Received May 10, 2018, accepted June 11, 2018, date of publication June 15, 2018, date of current version August 28, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2847632

BeeKeeper: A Blockchain-Based IoT System With Secure Storage and Homomorphic Computation

LIJING ZHOU, LICHENG WANG, YIRU SUN, AND PIN LV
State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China

Corresponding author: Licheng Wang (wanglc2012@126.com)

This work was supported in part by the National Key Research and Development Program under Grant 2016YFB0800602, in part by the National Natural Science Foundation of China (NSFC) under Grant 61502048, and in part by the Shandong provincial Key Research and Development Program of China under Grant 2018CXGC0701 and Grant 2018GGX106005.

ABSTRACT Currently, Internet of Things (IoT) and blockchain technologies are experiencing exponential growth in academia and industry. Generally, IoT is a centralized system whose security and performance mainly rely on centralized servers. Therefore, users have to trust the centralized servers; in addition, it is difficult to coordinate external computing resources to improve the performance of IoT. Fortunately, the blockchain may provide this decentralization, high credibility and high security. Consequently, blockchain-based IoT may become a reasonable choice for the design of a decentralized IoT system. In this paper, we propose a novel blockchain-based threshold IoT service system: BeeKeeper. In the BeeKeeper system, servers can process a user's data by performing homomorphic computations on the data without learning anything from them. Furthermore, any node can become a leader's server if the node and the leader desire so. In this way, BeeKeeper's performance can continually increase by attracting external computing resources to join in it. Moreover, malicious nodes can be scrutinized. In addition, BeeKeeper is fault tolerant since a user's BeeKeeper protocol may work smoothly as long as a threshold number of its servers are active and honest. Finally, we deploy BeeKeeper on the Ethereum blockchain and give the corresponding performance evaluation. In our experiments, servers can generate their response with about 107 ms. Moreover, the performance of BeeKeeper mainly depends on the blockchain platform. For instance, the response time is about 22.5 s since the block interval of Ethereum blockchain is about 15 s. In fact, if we use some other blockchain with short block interval, the response time may be obviously short.

INDEX TERMS IoT, blockchain, secret sharing, secure multi-party computing.

I. INTRODUCTION

With the emergence of the Internet of Things (IoT), the number of innovative applications is rapidly increasing [1]. Current IoT systems generally consist of designated low-power and lightweight devices with sensors. The devices are responsible for collecting data from the surrounding environment and may exchange data with other devices, servers or platforms. In conventional IoT systems, the collected data, which may be used in future processes, are stored in centralized cloud servers. Thereby, users have to trust that the centralized servers protect their private and sensitive data, which are generally unencrypted. Despite the indisputable benefits provided by these services, centralization IoT systems might face the following challenges:

- If the centralized servers stop functioning properly, the entire system faces the risk of becoming paralyzed [2]. Moreover, the system may suffer from DoS attacks.
- Generally, most data stored in servers are unencrypted. Therefore, users reveal sensitive data, such as identity, healthcare, behavior and private life information, to the corresponding servers [5]. Moreover, this can lead to an illegitimate use of personal information (as demonstrated by the Snowden incidence).
- A huge volume of data streams is produced by IoT devices at high speeds. According to a recent report by Gartner [3], about one million new IoT devices may be sold every hour by 2021. However, centralized servers may be neither sufficiently efficient nor sufficiently

scalable to address these enormous amounts of IoT data.

- Data stored by servers have a risk of being modified or deleted by the servers. In addition, users only have limited control over how their data are used or processed.

Blockchain, which is first proposed by Bitcoin [6], is a tamper-resistant timestamp ledger of blocks that is utilized to share and store data in a distributed manner. Blockchain has attracted enormous attention from academics and practitioners (e.g., in computer science, finance and law) due to its excellent properties such as decentralization, security, anonymity, privacy and tamper resistance [4]. Recently, blockchain has been widely utilized in non-monetary applications, including securing robotic swarms [7], verifying proof of location [8] and the sharing of healthcare data [9]. Due to the features of blockchain, blockchain may be a good solution to resolving problems facing centralized IoT. The advantages of blockchain for improving conventional IoT systems are described as follows:

- **Decentralization.** Blockchain decentralization denotes the following: (i) there are no centralized nodes controlling the system, (ii) blockchain nodes may freely join in or leave the blockchain network, and (iii) nodes may communicate with others directly without any third-party involvement. Therefore, a blockchain-based IoT may conveniently attract external nodes to join in the system to improve the system's performance. Moreover, blockchain-based IOT can effectively resist DOS attacks.
- **Collective verification and tamper resistance.** Collective verification means that all publicly verifiable data of a transaction should be verified by all record nodes before this transaction is recorded in the blockchain. Tamper resistance indicates that once some data have been recorded in the blockchain (based on PBFT consensus [22]), the data cannot be modified or deleted. Consequently, blockchain may provide high credibility to users and devices of IoT.
- **Privacy.** Some blockchain platforms provide anonymity and amount confidentiality (e.g., Zcash [23] and Monero [24]). The two aspects provide strong privacy to users. Therefore, blockchain-based IoT helps users and devices hide sensitive data such as identity and balance information.
- **Smart contracts** Blockchain offers a functionality of smart contracts [25], which are programs recorded on the blockchain and can be triggered by events. Thereby, smart contracts can assist devices in being more intelligent since the behavior of an IoT device can be specified by smart contracts.

In daily life, there is an interesting career called a beekeeper. Specifically, bees are mainly responsible for honestly bringing nectar from flowers to their beehive. The beekeeper may gather honey from the beehive at a suitable time, although he does not have to care about how the "beehive" processes the nectar into honey. Furthermore, the "beehive" also does not have to care about who will gather the honey and who sends nectar to the beehive. However, the "beehive" wishes to help the beekeeper to process the nectar since it can obtain some reward from the process. In this process, each party may perform simple tasks and only needs to focus on its own task.

Similar to the above process, we propose a blockchain-based threshold IoT system: BeeKeeper. In a basic instance of this system, there are three parties: a leader, the leader's devices and a certain number of servers. The leader is similar to the "beekeeper". Devices act like the "bees". Servers and the blockchain network are analogous to the "beehive". In this system, any blockchain node may become one of the leader's servers if both the node and leader desire so. Furthermore, servers can obtain a certain amount of reward according to their efforts in processing the leader's data. Moreover, BeeKeeper's users store data in the blockchain rather than in cloud servers.

A. OUR CONTRIBUTIONS

In summary, the contributions of this paper are as follows:

1) We propose a threshold secure multi-party computing (TSMPC) protocol. In TSMPC, servers may perform homomorphic computations on shares and then generate responses, although servers cannot learn anything from the shares. In addition, the leader may recover the desired result by collecting a threshold number of correct responses. Moreover, malicious nodes can be checked out since shares and responses are verifiable. Finally, the protocol may perform smoothly as long as a certain number of servers are active and honest.
2) We propose a blockchain-based threshold IoT system based on TSMPC: BeeKeeper. BeeKeeper has several additional features. First, if some data have been recorded in the blockchain, the data cannot be modified or deleted, and the publicly verifiable part of the data is credible. Second, nodes perform significantly less verification than TSMPC. Third, any node may become a leader's server if both the node and leader desire so. To the best of our knowledge, this is the first work to design a blockchain-based IoT system where servers may help users to process encrypted data without learning anything from the data. Fourthly, BeeKeeper may conveniently attract external computing resources to join in the system to improve the system performance. Finally, a server may obtain a certain amount of reward according to its efforts in processing data. Moreover, the clients of the leader, devices and servers do not have to keep a large amount of memory or use significant computing resources.
3) A prototype system is implemented to evaluate the feasibility of BeeKeeper. The prototype system is built upon the Ethereum private blockchain with four record nodes. Moreover, we use the transaction simulator to

simulate the leader, servers and devices to generate and send transactions to evaluate the performance of BeeKeeper.

B. ORGANIZATION

The remainder of the paper is organized as follows. An overview of BeeKeeper is given in Sec. III. Sec. IV briefly introduces the preliminaries. The system setting and model are discussed in Sec. V. We describe the construction of the BeeKeeper system in Sec. VI. A scenario is set up for the performance evaluation in Sec. VII. Finally, a short conclusion is given in Sect. VIII.

II. RELATED WORK

Currently, blockchain-based IoT is a hot topic. Previous works have mainly focused on four aspects: authentication (or access control), smart applications, data storage (or the integrity of transferred data) and cloud computing (or edge computing). To the best of our knowledge, previous papers did not include two aspects: (i) They did not realize that servers may perform homomorphic computations on encrypted data without decrypting the data. Moreover, in previous papers, data transferred to servers are typically unencrypted or servers (or computing nodes) can decrypt the transferred data. This leads to a risk of releasing sensitive information since the servers may learn the details of the received data. (ii) They also did not realize that external computing resources can conveniently join in the blockchain-based IoT system to improve the system performance. In greater detail, servers should perform some authentication from authorities before joining the system. Previous works are summarized as follows:

- **Authentication.** Authorities use blockchain and smart contracts to perform authentication or issue credentials to users or devices [10]–[13]. Sonnino *et al.* [13] used public and private attributes to control who has the ability to use the credentials. Specifically, only users, who own private and public attributes included in the credential, may use the credentials. In addition, smart applications were investigated in [1], [14], and [15] with the blockchain and smart contract. Lamichhane [14] designed a Decentralized Autonomous Organization (DAO) [25] to intelligently manage waste with smart contracts. While Dorri *et al.* [1], [15] proposed a partial centralized scheme since they used the private blockchain to record data.
- **Integrity.** In [16]–[19], the blockchain was used as a storage tool of IoT systems to record data that may be plaintext, ciphertext or hash values. In particular, Rahulamathavan *et al.* [16] used attribute-based encryption to ensure that encrypted data can be decrypted and verified only by specific miners or users that possess selected attributes. However, this protocol is partly centralized since attribute authorities are responsible for generating system parameters and generating attributes for users and miners. Therefore, this paper might face a risk in that the attribute authorities may decrypt all data. In addition, although the blockchain only records hash values of transferred data, the data receivers may decrypt the data to learn details about the data. In [17]–[19], the blockchain was simply used to prove the integrity of transferred data. Moreover, the data, which are typically unencrypted, are stored in receiver locations via a peer-to-peer file storage protocol. Briefly, [16]–[19] suffered a risk of releasing sensitive information.
- **Cloud computing and edge computing.** Sharma *et al.* [20] and Pahl *et al.* [21] proposed protocols wherein cloud servers or fog nodes can help users process data, and the blockchain was used to ensure the integrity of the transferred data. However, because that data, stored by servers, are unencrypted or can be decrypted by receivers, [20], [21] also faced the risk of releasing sensitive information.

III. AN OVERVIEW OF BeeKeeper

In a basic instance of the BeeKeeper system, a leader may perform a (t, n)-threshold BeeKeeper protocol among n servers, and it completely controls a certain number of devices (The record node is the full node who maintains a full copy of blockchain. While server, leader and device are light-clients of blockchain. All parties communicate with each other via transactions of blockchain). Specifically, the devices may send encrypted data to the blockchain, and each of the servers has the ability to perform homomorphic computations on the encrypted data. However, the servers cannot learn anything about the encrypted data as long as more than $n - t$ servers are honest. When the leader wants to obtain a result that can be calculated by the encrypted data, it will send a query to the servers via blockchain transactions. After this process, active servers may generate responses with the encrypted data according to the query. Then, they send encrypted responses to the leader via blockchain transactions. At this point, only the leader can decrypt the encrypted responses since only it has the corresponding decryption key. Finally, if the leader can collect at least t correct responses, it can recover the desired result. Then, the leader's smart contract will automatically return some reward to the corresponding servers. An overview of BeeKeeper's working process is described in Fig.1. The features of BeeKeeper are summarized as follows:

- **Decentralization.** There are no third-party authorities to provide any authentication. Moreover, any node may become a leader's server if both the node and the leader desire so. In addition, data are stored on the blockchain rather than in cloud servers.
- **Confidentiality, homomorphism and threshold.** In a basic instance, a leader may implement a (t, n)-threshold BeeKeeper protocol among n servers, and the leader's devices store encrypted data on the blockchain. First, the encrypted data are always confidential if more than $n - t$ servers are honest. Second, if t servers honestly respond to the leader's query by performing

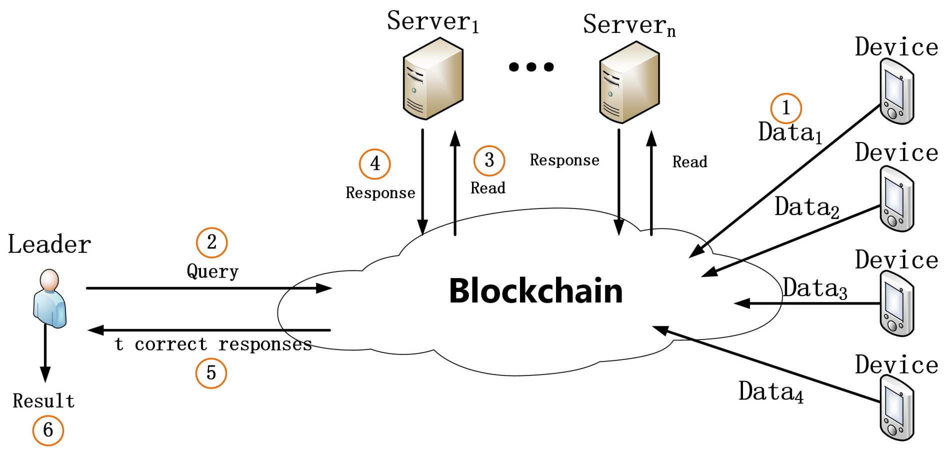


FIGURE 1. **An overview of BeeKeeper. After an initialize transaction has been presented in the blockchain, the users perform the following. Step 1: devices send record transactions to the blockchain network. Step 2: The leader sends a query transaction to query some result. Step 3: The servers read a query transaction and related record transactions from the blockchain. Step 4: After locally computing, the servers generate their responses and then send respond transactions to the blockchain network. Step 5: The leader collects respond transactions and obtains *t* correct responses. Step 6: The leader recovers the result with the *t* correct responses.**

homomorphic computations on the encrypted data, then the leader can recover his desired result with corresponding t responses. Finally, no one (including the leader) can learn anything with fewer than t responses. Moreover, the system is such that even if some of a user's servers are off-line, damaged or malicious, the user's protocol may still perform correctly as long as t servers are active and honest.

- **Verifiability or public verifiability.** Key data stored in the blockchain are verifiable. The verification can be off-chain computation or can be performed by smart contract. Specifically,
  - Anyone can verify that the verification key is valid.
  - A server can verify whether his core share is correctly computed by the corresponding leader.
  - The leader can verify whether responses are correctly computed by corresponding servers.
- **Credibility.** Before a transaction is recorded in the blockchain, record nodes may verify all publicly verifiable data of the transaction. Consequently, once a transaction has been recorded on the blockchain, the transaction's publicly verifiable data are credible. Moreover, due to the blockchain's tamper-resistant nature, data cannot be modified or deleted if they have been recorded on the blockchain.
- **Lightweight.** Indeed, record nodes of BeeKeeper are heavy since they are the full nodes who maintains a full copy of blockchain. However, the leader, leader's devices and servers are lightweight. Specifically, they do not require substantial memory and computation resources since (i) all related data are recorded on the blockchain, (ii) most verification computations are performed by the blockchain's record nodes, and (iii) servers help the leader process encrypted data to obtain the desired results.

IV. PRELIMINARIES

Bitcoin [6] is a decentralized payment scheme whereby every participant maintains its own local copy of the whole transaction history, called the blockchain, which is a ''chain'' of ''blocks''. The blockchain is maintained by anonymous record nodes, called miners, by executing the PoW that extends the blockchain. In Bitcoin, payers broadcast transactions, and record nodes collect transactions into their local blocks. A block contains two parts: a block body and a block header. Specifically, a block body contains transactions, while a block header contains the following: the hash value of the previous block, current Unix time, target value, nonce and merkle root of the transactions. The record nodes are connected by a reliable peer-to-peer network. Bitcoin consistency relies on the idea of computational puzzles—a.k.a. the moderately proof-of-work concept presented by Dwork and Naor [28]. Specifically, in Bitcoin, the computational puzzle is the following: a block is valid if its block header's cryptographic hash value is smaller than the preselected target value. To address the computational puzzle, each record node continually changes the nonce until it finds a solution (nonce) satisfying the computational puzzle. If a record node finds a solution to the cryptographic puzzle first, then its block becomes the new block of the blockchain. Because this process is very hard as well as the winner of each block competition may obtain a large reward; therefore, the record nodes of Bitcoin are also called miners, and the competition process is called mining. Then, when a miner finds a solution for a new block first, it will immediately broadcast its block including the solution to other nodes. Next, upon verifying the block, others will accept the block and add this block as a new block to its local blockchain. Then, all miners continue the mining process on their updated blockchain. The creator of the block is rewarded with bitcoins (the coins of Bitcoin) via the coinbase transaction. Consequently, bitcoins are created and distributed among miners. Moreover, this creator is also rewarded with the transactions fees of all transactions included in the block created by the creator. Furthermore, Bitcoin assumes that a majority of computational power is controlled by honest players.

The smart contract was proposed by Nick [27] in 1994. A smart contract is a user-defined program code that represents the implementation of a contractual agreement. With smart contracts, contractual parties may structure their relationships efficiently in a self-executing manner. A smart contract's code and its state are stored on the blockchain, and it is enforced by record nodes, who update the contract's state on the blockchain accordingly. The smart contract will be invoked whenever it receives a corresponding coin (or data) from a user or another smart contract. Furthermore, a user (or smart contract) may either receive messages (coins or data) from a smart contract or send coins (or data) to a smart contract.

Specifically, a smart contract consists of a storage file, program code and an account balance. Any user may generate a smart contract by sending a transaction to the blockchain network. Then, the smart contract cannot be modified or deleted once it has been recorded in the blockchain. For instance, a user may send coins (or data) to a smart contract by including the message and the address of the contract in its transaction. Then, the contract can send coins (or data) to the user (or another user) or another smart contract using a special instruction in its program code.

In the present paper, before the leader and servers begin operating, they should mortgage coins in smart contracts. Moreover, if a leader or server sends some inaccurate data, then anyone may input the corresponding evidence in the ''wrongdoer''s smart contract. After that, the finder may obtain a reward from the ''wrongdoer''s smart contract.

A. TRANSACTION AND BLOCK

In the blockchain, a transaction contains two parts: the *transaction header* and the *payload*. The transaction header and payload are shown in Table 1.

The structure of a block used in BeeKeepr can be described in Table 2.

TABLE 1. Format of transaction.

Transaction Header	
Hash	The transaction's hash value
Block number	Block containing the transaction
Order	The transaction's number in the block
Timestamp	Creation time of the transaction
Sender	Sender's ID
Receiver	Receiver's ID
Signature	Sig{the transaction's hash value}
Payload	
$\text{data}_1,\text{data}_2,\cdots,\text{data}_n$	

TABLE 2. Format of block.

Block Header	
Name	*Description*
Version	Block version number
Hash	The block's hash value
Parent hash	The previous block's hash value
Difficulty	The proof-of-work target difficulty
Timestamp	Creation time of the block
Merkle root	The root of Merkle Tree of transactions
Nonce	A random counter for proof-of-work
Block Body: Transactions	
$\text{Transaction}_1,\text{Transaction}_2\cdots,\text{Transaction}_n$	

V. SYSTEM SETTING AND MODEL

In this section, we introduce the system setting and model. We will first describe *Blockchain Network and Cryptographic Keys* used in the system.

A. BLOCKCHAIN NETWORK AND CRYPTOGRAPHIC KEYS

BeeKeeper consists of record nodes and light nodes. Specifically, all record nodes are connected by a reliable peer-to-peer network, and each light node connects to a certain number of record nodes. Record nodes are responsible for maintaining the blockchain via a Practical Byzantine Fault-tolerance (PBFT) consensus scheme and storing the entire blockchain list. Time is divided into epochs. In an epoch, record nodes collect and verify transactions sent to the blockchain network, and they record valid transactions in their local blocks. By performing the PBFT consensus scheme, some record node's block will become the valid block of the epoch. Then, all record nodes will join in the next epoch to build the next block. However, light nodes store all block headers, rather than the entire blockchain list.

Moreover, in BeeKeeper, there is no trusted public key infrastructure. Specifically, any node can generate an arbitrary number of key-pairs by itself. In the blockchain network, all users communicate with each other via transactions on the blockchain. Additionally, each record node can poll a random oracle [31] as a random bit source. Moreover, by mortgaging a certain amount of coins with a smart contract, a light node can become a leader or server with his address. Moreover, the security of the shares of our system relies on the security of Shamir's (t, n)-secret sharing (SSS) [29]. Specifically, we extend SSS to obtain a threshold secure multi-party computing (TSMPC) protocol that will be described in the Appendix, and the security of TSMPC is based on SSS.

In the implementation of BeeKeeper, we utilize secp256k1-based [33], which is an elliptic curve, ECDSA [34] as the signature scheme $\mathsf{Sig}(\cdot)$, secp256k1-based ECIES [35] as the encryption scheme $\mathsf{Enc}(\cdot)$ and SHA-256 [6] as the hash function H(·). In addition, we use elliptic curve point multiplication to compute commitments and use pairing computations [32] to verify the validations of commitments. Specifically, we utilize a high-speed elliptic curve library [30] to compute them with the 254-bit Barreto-Naehrig curve (BN-curve) [32], which can provide 128 bits of security.

Moreover, a node is honest if it follows all protocol instructions and is perfectly capable of sending and receiving information. On the other hand, a node is malicious if it can deviate arbitrarily from protocol instructions. Finally, in a blockchain system, all users communicate with each other via transactions of blockchain, and they only trust messages presented at blockchain.

B. ASSUMPTIONS

According to the PBFT [26] consensus scheme, the PBFT-based blockchain does not fork if at least $\frac{2}{3}$ of the record nodes are honest. To obtain a non-forked blockchain, we utilize the PBFT-based blockchain in the BeeKeeper system and assume that at least $\frac{2}{3}$ of the record nodes are honest. Therefore, once a transaction has appeared in the PBFT-based blockchain, the transaction cannot be modified or deleted. Moreover, we assume that the digital signature Sig(·), encryption scheme Enc(·) and hash function H(·) used in BeeKeeper are ideal such that no one can violate Sig(·), Enc(·) or H(·). In addition, we assume that the leader and servers are partially trusted. Specifically, the data sent by them should be verified; otherwise, the data are not credible. Finally, we assume that a leader may completely control his devices. In other words, the leader and his devices trust each other.

C. IoT DEVICE

In the BeeKeeper system, a device has limited storage and energy. Therefore, it is not suitable for storing the entire blockchain or all block headers. In this paper, a device's tasks are described as follows:

- Collect original data from out-of-system.
- Encrypt the original data.
- Generate transactions including the encrypted data.
- Send the transactions to the blockchain network.

VI. BeeKeeper

In this section, we present how BeeKeeper works. We will first describe *transactions* used in the system.

A. TRANSACTIONS

The payload of transaction might contain secret or public data that may be used in verifications or computations. According to the payload, transactions can be divided into 4 types, *initialize transaction, record transaction, query transaction and respond transaction*, which can be described by $T_{initialize}$, T_{record}, T_{query} and $T_{respond}$ as follows:

$T_{initialize}$
Transaction Header
Payload
VK C_{score}
$Server_1$'s ID C_{CF_1} C_{Ch_1} CM_{CF_1} CM_{Ch_1}
$Server_2$'s ID C_{CF_2} C_{Ch_2} CM_{CF_2} CM_{Ch_2}
$\cdots$
$Server_n$'s ID C_{CF_n} C_{Ch_n} CM_{CF_n} CM_{Ch_n}

T_{record}
Transaction Header
Payload
confidential data
confidential data
$\cdots$
confidential data

T_{query}
Transaction Header
Payload
$\text{ID}_{function}$ $\text{ID}_{T_{record}}$
$\text{ID}_{function}$ $\text{ID}_{T_{record}}$
$\cdots$

$T_{respond}$
Transaction Header
Payload
$\text{ID}_{T_{query}}$
CM_{Resp_i}
C_{Resp_i}

Time is also divided into epochs. In each epoch, record nodes will generate a block belonging to the epoch via the selected consensus scheme. In addition, record nodes are responsible for verifying all publicly verifiable data of transactions before the transactions are included in the blockchain. If any publicly verifiable data are invalid, then honest record nodes will reject the corresponding transactions. The transaction will then not be included in the blockchain. Moreover, because we adopt the Practical Byzantine Fault-tolerance consensus scheme, if a transaction has been presented to the blockchain, then all nodes can consider that the transaction's publicly verifiable data are credible.

Furthermore, record nodes may perform two types of verifications on transactions. The first type is *basic verification*, which should be performed on all transactions. Basic verifications have the following requirements:

- The transaction should be well-formed.
- The transaction's inputs should have not been used previously.
- The transaction's signature should be valid.
- The sum of the input coins should be equal to the sum of the output coins.

In addition to basic verifications, record nodes may perform *payload verifications* for initialize transactions and respond transactions. This means that in the payloads of initialize transactions and respond transactions, there are publicly verifiable data that can be verified by record nodes. If a transaction is found on the blockchain, then the record nodes have accepted the transaction's publicly verifiable data. Therefore, the transaction's receiver can assume that the transaction's publicly verifiable data are credible. Thus, the receiver simply needs to perform some other verifications that can be performed only by the receiver. In this way, most of the verification computations are performed by record nodes, which helps to decrease the servers' and leader's verification computations significantly. Fig.2 describes the verifications of initialize transactions, record transactions, query transactions and respond transactions.

B. CONSTRUCTION OF BeeKeeper

To clearly introduce the Beekeeper system, in this subsection, we describe a basic instance that contains a leader, the leader's device and n servers. Specifically, $Sr_1, Sr_2, \cdots, Sr_n$ denote the IDs of the n servers, ID_L is the leader's ID, and

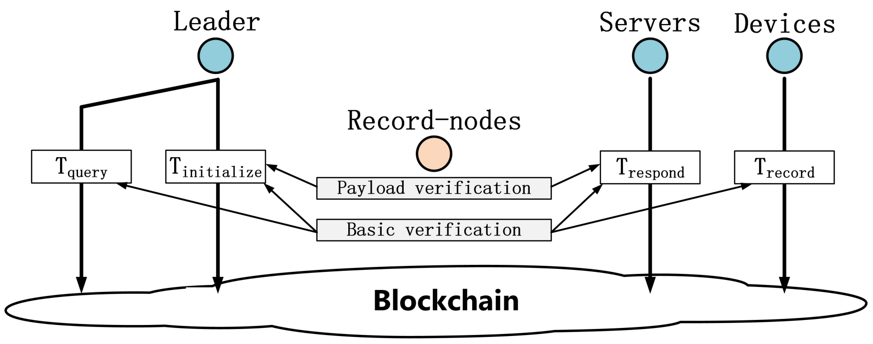


FIGURE 2. Record node verifications in BeeKeeper. All transactions must be verified by record nodes before they are written in the blockchain. For query transactions and record transactions, record odes simply need to perform basic verifications. Fr initialize transactions and respond transactions, record nodes should perform basic verifications and payload verifications.

TABLE 3. Symbols of BeeKeeper.

Symbol	Description
g	the generator of a cyclic group $\mathbb{G}$
e	the bilinear map, e: $\mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}$.
$\mathbb{F}_p$	the finite field with character p
ID_L	the leader's ID
Sr_i	the i-th server's IDs
ID_D	the device's ID
VK	the verification key
ID_{T_i}	the transaction T_i's ID
$\{pk_L, sk_L\}$	the $leader$'s key pair
$\{pk_i, sk_i\}$	the i-th server's key pair, for i from 1 to n
$\{pk_D, sk_D\}$	the $device$'s key pair
d_i	the i-th plaintext message protected by n servers
$\{CF_i, Ch_i\}$	the i-th server's core share
$Resp_i$	the i-th server's response

ID_D describes the device's ID. Essentially, more complex instances, containing multiple devices, can be constructed with a basic instance. The symbols used in the paper are shown in Table 3.

At first, the leader and n servers should have published smart contracts to mortgage a certain amount of guarantee coins on the blockchain. If someone sends inaccurate data that are verifiable, then the discoverer can send his evidence to the bumbler's smart contract to prove that the bumbler sent an inaccurate data. Then, the discoverer can automatically obtain a reward from the bumbler's smart contract.

In addition, the leader should publish a certain number of *pre-functions* and the *pre-functions' IDs*. Specifically, the pre-functions will be used to tell servers how the servers compute on the encrypted data. Moreover, the pre-functions' IDs are used to index the pre-functions.

After mortgaging guarantee coins and publishing pre-functions and the functions' IDs, the Beekeeper system can be performed as follows:

- **Step 1: Initialization.** The leader randomly samples a polynomial $F(x)$ of degree $t-1$ from $\mathbb{F}_p[x]$ as the following polynomial.

$$F(x) = a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + \cdots + a_1 x + s_{core},$$

where $s_{core}, a_1, \cdots, a_{t-1} \in \mathbb{F}_p$ and $a_{t-1} \neq 0$. Let

$$f(x) = a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + \cdots + a_1 x.$$

Then, we have $F(x) = f(x) + s_{core}$. The leader computes

$$f(x)^2 = q_{2t-2}x^{2t-2} + q_{2t-3}x^{2t-3} + \cdots + q_2 x^2.$$

After that, the leader randomly samples $l(x)$ of degree $t-1$ from $\mathbb{F}_p[x]$ as follow:

$$l(x) = c_{t-1}x^{t-1} + c_{t-2}x^{t-2} + \cdots + c_1 x.$$

Let

$$\begin{aligned} h(x) &= f(x)^2 - l(x) \\ &= b_{2t-2}x^{2t-2} + b_{2t-3}x^{2t-3} + \cdots + b_1 x. \quad (1) \end{aligned}$$

Then, the leader generates a verification key VK as follow:

$$VK = \{g, g^{a_{t-1}}, \cdots, g^{a_1}, g^{s_{core}}, g^{b_{2t-2}}, g^{b_{2t-3}}, \cdots, g^{b_1}, g^{c_{t-1}}, g^{c_{t-2}}, \cdots, g^{c_1}\}$$

For i from 1 to n, the leader does as follows:

– Compute $CF_i = F(Sr_i)$ and $Ch_i = h(Sr_i)$.
– Compute $CM_{CF_i} = g^{CF_i}$ and $CM_{Ch_i} = g^{Ch_i}$, which will be used to verify the correctness of CF_i and Ch_i.
– Encrypt CF_i, Ch_i with $Server_i$'s public key pk_i into $C_{CF_i} = Enc_{pk_i}(CF_i)$ and $C_{Ch_i} = Enc_{pk_i}(Ch_i)$ via ECIES.
– Encrypt s_{core} with the device's public key pk_D into $C_{s_{core}}$ via ECIES.

CF_i and Ch_i are $Server_i$'s *core share*. Moreover, for CF_i and Ch_i, only $Server_i$ can decrypt them since only S_i has the corresponding secret key sk_i. Then, the leader generates an initialize transaction $T_{initialize}$ as follows:

$T_{initialize}$				
Transaction Header				
Payload				
VK			C_{score}	
Sr_1	C_{CF_1}	C_{Ch_1}	CM_{CF_1}	CM_{Ch_1}
Sr_2	C_{CF_2}	C_{Ch_2}	CM_{CF_2}	CM_{Ch_2}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
Sr_n	C_{CF_n}	C_{Ch_n}	CM_{CF_n}	CM_{Ch_n}

After that, leader sends $T_{initialize}$ to blockchain network.

- **Step 2: Record node verify** $T_{initialize}$. Honest record nodes should verify all new distribute transactions before appending them to the blockchain. Specifically, when a record node receives a $T_{initialize}$, it will verify the transaction's verification key (VK) at first and then verify other data with the VK. If $T_{initialize}$ passes the verifications, then the record node accepts the $T_{initialize}$ and writes it to his local block; otherwise, the record node will reject $T_{initialize}$. The verifications are described as follows:
  - First, verify the verification key VK. The record node verifies whether the polynomials $f(x)$, $h(x)$ and $l(x)$, committed in the verification key, are well-formed. Specifically, the record node does as follows:
    - Randomly sample a number $x_0 \in \mathbb{F}_p$.
    - Compute

$$\begin{aligned} g_1 &= (g^{a_{t-1}})^{x_0^{t-1}} (g^{a_{t-2}})^{x_0^{t-2}} \cdots (g^{a_1})^{x_0} \\ &= g^{a_{t-1}x_0^{t-1}+\cdots+a_1x_0} \\ &= g^{f(x_0)} \\ g_2 &= (g^{b_{2t-2}})^{x_0^{2t-2}} (g^{b_{2t-3}})^{x_0^{2t-3}} \cdots (g^{b_1})^{x_0} \\ &= g^{b_{2t-2}x_0^{2t-2}+\cdots+b_1x_0} \\ &= g^{h(x_0)} \\ g_3 &= (g^{c_{t-1}})^{x_0^{t-1}} (g^{c_{t-2}})^{x_0^{t-2}} \cdots (g^{c_1})^{x_0} \\ &= g^{c_{t-1}x_0^{t-1}+\cdots+c_1x_0} \\ &= g^{l(x_0)} \end{aligned}$$

    - It is easy to see that

$$\begin{aligned} e(g_1, g_1) &= e(g^{f(x_0)}, g^{f(x_0)}) \\ &= e(g^{f(x_0)^2}, g), \ e(g_2g_3, g) \\ &= e(g^{h(x_0)}g^{l(x_0)}, g). \end{aligned}$$

    If

$$e(g_1, g_1) = e(g_2g_3, g); \qquad (2)$$

    then, the record node accepts that $f(x)$, $h(x)$ and $l(x)$ satisfy relationships and forms mentioned in Eq.1; otherwise, it rejects $T_{initialize}$ and stops his verifications.
  - Second, verify commitments CM_{CF_i}, CM_{Ch_i}, i from 1 to n. Specifically, the record node computes as follows:
    - Compute

$$\begin{aligned} CF_i^* &= (g^{a_{t-1}})^{Sr_i^{t-1}} \cdots (g^{a_1})^{Sr_i}(g^{s_{core}}) \\ Ch_i^* &= (g^{b_{2t-2}})^{Sr_i^{2t-2}} \cdots (g^{b_1})^{Sr_i} \end{aligned}$$

    - If

$$CF_i^* = CM_{CF_i} \text{ and } Ch_i^* = CM_{Ch_i}; \qquad (3)$$

    then, the record node accepts that CM_{CF_i} and CM_{Ch_i} are correctly computed by the leader; otherwise, it rejects $T_{initialize}$ and stop his verifications.

  If any data cannot pass corresponding verification, then the record node rejects the $T_{initialize}$.

Remark 1: Because the record node randomly samples the number x_0, Eq.2 and Eq.3 are sufficient to prove the validation of the verification key.

- **Step 3: Servers verify core shares.** i from 1 to n, when $Server_i$ sees $T_{initialize}$ at the blockchain, the server only needs to perform the following computations:
  - Decrypt C_{CF_i} and C_{Ch_i}. Then, the server obtains CF_i and Ch_i.
  - If

$$CM_{CF_i} = g^{CF_i} \text{ and } CM_{Ch_i} = g^{Ch_i},$$

  then the server accepts that $T_{initialize}$ is valid; otherwise, it can send its evidence (include $\text{ID}_{T_{initialize}}$, CF_i and Ch_i) to the leader's smart contract. After sending this evidence via a transaction, the server can obtain a reward from the leader's smart contract.
- **Step 4: Record.** After seeing $T_{initialize}$, the leader's device decrypts $C_{s_{core}}$ and then obtains the correct s_{core} since it possesses the leader's private key. Then, the device generates record transactions that are $T_1, T_2, \cdots, T_u$. Specifically, i from 1 to u, let $d_{i,1}, d_{i,2}, \cdots, d_{i,m_i}$ denote the device's i-th set of data that will be recorded in T_i. Then, with i from 1 to u and j from 1 to m_i, the device computes

$$s_{i,j} = d_{i,j} - s_{core}.$$

  Then, it generates transactions $T_1, T_2, \cdots, T_u$ as follows:

T_1		
Transaction Header		
Payload		
$s_{1,1}$ $s_{1,2}$	$\cdots$	s_{1,m_1}

T_2		
Transaction Header		
Payload		
$s_{2,1}$ $s_{2,2}$	$\cdots$	s_{2,m_2}

$\cdots\cdots$

T_u		
Transaction Header		
Payload		
$s_{u,1}$ $s_{u,2}$	$\cdots$	s_{u,m_u}

  Next, the device sends $T_1, T_2, \cdots, T_u$ to the blockchain network.
- **Step 5: Query.** When the leader wants to obtain a result $DATA$ that can be computed with the payloads of

$T_1, T_2, \cdots, T_u$, it can generate and send a query transaction T_{query} containing corresponding record transactions' IDs and pre-defined functions' IDs. Next, if t servers send t correct responses to the leader, then the leader can recover the correct *DATA*. For instance, leader wants to get a result *DATA* that can be obtained with T_1 and T_2, and *DATA* can be described as the following equation:

$$DATA = d_{1,1}d_{1,1} + d_{1,2}d_{1,2} + \cdots + d_{1,m_1}d_{1,m_1} + d_{2,1} + d_{2,1} + \cdots + d_{2,m_2}, \quad (4)$$

where $d_{i,j}$ is the plaintext of the $s_{i,j}$ and the servers only know $s_{i,j}$ rather than $d_{i,j}$. Then, the T_{query} can be described as follow:

T_{query}	
Transaction Header	
Payload	
$\text{ID}_{f_{mul}}$	ID_{T_1}
$\text{ID}_{f_{add}}$	ID_{T_2}

In the T_{query}, f_{mul} and f_{add} are pre-defined functions. Moreover, $\text{ID}_{f_{mul}}$ and $\text{ID}_{f_{add}}$ denote both functions' IDs. Specifically, f_{mul} means that leader wants to obtain $data_1$ as follows:

$$data_1 = d_{1,1}d_{1,1} + d_{1,2}d_{1,2} + \cdots + d_{1,m_1}d_{1,m_1}.$$

While f_{add} means that the leader want to obtain $data_2$ as follows:

$$data_2 = d_{2,1} + d_{2,2} + \cdots + d_{2,m_2}.$$

Overall, the query transaction T_{query} means that the leader wants to obtain $data_1 + data_2$. After that, the leader sends T_{query} to the blockchain network.

- **Step 6: Respond.** After T_{query} is recorded on the blockchain, any server can see it. Then, if a server wishes to respond to the query, it will generate a response according to T_{query}. Then, the server will secretly send his response to the leader via a transaction. If the leader collects at least t responses correctly computed by corresponding servers, then the leader can recover the correct *DATA* as mentioned in Eq.4. To introduce the process, without loss of generality, we assume that the t servers are $Server_1, Server_2, \cdots, Server_t$ and that they wish to respond to T_{query}. According to T_{query}, the servers can obtain $s_{1,1}, s_{1,2}, \cdots, s_{1,m_1}$ and $s_{2,1}, s_{2,2}, \cdots, s_{2,m_2}$, which are recorded in T_1 and T_2. First, with i from 1 to t, $Server_i$ computes as follows:

$$Resp_i = \sum\nolimits_{j=1}^{m_1}(CF_i + s_{1,j})(CF_i + s_j) - m_1 \cdot Ch_i + \sum\nolimits_{j=1}^{m_2}(CF_i + s_{2,j}).$$

Then, $Server_i$ encrypts $Resp_i$ into

$$C_{Resp_i} = Enc_{pk_L}(Resp_i)$$

with the leader's public key pk_L. Then, $Server_i$ computes a commitment about $Resp_i$ as follows:

$$CM_{Resp_i} = g^{Resp_i}.$$

Then, $Server_i$ generates a respond transaction $T^i_{respond}$ containing the leader's ID, CM_{Resp_i} and C_{Resp_i}. $T_{respond}$ can be described as follows:

$T^i_{respond}$		
Transaction Header		
Payload		
$\text{ID}_{T_{query}}$	CM_{Resp_i}	C_{Resp_i}

Overall, the servers $Server_1, Server_2, \cdots, Server_t$ generate $C_{Resp_1}, C_{Resp_2}, \cdots, C_{Resp_t}$ and $CM_{Resp_1}, CM_{Resp_2}, \cdots, CM_{Resp_t}$. Then, $Server_1, Server_2, \cdots, Server_t$ generate transactions $T^1_{respond}, T^2_{respond}, \cdots, T^t_{respond}$, respectively. Because only the leader has the corresponding secret key sk_L, only the leader can decrypt $C_{Resp_1}, C_{Resp_2}, \cdots, C_{Resp_t}$. After that, the servers send $T^1_{respond}, T^2_{respond}, \cdots, T^t_{respond}$ to the blockchain network.

- Step 7: **Record node verification** $T^1_{respond}$, $T^2_{respond}$, $\cdots$, $T^t_{respond}$. After receiving the respond transactions $T^1_{respond}$, $T^2_{respond}$, $\cdots$, $T^t_{respond}$, a record node should verify the validations of their CM_{Resp_1}, CM_{Resp_2} and CM_{Resp_t} (Record node can perform these verification with off-chain computation or smart contract. After that, the record node can determine whether received transactions are valid). Specifically, with i from 1 to t, the record node performs the following:
  - Compute

$$\begin{aligned} g^{CF_i} &= (g^{a_{t-1}})^{Sr_i^{t-1}} \cdots (g^{a_1})^{Sr_i} g^{s_{core}} \\ &= g^{a_{t-1}Sr_i^{t-1} + \cdots + a_1 Sr_i + s_{core}} \\ g^{Ch_i} &= (g^{b_{2t-2}})^{Sr_i^{2t-2}} \cdots (g^{b_1})^{Sr_i} \\ &= g^{b_{2t-2}Sr_i^{2t-2} + \cdots + b_1 Sr_i} \end{aligned}$$

  - With $s_{1,1}, s_{1,2}, \cdots, s_{1,m_1}, s_{2,1}, s_{2,2}, \cdots, s_{2,m_2}$ and the bilinear map e, the record node further computes

$$\begin{aligned} E_i^1 &= e(g^{CF_i}g^{s_{1,1}}, g^{CF_i}g^{s_{1,1}})e(g^{CF_i}g^{s_{1,2}}, g^{CF_i}g^{s_{1,2}}) \\ &\quad \cdots e(g^{CF_i}g^{s_{1,m_1}}, g^{CF_i}g^{s_{1,m_1}}) \\ E_i^2 &= e(g^{CF_i}g^{s_{2,1}}g^{CF_i}g^{s_{2,2}} \cdots g^{CF_i}g^{s_{2,m_2}}, g) \end{aligned}$$

  - If

$$E_i^1 E_i^2 / e(g^{Ch_i}, g^{m_1}) = e(CM_{Resp_i}, g),$$

  then the record node assumes that CM_{Resp_i} is valid; otherwise, it will reject invalid respond transactions.
- **Step 8: Recover.** Because $T^1_{respond}, T^2_{respond}, \cdots, T^t_{respond}$ are found on the blockchain, the transactions pass all previous verifications. Therefore, the leader simply needs to perform the final verification that can be performed only by itself. Specifically, with i from 1 to t,

the leader decrypts C_{Resp_i} and then obtains $Resp_i$. If

$$CM_{Resp_i} = g^{Resp_i},$$

then leader accepts that $Resp_i$ is correctly computed by $Server_i$; otherwise, it rejects the response and can send its evidence (including $\mathrm{ID}_{T_{respond}}$ and $Resp_i$) to the $Server_i$'s smart contract. The leader can then obtain a reward.

If all the t responses are correct, then the leader uses *Lagrange interpolation* to reconstruct a polynomial as follows:

$$\tilde{F}(x) = \sum_{i=1}^{t} Resp_i \prod_{j=1, j\neq i}^{t} \frac{x - Sr_j}{Sr_i - Sr_j}.$$

Finally, the leader calculates $\tilde{F}(0)$, which is the result *DATA*, as mentioned at Eq.4.

VII. PERFORMANCE EVALUATION

In this section, we give a performance evaluation of BeeKeeper, which may be broken up into three parts, by performing BeeKeeper on the Ethereum blockchain. The first part studies the processing time of the cryptographic and mathematical computations. The time needed for processing transactions is studied in the second part. The last part further analyzes the processing time of blocks when different transactions are sent to the blockchain network. The section starts with the prototype system setting.

A. PROTOTYPE SYSTEM SETTING

BeeKeeper's efficiency mainly depends on the blockchain platform, computing platform and performance of the cryptographic schemes. For instance, in this paper, we use the Ethereum blockchain as the platform. Specifically, in the Ethereum blockchain, a block may contain transactions of at most 62,360 bytes, its average block interval is approximately 15 s, and its transaction payload contains at most 1,014 bytes of data. Consequently, BeeKeeper's efficiency is significantly limited by the blockchain platform. Therefore, if we use a more efficient blockchain platform, we might obtain a better throughput.

We use laptops and virtual machines to implement the prototype system. Specifically, our laptop's configuration is described as follows: an Intel i5-5300 CPU at 2.30 GHz, 4 GB of memory, and the Windows 10 OS. On a local area network, we deploy a local blockchain via go-ethereum, which is a Go implementation of the Ethereum protocol [36]. In the blockchain network, we deploy four record nodes (miners), and we use a transaction simulator [36] to simulate the leader, servers and devices to generate and send transactions. Moreover, we record the BeeKeeper system's key data in the transaction's payload.

In the execution, we implement a (2,3)-threshold BeeKeeper prototype system that contains a leader, three servers and three devices. Specifically, if the leader can collect two correct responses from two servers, then the leader can obtain the correct desired result.

TABLE 4. Average time cost of cryptographic schemes.

Scheme	Time cost
BN-curve Point Mul	596.113 μs
BN-curve Point Add	2.328 μs
BN-curve Pairing	1.687 ms
Field Add	0.071 μs
Field Mul	0.531 μs
Secp256k1-curve ECDSA Sign	4.425 ms
Secp256k1-curve ECDSA Verify Sig	9.137 ms
Secp256k1-curve ECIES Encryption	8.745 ms
Secp256k1-curve ECIES Decryption	4.367 ms
Block Interval	15.2 s

Additionally, Ethereum possesses an embedded signature scheme: the ECDSA based on the secp256k1 elliptic curve [33]. For convenience, we use the scheme to sign messages. Furthermore, to encrypt key data recorded in the payloads of initialize transactions and respond transactions, we use ECIES, an encryption scheme, with the elliptic curve secp256k1 to encrypt some data via the receiver's public key, where the cipher block has a length of 64 bytes. This results in each encrypted message having a length of 64 bytes. Moreover, the encrypted data can only be decrypted by the corresponding receivers since only the receiver has the corresponding private key.

Furthermore, for committing data and verifying committed data, we utilize a high-speed pairing library [30] based on the BN-curve. Specifically, after selecting a basepoint G, data are committed via a point multiplication of G, and committed data are verified by their pairing computation. For instance, let G be the basepoint of the BN-curve. Then, the secret s can be committed as sG. Therefore, a commitment has a length of 64 bytes (512 bits) since any point on the BN-curve has two coordinates, and each coordinate is 32 bytes. Moreover, the bilinear map e (pairing computing) based on the BN-curve is used to verify the correctness of the commitments. We have $e(g^a, g^b) = e(g^{ab}, g)$. For instance, if we want to verify $ab = c$ and we do not want to reveal a, b and c, then we may use the following equation to verify $ab = c$.

$$e(g^a, g^b) = e(g^c, g).$$

B. PROCESSING TIME OF CRYPTOGRAPHIC SCHEMES

Generally, the time cost of performing cryptographic schemes will influence the time for processing transactions. Therefore, in this sub-section, we study the time cost of the cryptographic schemes. Specifically, we purely perform these cryptographic schemes without blockchain.

For each point multiplication, point addition, pairing, field addition, field multiplication, encryption, decryption, signing and signature verification, we perform 1000 experiments to obtain their average time cost, and the average time cost is described in Table 4.

C. GENERATE TRANSACTIONS

In the BeeKeeper system, different transactions may have different usages and payloads. For instance, an initialize transaction includes a verification key, some commitments, some server IDs and some encrypted messages. A respond transaction only contains a leader's ID, an encrypted response and a commitment. Moreover, the sizes of the payloads of initialize transactions and respond transactions are fixed, while the sizes of the payloads of record transactions and query transactions are variable. Therefore, transaction may have different generation processes and generation times. Consequently, in this sub-section, we study the generation time of transactions by deploying transaction simulator without sending them to the blockchain. We discuss initialize transactions first.

In the prototype system, we implement a (2,3)-threshold BeeKeeper instance. Therefore, the payload of $T_{initialize}$ may include a verification key, three servers' IDs, six commitments about secret polynomials, six encrypted data of six core data and six commitments of the six core data, as mentioned in Section VI-B. According to the last sub-section, these data have a length of 1056 bytes. However, the payload of a transaction, in the Ethereum blockchain, can include at most 1014 bytes. In other words, one transaction's payload cannot contain this much data, i.e., 1056 bytes. Therefore, we have to divide $T_{initialize}$ into $T^{VK}_{initialize}$ and $T^{non-VK}_{initialize}$ to record all the data. $T^{VK}_{initialize}$ and $T^{non-VK}_{initialize}$ have a same and unique sub-ID recorded in their transaction header. Others can use the sub-ID to determine whether $T^{VK}_{initialize}$ and $T^{non-VK}_{initialize}$ are generated from the same verification key. Specifically, both transactions can be described as follows:

$T^{VK}_{initialize}$	
Transaction Header	
Payload	
G	a_1G
$s_{core}G$	b_2G
b_1G	c_1G

$T^{non-VK}_{initialize}$				
Transaction Header				
Payload				
C_{score}				
Sr_1	C_{CF_1}	C_{Ch_1}	CM_{CF_1}	CM_{Ch_1}
Sr_2	C_{CF_2}	C_{Ch_2}	CM_{CF_2}	CM_{Ch_2}
Sr_3	C_{CF_3}	C_{Ch_3}	CM_{CF_3}	CM_{Ch_3}

Indeed, record transactions and query transactions may have payloads of variable size. In the implementation, the sizes of the record transactions and query transactions are fixed. Specifically, they can be described as follows:

T^1_{record}
Transaction Header
Payload
$s_{1,1}, s_{1,2}, \cdots, s_{1,31}$

T^2_{record}
Transaction Header
Payload
$s_{2,1}, s_{2,2}, \cdots, s_{2,31}$

T_{query}
Transaction Header
Payload
ID_{mul} $\mathrm{ID}_{T^1_{record}}$
ID_{add} $\mathrm{ID}_{T^2_{record}}$

TABLE 5. Payloads of transactions used in our simulation.

Transaction	Payload	Size
$T^{VK}_{initialize}$	a verification key	384 bytes
$T^{non-VK}_{initialize}$	3 IDs, 6 encrypted data and 6 commitments	928 bytes
T_{record}	1 to 31 confidential data	992 bytes
T_{query}	functions' IDs and record transactions' IDs	128 bytes
$T_{respond}$	a leader's ID, an encrypted response, a commitment	160 bytes

TABLE 6. Average time cost of processing transactions.

Operation on transaction	Computations	Time cost
Leader generates a $T^{VK}_{initialize}$	1S+5PM	8.227 ms
Leader generates a $T^{non-VK}_{initialize}$	1S+7E+6PM	70.038 ms
Device generates a T_{record}	1S+31FA	5.249 ms
Leader generates a T_{query}	1S	5.247 ms
Server generates a $T_{respond}$	1S+31FM+154FA+1E+1PM	14.615 ms
Record node verifies a $T^{VK}_{initialize}$	1V+5PM+2PA	12.122 ms
Record node verifies a $T^{non-VK}_{initialize}$	1V+12PM+6PA	16.304 ms
Record node verifies a T_{record}	1V	9.128 ms
Record node verifies a T_{query}	1V	9.131 ms
Record node verifies a $T_{respond}$	1V+125PA+66PM+34Pairing	106.129 ms
Server verifies a $T^{non-VK}_{initialize}$	2D++2PM	9.926 ms
Leader verifies a $T_{respond}$	1D+1PM	4.963 ms
Leader recovers the result	4FA+4FM	2.408 μs

In the table, S is a signing computation, V denotes a signature verification, PM describes a point multiplication on the ECC, PA is a point addition on the ECC, $Pairing$ means a pairing computation, E is an encryption, D denotes a decryption, FM describes a field multiplication, and FA is a field addition. For instance, "2PM+3PA+1V+6Pairing" denotes that the corresponding computations contain 2 point multiplications, 3 point additions, 1 signature verification and 6 pairing computations.

In the above query transaction, ID_{mul} and $\mathrm{ID}_{T^1_{record}}$ mean that the leader wants to obtain a result that can be described with the following equation.

$$data_1 = \sum_{i=1}^{31}(s_{core} + s_{1,i})(s_{core} + s_{1,i}).$$

Additionally, ID_{add} and $\mathrm{ID}_{T^2_{record}}$ mean that the leader wants to obtain a result that can be described with the following equation.

$$data_2 = \sum_{i=1}^{31}(s_{core} + s_{2,i}).$$

Finally, $data_1 + data_2$ is what the leader really wants to know. Specifically,

$$data_1 + data_2 = \sum_{i=1}^{31}(s_{core} + s_{1,i})(s_{core} + s_{1,i}) + \sum_{i=1}^{31}(s_{core} + s_{2,i}).$$

A respond transaction's payload contains a query transaction's ID, an encrypted response and a commitment of the response. Specifically, $Server_i$'s respond transaction can be described as follows:

$T^{i}_{respond}$		
Transaction Header		
Payload		
$\text{ID}_{T_{query}}$	CM_{Resp_i}	C_{Resp_i}

For the experiment, the sizes of the four transactions' payloads are shown in Table 5. For each of $T^{VK}_{initialize}$, $T^{non-VK}_{initialize}$, T_{record}, T_{query} and $T_{respond}$, we generate 1000 transactions to obtain their average generation time cost, and their average time costs are shown in Table 6.

D. VERIFY TRANSACTIONS

In the system, before a transaction is appended to the blockchain, record nodes must verify the transaction. Specifically, record nodes verify all publicly verifiable data of the transaction. Moreover, if a transaction has appeared in the blockchain, then the transaction's publicly verifiable data are credible. Consequently, others (e.g., the leader, servers and devices) do not have to verify the transaction's publicly verifiable data. This significantly reduces the verification computations of the leader, servers and devices. In this sub-section, we study the transactions' verification time cost. These transactions have been generated by the transaction simulator. Therefore, in this sub-section, we purely verify transactions without sending them to the blockchain. All publicly verifiable data of transactions are summarized as follows:

- All transactions' signatures are publicly verifiable data that can be verified by record nodes. Therefore, if a transaction has appeared on the blockchain, then the transaction's signature is credible, and others do not need to verify the signature.
- In addition to signatures, the payloads of the initialize transactions and respond transactions contain publicly verifiable data that can be verified by record nodes. Specifically, they are the initialize transaction's verification key, the commitments of core shares, and the respond transaction's commitments of responses. Consequently, if an initialize transaction (or a respond transaction) has appeared on the blockchain, then the transaction's publicly verifiable data are credible. Therefore, the transaction's receiver does not need to verify the publicly verifiable data.

In this way, the transaction's receiver simply needs to verify some key data that can be verified only by itself. Generally, in the system, the key data are very small and can be verified efficiently.

For each of $T^{VK}_{initialize}$, $T^{non-VK}_{initialize}$, T_{record}, T_{query} and $T_{respond}$, we generate 1000 transactions and then obtain their average verification time cost, which are shown in Table 6. Specifically, in Table 6, *S* is a signing computation, *V* denotes a signature verification, *PM* describes a point multiplication on the ECC, *PA* is a point addition on the ECC, *Pairing* means a pairing computation, *E* is an encryption, *D* denotes a decryption, *FM* describes a field multiplication, and *FA* is a field addition. For instance, "2PM+3PA+1V+6Pairing"

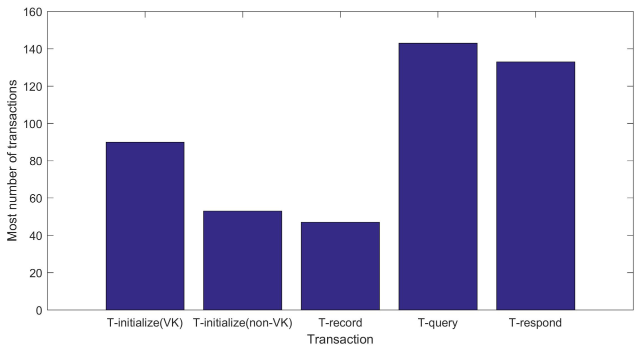


FIGURE 3. **The highest number of the same transactions recorded in a block.**

denotes that the corresponding computations contain 2 point multiplications, 3 point additions, 1 signature verification and 6 pairing computations.

However, if BeeKeeper is not based on the blockchain, then the system will lose several properties, such as tamper resistance and decentralization, and transaction receivers will perform more verification computations than the blockchain-based BeeKeeper. For convenience, the non-blockchain-based BeeKeeper is called the pure BeeKeeper. If the pure BeeKeeper system is deployed, then the following can occur:

- All data are stored by centralized nodes. Therefore, storage might be modified by the centralized nodes.
- All related users should independently verify all publicly verifiable data including the verification key and commitments.
- Servers and devices might be heavier than the blockchain-based BeeKeeper.

For instance, in a pure BeeKeeper system, if a leader receives a respond transaction, then it will verify all verifiable data by itself; otherwise, it would not trust the transaction's data. Therefore, it requires approximately 111.092 ms to verify the transaction's data.

However, if the BeeKeeper is based on a blockchain network, which is the key point of the paper, then the leader only needs 4.963 ms to verify some special data since other data have been verified by record nodes. Therefore, by combining with the blockchain, BeeKeeper significantly reduces the servers', leader's and devices' verification computations. Comparisons between the pure BeeKeeper and blockchain-based BeeKeeper are shown in Table 7.

E. ETHEREUM-BASED BeeKeeper

We run our BeeKeeper on the Ethereum blockchain. After generating a certain number of blocks, the block interval tends to be stable. Specifically, generating 1000 blocks takes approximately 4.3 hours. In other words, generating a block takes approximately 15.2 s on average. Furthermore, in the Ethereum blockchain, a block can record transactions of at most 62,360 bytes, a transaction with an empty payload is 308 bytes, and a transaction's payload can record at most

TABLE 7. Comparisons between pure BeeKeeper and blockchain-based BeeKeeper.

Comparative item	Time cost of pure BeeKeeper			Time cost of blockchain-based BeeKeeper			
	Leader	Server	Device	Leader	Server	Device	Record-node
Verifying $T^{vk}_{initialize}$	0 ms	12.122 ms	9.128 ms	0 ms	0 ms	0 ms	12.122 ms
Verifying $T^{non-vk}_{initialize}$	0 ms	26.229 ms	9.128 ms	0 ms	9.926 ms	0 ms	16.304 ms
Verifying T_{record}	9.128 ms	9.128 ms	0 ms	0 ms	0 ms	0 ms	9.128 ms
Verifying T_{query}	0 ms	9.128 ms	0 ms	0 ms	0 ms	0 ms	9.128 ms
Verifying $T_{respond}$	111.092 ms	0 ms	0 ms	4.963 ms	0 ms	0 ms	106.129 ms

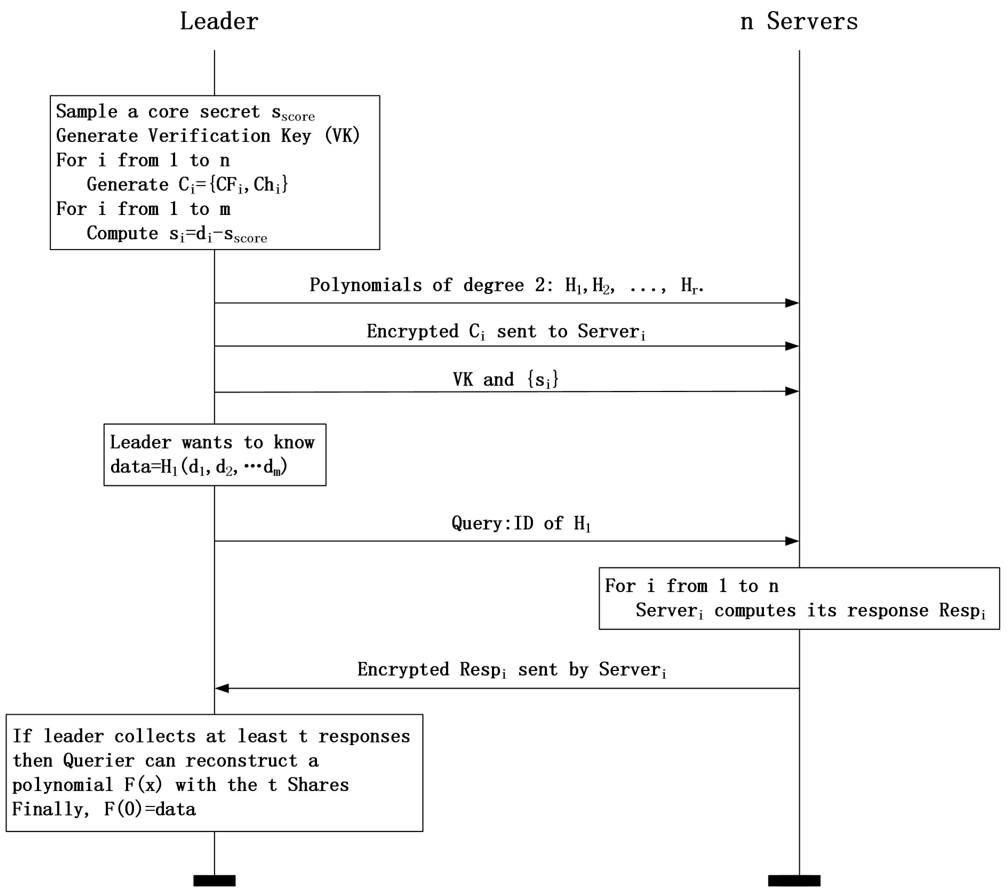

Leader
n Servers
Sample a core secret s_{score}
Generate Verification Key (VK)
For i from 1 to n
Generate $C_i=\{CF_i, Ch_i\}$
For i from 1 to m
Compute $s_i=d_i-s_{score}$
Polynomials of degree 2: $H_1, H_2, \ldots, H_r$.
Encrypted C_i sent to $Server_i$
VK and $\{s_i\}$
Leader wants to know
data=$H_1(d_1, d_2, \cdots d_m)$
Query:ID of H_1
For i from 1 to n
$Server_i$ computes its response $Resp_i$
Encrypted $Resp_i$ sent by $Server_i$
If leader collects at least t responses
then Querier can reconstruct a
polynomial F(x) with the t Shares
Finally, F(0)=data

FIGURE 4. (t,n)-threshold secure multi-parties computation (TSMPC) protocol.

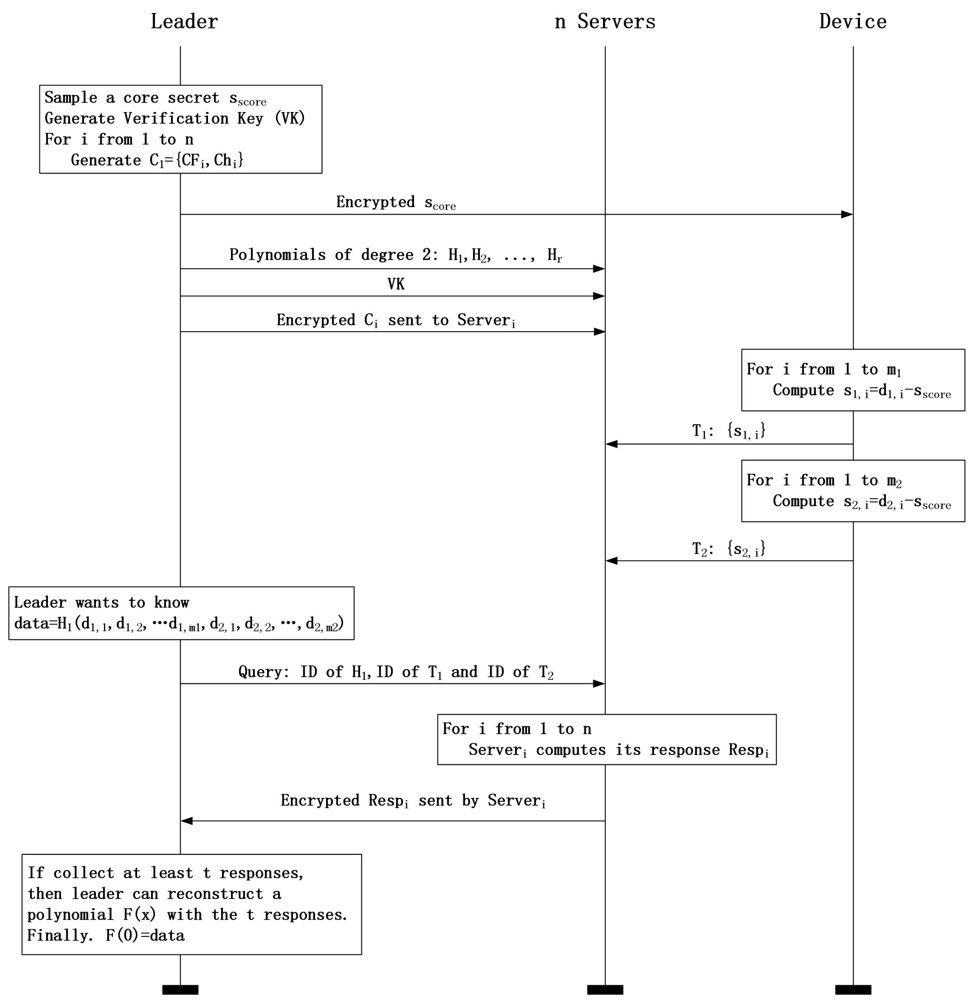
1014 bytes of data. Therefore, a transaction's size should be from 308 bytes to 308+1014=1322 bytes.

According to Table 5 and the above, in our implementation, any transaction's size can be calculated. The transactions sizes are shown in Table 8. A block can record transactions of at most 62,360 bytes. Therefore, if a block only records identical transactions, then the number of recorded transactions is limited. The limits are described in Fig.3.

In our experiments, because different transactions have different significance; therefore, the most significant transaction should be processed earliest. Moreover, more significant transactions should be processed more early than less significant transactions. In the Ethereum blockchain, record nodes (miners) earlier process the transaction with more fee. Therefore, we set different transactions having different

TABLE 8. Transaction sizes in our simulation.

Transaction	Size
$T^{VK}_{initialize}$	692 bytes
$T^{non-VK}_{initialize}$	1172 bytes
T_{record}	1300 bytes
T_{query}	436 bytes
$T_{respond}$	468 bytes



Leader
n Servers
Device
Sample a core secret s_{score}
Generate Verification Key (VK)
For i from 1 to n
Generate C_i={CF_i, Ch_i}
Encrypted s_{core}
Polynomials of degree 2: H_1, H_2, ..., H_r
VK
Encrypted C_i sent to $Server_i$
For i from 1 to m_1
Compute $s_{1,i}=d_{1,i}-s_{score}$
T_1: {$s_{1,i}$}
For i from 1 to m_2
Compute $s_{2,i}=d_{2,i}-s_{score}$
T_2: {$s_{2,i}$}
Leader wants to know
data=$H_1(d_{1,1}, d_{1,2}, \cdots d_{1,m1}, d_{2,1}, d_{2,2}, \cdots, d_{2,m2})$
Query: ID of H_1, ID of T_1 and ID of T_2
For i from 1 to n
$Server_i$ computes its response $Resp_i$
Encrypted $Resp_i$ sent by $Server_i$
If collect at least t responses,
then leader can reconstruct a
polynomial F(x) with the t responses.
Finally. F(0)=data

FIGURE 5. Work process of BeeKeeper.

TABLE 9. Transaction fee.

Transaction	Transaction fee
$T^{VK}_{initialize}$	0.002 ETH
$T^{non-VK}_{initialize}$	0.002 ETH
T_{record}	0.000001 ETH
T_{query}	0.0015 ETH
$T_{respond}$	0.001 ETH

transaction fees. When different transaction are pending in the record node transaction pool, transactions with higher fees will be recorded earlier. In the BeeKeeper system, the initialize transaction is the base of later transactions. Therefore, it should have the highest priority. To quickly respond to the leader's query, we set that query transaction as the second priority and respond transaction has the third priority. Finally, the record transaction has the lowest priority. In this way, the system's response rate will be obviously increased. For our experiments, their transaction fees are shown in Table 9.

In our experiments, after an initialize transaction has appeared on the blockchain, three devices send record transactions to the blockchain network at high frequency. The record transactions are the same as mentioned in Sec. VI-B. Every block can contain at most 47 record transactions. Let the data recorded in the payloads of the record transactions, which are generated by devices, be called "device data". Then, a block can store device data of at most 46,624 bytes. Because the blockchain generates a block per approximately 15 seconds on average, the system can record at most 3108.26 bytes of device data per second on average. At some later time, the leader sends a query transaction to the blockchain network, and the blockchain height is n at this time. The query transaction will be recorded on the blockchain in the $n + 1$-th block since a query transaction has a higher transaction fee than a record transaction. Then, servers may see the query transaction and then send the corresponding respond transactions to the blockchain network. Because a respond transaction has a higher transaction fee than record transactions, they can be recorded on the blockchain more quickly than record transactions. In addition, the time to verify a respond transaction is approximately 107 ms. Therefore, two respond transactions can be recorded on the blockchain in the $n+2$-th block. Consequently, when the $n+2$-th block is generated, the leader can obtain the two response transactions. Finally, the leader can recover his desired data. The process only takes approximately 22.5 seconds on average.

BeeKeeper's efficiency mainly depends on the blockchain platform. For instance, in this paper, we use the Ethereum blockchain as the platform. Specifically, Ethereum's blocks can contain transactions of at most 62,360 bytes, its average block interval is approximately 15 s, and its transaction payload contains at most 1014 bytes of data; therefore, the BeeKeeper's efficiency is significantly limited by the blockchain platform. Therefore, if we use some other, more suitable blockchain platform, then we might obtain a better performance.

VIII. CONCLUSION

In this paper, we propose a blockchain-based threshold IoT service system: BeeKeeper. In the BeeKeeper system, a leader may apply a (t, n)-threshold BeeKeeper protocol among n servers. After that, the leader's devices can send transactions, including encrypted data, of collected data to the blockchain network. The servers have abilities to perform homomorphic computations on the encrypted data. However, they cannot learn anything from the encrypted data as long as $n-t$ servers are honest. According to the leader's query, servers may generate responses for the leader. If the leader can collect at least t correct responses, then it can recover the desired result; otherwise, it cannot obtain anything. Moreover, most of the data recorded in transactions are verifiable or even publicly verifiable. Therefore, receivers may check whether received data are correctly computed, and record nodes help others to reduce their verification computations by verifying publicly verifiable data. Furthermore, because all data collected by devices are recorded in the blockchain and because servers can help the leader to process the data, the leader, leader's devices and servers do not need large memory and computational resources.

APPENDIX

In the following contents, we propose a threshold secure multi-parties computation (TSMPC) protocol that may be considered as a (t, n)-threshold verifiable homomorphic confidential storage scheme. The protocol contains two parties. They are a *leader* and n *servers*. Specifically, in the TSMPC protocol, the leader converts his data $\{d_1, d_2, \cdots, d_m\}$ into n sets and then sends the n sets to n servers, respectively. After that, when the leader wants to know some result that is $F(d_1, d_2, \cdots, d_m)$, it will send a query about the demand to servers. If leader can collect at least t correct responses from t servers, then it can recover the correct $F(d_1, d_2, \cdots, d_m)$. While if the leader can only collect less than t responses, then it cannot get any data. With the TSMPC, leader can obtain the desired result without computing the function $F(d_1, d_2, \cdots, d_m)$ by himelf.

TABLE 10. Symbols of TSMPC.

Symbol	Description
g	the generator of a cyclic group $\mathbb{G}$
e	the bilinear map, e: $\mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}$.
$\mathbb{F}_p$	the finite field with character p
L	the leader's ID
Sr_i	the i-th servers' IDs
VK	the verification key
$\{pk_L, sk_L\}$	the D's key pair
$\{pk_i, sk_i\}$	the i-th server's key pair, for i from 1 to n
d_i	the i-th plaintext data
CM_{F_i}	$CM_{F_i} = g^{F_i}$
CM_{h_i}	$CM_{h_i} = g^{h_i}$
$Resp_i$	the i-th server's response

A. CONSTRUCTION OF TSMPC

Symbols, used in the scheme, are summarized at Table 10.

The TSMPC can be described as follows:

- **Initialize.** Let $\mathbb{F}_p$ be a finite field with character p. Leader randomly samples a polynomial $F(x)$ of degree $t-1$ from $\mathbb{F}_p[x]$ as follows:
$$F(x) = a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + \cdots + a_1x + s_{core},$$
where $s_{core}, a_1, \cdots, a_{t-1} \in \mathbb{F}_p$ and $a_{t-1} \neq 0$. Let
$$f(x) = a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + \cdots + a_1x.$$
Then we have $F(x) = f(x) + s_{core}$. Leader randomly samples $l(x)$ of degree $t-1$ from $\mathbb{F}_p[x]$ as follow:
$$l(x) = c_{t-1}x^{t-1} + c_{t-2}x^{t-2} + \cdots + c_1x.$$
After that leader computes
$$\begin{aligned} h(x) &= f(x)^2 - l(x) \\ &= b_{2t-2}x^{2t-2} + b_{2t-3}x^{2t-3} + \cdots + b_1x. \end{aligned}$$
g is a generator of cyclic group $\mathbb{G}$. Then leader publishes a verification key VK as follow:
$$VK = \{g, g^{a_{t-1}}, \cdots, g^{a_1}, g^{s_{core}}, g^{b_{2t-2}}, g^{b_{2t-3}}, \cdots, g^{b_1}, g^{c_{t-1}}, g^{c_{t-2}}, \cdots, g^{c_1}\}$$
- **Verify committed polynomials.** Anyone, including servers, can verify whether polynomials $f(x)$, $h(x)$ and $l(x)$, committed in the verification key, are well-formed and sound. Specifically, it can do as follows:
  - Randomly sample t different numbers $x_1, x_1, \cdots, x_t \in \mathbb{F}_p$.
  - j from 1 to t, compute
$$\begin{aligned} g_j^f &= (g^{a_{t-1}})^{x_j^{t-1}} (g^{a_{t-2}})^{x_j^{t-2}} \cdots (g^{a_1})^{x_j} \\ &= g^{a_{t-1}x_j^{t-1} + a_{t-2}x_j^{t-2} + \cdots + a_1x_j} \\ g_j^h &= (g^{b_{2t-2}})^{x_j^{2t-2}} (g^{b_{2t-3}})^{x_j^{2t-3}} \cdots (g^{b_1})^{x_j} \\ &= g^{b_{2t-2}x_j^{2t-2} + b_{2t-3}x_j^{2t-3} + \cdots + b_1x_j} \end{aligned}$$

$$g_j^l = (g^{c_{t-1}})^{x_j^{t-1}}(g^{c_{t-2}})^{x_j^{t-2}}\cdots(g^{c_1})^{x_j} = g^{c_{t-1}x_j^{t-1}+c_{t-2}x_j^{t-2}+\cdots+c_1x_j}$$

– If $e(g_j^f, g_j^f) = e(g_j^h g_j^l, g)$, for j from 1 to t, then the verifier accepts that $f(x)$, $h(x)$ and $l(x)$, committed by verification key, are well-formed and sound. Otherwise, it rejects and return to step **Inilialize**.

- **Distribute.** For i from 1 to n, leader computes

$$CF_i = F(Sr_i) \; and \; Ch_i = h(Sr_i),$$

where Sr_i is the i-th server $Server_i$'s ID. After that, leader encrypts $\{CF_i, Ch_i\}$ into $C_i^{core} = Enc_{pk_i}(CF_i, Ch_i)$ with $Server_i$'s public key pk_i. Leader sends C_i^{core} to $Server_i$, respectively. For C_i^{core}, only $Server_i$ can decrypt the encrypted data since only it has the corresponding secret key. After obtaining CF_i and Ch_i, $Server_i$ can verify the soundness of $\{F_i, h_i\}$ with verification key

$$\{g, g^{a_{t-1}}, \cdots, g^{a_1}, g^{s_{core}}, g^{b_{2t-2}}, g^{b_{2t-3}}, \cdots, g^{b_1}, g^{c_{t-1}}, g^{c_{t-2}}, \cdots, g^{c_1}\}.$$

Specifically, it computes

$$CF_i^* = (g^{a_{t-1}})^{Sr_i^{t-1}}\cdots(g^{a_1})^{Sr_i}(g^{s_{core}})$$
$$Ch_i^* = (g^{b_{2t-2}})^{Sr_i^{2t-2}}\cdots(g^{b_1})^{Sr_i}$$

If $CF_i^* = g^{F_i}$ and $Ch_i^* = g^{h_i}$, then $Server_i$ accepts CF_i and Ch_i, otherwise it rejects.

- **Publish.** Leader computes

$$s_i = d_i - s_{core}$$

for $1 \leq i \leq m$. Then leader publishes $s_1, s_2, \cdots, s_m$ that can be seen by anyone including the servers. However, only the servers can use $s_1, s_2, \cdots, s_m$ to generate responses since they have CF_i and Ch_i, respectively.

- **Query.** If the leader wants to get a result, then it may send a query to servers. The query is that the leader wants to get a $DATA$ that can be calculated as the following equation:

$$DATA = d_{i_1}d_{i_2} + d_{i_3}d_{i_4} + \cdots + d_{i_{k_1-1}}d_{i_{k_1}} + d_{i_{k_1+1}} + \cdots + +d_{i_{k_1+k_2}}, \quad (5)$$

where $1 \leq i_1, i_2, \cdots, i_{k_1+k_2} \leq m$.

- **Respond.** $i \in [1, n]$, if $Server_i$ wishes to respond the leader's query, then it may generate a response $Resp_i$ by calculating with $s_1, s_2, \cdots, s_m$, CF_i and Ch_i as follows:

$$Resp_i = (CF_i + s_{i_1})(CF_i + s_{i_2}) + \cdots + (CF_i + s_{i_{k_1-1}}) \times (CF_i + s_{i_{k_1}}) + (CF_i + s_{i_{k_1+1}}) + \cdots + (CF_i + s_{i_{k_1+k_2}}) - \frac{k}{2}Ch_i$$

After that, $Server_i$ encrypts $Resp_i$ into $C_{Resp_i} = Enc_{pk_L}(Resp_i)$ with leader's public key pk_L. Then, $Server_i$ sends C_{Resp_i} to leader.

- **Recover.** If leader collects at least t correct and different responses, it can recover the $DATA$ as described in Eq. 5.

Without loss of generality, we assume the t resoibses come from $Server_1, Server_2, \cdots, Server_t$.

At first, the leader decrypts C_{Resp_1}, C_{Resp_2}, $\cdots$, C_{Resp_t} and then verify the validations of $Resp_1$, $Resp_2$, $\cdots$, $Resp_t$. Specifically, i from 1 to t, leader computes

$$g^{CF_i} = (g^{a_{t-1}})^{Sr_i^{t-1}}\cdots(g^{a_1})^{Sr_i}(g^{a_0}) = g^{a_{t-1}Sr_i^{t-1}+\cdots+a_1Sr_i+s_{core}}$$
$$g^{Ch_i} = (g^{b_{2t-2}})^{Sr_i^{2t-2}}\cdots(g^{b_1})^{Sr_i} = g^{b_{2t-2}Sr_i^{2t-2}+\cdots+b_1Sr_i}$$

After that, with $s_{i_1}, s_{i_2}, \cdots, s_{i_m}$ and bilinear map $e(\cdot,\cdot)$, leader further computes

$$E_i^1 = e(g^{CF_i}g^{s_{i_1}}, g^{CF_i}g^{s_{i_2}})e(g^{CF_i}g^{s_{i_3}}, g^{CF_i}g^{s_{i_4}}) \cdots e(g^{CF_i}g^{s_{i_{k_1-1}}}, g^{CF_i}g^{s_{i_{k_1}}})$$
$$E_i^2 = e(g_i^{CF_i}g^{s_{i_{k_1+1}}}g_i^{CF_i}g^{s_{i_{k_1+2}}}\cdots g_i^{CF_i}g^{s_{i_{k_1+k_2}}}, g)$$

If

$$E_i^1E_i^2/e(g^{Ch_i}, g^{\frac{k_1}{2}}) = e(g^{Resp_i}, g),$$

then leader considers that $Resp_i$ is correctly computed by $Server_i$. Essentially, if all $Resp_1, Resp_2 \cdots Resp_t$ are correctly computed by senders, then leader can recover the $DATA$ with $Resp_1, Resp_2$ and $Resp_t$. Specifically, leader can reconstruct a polynomial of degree $t-1$ by *Lagrange interpolating* as follow:

$$\widetilde{F}(x) = \sum_{i=1}^{t} Resp_i \prod_{j=1, j\neq i}^{t} \frac{Sr_j - x}{Sr_j - Sr_i}$$

Finally, $\widetilde{F}(0)$ equals to $DATA$. Consequently, leader obtains the correct $DATA$.

REFERENCES

[1] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an optimized blockchain for IoT," in *Proc. 2nd Int. Conf. Internet Things Design Implement.*, Apr. 2017, pp. 173–178.

[2] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh, "IoT security: Ongoing challenges and research opportunities," in *Proc. IEEE 7th Int. Conf. Service-Oriented Comput. Appl. (SOCA)*, Nov. 2014, pp. 230–234.

[3] *Top Strategic Predictions for 2017 and Beyond: Surviving the Storm Winds of Digital Disruption*. [Online]. Available: https://www.gartner.com/doc/3471568?ref=unauthreader

[4] M. Abramowicz, "Cryptocurrency-based law," *Ariz. L. Rev.*, vol. 58, p. 359, 2016.

[5] Y.-A. de Montjoye, E. Shmueli, S. S. Wang, and A. S. Pentland, "OpenPDS: Protecting the privacy of metadata through safeanswers," *PLoS ONE*, vol. 9, no. 7, p. e98790, 2014.

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

[7] E. C. Ferrer. (2016). "The blockchain: A new framework for robotic swarm systems." [Online]. Available: https://arxiv.org/abs/1608.00695

[8] G. Brambilla, M. Amoretti, and F. Zanichelli. (2016). "Using blockchain for peer-to-peer proof-of-location." [Online]. Available: https://arxiv.org/abs/1607.00174

[9] X. Yue, H. Wang, D. Jin, M. Li, and W. Jiang, "Healthcare data gateways: Found healthcare intelligence on blockchain with novel privacy risk control," *J. Med. Syst.*, vol. 40, no. 10, p. 218, 2016.

[10] L. Wu, X. Du, W. Wang, and B. Lin, "An out-of-band authentication scheme for Internet of Things using blockchain technology," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Mar. 2018, pp. 769–773.

[11] A. Ouaddah, A. A. Elkalam, and A. A. Ouahman, "Towards a novel privacy-preserving access control model based on blockchain technology in IoT," in *Europe and MENA Cooperation Advances in Information and Communication Technologies*. Cham, Switzerland: Springer, 2017, pp. 523–533.

[12] S. Huh, S. Cho, and S. Kim, "Managing IoT devices using blockchain platform," in *Proc. 19th Int. Conf. Adv. Commun. Technol. (ICACT)*, Feb. 2017, pp. 464–467.

[13] A. Sonnino, M. Al-Bassam, S. Bano, and G. Danezis. (2018). "Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers." [Online]. Available: https://arxiv.org/abs/1802.07344

[14] M. Lamichhane, "A smart waste management system using IoT and blockchain technology," Tech. Rep., 2017.

[15] A. Dorri, S. S. Kanhere, R. Jurdak, and P. Gauravaram, "Blockchain for IoT security and privacy: The case study of a smart home," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2017, pp. 618–623.

[16] Y. Rahulamathavan, R. C.-W. Phan, M. Rajarajan, S. Misra, and A. Kondoz, "Privacy-preserving blockchain based IoT ecosystem using attribute-based encryption," in *Proc. IEEE Int. Conf. Adv. Netw. Telecommun. Syst. (ANTS)*, Dec. 2017, pp. 1–6.

[17] M. Conoscenti, A. Vetrò, and J. C. De Martin, "Peer to peer for privacy and decentralization in the Internet of Things," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2017, pp. 288–290.

[18] B. Liu, X. L. Yu, S. Chen, X. Xu, and L. Zhu, "Blockchain based data integrity service framework for IoT data," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 468–475.

[19] M. A. Krishnan, C. G. Shankar, S. A. Raj, and A. Ragavan, "Peer to peer file sharing by blockchain using IOT," Tech. Rep., 2017.

[20] P. K. Sharma, M.-Y. Chen, and J. H. Park, "A software defined fog node based distributed blockchain cloud architecture for IoT," *IEEE Access*, vol. 6, pp. 115–124, 2017.

[21] C. Pahl, N. El Ioini, and S. Helmer, "A decision framework for blockchain platforms for IoT and edge computing," in *Proc. Int. Conf. Internet Things, Big Data Secur.*, 2018.

[22] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, 2002.

[23] E. B. Sasson *et al.*, "Zerocash: Decentralized anonymous payments from bitcoin," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 459–474.

[24] [Online]. Available: https://getmonero.org

[25] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Project, Tech. Rep., 2014.

[26] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. OSDI*, vol. 99, 1999, pp. 173–186.

[27] S. Nick, "Smart contracts," 1994, to be published.

[28] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Proc. Annu. Int. Cryptol. Conf.*, 1992, pp. 139–147.

[29] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[30] [Online]. Available: https://crypto.stanford.edu/pbc/download.html

[31] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proc. 1st ACM Conf. Comput. Commun. Secur.*, 1993, pp. 62–73.

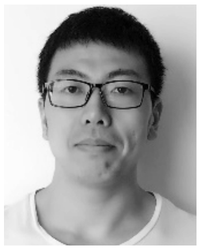
[32] P. S. L. M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *Selected Areas in Cryptography*. 2006.

[33] D. J. Bernstein and T. Lange. (2013). *SafeCurves: Choosing Safe Curves for Elliptic-Curve Cryptography*. [Online]. Available: http://safecurves.cr.yo.to

[34] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *Int. J. Inf. Secur.*, vol. 1, no. 1, pp. 36–63, 2001

[35] N. P. Smart, "The exact security of ECIES in the generic group model," in *Proc. IMA Int. Conf. Cryptogr. Coding*. Berlin, Germany: Springer, 2001, pp. 73–84.

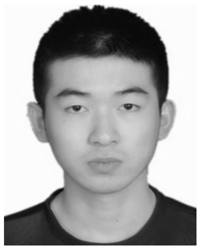
[36] [Online]. Available: https://github.com/ethereum/go-ethereum



LIJING ZHOU received the B.Sc. degree in mathematics and applied mathematics from Inner Mongolia Normal University, China, in 2009, and the M.S. degree in cryptography from Xidian University, China, in 2013. He is currently pursuing the Ph.D. degree in information security with the Beijing University of Posts and Telecommunications. His current research interests include blockchain technology and privacy, and cryptography.

LICHENG WANG received the B.S. degree in computer science from Northwest Normal University, China, in 1995, the M.S. degree in mathematics from Nanjing University, China, in 2001, and the Ph.D. degree in cryptography from Shanghai Jiao Tong University, China, in 2007. He is currently an Associate Professor with the Beijing University of Posts and Telecommunications.

YIRU SUN received the B.S. degree in mathematics and applied mathematics from Inner Mongolia Normal University, China, in 2009, and the M.S. degree in cryptography from Xidian University, China, in 2014. She is currently pursuing the Ph.D. degree in information security from the Beijing University of Posts and Telecommunications. Her current research interests include blockchain technology and privacy, and quantum cryptography.



PIN LV received the B.S. degree in software engineering from Jilin University, China, in 2015. He is currently pursuing the M.S. degree in computer science from the Beijing University of Posts and Telecommunications. His current research interests include blockchain technology and privacy, and cryptography.