.
[e] Key-homomorphic signatures.
[f] Application-agnostic circuit is utilized in Zapper. Only a single trusted setup is needed without the need of introducing a costly universal NIZK scheme.
[g] Zapper claimed that function privacy can be achieved by providing Zasm instructions as private inputs to an NIZK proof. Left as future work.
[h] Not to be confused with the PBFT consensus algorithm in blockchain. Byzantine assumptions in Pesca means that if more than t consensus nodes collude, they would have access to all secret data.
[i] As a theoretical work, Kachina does not have an NIZK implementation. However, they have considered Sonic [20] as a potential choice.

where users accordingly sends their private inputs and receives the new private states to/from the enclave. Secondly, when the enclave completes its computation, a quote is provided to convince miners that a multi-party state transition has been executed correctly. This is an alternative to using NIZK proofs in crypto-based PPSC schemes. Upon confirmation of the validity of the proof, miners accept the multi-party state transition, updating the public state and the hash/ciphertext of their private states. Apart from Hawk, Cloak [41] also works in this way. Fig. 3 demonstrate the multi-party state transition in TEE-based PPSC schemes.

4.2. Private state management

In TEE-based PPSC schemes, the symmetric encryption key for private states is established on a per-contract basis rather than per-user. As a comparison, Zerocash's DAP scheme, along with its variations, is widely utilized in crypto-based PPSC schemes. In these schemes, users either store their private data on their personal devices or retain the symmetric encryption key while storing the ciphertext on the blockchain. As a result, it brings a limitation that all private states must belongs to specific users. In contrast, TEE enclaves play a critical role as trusted parties which has resulted in the implementation of a simplified approach called distributed key generation (DKG) in TEE-based PPSC schemes. In this setup, per-contract key is applied, thereby supporting maintaining user-agnostic data in private.

Ekiden [42] is a typical TEE-based PPSC scheme that utilizes DKG for key management. In this scheme, a contract possesses both a long-term key and ephemeral short-term keys derived from the long-term key, providing forward secrecy.

(1) A key management committee is formed by a quorum of TEE nodes. When a new contract c is deployed, the committee employs the DKG protocol to generate a long-term key k_c, with each committee member storing a secret sharing of k_c. It is important to note that such a design greatly minimizing the potential for collusion and leakage of the long-term key k_c, even if a small fraction of TEEs gets corrupted via, e.g., a side-channel attack.
(2) Subsequently, when a TEE compute node requests the symmetric encryption key for accessing or creating private state data at epoch t, each committee member verifies the integrity of the requesting TEE node through remote attestation. This is followed by the execution of a distributed pseudo-random function to compute the corresponding short-term key $k_{c,t}$. The distributed nature of the pseudo-random function guarantees that no committee member possesses the complete short-term key. Instead, each member obtains a secret sharing of the calculated short-term key, which is then securely transmitted to the requesting TEE node via a communication channel. Once a sufficient number of secret sharings are received, the requesting TEE node can recover the short-term key $k_{c,t}$.
(3) By having a short-term key during each required epoch, the TEE node has the capability to both read and write the private state of the contract, enabling it to complete the state transition. A state updating message containing a encrypted state as well as a remote attestation quote is sent to miners. Upon receiving this message, miners verify the quote's validity. Once the validation is complete, the miners update the ciphertext of the private state of contract c on the blockchain.

It is worth mentioning that no individual party possesses direct access to the private state, ensuring its security and integrity. Only public outputs from a smart contract function is publicly visible, executed by such a TEE node, which is by design.

Some other TEE-based PPSC schemes have similar design to maintain a per-contract key. (1) In Phala [43], Gatekeepers are introduced to manage the root key, who work similarly with Ekiden's committee members. Periodical key rotation is proposed for re-elect Gatekeepers and updating the root key, to achieve forward secrecy. Though, in this scheme, the root key is not produced by DKG, but they have left applying a DKG scheme for key generation as future works. (2) In PDOs [44], when an enclave executes a state transition function, it obtains the relevant per-contract symmetric encryption key through the provisioning

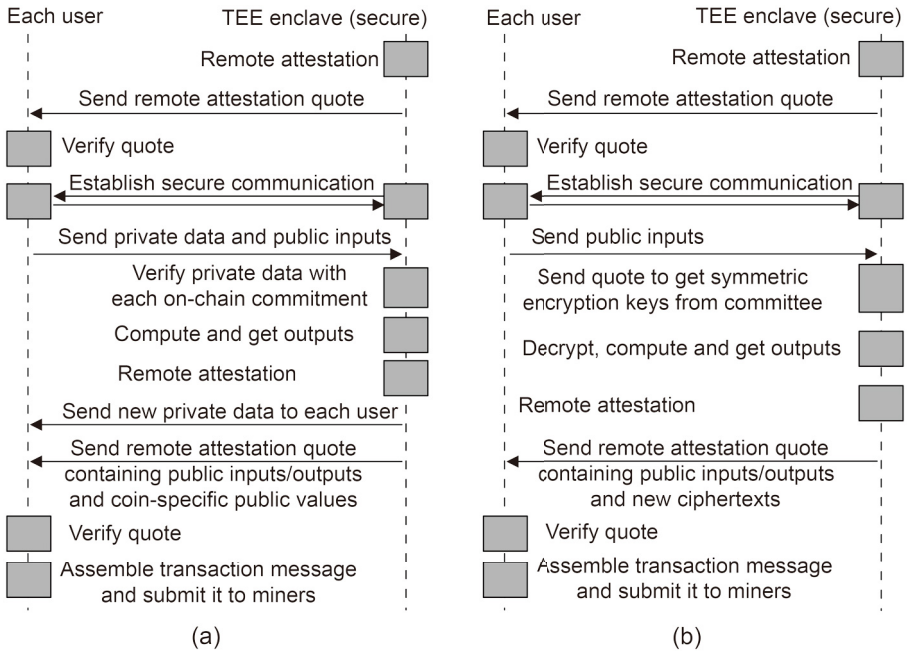

Fig. 3. Multi-party state transition in TEE-based PPSC schemes. (a) Apply TEE in DAP scheme. (b) Apply TEE in DKG scheme.

service, which derives the key from provisioned secrets. However, the authors have not provided comprehensive information regarding the specific mechanisms employed by the provisioning service for securely generating and storing these provisioned secrets. They assume that the service is owned by one or more potentially trustworthy organizations. (3) CCF [45] (formerly named CoCo) leverages trust in a consortium of governing members, who runs a replication protocol that supports both crash-fault tolerant (CFT) configuration and Byzantine-fault (BFT) tolerant configuration. In the protocol, master secrets are periodically generated. Some members are selected for key recovery, holding a secret sharing each, in case that protocol aborts because more node outages than its replication protocol can withstand. Similarly with Ekiden, pseudo-random function is utilized to compute the corresponding short-term key based on the corresponding master secret.

4.3. Function privacy

Two TEE schemes, namely PDOs [44] and ShadowEth [46], aim to achieve function privacy by ensuring that the smart contract codes are not publicly visible. The primary focus of PDOs [44] is to enable a group of untrusted parties to access and modify private data using policies. These policies are implemented through a smart contract c, the codes of which are shared among the parties. It is important to note that the codes do not necessarily need to be made public outside the group, ensuring confidentiality. This is achieved by securely loading and executing the smart contract codes in an SGX enclave, which protects against disclosure. However, it is crucial to emphasize that since the private data policies are agreed by this group, the private data only hold significance within the specific smart contract that is exclusively used by this group of parties. Individuals who do not belong to this group do not trust the data value. ShadowEth [46] ensure the confidentiality of smart contract codes by implementing TEE-DS, a decentralized secure storage system. This storage solution operates through a group of TEE nodes and maintains consistency with the help of a Paxos-like consensus algorithm. By utilizing TEE-DS, ShadowEth offers a reliable and secure environment for storing both codes and private states. In the end, we emphasize that function privacy cannot be solely guaranteed by maintaining the confidentiality of smart contract codes. We will engage in a discussion in Section 6.

4.4. Interactive multi-round computation

FastKitten [47] achieves a practical off-chain confidential smart contract execution over Bitcoin. In this scheme, the authors considered a special computation model as "multi-round contracts", which is different from the rest schemes. FastKitten is dedicated to optimizing the off-chain execution of multi-round contracts involving a defined set of parties. It specifically deals with contracts involving a group of n parties, who contribute initial inputs, and the contract proceeds through m subsequent rounds. In each round, the contract receives additional inputs from the same set of n parties and generates an output. Moreover, the contract in FastKitten handle coins[3] by receiving coins from each party at the initial round and transferring coins back to the parties according to the results of the contract execution. In their design, an operator capable of running a TEE is responsible for executing the contract and interacting with the involved parties.

However, this setup poses certain challenges. On the one hand, while TEE enclaves are generally trusted, it is essential to consider that they are executed within a TEE host (the operator) that could potentially act maliciously. This creates the risk of the operator deliberately disrupting the enclave by dropping network packages or abruptly terminating it, resulting in a Denial-of-Service (DOS) attack. On the other hand, a party providing inputs may also be malicious, as they could submit invalid inputs or even fail to provide necessary inputs.

Therefore, they have implemented a mechanism called challenge-response to address any malicious behaviors. In the event of a misbehavior, a penalty transaction is initiated to compensate the honest participants and penalize the unethical party or operator. Let us say a party maliciously provides incorrect inputs, in such cases, the operator publishes a challenge

[3] Not to be confused with *coins* in Zerocash's DAP scheme. Here, coins literally refers to currency.

Table 2
Overview of TEE-based PPSC schemes.

References	TEE implementation	TEE host	Blockchain requirements	Contract expressiveness	Root key managed by	Forward secrecy	Public verifiable outputs[a]	Function privacy
Hawk [13]	No	Managers	Modified EVM	NIZK circuits	User[b]	–[b]	Yes (NIZK)	No
Ekiden [42]	SGX	Executors	[c]	EVM contracts and SGX apps	Committee[d]	Yes	Yes	No
CCF [45]	SGX	Distributed Storage	Ledger[e]	EVM/Lua contracts and SGX apps	Storage	Yes	Yes	No
Phala [43]	SGX	Executors	Polkadot Parachains [c]	Phat Contract [48]	Committee[d]	Yes	Yes	No
ShadowEth [46]	SGX	Distributed Storage	[c]	EVM contracts	Storage	No	Yes	Yes
PDOs [44]	SGX	Executors	[c]	Interpretable codes	Provisioning Services	No	No	Yes
FastKitten [47]	SGX	Executors	Time-locked TX[f]	SGX apps[g]	No[h]	–[h]	No[h]	No
Cloak [41]	SGX	Executors	[c]	SGX apps	User[b]	–[b]	Yes	No
TZ4Fabric [49]	TrustZone[i]	Executors	Hyperledger Fabric	TrustZone apps	Unstated	Unstated	Yes	No

[a] Certain PPSC schemes prioritize convincing participants involved in a multi-party off-chain smart contract about the correctness of outputs, rather than aiming for consensus across the entire blockchain. In these schemes, the contract execution is not publicly verifiable, and therefore neither public nor private outputs are public can be recorded on chain as they are from a reliable data source.
[b] Hawk extends the DAP scheme from Zerocash, where users keep their private data off the chain. Similarly in Cloak, although not explicitly stated by authors, on-chain commitments of private data are generated and consumed, functioning similarly to the DAP scheme.
[c] Miners/consensus nodes should be able to verification remote attestation quotes. The requirement can be satisfied as smart contract support.
[d] Comprised of selected randomly executors.
[e] In CCF, blockchain acts as a dummy append-only ledger. They state "every one" is able to verify proofs.
[f] Blockchains supporting transactions carrying arbitrary data and can be timelocked, e.g., Bitcoin, Lightcoin, and smart contract enabled blockchains.
[g] With restriction: it must be possible to estimate an upper bound on the number of rounds and the maximum run time of any round.
[h] FastKitten focuses on interactive multi-round computation where all inputs are provided by users. No states are maintained.
[i] ARM TrustZone does not natively support remote attestation. TZ4Fabric provide possible workarounds.

transaction on the blockchain. If the party fails to respond to the challenge, the operator can use this as evidence of their malicious intent. Conversely, if the operator is the one being malicious, they may intentionally drop the message to falsely accuse a participant of wrongdoing. In order to maintain their integrity, the party can post a response transaction once the malicious operator has published the challenge transaction.

We have summarized the overview of TEE-based PPSC schemes as Table 2.

5. Challenges

In this section, we highlight the difficulties encountered when designing privacy-preserving smart contract schemes.

NIZK expressiveness and efficiency. Zerocash's DAP scheme is a widely utilized design for on-chain management of private states, typically demanding NIZK proofs. Nonetheless, it should be noted that NIZK implementations have a reduced level of expressiveness when compared to regular machines. For instance, zk-SNARKs provide support for proving statements represented as arithmetic circuits. However, implementing complex structures such as conditional/loop structures and memory addressing – commonly used in traditional computer structures – requires advanced skills. Additionally, NIZK implementations are generally inefficient, especially when a complex structure above is implemented. To effectively express privacy-preserving smart contracts in NIZK and minimize resource consumption, compiler toolchains and strategic workarounds are necessary.

Efficient multi-party transitions without disclosure. Generating an NIZK proof requires the prover's knowledge of all confidential data. This brings a conflict when a privacy-preserving smart contract transition involves multiple users, as each user possesses a portion of the data, and the prover must not acquire access to all private data. Consequently, discovering an efficient and privacy-preserving resolution to generate the necessary NIZK proofs remains a formidable, unresolved challenge.

Confidential data exchange across function-privacy contracts. There are situations where maintaining the confidentiality of a smart contract function becomes necessary. However, concealing the function poses a challenge in terms of trusting the private data generated by an unidentified smart contract. This is because a malicious function can generate false private data in violation of the protocol. Therefore, difficulties exist to develop a reliable model that ensures both function privacy and smart contract interoperability.

Forward-secrecy key management. TEE implementations may be vulnerable to data leakage, particularly through side-channel attacks. If an attacker manages to obtain the symmetric encryption key, they can decrypt all sensitive data, which poses a significant risk. To address this issue, it is crucial to implement a robust key derivation or key rotation mechanism to ensure forward secrecy in key management.

6. Discussion

In this section, we present a discussion on enhancing privacy-preserving smart contract schemes.

NIZK outsourcing. An NIZK implementation is generally inefficient, despite many efforts have been made to reduce the high resource requirement. For instance, ZeeStar stated that generating a Groth16 [24] zk-SNARK for Paillier encryption [50] with 2048-bit keys requires over 256 GB of RAM. Obviously, this is an obstacle for making crypto-based PPSC schemes practical, as a regular home PC usually have 16 GB of RAM, let alone cellphones. Servers, on the other hand, is capable of generating such an NIZK proof. However, it is a challenge for users to delegate the task to untrusted servers since all private data must be accessible in proof generation. One possible solution is to incorporate TEE in NIZK proof generation. While TEE may not be considered a reliable trust assumption in a crypto-based PPSC scheme, the system security remains unaffected if a user delegates proof generation to a TEE-capable server — in the worst case scenario, only this user's private data may be leaked. Apart from TEE, another attempt to outsource NIZK proof generation, zkSaaS [51], was proposed recently.

Malicious functions on TEE-based PPSC. Contrary to Zexe [37], TEE-based PPSC schemes lack an effective design for addressing malicious smart contracts, particularly when it comes to achieving function privacy. PPSC schemes with smart contract interoperability are susceptible to attacks or disruptions by malicious users who can create functions that compromise the functions and data of other users. For instance, an attacker can publish a smart contract that simply exposes a user's confidential data. Users need to trust a smart contract before using it with their private data, but establishing this trust, especially in the context of function privacy, can be challenging. Hence, the adoption of a TEE-based PPSC scheme for Zexe could provide a solution to protect users' private data against malicious functions.

Decentralized TEE attestation service. Intel SGX is the most practical and most widely deployed TEE schemes, where most TEE-based PPSC schemes base on. However, it is important to note that the remote attestation service of Intel SGX requires an Internet connection to Intel's online provisioning servers. This reliance on Intel's attestation service can be seen as a barrier since verifying a quote should be similar to verifying signatures. Furthermore, this online requirement impedes the ability of blockchain miners to accurately replay and validate PPSC transactions. Chen et al. proposed an alternative solution called OPERA [52]. This approach aims to eliminate Intel's single-point-of-verification, while still conducting attestation. By incorporating OPERA into TEE-based PPSC schemes, significant improvements in latency should be achieved, allowing for more reliable replay and validation of transactions.

Blockchain with heterogeneous TEE implementations. Although some TEE-based PPSC schemes [41,42,47] claim to have a TEE-agnostic design, they still rely on Intel SGX for its enclaves and remote attestation feature. Ideally, blockchain should be decentralized. It would become more reliable to incorporate multiple TEE implementations with chain. However, these alternative TEE implementations differ from Intel SGX. For instance, ARM TrustZone does not support remote attestation, leading TZ4Fabric [49] to develop workarounds for achieving PPSC. Another example is AMD SEV [53], which does not have the concept of an enclave. Other popular TEE implementations includes Keystone [54] and ARM CCV [55]. Therefore, incorporating heterogeneous TEE implementations into blockchain systems poses challenges. Recently, Zhao et al. introduced vSGX [56], a system that supports enclave deployment on AMD SEV. This innovation represents a crucial step towards decentralizing trust in hardware manufacturers.

7. Concluding remarks

In conclusion, our study delves into a comprehensive system overview of privacy-preserving smart contract (PPSC) schemes. We have classified these schemes into two categories, namely crypto-based PPSC schemes and TEE-based PPSC schemes, based on the techniques employed. While each scheme exhibits distinct designs, they encounter both common obstacles and individual challenges. Through our investigation, we have presented a comprehensive summary and comparison of the diverse approaches employed in constructing PPSC schemes. Additionally, we have highlighted on the associated challenges and discussed on potential avenues for augmenting these schemes further.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This study was partially supported by the National Key R&D Program of China (2022YFB4501000), the National Natural Science Foundation of China (62232010 and 62302266), Shandong Science Fund for Excellent Young Scholars, China (2023HWYQ-008), Shandong Science Fund for Key Fundamental Research Project, China (ZR2022ZD02), and the Fundamental Research Funds for the Central Universities, China.

References

[1] A. Al Omar, A.K. Jamil, A. Khandakar, A.R. Uzzal, R. Bosri, N. Mansoor, M.S. Rahman, A transparent and privacy-preserving healthcare platform with novel smart contract for smart cities, IEEE Access 9 (2021) 90738–90749.
[2] H.L. Pham, T.H. Tran, Y. Nakashima, A secure remote healthcare system for hospital using blockchain smart contract, in: 2018 IEEE Globecom Workshops, GC Wkshps, IEEE, 2018, pp. 1–6.
[3] N. Hynes, D. Dao, D. Yan, R. Cheng, D. Song, A demonstration of sterling: a privacy-preserving data marketplace, Proc. VLDB Endow. 11 (12) (2018) 2086–2089.
[4] T. Li, W. Ren, Y. Xiang, X. Zheng, T. Zhu, K.-K.R. Choo, G. Srivastava, FAPS: A fair, autonomous and privacy-preserving scheme for big data exchange based on oblivious transfer, Ether cheque and smart contracts, Inform. Sci. 544 (2021) 469–484.
[5] S. Tan, X. Wang, C. Jiang, Privacy-preserving energy scheduling for ESCOs based on energy blockchain network, Energies 12 (8) (2019) 1530.
[6] Q. Yang, H. Wang, Privacy-preserving transactive energy management for IoT-aided smart homes via blockchain, IEEE Internet Things J. 8 (14) (2021) 11463–11475.
[7] P.C.M. Arachchige, P. Bertok, I. Khalil, D. Liu, S. Camtepe, M. Atiquzzaman, A trustworthy privacy preserving framework for machine learning in industrial IoT systems, IEEE Trans. Ind. Inform. 16 (9) (2020) 6092–6102.
[8] Y. Jiang, C. Wang, Y. Wang, L. Gao, A privacy-preserving e-commerce system based on the blockchain technology, in: 2019 IEEE International Workshop on Blockchain Oriented Software Engineering, IWBOSE, IEEE, 2019, pp. 50–55.
[9] E.B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, M. Virza, Zerocash: Decentralized anonymous payments from bitcoin, in: 2014 IEEE Symposium on Security and Privacy, IEEE, 2014, pp. 459–474.
[10] S. Nakamoto, A. Bitcoin, A peer-to-peer electronic cash system, Bitcoin 4 (2) (2008) 15, https://bitcoin.org/bitcoin.pdf.
[11] G. Wood, et al., Ethereum: A secure decentralised generalised transaction ledger, Ethereum Proj. Yellow Pap. 151 (2014) (2014) 1–32.
[12] T. Kerber, A. Kiayias, M. Kohlweiss, Kachina–foundations of private smart contracts, in: 2021 IEEE 34th Computer Security Foundations Symposium, CSF, IEEE, 2021, pp. 1–16.
[13] A. Kosba, A. Miller, E. Shi, Z. Wen, C. Papamanthou, Hawk: The blockchain model of cryptography and privacy-preserving smart contracts, in: 2016 IEEE Symposium on Security and Privacy, SP, IEEE, 2016, pp. 839–858.
[14] A. Banerjee, M. Clear, H. Tewari, zkhawk: Practical private smart contracts from mpc-based hawk, in: 2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services, BRAINS, IEEE, 2021, pp. 245–248.
[15] W. Dai, PESCA: A privacy-enhancing smart-contract architecture, Cryptol. ePrint Arch. (2022).
[16] E. Ben-Sasson, I. Bentov, Y. Horesh, M. Riabzev, Scalable, transparent, and post-quantum secure computational integrity, Cryptol. ePrint Arch. (2018).
[17] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, N.P. Ward, Aurora: Transparent succinct arguments for R1CS, in: Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38, Springer, 2019, pp. 103–128.
[18] A. Chiesa, D. Ojha, N. Spooner, Fractal: Post-quantum and transparent recursive proofs from holography, in: Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39, Springer, 2020, pp. 769–793.
[19] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, I. Miers, Updatable and universal common reference strings with applications to zk-SNARKs, in: Annual International Cryptology Conference, Springer, 2018, pp. 698–728.
[20] M. Maller, S. Bowe, M. Kohlweiss, S. Meiklejohn, Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 2111–2128.
[21] A. Gabizon, Z.J. Williamson, O. Ciobotaru, Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge, Cryptol. ePrint Arch. (2019).

[22] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, N. Ward, Marlin: Preprocessing zkSNARKs with universal and updatable SRS, in: Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39, Springer, 2020, pp. 738–768.
[23] E. Ben-Sasson, A. Chiesa, E. Tromer, M. Virza, Succinct non-interactive zero knowledge for a von Neumann architecture, in: 23rd {USENIX} Security Symposium, {USENIX} Security 14, 2014, pp. 781–796.
[24] J. Groth, On the size of pairing-based non-interactive arguments, in: Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35, Springer, 2016, pp. 305–326.
[25] J. Groth, M. Maller, Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs, in: Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part II, Springer, 2017, pp. 581–612.
[26] C. Baum, I. Damgård, C. Orlandi, Publicly auditable secure multi-party computation, in: Security and Cryptography for Networks: 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings 9, Springer, 2014, pp. 175–196.
[27] M. Brandenburger, C. Cachin, R. Kapitza, A. Sorniotti, Blockchain and trusted computing: Problems, pitfalls, and a solution for hyperledger fabric, 2018, arXiv preprint arXiv:1805.08541.
[28] Y. Wang, J. Li, S. Zhao, F. Yu, Hybridchain: A novel architecture for confidentiality-preserving and performant permissioned blockchain using trusted execution environment, IEEE Access 8 (2020) 190652–190662.
[29] S. Steffen, B. Bichsel, M. Gersbach, N. Melchior, P. Tsankov, M. Vechev, Zkay: Specifying and enforcing data privacy in smart contracts, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 1759–1776.
[30] S. Steffen, B. Bichsel, R. Baumgartner, M. Vechev, Zeestar: Private smart contracts by homomorphic encryption and zero-knowledge proofs, in: 2022 IEEE Symposium on Security and Privacy, SP, IEEE, 2022, pp. 179–197.
[31] S. Steffen, B. Bichsel, M. Vechev, Zapper: Smart contracts with data and identity privacy, in: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, 2022, pp. 2735–2749.
[32] G. Zyskind, O. Nathan, A. Pentland, Enigma: Decentralized computation platform with guaranteed privacy, 2015, arXiv preprint arXiv:1506.03471.
[33] C. Baum, J.H.-y. Chiang, B. David, T.K. Frederiksen, Eagle: Efficient privacy preserving smart contracts, Cryptol. ePrint Arch. (2022).
[34] Q. Ren, H. Liu, Y. Li, H. Lei, Demo: Cloak: A framework for development of confidential blockchain smart contracts, in: 2021 IEEE 41st International Conference on Distributed Computing Systems, ICDCS, 2021, pp. 1102–1105.
[35] A. Banerjee, H. Tewari, Multiverse of HawkNess: A universally-composable MPC-based hawk variant, Cryptography 6 (3) (2022) 39.
[36] R. Solomon, R. Weber, G. Almashaqbeh, smartfhe: Privacy-preserving smart contracts from fully homomorphic encryption, Cryptol. ePrint Arch. (2021).
[37] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, H. Wu, Zexe: Enabling decentralized private computation, in: 2020 IEEE Symposium on Security and Privacy, SP, IEEE, 2020, pp. 947–964.
[38] A.L. Xiong, B. Chen, Z. Zhang, B. Bünz, B. Fisch, F. Krell, P. Camacho, VERI-ZEXE: Decentralized private computation with universal setup, Cryptol. ePrint Arch. (2022).
[39] R. Del Pino, V. Lyubashevsky, G. Seiler, Short discrete log proofs for FHE and ring-LWE ciphertexts, in: IACR International Workshop on Public Key Cryptography, Springer, 2019, pp. 344–373.
[40] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, G. Maxwell, Bulletproofs: Short proofs for confidential transactions and more, in: 2018 IEEE Symposium on Security and Privacy, SP, IEEE, 2018, pp. 315–334.
[41] Q. Ren, Y. Wu, H. Liu, Y. Li, A. Victor, H. Lei, L. Wang, B. Chen, Cloak: Transitioning states on legacy blockchains using secure and publicly verifiable off-chain multi-party computation, in: Proceedings of the 38th Annual Computer Security Applications Conference, 2022, pp. 117–131.
[42] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, D. Song, Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts, in: 2019 IEEE European Symposium on Security and Privacy, EuroS&P, IEEE, 2019, pp. 185–200.
[43] H. Yin, S. Zhou, J. Jiang, Phala network: A confidential smart contract network based on polkadot, 2019.
[44] M. Bowman, A. Miele, M. Steiner, B. Vavala, Private data objects: an overview, 2018, arXiv preprint arXiv:1807.05686.
[45] M. Russinovich, E. Ashton, C. Avanessians, M. Castro, A. Chamayou, S. Clebsch, M. Costa, C. Fournet, M. Kerner, S. Krishna, et al., CCF: A Framework for Building Confidential Verifiable Replicated Services, Technical Report, Microsoft Research and Microsoft Azure, 2019.
[46] R. Yuan, Y.-B. Xia, H.-B. Chen, B.-Y. Zang, J. Xie, Shadoweth: Private smart contract on public blockchain, J. Comput. Sci. Tech. 33 (2018) 542–556.
[47] P. Das, L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, A.-R. Sadeghi, FastKitten: Practical smart contracts on bitcoin, in: USENIX Security Symposium, 2019, pp. 801–818.
[48] Phat Contract, Phala Network, https://phala.network/phat-contract. (Accessed 28 July 2023).
[49] C. Müller, M. Brandenburger, C. Cachin, P. Felber, C. Göttel, V. Schiavoni, TZ4fabric: Executing smart contracts with ARM TrustZone:(practical experience report), in: 2020 International Symposium on Reliable Distributed Systems, SRDS, IEEE, 2020, pp. 31–40.
[50] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in: International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 1999, pp. 223–238.
[51] S. Garg, A. Goel, A. Jain, G.-V. Policharla, S. Sekar, zkSaaS: Zero-knowledge SNARKs as a service, Cryptol. ePrint Arch. (2023).
[52] G. Chen, Y. Zhang, T.-H. Lai, Opera: Open remote attestation for intel's secure enclaves, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 2317–2331.
[53] AMD Secure Encrypted Virtualization (SEV), AMD, https://www.amd.com/en/developer/sev.html. (Accessed 29 July 2023).
[54] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, D. Song, Keystone: An open framework for architecting trusted execution environments, in: Proceedings of the Fifteenth European Conference on Computer Systems, 2020, pp. 1–16.
[55] Arm Confidential Compute Architecture, Arm, https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture. (Accessed 29 July 2023).
[56] S. Zhao, M. Li, Y. Zhang, Z. Lin, vsgx: Virtualizing sgx enclaves on amd sev, in: 2022 IEEE Symposium on Security and Privacy, SP, IEEE, 2022, pp. 321–336.