RPSC: Regulatable Privacy-Preserving Smart Contracts on Account-based Blockchain

Zoe L. Jiang, *Member, IEEE,* Min Xie, Hanlin Chen, Yijian Pan, Jiazhuo Lyu, Man Ho Au, *Member, IEEE,* Junbin Fang, Yang Liu and Xuan Wang

***Abstract*—Smart contracts have been widely used to develop decentralized applications on account-based blockchain. The privacy issues of smart contracts have also received attention from researchers, and many privacy-preserving schemes and applications have been proposed. However, most existing schemes cannot achieve flexible conversion between private and public data. And the overly secure privacy-preserving scheme directly makes the regulation impossible. To mitigate these limitations, we propose a flexible privacy-preserving smart contracts with regulation (RPSC) system over the account-based blockchain. We first design a two-layer commitment structure that enables the fine-grained privacy protection (identity anonymity and data confidentiality) and flexible data state transitions. Then we combine a public-key encryption scheme with a zk-SNARKs scheme to achieve regulation property while keeping user's identity from others. Moreover, we prove that our scheme is secure, including privacy, soundness and traceability. Finally, we integrate RPSC into an account-based blockchain and implement two applications to evaluate the system performance. The evaluation results show our system performs effectively in practical settings.**

***Index Terms*—regulatable smart contracts, account-based blockchain, identity anonymity, data privacy, zk-SNARKs.**

I. Introduction

CRYPTOCURRENCY, which is based on public-key cryptography, is considered one of the most promising digital currencies. Cryptocurrency enables parties to conduct transactions and verify them directly in a decentralized network via the well-known blockchain technology [1]. Ideally, users expect that (1) their transactions can be processed efficiently; (2) their private information can be protected; and (3) malicious behaviors (e.g., double spending attacks and forging extra currency) can be detected and prohibited. Cryptocurrencies have attracted the attention of many scholars, focusing on the confidentiality of (transaction) data and the anonymity of user identities.

Bitcoin [2], one of the most representative cryptocurrencies, protects user identity through the pseudonym mechanism. However, it is easy for attackers to trace transactions and deanonymise users [3] [4]. Zerocash [5] and Zerocoin [6] solve the problem of privacy disclosure by replacing actual information of transaction with commitment and utilise zero-knowledge proof to ensure transaction correctness at the expense of efficiency. Hawk [7], which is based on a cryptocurrency similar to ZeroCash, effectively addresses the privacy disclosure issue and provides a prototype for the design of blockchain privacy protection solutions. Nonetheless, this design loses the flexibility to disclose private data. It forces all data in a transaction to be either public or private, with no flexibility to reveal some private data.

In addition, while anonymity is a desirable requirement for user privacy protection in cryptocurrencies [8] [9], it also poses a great risk that malicious users will abuse the blockchain to escape punishment, or even commit crimes such as money laundering, drug trafficking, and extortion. As a result, accountability measures are considered acceptable to protect the rights and interests of users when private data is at stake. However, most of the previous schemes lack accountability, making it vulnerable to abuse, using it in illegal transaction. Furthermore, attacker may launch attack on blockchain without the fear of being caught (e.g., DDoS attacks in Bitcoin [10] and Ethereum [11]).

There are two popular types of blockchains: UTXO-based and account-based. In UTXO-based blockchains, transaction outputs are directly referenced as inputs in subsequent transactions. Conversely, in account-based blockchains, coins are represented as balances within accounts, rather than uniquely referenced transactions. This structure makes account-based blockchains inherently more adaptable to smart contracts, which are crucial for tailoring blockchain applications to diverse scenarios, and even the core of Web3.0[1]. For instance, Ethereum, the leading account-based blockchain, dominates the smart contract platform market, hosting the majority of decentralized applications (DApps) and managing billions in transaction value annually[2], and gradually replacing Bitcoin to capture more market share [12]. However, many existing

Manuscript received 17 September, 2023; revised 14 May, 2024; accepted 26 June 2024. This work was supported in part by the Shenzhen Science and Technology Major Project under Grant KJZD20230923114908017, in part by the Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies under Grant 2022B1212010005, in part by the PolyU SHS under Grant P0046340 and in part by the RGC under Grant C1029-22GF. Recommended for acceptance by Dr. Y. Wu.(Corresponding author: Junbin Fang.)

Zoe L. Jiang and Xuan Wang are with Harbin Institute of Technology, Shenzhen, Shenzhen 518055, and Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies, Shenzhen, China.(E-mail: zoeljiang@hit.edu.cn, wangxuan@cs.hitsz.edu.cn.)

Min Xie, Hanlin Chen, Yijian Pan are with Harbin Institute of Technology, Shenzhen, Shenzhen 518055, China.(E-mail: minxie@stu.hit.edu.cn, 19S051042@stu.hit.edu.cn, panyijian@stu.hit.edu.cn.)

Man Ho Au and Jiazhuo Lyu are with The Hong Kong Polytechnic University, Hong Kong, China.(E-mail: mhaau@polyu.edu.hk, jiazhuo.lyu@connect.polyu.hk.)

Junbin Fang is with the School of Science and Technology, Jinan University, Guangzhou 510632, China.(E-mail: tjunbinfang@jnu.edu.cn.)

Yang Liu is with the Department of Computer Science, Swansea University, Swansea SA2 8PP, UK.(E-mail: liu.yang@hit.edu.cn.)

[1]https://eduhubcommunity.hashnode.dev/smart-contracts-the-core-of-web3
[2]https://www.prudentmarkets.com/report/ethereum-market/152880/

TABLE I: Functional comparison of different frameworks

Scheme	Anonymity	Transaction Privacy	Programmability	Fine Grained Privacy Protection	Regulatory	blockchain type
Zerocoin [6]	yes	no	no	no	no	UTXO
Zerocash [5]	yes	yes	no	no	no	UTXO
Hawk [7]	yes	yes	yes	no	no	account
zkay [13]	no	yes	yes	no	no	account
Zether [14]	no	yes	yes	no	no	account
Zexe [15]	yes	yes	no	no	no	account
DSC [16]	no	yes	yes	no	no	account
VeriZexe [17]	yes	yes	yes	no	no	account
This paper	yes	yes	yes	yes	yes	account

privacy-preserving techniques are designed to work on UTXO-based blockchains.

A. Our contribution

This paper aims to provide a fine-grained privacy-preserving smart contract with regulatory measures. Specifically, we design a Regulatable Privacy-Preserving Smart Contracts (RPSC) system with identity anonymity and fine-grained data confidentiality. In addition, the system supports built-in regulatory measures, i.e., users can be deanonymised and the confidential data can be revealed by regulators if needed. Finally, we implement the system on account-based blockchain (with smart contracts) with performance evaluation. The contributions of this paper are summarized as follows.

Identity anonymity. RPSC implements its own identity mechanism inside smart contract. The identity anonymity in RPSC is guaranteed by its own zero-knowledge proof protocol.

Fine-grained data confidentiality. RPSC ensures the correct execution of a transaction without disclosing the knowledge about the private data and provides a flexible method for the data owner to selectively disclose some private data in the transaction to the public.

Regulatory mechanism. RPSC enables the regulators to disclose user identity or reveal private data involved in transactions if needed, e.g., under court order or during dispute handling.

Account-based blockchain platform compatibility. RPSC is designed and implemented on Ethereum smart contract, meaning that it can be deployed to all account-based blockchain platforms supporting Ethereum Virtual Machine (EVM). More importantly, we implement two applications using RPSC for blockchain-based blind auctions and electronic voting, supporting on-chain verification on smart contract. This evaluation demonstrates that RPSC is practical for account-based blockchains.

B. Related Work

Cryptocurrency transaction privacy. Cryptocurrency transaction privacy, which is normally considered as the anonymity of user identities and the confidentiality of data, has attracted a lot of attention [5], [6], [18]–[24]. For most cryptocurrencies like Bitcoin, coin mixing [18] [19] is always recommended or forced before a transaction. CryptoNote is a protocol for the implementation of untraceable and unlinkable cryptocurrencies using ring signatures. The Ring Confidential Transactions (RingCT) was first proposed in [20], then improved in [21]. Monero [23] [24] combines one-time address with the confidential transactions protocol [22] to achieve both identity anonymity and data confidentiality. Zerocoin [6] is an extension of bitcoin which applies zero-knowledge arguments to achieve identity anonymity. Zerocash [5] is a cryptocurrency protocol implementing fully identity anonymous based on SNARKs, which not only protects the identity of users, but also hides the transaction relationships and amounts between them. However, these privacy-preserving mechanisms in these schemes are mainly for UTXO blockchains and cannot be adapted to account-based blockchains. To achieve transaction privacy, RPSC implements its own privacy preserving mechanism at the smart contract layer, making it possible to work on account-based blockchain with smart contract.

Smart contract privacy. Kosba et al. proposed Hawk [7], a system that achieves both transaction privacy and low computational complexity on blockchain by introducing manager roles and zero knowledge proof protocols. Zkay [13] is a language for implementing smart contracts, which introduces privacy types defining owners of private values. Zero-Knowledge EXEcution (ZEXE) [15] is an account-based system supporting private smart contract on a distributed ledger. In ZEXE, a user conducts offline computation and uploading a proof of correct execution to the ledger (which hides additionally which smart contract has been executed). However, stateful computations are not supported in ZEXE, meaning that only partial smart contract functionality is provided. VeriZexe [17] is introduced to provide a decentralized private computation with a universal setup, implying supporting general computations. DSC [16] applied homomorphic encryption and zero-knowledge arguments to protect balance and transaction amounts with a mechanism for programmability. Eagle [25] focuses on token amounts and auxiliary data private in smart contract by using multi-party computation (MPC). To address the expressive limitations of existing privacy smart contracts, ZeeStar [26] is a smart contract compiler that supports operations on private foreign data and integrates homomorphic encryption, making it suitable for applications that require computations on encrypted values. However, most of these schemes are for account-based blockchains and do not support identity anonymity due to the relative ease of linking accounts in the account-based blockchain. Recently, Zapper [27] is introduced to hide not only the identity of its users but also the objects they access to prevent deanonymization attacks by using oblivious Merkle tree construction. Unfortunately, account balances, private data, and identity anonymity are defined separately, making providing transaction privacy even more complicated if smart contract transactions involve simultaneous modifications of these state values.

Regulatable and privacy-preserving blockchains. Decentralized auditing schemes like zkLedger [28], Solidus [29], etc [30]–[32] consider auditing in distributed ledger applications. In these schemes, financial institutions, such as banks, act as regulators and are responsible for managing contingencies in the audit process. However, the regulation of these schemes is conducted mainly for the audit of payment transactions and is not fully suitable for other types of applications. Garman et al. proposed Decentralized Anonymous Payment (DAP) systems [33], an extension of Zerocash supporting regulatory measures. Similar to DAP, RPSC combines encryption and zero-knowledge arguments to implement a regulatory mechanism for smart contract functions besides payment, such as e-auction or e-voting.

In Table I we compare the related work with the work in this paper functionally. RPSC realizes anonymity, transaction confidentiality and programmability, and fine-grained privacy preserving.

Organization. Section 2 reviews cryptographic primitives in the system. The RPSC system model is introduced in Section 3. In Section 4 we present the construction of the RPSC system including user algorithms, regulator algorithms and the smart contract, followed by presenting its comparison with prior works. Security analysis and system implementation with performance evaluation are illustrated in Section 5 and 6, respectively. Finally, we draw a conclusion of this paper in Section 7.

II. Preliminaries

In this section, we describe the cryptography primitives employed in the system.

A. Zk-SNARKs

The acronym zk-SNARKs [34] stands for “Zero-Knowledge Succinct Non-Interactive Argument of Knowledge”, which allows someone to prove possession of certain information, e.g., a secret key, without revealing that information, and without interacting with the verifier. We describe the Zk-SNARKs using the following algorithms:

- $(pk, vk) \leftarrow KeyGen(1^\lambda, C)$. On input a security parameter λ and an arithmetic circuit C, the algorithm outputs the proving key pk and the verification key vk.
- $\pi \leftarrow Prove(pk, x, a)$. On input a proving key pk, a statement x and a witness w, where x and w satisfied the relationship $\mathcal{R}_C$ and $\mathcal{R}_C$ is the set of (x, a) such that $C(x, a) = 1$. The algorithm outputs the proof π for the statement x.
- $1/0 \leftarrow Verify(vk, x, \pi)$. On input a verification key vk, a statement x and the corresponding proof π. the algorithm outputs 1 or 0 to indicate whether the statement x is true or not.

ZK-SNARKs should satisfy the following properties:

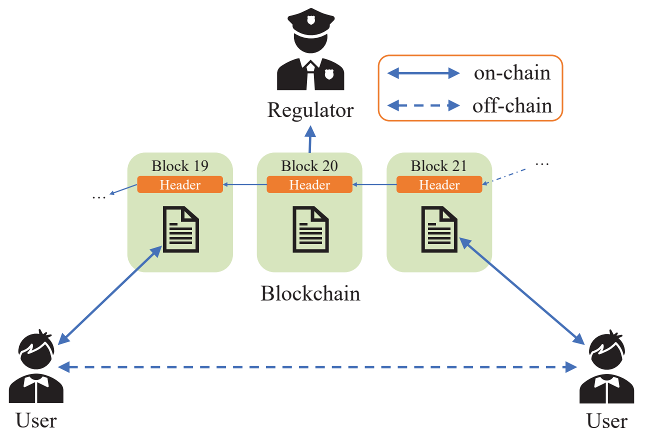


Fig. 1: The system model of the RPSC system

- Completeness. For all security parameter λ, any arithmetic circuit C, and any $(x, a) \in \mathcal{R}_C$, an honest prover who holds the witness a always convinces an honest verifier that the statement x is true.
- Succinctness. An prover generates a proof π has $O_\lambda(1)$ bits and Verifier run $Verify(vk, x, \pi)$ in time $O_\lambda(|x|)$.
- Proof of knowledge (and soundness). For all $poly(\lambda)$-size adversary $\mathcal{A}$, if $\mathcal{A}$ can output a' with a non-negligible probability and convince the verifier, there is a $poly(\lambda)$-size extractor $\mathcal{E}$ can extract a' with a non-negligible probability, satisfying $(x, a') \in \mathcal{R}_C$.
- Perfect zero knowledge. There is a polynomial-time simulator Sim such that for all distinguishers $\mathcal{D}$, the probability of distinguishing whether a proof π is generated by the simulator Sim or an honest prover is negligible.

B. Commitment

A commitment scheme [35] allows someone to commit to a chosen value while keeping it hidden to others, with the ability to reveal the committed value later. Generally, the commitment scheme is composed of a pair of PPT algorithms (Gen, Com) with binding and hiding properties which are defined as follows.

- $pp \leftarrow Gen(1^\lambda)$. On input a security parameter 1^λ, the algorithm outputs public parameters pp.
- $commit \leftarrow Com(pp, m, r)$. On input public parameters pp, a message m and a random number r, the algorithm outputs the commitment $commit$ for the message m.

The binding and hiding properties are described as follows.

- Binding. Given parameter pp, for all PPT adversaries $\mathcal{A}$, $Pr[(m_0, r_0, m_1.r_1) \leftarrow \mathcal{A}(pp)|Com(pp, m_0, r0) = Com(pp, m_1, r_1) \wedge m_0 \neq m_1] \leq negl(\lambda)$.
- Hiding. For any two messages m and m', for all adversaries $\mathcal{A}$, $Com(pp, m, r)$ and $Com(pp, m', r')$ with uniformly random r, r' are indistinguishable.

III. System Architecture

A. System Model

In the RPSC system model (Fig. 1), there are three entities (“User”, “Regulator” and “Blockchain”), together with two communication methods (“off-chain” and “on-chain”). The “Off-chain” communications will be conducted over channels secured by cryptographic protocols, such as Transport Layer Security (TLS), and the “on-chain” is achieved by publishing a transaction using Peer-To-Peer (P2P) communication.

- *User*: Users in the system are the main participants, who own their respective key pairs (each key pair comprises a public key and a private key). The public keys are users’ identifications and the private keys are used to generate privacy transactions. Users mainly use smart contracts on the blockchain, and their capabilities depend on the role in the specific application. For example in voting scenarios, there are two types of users, voters and vote-counters. Voters cannot perform counting operations while vote-counters can.
- *Regulator*: Regulators, typically government departments, can view data on the blockchain, more specifically private data within contracts, and track privacy transactions on the chain. The regulator holds a pair of public and private keys. However, regulators cannot manage the data on the blockchain. This entity can only check the legitimacy of transactions on the blockchain.
- *Blockchain*: Blockchain is regarded as a transparent and append-only public distributed ledger maintained by all nodes named miners. In the system, any blockchain system supporting the functionality of smart contract can be adopted. A smart contract is deployed in blockchain and then automatically executed once being triggered by users.

B. Syntax

RPSC is composed of user algorithms, regulator algorithms and the smart contract. The user algorithms and regulator algorithms are executed locally by users and the regulator, respectively. The algorithms in the smart contract are triggered to automatically execute on chain. In particular, RPSC takes $\{\mathcal{P}, \mathcal{C}, \mathcal{U}\}$ as the application parameters $\mathbf{ap}$, where sets $\mathcal{P}, \mathcal{C}, \mathcal{U}$ are used to store private data records, public data records and registered users, respectively.

Regulator algorithms

- $(PK_{reg}, SK_{reg}, pp) \leftarrow regulatorSetup(\lambda)$. The algorithm takes the security parameter λ as input and outputs the regulator’s public-private key pair (PK_{reg}, SK_{reg}) and public parameters *pp*. Moreover, *pp* will be provided as an implicit input to the remaining algorithms.
- $(cm_{r_{prev}}, PK_{user}) \leftarrow revealRecord\ (r, SK_{reg})$. The algorithm is run by regulator that can trace the current owner of a record and disclose the commitment of the previous record. On input a record r and the regulator’s private key SK_{reg}, this algorithm outputs the commitment for the previous record $cm_{r_{prev}}$ and the owner PK_{user} of the record r.

User algorithms

- $(PK_{user}, SK_{user}) \leftarrow userSetup(\lambda)$. The algorithm takes the security parameter λ as input and generates a key pair PK_{user}, SK_{user}. Note that PK_{user} is recorded in application parameters $\mathbf{ap}$ and published in blockchain.
- $(r, cm_r, \pi, ct) \leftarrow createRecord\ (data, PK_{user}, PK_{reg}, ProK, flag)$. The algorithm is run by the user who wants to create a record r for the data $data$. On input a data $data$, the public key of a registered user PK_{user}, the regulator’s public key PK_{reg}, the proving key $ProK_{create}$ for zero-knowledge proof and a flag $flag$ to indicate whether m is private, this algorithm generates a new record r of $data$, the commitment cm_r for r, a ciphertext ct by PK_{reg} and a proof π for proving the computation of *createRecord*. In particular, r is in the possession of the owner, and cm_r, π, ct are released to the blockchain for verification by the smart contacts. For simplicity, this procedure is treated as the default operation in the following algorithm, which has similar output.
- $(r', cm_{r'}, \pi, ct) \leftarrow updatePrivateData(r, cm_r, data_{new}, PK_{user}, SK_{user}, PK_{reg}, ProK_{update}, MTree)$. The algorithm is run by the owner PK_{user} of the record r who updates the private data without exposing the previous data of r. On input a record r, the corresponding commitment cm_r, the data to be updated $data_{new}$, the key pair of the record owner (PK_{user}, SK_{user}), the regulator’s public key PK_{reg}, the proving key $ProK_{update}$ and the Merkle tree MT build over all $records$ on the blockchain. this algorithm generates a updated record r' of $data_{new}$, the commitment $cm_{r'}$ for r', a ciphertext ct by PK_{reg} and a proof π for proving computation of *updatePrivateData*.
- $(r', cm_{r'}, \pi, ct) \leftarrow transferDataOwnership(r, cm_r, PK_{user}, SK_{user}, PK_{user'}, PK_{reg}, ProK_{transfer}, MTree)$. The algorithm is run by the owner PK_{user} of the record r who wants to transfer ownership of r to another user $PK_{user'}$. On input a record r, the corresponding commitment cm_r, the key pair of the record owner (PK_{user}, SK_{user}), the transferred user’s public key $PK_{user'}$, the regulator’s public key PK_{reg}, the proving key $ProK_{transfer}$ for *transferDataOwnership* and the Merkle tree MT build over all $records$ on the blockchain. This algorithm generates a transferred record r' for $PK_{user'}$, the commitment $cm_{r'}$ for r', a ciphertext ct by PK_{reg} and a proof π for proving computation of *transferDataOwnership*.
- $(r', cm_{r'}, \pi, ct) \leftarrow revealPrivateData(r, cm_r, PK_{user}, SK_{user}, PK_{reg}, ProK_{reveal}, MTree)$. The algorithm is run by the owner PK_{user} of the record r who reveal the private data in the commitment cm_r. On input a record r, the corresponding commitment cm_r, the key pair of the record owner (PK_{user}, SK_{user}), the regulator’s public key PK_{reg}, the proving key $ProK_{reveal}$ for *revealPrivateData* and the Merkle tree MT build over all $records$ on the blockchain. This algorithm generates a record r', a commitment $cm_{r'}$ for public data $data$, a ciphertext ct by PK_{reg} and a proof π for proving computation of *revealPrivateData*.
- $(R_{create}, CM, \pi, ct) \leftarrow compute(R_{con}, COM_{con}, F, aux, PK_{user}, SK_{user}, PK_{reg}, ProK_{compute}, flag)$. The algorithm is run by the owner PK_{user} of the record set R_{con} who attempts to perform arbitrary operations modeled as function F. On input a record set R_{con}, the

corresponding commitment set COM_{con}, a function F, the auxiliary information aux, the key pair of the owner (PK_{user}, SK_{user}), the regulator's public key PK_{reg}, the proving key $ProK_{compute}$ for *compute* and a flag $flag$ to indicate whether the result of F is private. this algorithm generates a new record set R_{create}, a commitment set CM, a ciphertext *ct* by PK_{reg} and a proof π for proving computation of *compute*.

Smart contract

- $1/0 \leftarrow verify(x, \pi, VK)$. The algorithm is automatically triggered by the smart contract, which takes as input the statement x for new records, the proof π for the statement, and the verification key VK, and outputs 1/0 to indicate whether the record is valid or not. Note that VK is always public.

C. Security Requirements

The RPSC system needs to satisfy the following fundamental security requirements [7], [13], [17].

1) **On-chain privacy.** On-chain privacy includes transaction privacy and identity anonymity. No one, except the regulator, should be able to determine the user's identity or transaction content.
2) **Off-chain soundness.** Any user, even the owner of a data record, should not falsify or tamper with the content of the data record.
3) **Data record unlinkablity.** No one, except the regulator, should be able to access to transaction content or determine user's identity.
4) **Regulatory traceability.** It is allowed for regulator to track transactions of data records with regulatory private key.

IV. System Construction

A. High-level Description

Building on top of any EVM-compatible smart contract platforms, the RPSC system enables the privacy preserving of smart contracts and preserves the programmability of smart contracts. The RPSC allows users to perform off-chain privacy computation and generate a publicly verifiable transaction to prove the correctness of the computation. In the transaction, the user is free to choose whether to disclose privacy-protected data and identity. Further, the system provides a regulatory mechanism that allows the regulator to track the link of private transactions and trace the detailed transaction data and identity information.

Similar to ZEXE, we use Record structure to generalise the idea of Zerocash to support general data. We implemented this structure of Record on the smart contract, and the user performs data computation by consuming and generating Records. The advantage of implementing the Record structure on the smart contract is clear: (1) it does not require the underlying blockchain to support functional privacy; and (2) we retain the full programmability of the smart contracts.

We further design a mechanism to give regulator access to private data and identity of sender. The basic idea is using the regulator's public key to encrypt a copy of the private transaction data, so that the regulator can use the private key to decrypt and get the detailed data to regulate the content of the transaction.

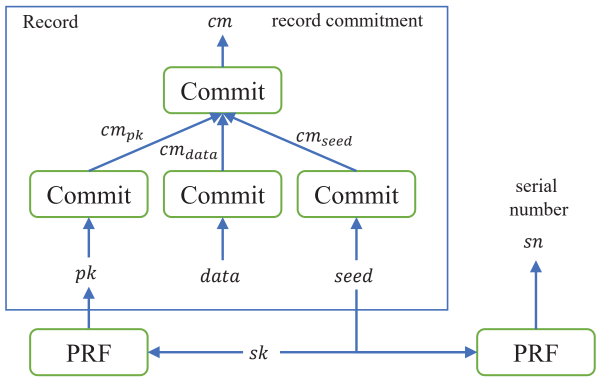


Fig. 2: Construction of multi-layer record

B. Fine-grained Data Record Structure

Generally, in the existing privacy-preserving blockchain, private data is represented as a specific data structure and public data is shown directly. Therefore, in some application scenarios, it needs to be pre-determined whether the data is public or not. When private data needs to be made public due to changing requirements, the application system may need to be redesigned or a revealing transaction with proof will be published. The main reason is the lack of consistent representation of public and private data, i.e. the privacy-preserving scheme only provides coarse-grained protection.

The RPSC system uses a similar data structure, i.e. **Record** structure, as ZEXE [15] to represent private data and extends it to represent both private and public data. The extended data structure is summarized in Fig. 2. Specifically, a record without privacy protection consists of three parts: the public key of the data owner, the data, and a serial number seed, where the seed is used to identify the record. Compared to the ZEXE's Record structure, the most significant change in the RPSC is to turn a single-layer structure into a multi-layer one. In a single-layer record, only the record commitment cm is stored in the contract, while in a multi-layer one $(cm_{pk}, cm_{data}, cm_{seed}, cm)$ are sent to the contract. If the user wants to disclose private data, cm_{data} could be replaced with $data$, then the smart contract verification only needs to compute $cm_{data} = Com(pp, data)$ firstly, and continue to verify the record commitment.

In RPSC, a user can easily implement fine-grained disclosure of private data by calling $compute$ or $revealPrivateData$. By setting the computation function F, $compute$ can equivalently split the original private data record. The user can selectively reveal the private data in the split data record, thus achieving fine-grained data privacy.

C. Concrete construction

Users in RPSC are assumed to be stateful, who record their transaction scripts for subsequent use. Additionally, the main cryptographic building blocks used in RPSC include commitment protocols ($C.Gen$, $C.Com$, $C.Ver$), zk-SNARKS protocols ($ZK.Gen$, $ZK.Pro$, $ZK.Ver$), pseudo-random functions (PRF), symmetric encryption schemes ($Sym.Gen$, $Sym.Enc$, $Sym.Dec$), random generation function (Ran) and public key encryption scheme($PK.Gen$, $PK.Enc$, $PK.Dec$). Furthermore, we assume that PRF is collision resistant, in the sense that it is infeasible to find $x \neq x^{'}$, such that $PRF(x) = PRF(x^{'})$.

Algorithm 1: $revealRecord$

```
Input: $r$, $sk_{reg}$
Output: the commitment $cm_{r_{prev}}$ of the previous record $r_{prev}$, the owner $PK_{user}$ of the current record $r$
1  let $Tx_{all}$ be the set of all transactions
2  $sk_{sym} = PK.Dec(sk_{reg}, keyCt)$
3  $plaintext = SymDec(sk_{sym}, dataCt)$
4  Parse $plaintext$ as $PK_{user}, seed_{prev}$
5  if $seed_{prev} = null$ then
6      $cm_{r_{prev}} = null$
7  else
8      $cm_{seed} = C.Com(pp, seed_{prev})$
9      Find $cm_{r_{prev}}$ by traversing $Tx_{all}$ with $cm_{seed}$
10 end
11 return $(cm_{r_{prev}}, PK_{user})$
```

Algorithm 2: $createRecord$

```
Input: $data$, $PK_{user}$, $PK_{reg}$, $ProK_{create}$, $flag$
Output: Newly created record $r$, record commitment $cm_r$, and proof $\pi$, cipher text $ct$
1  $seed = Ran(pp)$
2  $r = (PK_{user}, data, seed)$
3  $cm_{seed} = C.Com(pp, seed)$
4  $cm_{pk} = C.Com(pp, Pk_{user})$
5  $cm_{data} = C.Com(pp, data)$
6  if $flag = private$ then
7      $cm = C.Com(pp, cm_{pk}||cm_{data}||cm_{seed})$
8      $cm_r = (cm_{pk}, cm_{data}, cm_{seed}, cm, flag)$
9  else
10     $cm = C.Com(pp, cm_{pk}||data||cm_{seed})$
11     $cm_r = (cm_{pk}, data, cm_{seed}, cm, flag)$
12 end
13 $sk_{sym} = Sym.Gen(pp)$
14 $seed_{prev} = null$
15 $dataCt = Sym.Enc(sk_{sym}, PK_{user}||seed_{prev})$
16 $keyCt = PK.Enc(PK_{reg}, sk_{sym})$
17 $ct = keyCt||dataCt$
18 $x = (cm, ct, PK_{reg}, pp)$
19 $a = (r, sk_{sym})$
20 $\pi = ZK.Pro(ProK_{create}, x, a)$
21 return $(r, cm_r, \pi, ct)$
```

Regulator algorithms. Regulator algorithms are specifically how the regulator tracks the valid records, including $regulatorSetup$ and $revealRecord$. $regulatorSetup$ algorithm takes the security parameter λ as input and generates (PK_{reg}, SK_{reg}) by running public key encryption scheme $PK.Gen(\lambda)$. $revealRecord$ algorithm is detailed in Algorithm 1.

User algorithms. User algorithms are always used for users to handle their own records and generate the new records with corresponding proofs, which are released to the blockchain. Specifically, users can flexibly disclose the private data of valid records, or use valid records as input to compute any function without disclosing private records. Remarkably, a record is allowed to be handed over to another user in RPSC, whereas a record can only be held by one user at a time. The user algorithms are detailed in Algorithm 2,3,4,6,5. As for the $userSetup$ algorithm, it generates a private key $SK_{user} \leftarrow Ran(pp)$ by running a random generation function Ran with public parameters pp as input. Subsequently, the user obtains the public key $PK_{user} \leftarrow PRF(\lambda, SK_{user})$ by using PRF to input the security parameters λ and the private key SK_{user}.

Algorithm 3: $updatePrivateData$

```
Input: $r.seed$, $cm_r$ , $data_{new}$, $PK_{user}$, $SK_{user}$, $ProK_{update}$, $PK_{reg}$, $MT$
Output: Created record $r'$ and its commitment $cm'_r$, a proof $\pi$, and regulatory ciphertext $ct$
1  let $branch_r$ be the branch of $cm_r.cm$ in $MT$
2  $sn_r = PRF(SK_{user}, r.seed)$
3  $seed' = Ran(pp)$
4  $r' = (PK_{user}, data_{new}, seed')$
5  $cm_{seed'} = C.Com(pp, seed')$
6  $cm_{data_{new}} = C.Com(pp, data_{new})$
7  $cm' = C.Com(pp, cm_r.cm_{pk}||cm_{data_{new}}||cm_{seed'})$
8  let $flag$ be $private$
9  $cm_{r'} = (cm_r.cm_{pk}, cm_{data_{new}}, cm_{seed'}, cm', flag)$
10 $plaintext = PK_{user}||r.seed$
11 $sk_{sym} = Sym.Gen(pp)$
12 $dataCt = Sym.Enc(sk_{sym}, plaintext)$
13 $keyCt = PK.Enc(PK_{reg}, sk_{sym})$
14 $ct = keyCt||dataCt$
15 $x = (MT.root, sn_r, cm_{r'}, ct, pp, PK_{reg})$
16 $a = (r, r', sk_{sym}, SK_{user}, set_{branch})$
17 $\pi = NIZK.Proof(ProK_{update}, x, a)$
18 return $(r', cm_{r'}, \pi, ct, sn_r)$
```

To protect user anonymity and data confidentiality, zero-knowledge proof is involved in RPSC. In particular, the statement of zero-knowledge proof in user algorithms are described as follows.

Let r is a newly generated record, which contains plaintext data $data$, user identity Pk_{user} and private seed $seed$ for r. cm is considered as the commitment of r, $dataCT$ and $keyCT$ are encryption under symmetric key sk_{sym} with Pk_{user} and the private seed of the previous record $seed_{prev}$ as input, and encryption under the public key of the regulator PK_{reg} with sk_{sym} as input. In Algorithm 2, if a user attempts to convince others that the new record r is valid without revealing it, he must provide a proof that: 1) cm is indeed the commitment of r; 2) the user identity Pk_{user} in the ciphertext $dataCT$ and contained in r are the same; 3) The sk_{sym} encrypted in $keyCT$by PK_{reg} is indeed the private key of $dataCT$. Formally, a statement for Algorithm 2 is proved as follows, in which the data is private:

$$\left\{\begin{array}{l} \big(data, PK_{user}, seed, sk_{sym}\big): \\ r = (Pk_{user}, data, seed) \wedge cm_{seed} = C.Com(pp, seed) \\ \wedge cm_{pk} = C.Com(pp, Pk_{user}) \wedge \\ cm_{data} = C.Com(pp, data) \wedge \\ cm = C.Com(pp, cm_{pk}||cm_{data}||cm_{seed}) \wedge \\ dataCt = Sym.Enc(sk_{sym}, PK_{user}||seed_{prev}) \wedge \\ keyCt = PK.Enc(PK_{reg}, sk_{sym}) \\ \wedge ct = keyCt||dataCt \end{array}\right\}$$

Algorithm 4: $revealPrivateData$

```
Input: $r, cm_r, PK_{user}, SK_{user}, ProK_{reveal}, PK_{reg}, MT$
Output: Public data record $r'$, $cm_r'$, a proof $\pi$ and regulatory ciphertext $ct$
1  let $branch_r$ be the branch of $cm_r.cm$ in $MT$
2  $sn_r = PRF(SK_{user}, r.seed)$
3  $seed' = Ran(pp)$
4  $r' = (PK_{user}, r.data, seed')$
5  $cm_{seed'} = C.Com(pp, seed')$
6  $cm' = C.Com(pp, cm_r.cm_{pk}||r.data||cm_{seed'})$
7  let $flag$ be $public$
8  $cm_{r'} = (cm_r.cm_{pk}, r.data, cm_{seed'}, cm', flag)$
9  $plaintext = PK_{user}||r.seed$
10 $sk_{sym} = Sym.Gen(pp)$
11 $dataCt = Sym.Enc(sk_{sym}, plaintext)$
12 $keyCt = PK.Enc(PK_{reg}, sk_{sym})$
13 $ct = keyCt||dataCt$
14 $x = (MT.root, sn_r, cm_{r'}, ct, pp, PK_{reg})$
15 $a = (r, r', sk_{sym}, SK_{user}, set_{branch})$
16 $\pi = NIZK.Proof(ProK - reveal, x, a)$
17 return $(r, cm_{r'}, sn_r, \pi, ct)$
```

The following statement is used in Algorithms 3,4,6,5. The statements of these algorithms are similar to the described above, and Algorithm 6 is considered as a generalization. For brevity, we state the statement of Algorithm 6. In Algorithm 6, the user is asked to provide a proof, which is the same as Algorithm 3 except that the user has to disclose the unique record serial numbers SN to consume the set R_{con} of valid records $\{r_1, ..., r_k\}$ he holds. Note that the commitment CM_{con} of R_{con} is contained in the Merkle tree consisting of all records whose root is $Mroot$. Here we set the function F as an arbitrary computation that consumes the record set R_{con} and outputs a new record set $R_{create} = \{r_1{}', \ldots, r_{k'}{}'\}$, which is generated by R_{con}. Hence, he has to provide an additional proof: 4) The new set R_{create} of records $\{r_1{}', \ldots, r_{k'}{}'\}$ is computed using the records R_{con} by the function F; 5) All records $\{r_1, ..., r_k\}$ contained in R_{con} have a path $branch_i$ to $Mroot$;6) the private record serial numbers SN for R_{con} are held. A formal statement is provided below, where the data is kept private:

$$\left\{\begin{array}{c} (R_{con}, R_{create}, \{branch_i\}_{1,\ldots,k}, SK_{user}, seed, sk_{sym},): \\ \left(\begin{array}{c} \forall r_i \in R_{con}, sn_i \in SN \\ sn_i = PRF(SK_{user}, r_i.seed) \\ \wedge cm^{r_i}_{seed} = C.Com(pp, r_i.seed) \\ \wedge cm^{r_i}_{pk} = C.Com(pp, r_i.Pk_{user}) \\ \wedge cm^{r_i}_{data} = C.Com(pp, r_i.data) \\ \wedge cm_{r_i} = C.Com(pp, cm^{r_i}_{seed}||cm^{r_i}_{pk}||cm^{r_i}_{data}) \wedge \\ True = MerkleBrranch(Mroot, cm_{r_i}, branch_i) \end{array}\right) \\ \wedge\{r_1{}', \ldots, r_{k'}{}'\} = F(r_1, \ldots, r_k, aux) \wedge \\ \left(\begin{array}{c} \forall r_i{}' \in R_{create} cm^{r_i{}'}_{seed} = C.Com(pp, r_i{}'.seed) \\ \wedge cm^{r_i{}'}_{pk} = C.Com(pp, r_i{}'.Pk_{user}) \\ \wedge cm^{r_i{}'}_{data} = C.Com(pp, r_i{}'.data) \\ \wedge plaintext = plaintext||(r_i{}'.PK_{user}, plaintext_{pr}) \\ \wedge CM = CM||cm_{r_i{}'} \end{array}\right) \\ \wedge dataCt = Sym.Enc(sk_{sym}, plaintext) \\ \wedge keyCt = PK.Enc(PK_{reg}, sk_{sym}) \\ \wedge ct = keyCt||dataCt \end{array}\right\}$$

Algorithm 5: transferDataOwnership

```
Input: $r.seed$, $cm_r$ , $PK_{user}$, $SK_{user}$ , $PK_{user'}$, $ProK_{transfer}$, $PK_{reg}$, $MT$
Output: Created data record $r'$, $cm_{r'}$d, a proof $\pi$, the series number $sn_r$ and regulatory ciphertext $ct$
1  let $branch_r$ be the branch of $cm_r.cm$ in $MT$
2  $sn_r = PRF(SK_{user}, r.seed)$
3  $seed' = Ran(pp)$
4  $r' = (PK_{user'}, r.data, seed')$
5  $cm_{seed'} = C.Com(pp, seed')$
6  $cm_{pk'} = C.Com(pp, PK_{user'})$
7  $cm' = C.Com(pp, cm_{pk'}||cm_r.cm_{data}||cm_{seed'})$
8  $cm_{r'} = (cm_{pk'}, cm_r.cm_{data}, cm_{seed'}, cm', cm_r.flag)$
9  $plaintext = PK_{user}||r.seed$
10 $sk_{sym} = Sym.Gen(pp)$
11 $dataCt = Sym.Enc(sk_{sym}, plaintext)$
12 $keyCt = PK.Enc(PK_{reg}, sk_{sym})$
13 $ct = keyCt||dataCt$
14 $x = (MT.root, sn_r, cm_{r'}, ct, pp, PK_{reg})$
15 $a = (r, r', sk_{sym}, SK_{user}, set_{branch}, PK_{user'})$
16 $\pi = NIZK.Proof(ProK_{transfer}, x, a)$
17 return $(r', cm_{r'}, \pi, ct, sn_r)$
```

Smart contract. The smart contract is triggered to record the user's registration and verify that the record is valid, which consists of the $verify$ algorithms. Roughly speaking, the $verify$ algorithm is viewed as a universal verification of all involved zero-knowledge proofs, and we believe that all involved proofs can be covered in the statement of Algorithm 6. The algorithm is detailed in Algorithm 7.

```
Algorithm 6: compute
  Input: $R_{con} = \{r_1, \ldots, r_k\}$,
         $CM_{con} = \{cm_{r_1}, \ldots, cm_{r_k}\}$, $F$, $aux$,
         $PK_{user}$,$SK_{user}$, $PK_{compute}$,$PK_{reg}$, $flag$
  Output: set of commitment of created records, a
          proof, , and regulatory ciphertext $ct$
 1 let $MT$ be a Merkle tree build over all $records$
 2 $set_{branch} = null$
 3 $SN = null$
 4 $CM = null$
 5 $plaintext_{pr}, plaintext = null$
 6 for $r_i \in R_{con}$ do
 7     let $branch_{r_i}$ be the branch of $cm_{r_i}.cm$ in $MT$
 8     $set_{branch} = set_{branch}||branch_{r_i}$
 9     $SN = SN||PRF(SK_{user}, r_i.seed)$
10     $plaintext_{pr} = plaintext_{pr}||r_i.seed$
11 end
12 $R_{create} = \{r_1{}', \ldots, r_{k'}{}'\} = F(r_1, \ldots, r_k, aux)$
13 for $r_i{}' \in R_{create}$ do
14     $cm^{r_i{}'}_{seed} = C.Com(pp, r_i{}'.seed)$
15     $cm^{r_i{}'}_{pk} = C.Com(pp, r_i{}'.Pk_{user})$
16     $cm^{r_i{}'}_{data} = C.Com(pp, r_i{}'.data)$
17     if $flag = private$ then
18         $cm' = C.Com(pp, cm^{r_i{}'}_{pk}||cm^{r_i{}'}_{data}||cm^{r_i{}'}_{seed})$
19         $cm_{r_i{}'} = (cm^{r_i{}'}_{pk}, cm^{r_i{}'}_{data}, cm^{r_i{}'}_{seed}, cm', flag)$
20     else
21         $cm' = C.Com(pp, cm^{r_i{}'}_{pk}||r_i{}'.data||cm^{r_i{}'}_{seed})$
22         $cm_{r_i{}'} = (cm^{r_i{}'}_{pk}, r_i{}'.data, cm^{r_i{}'}_{seed}, cm', flag)$
23     end
24     $plaintext = plaintext||(r_i{}'.PK_{user}, plaintext_{pr})$
25     $CM = CM||cm_{r_i{}'}$
26 end
27 $sk_{sym} = Sym.Gen(pp)$
28 $dataCt = Sym.Enc(sk_{sym}, plaintext)$
29 $keyCt = PK.Enc(PK_{reg}, sk_{sym})$
30 $x = (MT.root, SN, CM, keyCt||dataCt, pp, PK_{reg})$
31 $a = (R_{con}, R_{create}, sk_{sym}, SK_{user}, set_{branch})$
32 $\pi = NIZK.Proof(PK_{compute}, x, a)$
33 $ct = keyCt||dataCt$
34 return $(R_{create}, SN, CM, \pi, ct)$
```

```
Algorithm 7: verify
  Input: $x$ , $\pi$ ,$VK$
1 parse $x$ as $(MT.root, SN, CM, ct, pp, PK_{reg})$
2 if $ZK.Ver(VK, x, \pi)$ = 1 then
3     $\mathcal{C}.append(CM)$
4     $\mathcal{P}.append(SN)$
5     return 1
6 else
7     return 0
8 end
```

D. Workflow

The RPSC system consists of three phases, namely: **Initialization**, **Register**, and **Computation**.

We give a use case of RPSC to illustrate how the algorithm described above realizes the properties of RPSC.

Voting Example: Suppose there are n voters denote $P_i(i = 1, 2, ..., n)$ and a voting manager. $data = 0$ means voter votes against and 1 for agreement. The voting is divided into 3 stages: Initialization, Register, and Computation. We suppose the public key of the regulator (Pk_{reg}) is published.

- Initialization

1) All voters $P_i(i = 1, 2, ..., n)$ call $userSetup$ to get (PK_i, SK_i), the voting manager call $userSetup$ to get (PK_v, SK_v).

2) The voting manager calls $ZK.Gen(1^\lambda, C)$ to get $(ProK, VerK)$ where C is related to R_L.

3) The voting manager sets the voting smart contract to the blockchain.

- Registration

1) For each voter $P_i(i = 1, 2, ..., n)$, P_i sends public key PK_i to the smart contract.

2) The voting manager sends PK_v to the smart contract.

- Computation

1) The Voting manager calls $createRecord(null, PK_v, PK_{reg}, ProK, private)$ to get (r_i, cm_i, π_i, ct_i) for each voter P_i.

2) The voting manager sends the content of $(rt, null, cm_i, \pi_i, ct_i)$ to the smart contract on the blockchain. The smart contract calls $verify$ to verify each transaction.

3) The voting manager calls $transferDataOwnership$ $(r.seed_i, cm_i, PK_v, SK_v, PK_i, ProK, PK_{reg}, MT)$ and get $(r_{n+i}, cm_{n+i}, \pi_{n+i}, ct_{n+i}, sn_i)$ for each voter P_i.

4) The voting manager sends the transaction of content $(rt, sn_i, cm_{n+i}, \pi_{n+i}, ct_{n+i})$ to the smart contract. In addition, the voting manager sends r_{n+i} to voter P_i privately.

5) Each Voter P_i calls $updatePrivateData(r_{n+i}.seed, cm_{n+i}, data_{new_i}, PK_i, SK_i, ProK, PK_{reg}, MT)$ to get $(r_{2n+i}, cm_{2n+i}, \pi_{2n+i}, ct_{2n+i}, sn_{n+i})$.

6) Each voter sends the transaction of content $(rt, sn_{n+i}, cm_{2n+i}, \pi_{2n+i}, ct_{2n+i})$ to the smart contract.

7) Each voter P_i calls $transferDataOwnership$ $(r_{2n+i}.seed, cm_{2n+i}, PK + i, SK_i, PK_v, Pro_k, PK_{reg}, MT)$ to get $(r_{3n+i}, cm_{3n+i}, \pi_{3n+i}, ct_{3n+i}, sn_{3n+i})$.

8) Each Voter P_i sends the transaction of content $(rt, sn_{3n+i}, cm_{3n+i}, \pi_{3n+i}, ct_{3n+i})$ to the smart contract. In addition, P_i sends record r_{3n+i} to the voting manager privately.

9) The voting manager checks whether the record r_{3n+i} is corresponding to the cm_{3n+i} on the blockchain. The Voting manager set $F(r_1, r_2, ..., r_n, aux) = r_1.data + r_2.data + ... + r_m.data$. After that, the voting manager calls $compute(R_{con} = \{r_{3n+1}, r_{3n+2}, ..., r_{3n+n}\}, CM_{con} = \{cm_{3n+1}\ , cm_{3n+2}, ..., cm_{3n+n}\}, F, aux, PK_v, SK_v, Pro_K, PK_{reg}, public)$ to get $(r_{res}, sn_{res}, cm_{res}, \pi_{res}, ct_{res})$.

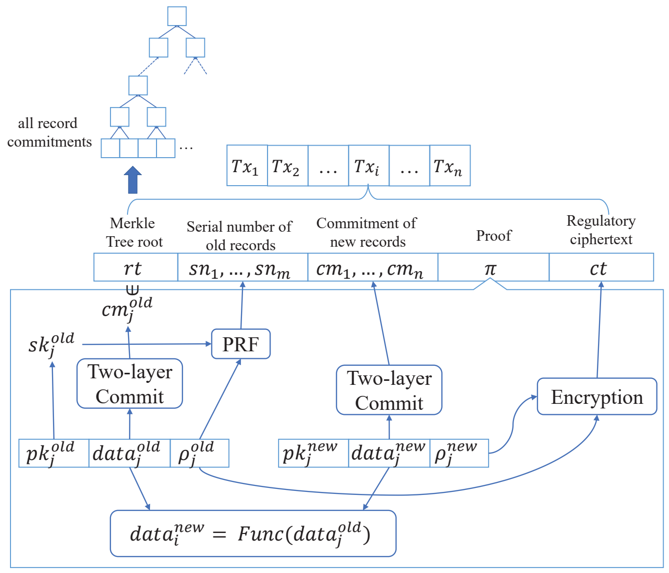

all record commitments
Tx_1 Tx_2 … Tx_i … Tx_n
Merkle Tree root
Serial number of old records
Commitment of new records
Proof
Regulatory ciphertext
rt $sn_1,\dots,sn_m$ $cm_1,\dots,cm_n$ π ct
cm_j^{old}
sk_j^{old}
PRF
Two-layer Commit
Two-layer Commit
Encryption
pk_j^{old} $data_j^{old}$ ρ_j^{old}
pk_j^{new} $data_j^{new}$ ρ_j^{new}
$data_i^{new} = Func(data_j^{old})$

Fig. 3: Construction of a transaction Tx_i

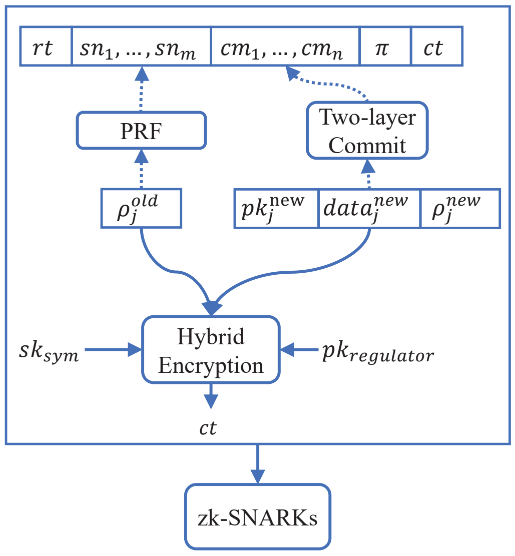

rt $sn_1,\dots,sn_m$ $cm_1,\dots,cm_n$ π ct
PRF
Two-layer Commit
ρ_j^{old}
pk_j^{new} $data_j^{new}$ ρ_j^{new}
sk_{sym}
Hybrid Encryption
$pk_{regulator}$
ct
zk-SNARKs

Fig. 4: Overview of transaction regulation

10) The voting manager sends the transaction of content $(rt, sn_{res}, cm_{res}, \pi_{res}, ct_{res})$ to the smart contract. In addition, the voting manager sends r_{n+i} to voter P_i privately.

11) At the time of revealing the voting result, the voting manager calls $reavelPrivateData(r_{res}, cm_{res}, PK_v, SK_v, Pro_k, PK_{reg}, MT)$ to get $(r_{res}, sn_{res'}, cm_{res'}, \pi_{res'}, ct_{res'})$.

12) The voting manager sends the transaction of content $(rt, sn_{res'}, cm_{res'}, \pi_{res'}, ct_{res'})$ to the smart contract.

E. Transaction Construction with accountability

Each two-layer record is associated with a public key pk, which implies consuming a record requires knowing the corresponding secret key sk. A record consists of a public key, data and a serial number seed. Unlike the ZEXE system, these records lack birth and death predictions due to challenges in function privacy within the account-based blockchain, which avoids increasing system complexity. The record serial number will appear on the smart contract when the record is consumed. Notably, the serial number, generated using a pseudorandom function PRF, combines the serial number seed and the secret key of the owner. This ensures determinism for the serial number when the record is created, but the secret key of the owner prevents unauthorized computation.

Records are consumed through transactions that replace old ones with new ones in a smart contract. As shown in Fig. 3, each transaction contains the consumed record's serial number, commitments of the new record, and a proof attesting to the correctness of the transaction. Record commitments form a Merkle tree in the contract, facilitating efficient zero-knowledge proofs of record commitment existence with a suitable authentication path, which indicates that each serial number can only appear once in a contract, preventing double consumption.

For security, the system's transaction regulation meets regulatory traceability requirements. Similar to [33], the basic idea of the RPSC for achieving traceability is that regulators can collects blockchain data for tracking (See Fig. 4). Transaction details are encrypted using regulatory keys and proofs from zk-SNARKs protocol ensure encrypted data consistency. The smart contract checks the encryption's correctness. To enhance efficiency, transaction details use a symmetric key encrypted under the regulatory public key. In details, Algorithm 1 implements transaction supervision. It generates a zk-SNARKs proof and verifies regulatory ciphertext correctness. The consumed record's serial number seed is encrypted for transaction traceability. The created record's data and public key are encrypted for data and identity traceability. Transaction links are recoverable through serial number seed commitments, allowing extraction of predecessor transaction data and identity for the current transaction.

V. Security Analysis

The proposed RPSC system can satisfy all the security requirements described in Section III-C.

1) **On-chain privacy:** On-chain privacy is composed of transaction privacy and identity anonymity: (1) For transaction privacy, it is required to protect input record, output record, transaction information. The basic idea of the RPSC system comes from that of Zerocash and ZEXE. In brief, the Merkle tree and pseudo-random function hide the details information of input records in the transaction. There is no PPT adversary that can get the input records information from the Merkle tree root and the serial number of the records. The commitment protocol hides the privacy of the output records. The semantic security of the encryption algorithm protects the transaction information. And the zero-knowledge proof guarantees the correctness of the overall transaction information without revealing any additional information. (2) For identity anonymity, it should not determine user's identity from public key and serial number. In brief,

the hiding property of the commitment protocol ensures that no additional information about the public key of a transaction can be recovered from the commitment in the transaction. The security of the pseudo-random function also ensures that a serial number does not reveal any information about the corresponding private key. Therefore, no attacker can obtain any new information about user's identity and thus identity anonymity holds.

2) **Off-chain soundness:** All the operations to generate new data records except $createRecord$ need to spend some existing data records. On one hand, it is required to guarantee the relationship between the new generated data records and the existing ones. The zero-knowledge proof generated by the off-chain computation can provide the proof since no PPT adversary that can forge a ZK-SNARKs proof with malicious data. On the other hand, it should guarantee the identity of the decrypted message and the original if the key is correct. Perfectly correct encryption algorithm can guarantee it as the decrypted message and the plaintext are identical if the key is correct. Therefore, the off-chain soundness of the system holds.
3) **Data record unlinkablity:** For any blockchain node, the only data recorded on the blockchain is the committed value cm_r of the record. When a data record transaction occurs, the user generates a zero-knowledge proof of a Merkle tree to prove that he owns some record in the set of the previous data records. Even if an adversary knows the set of previous data records, he cannot tell which one the current data record actually is. In addition, only the data record serial number sn is appended to the used data record list after each transaction. Due to the properties of the random function PRF, no PPT adversary can link the data record commitment to the corresponding data record serial number. Therefore, the property of data record unlinkablity holds.
4) **Regulatory traceablity:** On one hand, as the commitment scheme is computationally binding, the probability for a PPT adversary to discover another $m^{'}$ satisfying $Com(pp, m) = Com(pp, m^{'})$ negligible. The knowledge soundness of zk-SNARKs ensures that for a false statement, no PPT adversary can convince a verifier with a non-negligible probability. Hence, adversary cannot generate an incorrect or meaningless ciphertext. On the other hand, a regulator can always decrypt the regulatory ciphertext by his private key and get enough information to trace data. In details, the random seed of records used in transaction can be decrypted from the ciphertext. The commitment scheme ensures $Com(pp, seed)$ cannot be changed, and thus the regulator can always find the previous transactions.

VI. Implementation and Evaluation

This section presents a performance analysis and evaluation where we implement and evaluate our system alongside replicates of Hawk [7] and zkay [13] to compare their performance metrics directly. In particular, we present two applications with RPSC to improve the accountability of privacy preserving auction and e-voting. The integration of our RPSC into account-based blockchain applications will result in great performance and fine-grained privacy.

A. Performance Analysis and System Implementation

We first present the efficiency analysis of the RPSC, and then provide our implementation. For clarity, we analyzed the *compute* function and *verify* function (on-chain) for general computation f, and Table II summarizes their communication and time complexity. The parameters in Table II are defined as follows: k represents the number of consumed records; k' represents the number of generated records ; exp_1 represents exponentiation operations in the group G_1; P represents bilinear mapping operations; sym represents to symmetric operations; $|C|$ represents the circuit size of an arbitrary function f.

TABLE II: Performance Analysis of RPSC

compute	*verify*	*communication*
$1\ sym + (O(\|C\|) + 4k' + 2)\ exp_1$	$4P + O(k')\ exp_1$	$k\ Z_p + (5k'+4)G_1 + 1G_2$

In Table II, we only focus on the expensive operations in the scheme, including symmetric operations, exponentiation, and bilinear pairing, which mainly involve encryption and zk-SNARKs, while ignoring the cost of other light computations. For the *compute* function along with the proofs, ct requires symmetric encryption instantiated by *AES encryption*, and public key encryption using *ElGamal encryption*, costing $1\ sym + 2\ exp_1$. Furthermore, CM demands commitments instantiated by *Pedersen commitment*, incurring $4k'exp_1$. Moreover, π is used by zk-SNARKs, leading to a complexity of $O(|C|)exp_1$. The $verify$ algorithm merely utilizes verification in the zk-SNARKs [34], costing $4P + O(k')exp_1$.

Regarding communication overhead, we have k elements of Z_P and k' elements of G_1 from $dataCT$, 2 $G1$ elements from $keyCT$, $4k'$ G_1 elements from the commitment set CM, and 2 G_1 elements and 1 G_2 element from the proof π based on the zk-SNARKs [34].

We implement RPSC on the Ubuntu 20.04 LTS system with Intel(R) Xeon(R) Platinum 8369HB CPU of 3.30GHz and 256-GB RAM. Note that to highlight the efficiency of the system, we implement the prototype of RPSC in two parts, an on-chain implementation and an off-chain implementation. Off-chain operations are implemented in Java to test the cost of local computation, while on-chain operations are implemented in Solidity to test the cost of blockchain involvement.

For off-chain operations, we use the jsnark[3] library to design the arithmetic circuits for the smart contracts, and use the libsnark[4] library to generate zero-knowledge proofs. Here, we implement zk-SNARKs [34] using the pairing-friendly Type-3 alt_bn128, which is also used to implement bilinear maps in Ethereum pre-compiled contracts[5]. For on-chain operations,

[3]https://github.com/akosba/jsnark
[4]https://github.com/scipr-lab/libsnark
[5]https://github.com/ethereum/EIPs/blob/master/EIPS/eip-197.md

TABLE III: Performance for different scenarios

	Register	Auction			Electronic voting			
	createRecord	bidding	finalization		voting		counting	
# Parties	-	-	10	50	10	50	10	50
# Constraints(10^4)	32.75	117.41	542.16	2670.08	547.45	2697.67	316.84	1442.08
KeyGen(s)	6.93	20.11	207.92	900.25	207.65	908.30	114.85	520.89
Prove(s)	60.21	192.23	993.22	4832.71	983.33	4831.47	587.37	2661.33
Verify(ms)	30.6	37.6	60.0	183.1	60.7	182.5	51.0	121.4
ProveKey(MB)	82.18	274.82	1274.50	6333.15	1285.09	6388.18	767.34	3406.46
VerifyKey(KB)	14.81	26.14	87.40	386.46	85.96	385.02	59.79	209.31
Proof(KB)	0.38	0.38	0.38	0.38	0.38	0.38	0.38	0.38
Stmt(KB)	22.82	40.87	138.10	614.35	136.11	612.36	94.44	332.57

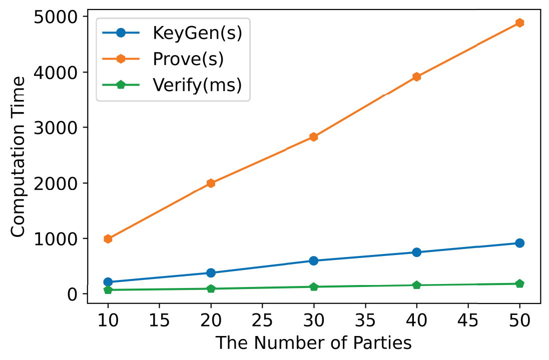


Fig. 5: Time cost of zk-SNARK for different sizes of the finalization circuit in the auction application

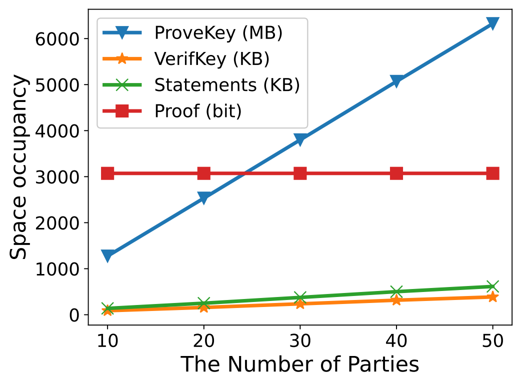


Fig. 6: Storage occupancy of zk-SNARK for different sizes of the finalization circuit in the auction application.

we configure the private network on Ethereum[6] and deploy $verification$ contract written in solidity.

To demonstrate the practicality of RPSC, we implement a blockchain-based privacy preserving e-voting and auction application by integrating RPSC into blockchain, which is one of the most representative applications atop blockchain. Moreover, we replicate Hawk [7] and zkay [13] for auction application in the same experimental configuration and jsnark library, and performe a comparative analysis of the time cost to establish a general benchmark. In short, the e-voting scenario is divided into three main phases, namely the issuance, voting and counting phases. Similar to voting, the auction scenario consists of the issuance of signage, bidding, and finalization phases. Note that the issuance in e-voting and auction are consistent in RPSC by calling the $createRecord$ algorithm. In addition, voting and counting are affected by the size of the application participants in the voting application, similar to the finalization in auctions.

B. Performance Evaluation

Off-chain Evaluation. We evaluate off-chain operations for both e-voting and auction applications, and the results are illustrated in Table III .We focus on evaluating the zk-SNARKs performance since all other computation time is negligible in comparison.

Here we show the time and space cost of off-chain computation, that is, proof generation, for both e-voting and auction applications. As shown in Table III, we set the number of participants to 10 and 50, respectively, to indicate the effect of the number of participants on these two applications. In particular, we have described the workflow of e-voting by invoking the RPSC algorithm in Section 4.4.When the number of participants reaches 10, the time cost of the counting phase in the e-voting is 587.37s. To further demonstrate the impact of the number of participants on the auction, we set the number of participants to increase from 10 to 50 in steps of 5, to test the overhead of time and space. The results for the auction are shown in Fig. 5 and Fig. 6, the time overhead in the finalization phase increases significantly with respect to other computations, as well as the required spatial storage risk. During the computation, larger memory and CPU resources are required, and as the number of constraints increases, the size of the proof key increases, along with the time required for the proof. In addition, to visually demonstrate the $verify$ algorithm, we test the time cost of this algorithm locally, which is also implemented on-chain. The overhead of the $verify$ algorithm is considered cheap in all scenarios. The verification time is mainly related to the size of the public statement, i.e., the public wire value of the circuit. However, the size of the statement is far smaller than the size of the circuit, which results in a small verification time.

The verification key, proof and statement will be public transmitted in the P2P network, which has a limit on the block size. In the implementation, the proof size is fixed at 0.38KB due to the selected zk-SNARKs proposed by Groth [34] in

[6]https://github.com/ethereum/go-ethereum/

TABLE IV: Performance comparison of different schemes for the finalization in auction scenarios with 10 and 50 bidders

# Parties	10				50			
Schemes	Hawk [7]	zkay [13]	Our(Opt.)	Our	Hawk [7]	zkay [13]	Our(Opt.)	Our
# Constraints(10^4)	269.00	85.32	604.34	542.16	1343.87	426.60	2934.38	2670.08
Prove(s)	469.32	140.55	1076.70	993.22	2355.43	702.80	5270.26	4832.71
Verify(ms)	38.9	34.3	42.9	60.0	79.0	76.9	77.2	183.1
ProveKey(MB)	634.34	199.74	1395.72	1274.50	3201.35	970.83	6848.63	6333.15
VerifyKey(KB)	26.96	25.44	28.67	87.40	126.65	125.13	128.36	386.46
Proof(KB)	0.38	0.38	0.38	0.38	0.38	0.38	0.38	0.38
Stmt(KB)	41.86	39.69	44.83	138.10	200.61	198.44	203.58	614.35

TABLE V: Execution gas cost in $verification$ contract for different phases with 10 participants

application	Register	Auction			Electronic voting	
phase	createRecord	bidding	finalization(Y)	finalization(N)	voting	counting
Gas	279,959	356,260	426,695	1,016,972	947,495	755,749

Type-3 alt_bn128 , which only consists of 2 elements in G_1 and 1 element in G_2 for the proof[7]. And the verification key size does not exceed 500KB, which is a very small value based on the current network bandwidth and storage capacity.

As the number of parties increases, the size of the statement increases correspondingly, in part due to the inclusion of the regulatory ciphertexts. To address this, we reduce the statement size by using a hash function to generate a digital digest of the regulatory ciphertexts, which ensures that the space occupied by the ciphertexts within the statement remains constant, specifically equal to the output size of the hash function. The circuit design optimization mentioned above significantly decreases the overhead of on-chain computation, as showed in Table IV, which indicates a marked reduction in both the statement size and on-chain computational costs. While hashing the entire statement optimizes the on-chain overhead, this approach leads to an increased off-chain cost. Therefore, a trade-off must be struck between optimizing statement size and managing off-chain overhead for the practically.

General benchmark. Here we present a general benchmark by comparing our system with Hawk [7] and zkay [13], two of the most prominent frameworks. To obtain a fair and comprehensive comparison, we replicated both frameworks within our experimental environment and conducted comparative analyses of the time costs in the voting scenario, including the $verify$ function (on-chain operation).

As shown in Table IV, we focus on the comparison of our optimized implementation with Hawk [7] and zkay [13]. Notably, the time required for proof generation increases due to the increased circuit size compared to Hawk [7] and zkay [13], which is a consequence of capturing programmable smart contracts with fine-grained privacy and regulatory features in RPSC. Fortunately, the time overhead for $verify$ function (on-chain) shows only a slight increase. Moreover, the size of $VerifyKey$ is only slightly increased compared to Hawk [7] and zkay [13] since the minor expansion in the statement. These observations indicate that our system maintains a similar level of efficiency on-chain compared to Hawk [7] and zkay [13], which is acceptable for the blockchain. As for the increased time cost in proving and the increased size of $ProofKey$, which can be easily mitigated with solutions such as cloud computing, they do not affect the generation and verification of privacy preserving transactions on the blockchain. This balance ensures that while our system introduces some additional overhead off-chain for more functionality, its on-chain operations do not compromise blockchain efficiency.

On-chain Evaluation. Our $verification$ contract focuses on the verification of off-chain operations in two applications, including createRecord, bidding, and finalization in the auction application, and voting, counting in the e-voting application. We set the number of participants to 10 and evaluated the execution gas cost of $verify$ algorithm in the above different phases, as shown in Table V.

At the beginning of these applications, createRecord is invoked to register participants, which costs 279,959 gas to verify the correctness of the records. Subsequently, in the auction, the commitment and proof are uploaded to the blockchain after the participant completes the off-chain bidding. The proof is verified at a cost of 356,260 gas. After the finalization result is uploaded to the blockchain, the proof of finalization is verified, which costs 426,695 gas. Specifically, we take advantage of the above optimization to greatly reduce the on-chain computation, which only requires half. Similar to auctions, in electronic voting, the verification of privacy-preserving voting on the chain costs 947,495 gas. More importantly, the verification of the voting results requires 75,5749 gas, which is acceptable. Note that the verifier key greatly affects the verification cost, resulting in linear field operations and storage overhead on the chain, which is expected.

VII. Conclusion

In this paper, we focus on designing a regulatable privacy-preserving smart contracts (RSPC) system for account-based blockchain. We first present the system model of RPSC and then provided a concrete construction of RPSC based on the two-layer commitment structure and transaction regulation design. Moreover, we demonstrated how the system can satisfy the related security requirements and evaluated the perfor-

[7]Given the parameter settings for Type-3 alt_bn128, each element in G_1 is 96 bytes, and each element in G_2 is 192 bytes.

mance of the prototype, which can meet the need for practical application.

We have also tried to improve the security of the system with cryptographic algorithms that are resistant to quantum attacks, such as lattice-based zero-knowledge proofs. However, the overall efficiency of the system is still not practically applicable. Further reduction of the computation time of the proof is achievable. Moreover, the current regulatory mechanism relies on a trusted government agency. Multi-party computing (MPC) can be applied to decentralize the regulatory private key into multiple parties for joint regulation.

References

[1] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14(4):352–375, 2018.

[2] Jordi Herrera-Joancomartí. Research and challenges on bitcoin anonymity. In *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*, pages 3–16. Springer, 2014.

[3] Tiffany Hyun-Jin Kim and Joshua Lampkins. Ssp: Self-sovereign privacy for internet of things using blockchain and mpc. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 411–418. IEEE, 2019.

[4] Robert Werner, Sebastian Lawrenz, and Andreas Rausch. Blockchain analysis tool of a cryptocurrency. In *Proceedings of the 2020 The 2nd International Conference on Blockchain Technology*, pages 80–84, 2020.

[5] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

[6] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411. IEEE, 2013.

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

[8] Guy Zyskind, Oz Nathan, et al. Decentralizing privacy: Using blockchain to protect personal data. In *2015 IEEE Security and Privacy Workshops*, pages 180–184. IEEE, 2015.

[9] Hisham S Galal and Amr M Youssef. Verifiable sealed-bid auction on the ethereum blockchain. In *International Conference on Financial Cryptography and Data Security*, pages 265–278. Springer, 2018.

[10] Marie Vasek, Micah Thornton, and Tyler Moore. Empirical analysis of denial-of-service attacks in the bitcoin ecosystem. In *International conference on financial cryptography and data security*, pages 57–71. Springer, 2014.

[11] Xu Wang, Xuan Zha, Guangsheng Yu, Wei Ni, Ren Ping Liu, Y Jay Guo, Xinxin Niu, and Kangfeng Zheng. Attack and defence of ethereum remote apis. In *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2018.

[12] Kei Nakagawa and Ryuta Sakemoto. Market uncertainty and correlation between bitcoin and ether. *Finance Research Letters*, 50:103216, 2022.

[13] Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin Vechev. zkay: Specifying and Enforcing Data Privacy in Smart Contracts. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1759–1776, London, UK, 2019. ACM.

[14] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *International Conference on Financial Cryptography and Data Security*, pages 423–443. Springer, 2020.

[15] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 947–964. IEEE, 2020.

[16] Shunli Ma, Yi Deng, Debiao He, Jiang Zhang, and Xiang Xie. An efficient nizk scheme for privacy-preserving transactions over account-model blockchain. *IEEE Transactions on Dependable and Secure Computing*, 18(2):641–651, 2020.

[17] Alex Luoyuan Xiong, Binyi Chen, Zhenfei Zhang, Benedikt Bünz, Ben Fisch, Fernando Krell, and Philippe Camacho. Verizexe: Decentralized private computation with universal setup. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 4445–4462. USENIX Association, 2023.

[18] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. Coinshuffle: Practical decentralized coin mixing for bitcoin. In *European Symposium on Research in Computer Security*, pages 345–364. Springer, 2014.

[19] Sarah Meiklejohn and Claudio Orlandi. Privacy-enhancing overlays in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 127–141. Springer, 2015.

[20] Shen Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098, 2015.

[21] Shi-Feng Sun, Man Ho Au, Joseph K Liu, and Tsz Hon Yuen. Ringct 2.0: A Compact Accumulator-Based (Linkable Ring Signature) Protocol for Blockchain Cryptocurrency Monero. In *Proceedings of the European Symposium on Research in Computer Security*, pages 456–474, Oslo, Norway, 2017. Springer.

[22] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In *International Conference on Financial Cryptography and Data Security*, pages 43–63. Springer, 2018.

[23] N. van Saberhagen. CryptoNote v 2.0, 2013. [Online]. Available: https://cryptonote.org/.

[24] Arijit Dutta, Suyash Bagad, and Saravanan Vijayakumaran. Mprove+: Privacy enhancing proof of reserves protocol for monero. *IEEE Trans. Inf. Forensics Secur.*, 16:3900–3915, 2021.

[25] Carsten Baum, James Hsin-yu Chiang, Bernardo David, and Tore Kasper Frederiksen. Eagle: Efficient privacy preserving smart contracts. In *Financial Cryptography and Data Security - 27th International Conference, FC 2023, Bol, Brač, Croatia, May 1-5, 2023*, volume 13950, pages 270–288. Springer, 2023.

[26] Samuel Steffen, Benjamin Bichsel, Roger Baumgartner, and Martin T. Vechev. Zeestar: Private smart contracts by homomorphic encryption and zero-knowledge proofs. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 179–197. IEEE, 2022.

[27] Samuel Steffen, Benjamin Bichsel, and Martin T. Vechev. Zapper: Smart contracts with data and identity privacy. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2735–2749. ACM, 2022.

[28] Neha Narula, Willy Vasquez, and Madars Virza. zkledger: Privacy-preserving auditing for distributed ledgers. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 65–80, 2018.

[29] Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed Kosba, Ari Juels, and Elaine Shi. Solidus: Confidential distributed ledger transactions via pvorm. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 701–717, 2017.

[30] Jiajun Zhou, Chenkai Hu, Jianlei Chi, Jiajing Wu, Meng Shen, and Qi Xuan. Behavior-aware account de-anonymization on ethereum interaction graph. *IEEE Trans. Inf. Forensics Secur.*, 17:3433–3448, 2022.

[31] Huikang Cao, Ruixuan Li, Wenlong Tian, Zhiyong Xu, and Weijun Xiao. Blockchain-based accountability for multi-party oblivious ram. *Journal of Parallel and Distributed Computing*, 137:224–237, 2020.

[32] Yan Wu, Can Zhang, and Liehuang Zhu. Privacy-preserving and traceable blockchain-based charging payment scheme for electric vehicles. *IEEE Internet Things J.*, 10(24):21254–21265, 2023.

[33] Christina Garman, Matthew Green, and Ian Miers. Accountable privacy for decentralized anonymous payments. In *International Conference on Financial Cryptography and Data Security*, pages 81–98. Springer, 2016.

[34] Jens Groth. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 305–326. Springer, 2016.

[35] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

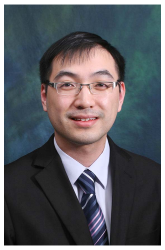
Zoe L. Jiang received the Ph.D. degree from The University of Hong Kong, Hong Kong, in 2010. She is currently a professor with School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. Her research interests include information security and applied cryptography.

Min Xie is currently pursuing the Ph.D. degree with School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. His research interests include zero-knowledge proof, anonymous credential and blockchain technology.

Hanlin Chen received the master's degree from Harbin Institute of Technology in 2022. His research interests during his master's degree included cryptography, network security and blockchain. He is current an software engineer working in Tencent company.

Yijian Pan received the master's degree from Harbin Institute of Technology in 2023. His research interests include applied cryptography, network security and machine learning. He is current an software engineer working in Aliyun company.

Jiazhuo Lyu received the master's degree from Harbin Institute of Technology in 2019. He is currently pursuing the Ph.D. degree with the Department of Computing, The Hong Kong Polytechnic University. His current research interests include cryptography, secure multi-party computation, and blockchain.



Man Ho Au is currently a full professor in the Department of Computing, The Hong Kong Polytechnic University. He has published over 190 refereed papers in top journals and conferences, including IEEE TDSC, TIFS, TKDE, TC and IEEE S&P, ACM CCS, SIGMOD, SOSP, NDSS, CRYPTO, INFOCOM, etc. His research interests include cybersecurity, applied cryptography, blockchain technology, and their applications. He is a recipient of the 2009 PET runner-up award for outstanding research in privacy enhancing technologies.

Junbin Fang is now a professor in Department of Optoelectronic Engineering at Jinan University, Guangzhou, China. His research interests include artificial intelligence security, blockchain, visible light communication and quantum information.

Yang Liu received his D.Phil (PhD) degree in Computer Science from University of Oxford in July 2018. Prior to joining Oxford, He received an MSc in Software Engineering from Peking University and a B.Eng in Computer Science from Ocean University of China. He is currently a Reader in Department of Computer Science, Swansea University, UK. His research interests include security and privacy problems and, in particular, the privacy issues related to mobile and IoT devices.

Xuan Wang received his M.S. and Ph.D. degrees in Computer Sciences from Harbin Institute of Technology in 1994 and 1997 respectively. He is a professor in Harbin Institute of Technology, Shenzhen, China. His research interests include information security, artificial intelligence, and computational linguistics.