BeeHive: Double Non-interactive Secure Multi-party Computation

Lijing Zhou[a], Licheng Wang[a,*], Yiru Sun[a], Tianyi Ai[a]

[a] *Beijing University of Posts and Telecommunications, Bei Jing* 100876*, P.R. China.*

Abstract

Currently, round complexity and communication complexity are two fundamental issues of secure multi-party computation (MPC) since all known schemes require communication for each multiplication operation. In this paper, we propose a double non-interactive secure multi-party computation, called BeeHive, that essentially addresses the two fundamental issues. Specifically, in the proposed scheme, the first non-interactivity denotes that shareholders can independently generate shares (these shares will be responses that are sent to the dealer) of any-degree polynomial of secret numbers without interaction. Furthermore, the second non-interactivity indicates that the dealer can verify correctness of responses sent by shareholders without interaction. To the best of our knowledge, it is the first work to realize that shareholders can generate shares of any-degree polynomial of secret numbers without interaction. Existing secure MPCs are not suitable for blockchain due to their issues of round complexity and communication complexity, while the proposed Beehive is very suitable. Therefore, we present a general architecture of blockchain-based Beehive. Finally, we implemented the proposed scheme in Python with a detailed performance evaluation.

Keywords: Secure multi-party computation, secret sharing, non-interactive, full homomorphism.

---

[*] Corresponding author
Email address: `wanglc2012@126.com` (Licheng Wang)

1. Introduction

Currently, secure multi-party computation (MPC) [1] is a very significant technology in the blockchain security. On the one hand, by using blockchain technology, distrustful players can achieve a consensus on history. On the other hand, secure MPC allows that distrustful players compute an agreed function of their inputs in a secure way. Even if some malicious players cheat, MPC can guarantee the correctness of output as well as the privacy of players' inputs. Therefore, the combination of secure MPC and blockchain has both academic and industrial significance.

However, there is a long-term problem that all existing information-theoretic secure MPCs have large round and communication complexity [2, 17, 3, 7, 8, 9, 10, 11, 12, 13, 14, 15]. In these constructions, it is the case that multiplication gates require communication to be processed (while addition/linear gates usually do not). In CRYPTO 2016, Damgård et al. [17] proposed that the number of rounds should be at least the (multiplicative) depth of the circuit, and the communication complexity is $O(ns)$ for a circuit of size s (n and s are the number of participants and the number of multiplication gates respectively).

Specifically, the issue of round and communication complexity existed because all such protocols follow the same typical "gate-by-gate" design pattern [17]: Players work through an arithmetic (boolean) circuit on secretly shared inputs, such that after they execute a sub-protocol that processes a gate, the output of gate is represented as a new secret sharing among these players. In particular, a Multiplication Gate Protocol (MGP) basically takes random shares of two values a, b from a field as input and random shares of ab as output.

In current blockchain platforms (e. g., Bitcoin [4], Ethereum [5] and EOS [6]), throughput is an important indicator for evaluating performance of blockchain platform. If every multiplication of secret numbers needs $O(n)$ communication complexity, the throughput will be seriously affected. Therefore, existing secure MPCs are not suitable for the requirements of blockchain since they require a certain number of interaction per multiplication of secret numbers.

In this paper, we mainly focus on non-interactive secure MPC, where players can process any-degree multiplication of secret numbers without interaction. It will have great significance for the application of secure MPC in the blockchain.

1.1. Our Results

Our contributions are summarized as follows:

- We propose a *Double Non-interactive Secure Multi-party Computation*, called BeeHive. The first non-interactivity denotes that shareholders can locally generate shares (these shares will be the responses sent to dealer) of any-degree polynomial of secret numbers without interaction. The second non-interactivity indicates that dealer can verify the correctness of responses sent by shareholders without interaction.

- We present detailed performance evaluation of BeeHive by deploying BeeHive on a Ubuntu 16.04 environment laptop. Specifically, the proposed BeeHive was implemented in Python on a two core of a 2.60GHz Intel(R) Core (TM) i7-6500U CPU with 8G RAM. We used high-speed Python Pairing-Based Cryptography (PBC) library [30] to compute point multiplication of elliptic curve and pairing, and utilized Python GNU Multiple Precision (GMP) Arithmetic Library [31] to calculate big number computation. According to the performance evaluation, the performance of proposed BeeHive is satisfactory. For instance, when the request is a 10-degree polynomial of secret value, generating a response takes about 0.0017263 s; verifying a response takes about 0.1221394 s; recovering a result takes about 0.0003862 s.

- A security analysis of BeeHive is presented. According to this security proof, we proved that malicious shareholders cannot get any information if the number of malicious shareholders is less than the threshold number.

1.2. Related Work

The round complexity and communication complexity of secure MPC have been two fundamental issues in cryptography. There are many studies about these two aspects. In this subsection, we will present related work about our study at first, then some comparisons between our previous paper [26] and this paper will be presented.

Round complexity. The round complexity of an ordered gate-by-gate protocol must be at least proportional to the multiplicative depth of the circuit [7]. The work of constant-round protocols for MPC was initially studied by Beaver et al. [18]. Subsequently, a long sequence of works constructed constant-round MPCs (e.g., 2-round [19, 15, 27, 3], 3-round [20], 4-round [21, 7, 12], 5-round [22, 16, 8] and 6-round [22]). In particular, in Eurocrypt 2004, Katz and Ostrovsky [16] established the exact round complexity of secure two-party computation with respect to blackbox proofs of security. In CRYPTO 2015, Ostrovsky et al. [12] provided a 4-round secure two-party computation protocol based on any enhanced trapdoor permutation, and Ishai et al. [15] obtained several results on the existence of 2-round MPC protocols over secure point-to-point channels, without broadcast or any additional setup. In Ecrypt 2017, Garg et al. [8] proposed several 5-rounds protocols by assuming quasi-polynomially-hard injective one-way functions (or 7 rounds assuming standard polynomially-hard collision-resistant hash functions). However, our scheme can solve any request of any-degree polynomial of secret numbers in 1-round.

Communication complexity. Initially, Rabin et al. [23] proposed that: To securely compute a multiplication of two secretly shared elements from a finite field based on one communication round, players have to exchange $O(n^2)$ field elements since each of n players must perform Shamir's secret sharing as part of the protocol. After that, Cramer et al. [24] further proposed a twist on Rabin's idea that enables one-round secure multiplication with just $O(n)$ bandwidth in certain settings, thus they reduced the communication complexity from quadratic to linear. Recently, in CRYPTO 2016, Damgård et al. [17] further presented that: In the honest majority setting, as well as for dishonest majority with preprocessing, any gate-by-gate protocol must communicate $O(n)$ bits for every multiplication gate, where n is the number of players. While, servers (shareholders) of our scheme can generate responses of any-degree polynomial of secret numbers without any interaction.

Comparisons with [26]. Currently, IoT and blockahin are experiencing exponential growth in industry and academia [25]. Recently, [26] proposed a blockchain-based IoT system with secure storage and homomorphic computation, called BeeKeeper. A key innovation of [26] is a secure multi-party computation scheme, which realized that shareholders can generate shares of two-degree polynomials of secret numbers without interaction. In this paper, we extend the secure MPC of [26] to propose a novel secure MPC scheme, called BeeHive. To the best of our knowledge, it is the first work to realize that shareholders can generate shares of any-degree polynomial of secret numbers without interaction. Essentially, BeeHive can be used to replace the secure MPC previously used in BeeKeeper [26] since the way of combining is the same. Consequently, we will mainly present the work principle and security analysis of BeeHive in this paper since how MPC is used in BeeKeeper can be found in [26]. By using BeeHive, servers of BeeKeeper can process any-degree (not just two-degree realized by [26]) polynomial of secret numbers and the verification of responses will be obviously more efficient than [26]. Temporarily, the secure MPC scheme proposed in [26] is called Pre-Scheme. In the following text, we will present the Pre-Scheme roughly at first. Then, we will show what BeeHive has improved.

Pre-Scheme has the following limitations:

- Servers (shareholders) can only generate shares of two-degree polynomial of secret numbers. In other words, servers cannot get any shares of k-degree ($k > 2$) polynomial of secret numbers.
- Pre-Scheme used the pairing (pairing is an expensive computation) to verify the correctness of responses (these responses are shares of two-degree polynomial of secret numbers) sent by servers.
- [26] did not include a complete security analysis of Pre-Scheme.

Based on Pre-Scheme, in this paper, we propose the BeeHive. Improvements of BeeHive are described as follows:

- Theoretically, servers can generate shares of any-degree polynomial of secret numbers without communicating with anyone.
- The dealer can verify the correctness of responses (shares of any-degree polynomial of secret numbers) sent by servers. In this verification process, BeeHive does not use pairing to verify responses, while Pre-Scheme used.
- We will present a complete security analysis of BeeHive. Moreover, this proof is also valid for the Pre-Scheme [26].

Organization. The remainder of the paper is organized as follows. An overview of BeeHive is shown in Sec. 2. Sec. 3 briefly presents preliminaries. We introduce BeeHive without verifiability and the verifiability of BeeHive in Sec. 4.1 and 4.2, respectively. Moreover, blockchain-based BeeHive is shown in Sec. 4.3. A detailed performance evaluation is shown in Sec. 5. A security analysis is presented in Sec. 6. Finally, a short conclusion is presented in Sect. 7.

2. An Overview of BeeHive

In the real world, there may be such a situation like the following one. That is, a user does not want to store data, and he does not want others to get the data, but he wants to use the data for calculation at any time. Therefore, the user has to store his encrypted data on some servers, and he hopes he can get any result generated with his data by requesting these servers. In this process, plaintext data and result of request should not be achieved by servers. Besides, the user may not hope there is only one server since if the single server is offline or malicious, the scheme will not work properly. Therefore, threshold scheme with multiple servers may be a reasonable choice. Specifically, if the user can achieve a threshold number of correct responses from threshold servers, he will obtain his desired result even if a part of these servers are offline or malicious. Finally, in order to check out malicious participants and incorrect messages, all key messages should be verifiable.

To address the above situation, in this paper, we propose a double non-interactive secure multi-party computation, called BeeHive. In BeeHive, dealer does not need to keep data (encrypted data are stored by servers), and he can achieve any result generated with his data by requesting servers as long as a threshold number of servers send correct responses to him. An overview of BeeHive is shown in Fig.1.

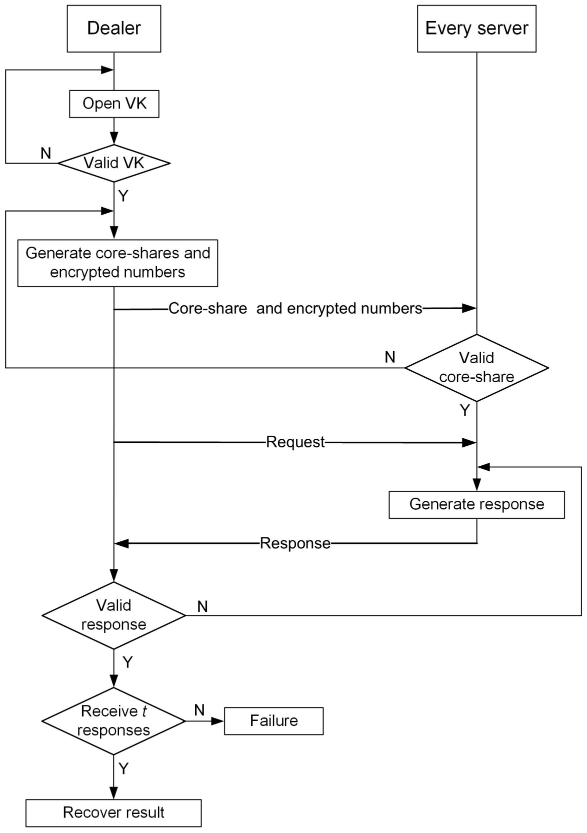

Figure 1: An overview of BeeHive

We hope to take the (t, n) BeeHive as an example to present the working process of the proposed scheme as follows:

- Step 1: The dealer generates n sets of core-shares and a verification key (VK). After that, he opens VK, then anyone (including servers) can verify whether VK is correctly computed by dealer. If VK is invalid, then the dealer has to regenerate the core-shares and VK, else the participants join in the next step.

- Step 2: The dealer secretly sends these n sets of core-share to n servers respectively. After receiving a core-share, a server can verify whether his core-share is valid by using VK. If the server's core-share is invalid, then he can ignore it and ask dealer to resend a core-share to him.

- Step 3: The dealer encrypts secret numbers into encrypted numbers, then he sends the encrypted numbers to servers.

- Step 4: When the dealer needs to get a result that is a polynomial of secret numbers, he will send a query to n servers.

- Step 5: According to the query sent by dealer, an active server will independently generate a response with his core-share (this process has no interaction with other servers), then the server will send his response to dealer securely.

- Step 6: After receiving responses, the dealer can verify whether responses are correctly computed by corresponding servers. These verifications do not need interaction with other servers. If a response is invalid, then the dealer can ignore this response or ask the corresponding server to resend a response to him. Finally, the dealer can recover the desired result if he can collect at least t correct responses.

BeeHive mainly has the following features:

- **Full homomorphism.** Servers can perform efficient homomorphic additions and multiplications on encrypted numbers without decrypting them.

- **Confidentiality.** Secret numbers shared by dealer with core-shares are always confidential as long as less than threshold number of servers are malicious.

- **Verifiability.** Verification key, core-shares and responses are verifiable.
  - *Verification key.* When the verification key is opened, anyone can verify its validity.
  - *Core-shares.* When a server receives a set of core-share, he can verify whether his core-share is correctly computed by the dealer by using VK. Moreover, in this method, the malicious dealer and incorrect core-shares can be checked out.
  - *Responses.* When the dealer gets a response sent by a server, the dealer would verify whether this response is correctly computed by the server. In this way, malicious servers and incorrect responses can be checked out.

3. Preliminaries

In this section, we hope to present basic cryptography techniques of BeeHive and the adversary model.

3.1. Shamir's (t, n) Secret Sharing

Alice wants to secretly share a secret value s with n participants, and arbitrary t of the n participants can recover s, but less than t participants cannot get anything. In order do this, Alice needs to generate n shares of s, then secretly sends the n shares to the n participants respectively. After that, if someone can collect at least t correct shares, then he can recover the secret value s. This problem can be resolved by Shamir's (t, n) secret sharing (SSS) [28]. In this subsection, we will present the working process of the SSS.

Firstly, Alice randomly samples a polynomial $f(x)$ of degree t-1 from $\mathbb{F}_p[x]$ (p is a big prime number) as the following polynomial:

$$f(x) = a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + \cdots + a_1x + s,$$

where s is the secret value as well as $a_1, \cdots, a_{t-1} \in \mathbb{F}_p$, $a_{t-1} \neq 0$.

Secondly, let P_1, P_2,...,P_n be the n participants and ID_i $(i = 1, 2, ..., n)$ denote P_i's address. Alice generates P_i's share as follow:

$$Share_i = f(ID_i),$$

where i=1, 2, ...,n. Then, Alice secretly sends $Share_1$, $Share_2$, ...,$Share_n$ to the n participants, respectively.

Finally, if someone collects t correct shares, then he can use the *lagrange interpolation* to reconstruct the polynomial $f(x)$. Without loss of generality, let the t shares be $Share_1$, $Share_2$, ...,$Share_t$. He can reconstruct the polynomial $f(x)$ as follow:

$$f(x) = \sum_{i=1}^{t} Share_i \prod_{j=1, j\neq i}^{t} \frac{x - ID_j}{ID_i - ID_j}.$$

Consequently, he can get $s = f(0)$.

Addition homomorphism of SSS. SSS naturally has the additional homomorphism. It means that the sum of shares is the share of the sum of corresponding secrets. Moreover, the threshold number is always immutable during this process since the degree of the sum of shared polynomials is equal to the degree of shared polynomials. Therefore, if a dealer can collect threshold number of sum shares, he can reconstruct the corresponding polynomial and then get the sum of secrets. Consequently, SSS naturally has the additional homomorphism.

Multiplication homomorphism of SSS. Similarly, SSS naturally also has the multiplicative homomorphism. It denotes that the product of shares is the share of the product of corresponding secrets. However, the multiplicative homomorphism has a big limitation that is, with the degree growth of product of secrets, the degree result polynomial will become larger and larger. Under this process, it will eventually arrive at a threshold larger than n so that the final result cannot be reconstructed. Finally, the multiplicative homomorphism of SSS is restricted.

3.2. Pairing

In BeeHive, the pairing computation is only used in the verification process of verification key. After that, pairing will not be used anymore. Namely, it however will not be used in the verification processes of core-shares and respons-es.

Let $\mathbb{G}$ and $\mathbb{G}_T$ be the cyclic groups of a large prime order q. G is the generator of $\mathbb{G}$. A cryptography pairing [29] e (bilinear map): $\mathbb{G}\times\ \mathbb{G} \rightarrow \mathbb{G}_T$ is a map that has a property of bilinearity. The bilinearity means that

$$e(aG, bG) = e(G, G)^{ab},$$

where $a, b \in \mathbb{Z}_q$.

Remark 1. *In the proposed scheme, pairing is only used in verifying VK. In other words, it will not be used in any other process anymore.*

3.3. Adversary Model

In this subsection, we will take the (t, n) BeeHive as an example to present the adversary model of the proposed scheme. In this adversary model, there are n servers. In these n servers, we assume that more than $n-t$ servers are honest. Otherwise, this scheme is not secure. Specifically, t malicious servers can recover secret numbers from their core-shares, but if the number of malicious servers is less than t, these malicious servers would get nothing.

Honest. An honest user means that (i) he honestly processes his data, (ii) he does not jointly work with others and (iii) he secretly keeps his secret data without exposing it to others.

Malicious. A malicious user denotes that (i) he can dishonestly process his data, (ii) he can jointly work with other malicious users and (iii) he can expose his secret data to other malicious users.

In the full version of BeeHive, we have the following assumptions:

- The dealer could dishonestly generate the verification key (VK) and core-shares, but he can not expose any secret data to servers.

- Some malicious servers may not generate responses or respond incorrectly, but it is crucial to have at least $n - t$ servers honest and active.

4. Construction of BeeHive

In this section, to clearly present the work process of BeeHive, we will present the BeeHive without verifiability at first. Then we will give out the verifiability of BeeHive. Finally, basic applications of BeeHive combined with blockchain will be illustrated.

4.1. BeeHive without verifiability

In this subsection, we will present the BeeHive without verifiability, where data-senders (dealer and servers) are all honest. Namely, all data-recepients (servers and dealer) do not need to verify data received. While, in the next section, we will specifically show the verification processes of BeeHive, where the dealer and servers could be dishonest, and a (t, n) BeeHive will be taken as an example to present the scheme without verifiability. It contains a dealer and n servers. Let Sr_i denote the i-th server and ID_i be the ID of Sr_i.

4.1.1. Generation of Core-share, Request, Response and Result

Assume the dealer wants to get $V = \sum_{i=0}^{k} b_i s^i$, where s is the key secret value shared among servers. Therefore, the dealer needs to send $request = \{b_k, b_{k-1}, ..., b_1, b_0\}$ to every server. According to the *request*, a server can use his data to generate a response of V for dealer. It must be pointed that servers cannot get s or V in this process although they get the *requst*. Before presenting the real working process, we will provide some mathematical principles at first, which can help to understand the process of generating responses.

- Let $f(x) = w_{t-1}x^{t-1} + w_{t-2}x^{t-2} + ... + w_1 x + s$ be a random $(t-1)$-degree polynomial over $\mathbb{F}_q$.
- Let
$$S(x) = \sum_{i=1}^{k} b_i f(x)^i + b_0.$$
We know that $S(0) = \sum_{i=0}^{k} b_i s^i$ since $f(0) = s$. However, the degree of $S(x)$ is $kt - k$.

- In order to reduce the degree of $S(x)$ to $t-1$ as well as keep its constant term being $\sum_{i=0}^{k} b_i s^i$, we now present polynomials $h_2(x), h_3(x), ..., h_k(x)$. They can be constructed as follows:
  - Randomly sample $(t-1)$-degree polynomials $c_2(x), c_3(x), ..., c_k(x)$, $c_i(0) = 0$, $i = 2, 3, ..., k$.
  - i from 2 to k, construct

$$h_i(x) = f(x)^i - c_i(x) - s^i.$$

- Compute

$$\begin{aligned} H(x) = &\ S(x) - \textstyle\sum_{j=2}^{k} b_j h_j(x) \\ = &\ \textstyle\sum_{i=1}^{k} b_i f(x)^i + b_0 - \sum_{j=2}^{k} b_j h_j(x) \\ = &\ \textstyle\sum_{j=2}^{k} b_j (c_j(x) + s^j) + b_1 f(x) + b_0. \end{aligned} \quad (1)$$

- $H(x)$ is a polynomial of $t-1$ since $c_k(x)$, $c_{k-1}(x)$,..., $c_2(x)$ and $f(x)$ are of degree $t-1$. Moreover, we have

$$H(0) = \sum_{i=1}^{k} b_i s^i + b_0 = V.$$

Thereby, $H(x)$ is the desired polynomial. $H(x)$ can be reconstructed if someone can obtain at least t correct shares of $H(x)$, then he can obtain $\sum_{i=0}^{k} b_i s^i$ by computing $H(x)|_{x=0}$.

Question: How does a server generate his share of the $H(x)$as his response?

According to the above process of computing $H(x)$, servers can work as follows to help dealer to secretly obtain $\sum_{i=1}^{k} b_i s^i + b_0$:

- **Core-shares** The dealer randomly samples $f(x)$, $c_2(x)$, $c_3(x)$, ..., $c_k(x)$. They are polynomials of degree $t-1$ and $f(0) = s$, $c_j(0) = 0$ $(j = 2, 3, ..., k)$. Then, the dealer generates polynomials $h_2(x), h_3(x), ..., h_k(x)$ as above process, and then generates core-share for each server. For instance, Sr_i's core-share is

$$\text{core-share}_i = \{f(ID_i)||h_2(ID_i)||h_3(ID_i)||...||h_k(ID_i)\}$$

, $i = 1, 2, ..., n$. Then, the dealer secretly sends core-share$_i$ to Sr_i.

- **Request** Assume that the dealer wants to get the result $V = \sum_{i=0}^{k} b_i s^i$. Therefore, he will send a request to n servers to get feedback from them. Specifically, the request includes the following numbers:

  $$\{b_k, b_{k-1}, ..., b_1, b_0\}$$

  According to the request, an active server will know that the dealer wants to get the result $\sum_{i=1}^{k} b_i X^i + b_0$, where X is the secret value s. However, servers do not know what X is.

- **Responses** If Sr_i wants to respond the request, he can use $f(ID_i)$, $h_2(ID_i)$, $h_3(ID_i)$,..., $h_k(ID_i)$ to generate his response $Resp_i$ (it is a share of $H(x)$) as follow:

  $$Resp_i = \sum_{j=1}^{k} b_j f(ID_i)^j + b_0 - \sum_{j=2}^{k} b_j h_j(ID_i).$$

  Then, Sr_i sends $Resp_i$ to the dealer secretly.

- **Result** If the dealer can collect t responses like $Resp_i$, then he can use the lagrange interpolating to recover the $t-1$-degree polynomial $H(x)$. Finally, the dealer can get the desired result $\sum_{i=0}^{k} b_i s^i$ by computing $H(x)|_{x=0}$.

Remark 2. *We know that the degree of request $\sum_{i=0}^{k} b_i s^i$ is k. However, the value k is unlimited. Namely, the dealer can purposefully set the k according to his requirements by providing enough core-shares to servers. Therefore, servers of BeeHive can process any-degree polynomials of secret numbers in theoretically as long as the dealer can provide enough core-shares to servers. For instance, servers can generate responses of 50-degree polynomials of secret number if their core-shares are similar to $f(ID_i), h_2(ID_i), ..., h_{50}(ID_i)$.*

4.1.2. BeeHive with Sharing Encrypted Numbers

In this subsection, we will add a feature of sharing encrypted numbers on BeeHive. Specifically, the dealer has a set of secret numbers that are

$d_1, d_2, ..., d_m$. After randomly sampling $f(x)$ ($f(x)$ is a (t-1)-degree polynomial and $f(0) = s$), the dealer performs as follows:

- Encrypt $d_1, d_2, ..., d_m$ into $a_1, a_2, ..., a_m$ as follows:

$$a_j = d_j - s, j = 1, 2, ..., m.$$

- Secretly sending core-share$_i$ to Sr_i, i from 1 to n.
- Open $a_1, a_2, ..., a_m$.

After that, the dealer will send a request about the encrypted numbers $a_1, a_2, ..., a_m$. Then servers can generate corresponding responses according to the request. Next, we will present how dealer and servers work with $a_1, a_2, ..., a_m$.

At first, assume that: i) the largest degree of addressable request is k, ii) the dealer has secretly sent $\{f(ID_i)||h_2(ID_i)||h_3(ID_i)||...||h_k(ID_i)\}$ to Sr_i, $i = 1, 2, ..., n$, and iii) he has also opened $\{a_1, a_2, ..., a_m\}$. Then, servers can help the dealer to get any result like the following formula:

$$\sum_{t=1}^{w} \prod_{j=1}^{v_t} \mu_{t,d} d_{j_{t,d}},$$

where $v_t \le k$ and $\mu_{t,d} \in \mathbb{F}_q$, $j_{t,d} \in \{1, 2, ..., m\}$. The dealer sends a string to servers like the following one:

$$\sum_{t=1}^{w} \prod_{j=1}^{v_t} \mu_{t,d}(X + a_{j_{t,d}}),$$

due to $d_{j_{t,d}} = s + a_{j_{t,d}}$.

After receiving the above request, Sr_i can transmit the string into a polynomial of x as follow:

$$W(x) = \sum_{t=1}^{w} \prod_{j=1}^{v_t} \mu_{t,d}(x + a_{j_{t,d}}) = \sum_{j=0}^{k} b_j x^j.$$

At this moment, the polynomial $W(x)$ can be seen as the request mentioned in Sec. 4.1.1. Therefore, servers can use $W(x)$ to generate responses, and the subsequent work is the same as the corresponding work mentioned in Sec. 4.1.1.

Remark 3. *Servers and dealer may transmit the string of request into a addressable polynomial. After that, the processes of generating responses and verifying responses are the same as the original BeeHive. Servers cannot get* $d_1, d_2, ..., d_m$ *from* $a_1, a_2, ..., a_m$ *as long as the key secret value* s *is secretly protected by servers. We will analyze the security of BeeHive in Sec. 6.*

4.2. Verifiability of BeeHive

In Sec. 4.1, we described the BeeHive without verification, and we assumed that data-senders (dealer and servers) are honest. However, in practical applications, data-senders might incorrectly compute data which would lead to the corresponding data-recepients generates wrong results. Therefore, data-recepients (servers or dealer) should verify whether received data (core-shares or responses) are correctly computed by corresponding data-senders. In this way, malicious data-senders and incorrect data can be checked out. Therefore, in this section, we will present how data-recepients verify received data.

Specifically, compared with the BeeHive without verifiability mentioned in Sec.4.1, the full BeeHive adds four parts: (i) the dealer generates and opens the verification key; (ii) anyone can verify the correctness of the verification key; (iii) the server can verify the correctness of his core-share; (iv) the dealer can verify the correctness of responses sent by servers.

Without loss of generality, we take the $(2, 3)$ BeeHive as an example (The (t, n) BeeHive can be similarly constructed since it is similar to $(2, 3)$ BeeHive.). The BeeHive contains a dealer and three servers as well as a server can respond at most k-degree request included in the query. Let Sr_1, Sr_2, Sr_3 denote the three servers. Furthermore, the dealer can recover the desired result if at least two servers generate responses to the dealer honestly. Moreover, ID_i is the ID of Sr_i, $i = 1, 2, 3$. Furthermore, in the underlying contents, let g denote a generator of a cyclic group. We will use g^a to compute a commitment to hide a. In the next text, we will present how to verify verification key (VK), core-shares and responses.

4.2.1. Verify Verification Key

Before the dealer sends the core-shares to servers, he would generates a verification key (VK) that will be used in the future verifications. The VK is constructed as follows:

- The dealer randomly samples $f(x), c_2(x), ..., c_k(x)$ from $\mathbb{F}_p[x]$. $f(x), c_2(x), ..., c_k(x)$ are polynomials of degree $t-1$ as follows:

$$\begin{aligned} f(x) &= b_2^f x^2 + b_1^f x + s, \\ c_2(x) &= b_2^{c_2} x^2 + b_1^{c_2} x, \\ c_3(x) &= b_2^{c_3} x^2 + b_1^{c_3} x, \\ &.... \\ c_k(x) &= b_2^{c_k} x^2 + b_1^{c_k} x, \end{aligned} \tag{2}$$

  Then, the dealer computes $h_r(x)$ $(r = 2, 3, ..., k)$ as follow:

$$h_r(x) = f(x)^r - c_r(x) - s^r = \textstyle\sum_{j=1}^{2r} b_j^{h_r} x^j. \tag{3}$$

- Let CM_X denote the commitment of X. X may be a constant or polynomial. Specifically,

  - $CM_a = g^a$ when a is a constant.
  - $CM_{\{h(x)\}} = \{g^{b_i} | h(x) = b_r x^r + b_{r-1} x^{r-1} + ... + b_1 x + b_0, i = 0, 1, 2, ..., r\}$ when $h(x)$ is a polynomial of x. For instance, if $h(x) = b_3 x^3 + b_2 x^2 + b_1 x + b_0$, then

$$CM_{\{h(x)\}} = \{g^{b_3} || g^{b_2} || g^{b_1} || g^{b_0}\}.$$

  Let $Hash(\cdot)$ be a hash function. The dealer computes the following commitments:

  - $CM_{\{f(x)\}}$.
  - $CM_{\{c_j(x)\}}$, $CM_{\{h_j(x)\}}$ and CM_{s^j}, $j = 2, 3, ..., k$.
  - $CM_{f(r)^j}$, where $r = Hash(CM_{\{f(x)\}})$ and $j = 2, 3, ..., k$.

- The verification key (VK) is as follow:

Verification key			
	$CM_{\{f(x)\}}$		
$CM_{\{c_2(x)\}}$	$CM_{\{c_3(x)\}}$	...	$CM_{\{c_k(x)\}}$
$CM_{\{h_2(x)\}}$	$CM_{\{h_3(x)\}}$	...	$CM_{\{h_k(x)\}}$
CM_{s^2}	CM_{s^3}	...	CM_{s^k}
$CM_{f(r)^2}$	$CM_{f(r)^3}$	...	$CM_{f(r)^k}$

Anyone (include servers) can verify the correctness of verification key (VK). The correctness of VK means that commitments of $f(x), c_2(x), h_2(x), c_3(x), h_3(x), ..., c_k(x), h_k(x)$ satisfy the following requirements:

- $f(x), c_2(x), c_3(x), ..., c_k(x)$ are polynomials of degree $t-1$.
- $c_j(0) = 0$, $j = 2, 3, ..., k$.
- $h_j(x) = f(x)^j - c_j(x) - s^j$, $j = 2, 3, ..., k$.

Specifically, a verifier can verify VK as follows:

- j from 2 to k, if the following equation holds, then the commitment of s^j is valid.

$$e(CM_s, CM_{s^{j-1}}) = e(CM_{s^j}, g).$$

  If above equation does not holds for any j, the verifier would stop his verifications of VK and concludes that the dealer did not generate the VK honestly.

- Compute $r' = Hash(CM_{\{f(x)\}})$.
- Compute $g^{f(r')}$ by using the following equation:

$$g^{f(r')} = (CM_{b_2^f})^{r'^2}(CM_{b_1^f})^{r'}CM_s.$$

- j from 2 to k, if the following equation holds, the commitment of $f(r)^j$ is correct.

$$e(g^{f(r')}, CM_{f(r)^{j-1}}) = e(CM_{f(r)^j}, g).$$

If above equation does not holds for any j, the verifier would stop his verifications of VK and concludes that the dealer did not generate the VK honestly.

- j from 2 to k, the verifier computes $g^{c_j(r')}$ by using the following equation:

$$g^{c_j(r')} = \prod_{t=1}^{2} (CM_{b_t^{c_j}})^{r'^t},$$

where $CM_{b_t^{c_j}}$ is included in the $CM_{\{c_j(x)\}} = \{g^{b_1^{c_j}} || g^{b_2^{c_j}}\}$.

- j from 2 to k, compute $g^{h_j(r')}$ by using the following equation:

$$g^{h_j(r')} = \prod_{t=1}^{2j} (CM_{b_t^{h_j}})^{r'^t},$$

where $CM_{b_t^{h_j}}$ is included in the $CM_{\{h_j(x)\}} = \{g^{b_1^{h_j}} || g^{b_2^{h_j}} || ... || g^{b_{2j}^{h_j}}\}$.

- j from 2 to k, if the following equation holds, then the commitments of $c_j(x)$ and $h_j(x)$ are correct.

$$g^{h_j(r')} = CM_{f(r)^j} / (g^{c_j(r')} CM_{s^j}).$$

If above equation does not holds for any j, the verifier would stop his verifications of VK and concludes that the dealer did not generate the VK honestly.

Finally, if the VK passes all the above verifications, then it can be seen valid. After that, the verifier can use the VK to verify core-shares and responses.

4.2.2. Verify Core-shares

In this subsection, we will present how Sr_i verifies his core-share. Assume that the VK has been verified and it is valid. In BeeHive, the core-share of Sr_i are as follows:

$$f(ID_i), h_2(ID_i), h_3(ID_i), ..., h_k(ID_i).$$

Because commitments of coefficients of $f(x)$, $c_2(x)$, ...,$c_k(x)$, $h_2(x)$, ..., $h_k(x)$ have been provided in VK as well as VK is valid, so Sr_i can verify his core-shares with these commitments and ID_i. Moreover, the verification processes of

$f(ID_i)$, $h_2(ID_i)$, $h_3(ID_i)$, ..., $h_k(ID_i)$ are similar to each other. Therefore, we are going to take the verification process of $f(ID_i)$ as an example. Specifically, if the following equation holds, then the $f(ID_i)$ is correct.

$$g^{f(ID_i)} = (CM_{b_2^f})^{ID_i^2}(CM_{b_1^f})^{ID_i}CM_s.$$

Anyone of $f(ID_i)$, $h_2(ID_i)$, $h_3(ID_i)$, ..., $h_k(ID_i)$ can be verified as the same process above. For another instance, if the following equation holds, then $h_3(ID_i)$ can be seen correct.

$$g^{h_3(ID_i)} = \prod_{j=1}^{6}(CM_{b_j^{h_3}})^{ID_i^j}.$$

4.2.3. Verify Responses

In BeeHive, the dealer can verify whether a response is correctly computed by the corresponding server. In this subsection, we will take the case of request being $\sum_{i=1}^{k} b_i s^i + b_0$ as an example to present the process of verifying response. Specifically, the dealer verifies the response $Resp_i$ generated by Sr_i $(i = 1, 2, ..., n)$ as follows:

According to Sec. 4.1.1, we know

$$Resp_i = \textstyle\sum_{t=2}^{k} b_t(c_t(ID_i) + s^t) + b_1 f(ID_i) + b_0.$$

Consequently, the dealer can verify the $Resp_i$ as follows:

- Compute $CM_{f(ID_i)}$ as follows:

$$CM_{f(ID_i)} = \prod_{j=0}^{2}(CM_{b_j^f})^{ID_i^j}.$$

- Compute $CM_{c_t(ID_i)}$ $(t = 2, 3, ..., k)$ as follows:

$$CM_{c_t(ID_i)} = \prod_{j=1}^{2}(CM_{b_j^{c_t}})^{ID_i^j}.$$

- If the following equation holds, then the response $Resp_i$ is correct.

$$g^{Resp_i} = \prod_{t=2}^{k}[CM_{c_t(ID_i)}CM_{s^t}]^{b_t}[CM_{f(ID_i)}]^{b_1}CM_{b_0}.$$

4.3. Blockchain-based BeeHive

Currently, blockchain [32] is experiencing exponential growth in industry and academia. Blockchain can provide decentralization and high credibility to users due to its collective verification and tamper resistance. Therefore, it can also provide high credibility and convenience to users of BeeHive. Benefits of blockchain-based BeeHive are as follows:

- Blockchain provides tamper-resistance to all users of BeeHive. Namely, once data have been recorded in the blockchain, they can be seen immutable. Besides, verification key (VK) is the security base of BeeHive. Therefore, if the correctness and tamper-resistance of VK cannot be guaranteed, then scheme would be insecure. For instance, if VK of BeeHive is opened on some centralized data center, the center could modify or delete this VK since the center could be corrupted by some malicious adversary. However, if VK is opened in the blockchain, then all users can consider that VK as credible and immutable.

- Verifiers of blockchain can help users of BeeHive to verify all publicly verifiable data. Thereby, verifiers of blockchain can verify the VK included in the transaction before it is recorded in the blockchain. Consequently, only valid VK can be recorded in the blockchain. Similarly, most verification of commitments of core-shares and responses are publicly verifiable, and the process of these verification does not release the plain-text of core-shares and responses, thus this verification can also be performed by verifiers of blockchain. In that way, invalid commitments of core-shares and responses cannot be recorded in the blockchain. Therefore, if commitments of core-shares and responses have been recorded in the blockchain, the corresponding receivers can consider these commitments as valid, now they only need to decrypt the encrypted core-shares or responses, then compare the commitments of plain-text core-shares or responses to verify they are the same as the commitments on the blockchain.

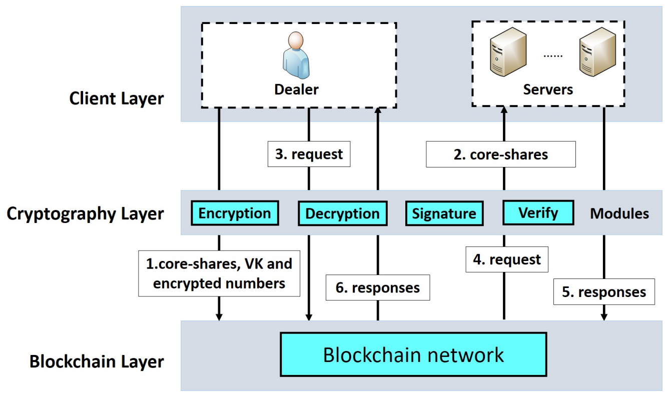

Figure 2: Architecture of blockchain-based BeeHive

- Processing power limitation is broken. Generally speaking, in some specific application, the processing power of servers has a upper-limit since servers are controlled by some company. Therefore, in this situation, external computing resources are hard to join in this system of servers to increase the processing power of the entire system. However, in blockchain-based BeeHive, a user can arbitrarily choose some nodes as his servers as long as these nodes are willing. Thus if the blockchain-based BeeHive can be applied in practice, the processing power upper-limit of the entire system may gradually increase with the number of server nodes increases.

- Servers and dealer do not need to store encrypted data in their location since these encrypted can be stored by the blockchain. In this way, servers and dealer do not need a lot of storage space.

Fig. 2 presents a illustrative architecture of blockchain-based BeeHive. We would not present too much details of blockchain-based BeeHive since it is similar to BeeKeeper [26]. The details can be found in [26]. In the following text, we will present the rough working process.

In this architecture, there are two groups of participants. The first group contains the record-nodes of blockchain. Record-nodes are full nodes of blockchain who are responsible for verifying all publicly verifiable data. Once the data are confirmed to be valid, related transactions including these data will be recorded in the blockchain by record-nodes. Otherwise, the related transaction will not be recorded. The second group contains "dealer and servers". All these participants only trust information recorded in the blockchain, and they communicate to others with the payload field of a transaction in the blockchain system.

The working process of blockchain-based BeeHive are as follows:

1. Dealer generates VK, core-shares and encrypted numbers. He writes VK and encrypted numbers in transaction T_{VK} and $\mathrm{T}_{enumbers}$ respectively. After that, he computes commitments of core-shares and encrypts core-shares with corresponding servers' public keys, then he writes these commitments and encrypted core-shares in $\mathrm{T}_{core-share}$. Finally, he sends T_{VK}, $\mathrm{T}_{enumbers}$ and $\mathrm{T}_{core-share}$ to blockchain network. After record-nodes receive T_{VK} and $\mathrm{T}_{core-share}$, they verify whether VK and commitments of core-shares are valid. If they are valid, record-nodes record these data in the blockchain, else they reject corresponding transactions.
2. After seeing $\mathrm{T}_{core-share}$ in the blockchain, servers can consider these commitments of cores-shares are valid. Servers can decrypt encrypted core-shares by using their private keys respectively. After that, a server just need to check whether the commitments of plain-text of core-share are equal to commitments recorded in the $\mathrm{T}_{core-share}$. If they are equal to each other, the server's core-share can be seen as valid, else the server can ask dealer to resend his core-share.
3. When the dealer wants to get a result, he can send a transaction $\mathrm{T}_{request}$ including his request to the blockchain network.
4. After seeing $\mathrm{T}_{request}$ in the blockchain, an active server will generate a response according dealer's request with encrypted numbers recorded in the blockchain, then he would encrypt his response with dealer's public key. After that, he would send a transaction $\mathrm{T}_{response}$ including the encrypted response and a commitment of this response to the blockchain network. After record-nodes receive $\mathrm{T}_{response}$, they verify whether the commitment

of response is valid. If the commitment is valid, then record-nodes will record T$_{response}$ in the blockchain, otherwise they reject this transaction.

5. After seeing T$_{response}$ in the blockchain, the dealer can consider that the commitment of response included in the T$_{response}$ is valid. Then the dealer decrypts the encrypted response with his private key. After that, the dealer just needs to check whether the commitment of the plain-text response is equal to the commitment recorded in T$_{response}$. If the two commitments are the same, the response can be seen as valid, otherwise the dealer can ask corresponding server to resend a response to dealer.
6. If the dealer can collect at least threshold number of responses, the desired result can be recovered correctly, else the scheme fails.

5. Performance Evaluation

In this section, we will present a performance evaluation of BeeHive by deploying BeeHive on a Ubuntu 16.04 environment laptop. Specifically, the BeeHive was implemented in Python on a two core of a 2.60GHz Intel(R) Core (TM) i7-6500U CPU with 8G RAM. We used high-speed Python Pairing-Based Cryptography (PBC) library [30] to compute point multiplication of elliptic curve and pairing, and utilized Python GNU Multiple Precision (GMP) Arithmetic Library [31] to calculate big number computation.

In the our experiments, BeeHive was divided into seven functions: *Gen_VK, Ver_VK, Gen_core-share, Ver_core-share, Gen_response, Ver_response* and *Rec_result*. These functions are used as follows:

- *Gen_VK:* Dealer uses *Gen_VK* to generate verification key (VK).
- *Ver_VK:* Servers can use *Ver_VK* to verify the validation of VK.
- *Gen_core-share:* Dealer uses *Gen_core-share* to generate core-shares for servers.
- *Ver_core-share:* Servers can use *Ver_core-share* and VK to verify core-shares sent by dealer.

- *Gen_response:* Servers can use *Gen_response* to generate responses according to the request sent by dealer.
- *Ver_response:* Dealer can use *Ver_response* and VK to verify the validation of responses sent by servers.
- *Rec_result:* Dealer can use *Rec_result* to recover desired result with the threshold number of correct responses.

In practical applications, these functions belong to different participants (dealer and servers). The affiliation of these functions is shown in Table 1.

Table 1: Functions of Participant

Participant	**Functions of Participant**
Dealer	*Gen_VK, Gen_core-share, Ver_VK* *Ver_response, Rec_result*
Server	*Ver_VK, Ver_core-share, Gen_response*

Essentially, we performed two types of tests as follows:

- **Test 1:** We deployed (3,7) BeeHive (a total 7 servers, and the desired result can be recovered with at least 3 valid responses) on our laptop. Let k be the largest degree of addressable request, we set k from 4 to 10. We tested the performance of *Gen_core-share, Gen_VK, Ver_core-share* and *Ver_VK*. The results of Test 1 are shown in Table 2.
- **Test 2:** We also deployed (3,7) BeeHive on our laptop. Let the largest degree of addressable request be constant 10. We set the degree of request from 2 to 10. We tested the performance of *Rec_result, Gen_response* and *Ver_response*. The results of Test 2 are shown in Table 3.

Table 2: Performance of functions with the change of the largest degree of addressable request (second)

k	*Gen_core-share*	*Ver_core-share*	*Gen_VK*	*Ver_VK*
4	0.000329733	0.003045559	0.127948046	0.148387432
5	0.000474215	0.004627943	0.155535466	0.237745762
6	0.000656843	0.005952125	0.201929808	0.368674755
7	0.000918154	0.007400751	0.259971857	0.536798954
8	0.001402143	0.010572672	0.317357063	0.766717434
9	0.001489878	0.012337208	0.375114679	1.056947708
10	0.002088308	0.014226913	0.435570243	1.331381321
11	0.002505302	0.017073154	0.493043661	1.681715012
12	0.003757325	0.021838427	0.572894812	2.194091082

We deployed (3,7) BeeHive to show the performance of *Gen_core-share*, *Gen_VK,* *Ver_core-share* and *Ver_VK*. **k** denotes the largest degree of addressable request. The finite field is based on a 256-bit big prime number.

6. Security Analysis

In this section, we will take the (t, n) BeeHive as an example to discuss the confidentiality of the proposed BeeHive, and the highest degree of polynomial that the dealer can query is k. Because the BeeHive is a threshold cryptography scheme, its confidentiality means that $t-1$ malicious servers cannot jointly recover the key secret value s, unless the number of servers reaches t. According to Sec. 4.1, the secretly shared polynomials are:

$$f(x), h_2(x), h_3(x), ..., h_k(x).$$

Moreover, each honest server secretly keeps his core-share. For instance, the server Sr_i $(i = 1, 2, ..., n)$ secretly keeps his core-share:

$$f(ID_i), h_2(ID_i), h_3(ID_i), ..., h_k(ID_i).$$

Besides, we know $c_j(x) = f(x)^j - h_j(x) - s^j$ $(j = 2, 3, ..., k)$ due to $h_j(x) = f(x)^j - c_j(x) - s^j$ as mentioned in Sec. 4.1. Therefore, secretly shared polyno-

Table 3: Performance of functions with the change of degree of request (second)

Degree of request	*Rec_result*	*Gen_response*	*Ver_response*
2	0.000478268	0.001681805	0.007747173
3	0.000378373	0.001621246	0.016930582
4	0.000452042	0.001915455	0.023883823
5	0.000365019	0.001704931	0.032199383
6	0.000339508	0.001774073	0.049633026
7	0.000391245	0.001723289	0.065804005
8	0.000404119	0.001615524	0.081865788
9	0.000323057	0.001732349	0.096564293
10	0.000386238	0.001726389	0.122139454

We deployed (3,7) BeeHive with the largest degree of addressable request being 10 to show the performance of *Rec_result, Gen_response* and *Ver_response*. Moreover, the finite field is based on a 256-bit big prime number.

mials are equivalent to:

$$f(x), c_2(x), c_3(x), ..., c_k(x).$$

The $t-1$ malicious servers do not know anything about $f(x), c_2(x), c_3(x), ..., c_k(x)$ except that $f(x), c_2(x), c_3(x), ..., c_k(x)$ are polynomials of degree $t-1$ and $c_j(0) = 0$, j=2,3,...,k. In other words, they only know $f(x), c_2(x), c_3(x), ..., c_k(x)$ have the following expressions, but they have no idea about the coefficients of these equations.

$$\begin{array}{rcl} f(x) & = & b_{t-1}^{f}x^{t-1} + b_{t-2}^{f}x^{t-2} + ... + b_1^{f}x + s, \\ c_2(x) & = & b_{t-1}^{c_2}x^{t-1} + b_{t-2}^{c_2}x^{t-2} + ... + b_1^{c_2}x, \\ c_3(x) & = & b_{t-1}^{c_3}x^{t-1} + b_{t-2}^{c_3}x^{t-2} + ... + b_1^{c_3}x, \\ ...... & .. & .... \\ c_k(x) & = & b_{t-1}^{c_k}x^{t-1} + b_{t-2}^{c_k}x^{t-2} + ... + b_1^{c_k}x. \end{array}$$

Essentially, we want to discuss why the $t-1$ malicious servers cannot recover s by using their core-shares although they work together. Without loss of generality, we assume that $Sr_1, Sr_2, ..., Sr_{t-1}$ are the $t-1$ malicious servers as well as other servers are honest. According to Sec. 4.1, we know that core-share kept by Sr_i $(i = 1, 2, ..., t-1)$ is $f(ID_i)$, $h_2(ID_i)$, $h_3(ID_i)$, ..., $h_k(ID_i)$ as shown in Table 4.

Table 4: Core-shares of servers

Server	Sr_1	Sr_2	...	Sr_{t-1}
Core-shares	$f(ID_1)$	$f(ID_2)$	...	$f(ID_{t-1})$
	$h_2(ID_1)$	$h_2(ID_2)$	...	$h_2(ID_{t-1})$
	$h_3(ID_1)$	$h_3(ID_2)$	...	$h_3(ID_{t-1})$
	...	...	...	...
	$h_k(ID_1)$	$h_k(ID_2)$	...	$h_k(ID_{t-1})$

In order to solve coefficients of $f(x), c_2(x), c_3(x), ..., c_k(x)$, the $t-1$ malicious servers can construct linear equations by using their core-shares as follows:

$$\begin{cases} b^f_{t-1}(ID_1)^{t-1} + ... + b^f_1 ID_1 + s = & f(ID_1) \\ b^f_{t-1}(ID_2)^{t-1} + ... + b^f_1 ID_2 + s = & f(ID_2) \\ \text{.............................} & \text{...........} \\ b^f_{t-1}(ID_{t-1})^{t-1} + ... + b^f_1 ID_{t-1} + s = & f(ID_{t-1}) \end{cases} \tag{4}$$

$$\begin{cases} b^{c2}_{t-1}(ID_1)^{t-1} + ... + b^{c2}_1 ID_1 + s^2 = & f(ID_1)^2 - h_2(ID_1) \\ b^{c2}_{t-1}(ID_2)^{t-1} + ... + b^{c2}_1 ID_2 + s^2 = & f(ID_2)^2 - h_2(ID_2) \\ \text{.............................} & \text{...........} \\ b^{c2}_{t-1}(ID_{t-1})^{t-1} + ... + b^{c2}_1 ID_{t-1} + s^2 = & f(ID_{t-1})^2 - h_2(ID_{t-1}) \end{cases} \tag{5}$$

$$\begin{cases} b^{c3}_{t-1}(ID_1)^{t-1} + ... + b^{c3}_1 ID_1 + s^3 = & f(ID_1)^3 - h_3(ID_1) \\ b^{c3}_{t-1}(ID_2)^{t-1} + ... + b^{c3}_1 ID_2 + s^3 = & f(ID_2)^3 - h_3(ID_2) \\ \text{.............................} & \text{...........} \\ b^{c3}_{t-1}(ID_{t-1})^{t-1} + ... + b^{c3}_1 ID_{t-1} + s^3 = & f(ID_{t-1})^3 - h_3(ID_{t-1}) \end{cases} \tag{6}$$

....

$$\begin{cases} b^{ck}_{t-1}(ID_1)^{t-1} + ... + b^{ck}_1 ID_1 + s^k = & f(ID_1)^k - h_k(ID_1) \\ b^{ck}_{t-1}(ID_2)^{t-1} + ... + b^{ck}_1 ID_2 + s^k = & f(ID_2)^k - h_k(ID_2) \\ \text{.............................} & \text{...........} \\ b^{ck}_{t-1}(ID_{t-1})^{t-1} + ... + b^{ck}_1 ID_{t-1} + s^k = & f(ID_{t-1})^k - h_k(ID_{t-1}) \end{cases} \tag{7}$$

For Eq.4, there are $t-1$ equations and t variables. The t variables are $b^f_{t-1}, b^f_{t-2}, ..., b^f_1, s$. Moreover, according to the theory of linear algebra, we know that s is a free variable. Namely, s can be any number. Consequently, s cannot be determined by Eq.4.

For Eq.5, there are $t-1$ equations and t variables. The t variables are $b^{c_2}_{t-1}, b^{c_2}_{t-2}, ..., b^{c_2}_{1}, s$. Moreover, according to the theory of linear algebra, we can also know that s is a free variable. Consequently, s cannot be determined by Eq.4 and Eq.5.

Similarly, for Eq.6 and Eq.7, s is a free variable still. Consequently, s cannot be determined by Eq.4, Eq.5, Eq.6..., Eq.7.

Finally, s is always un-determined, therefore, the $t-1$ malicious servers cannot recover the key secret value s although they jointly work together.

7. Conclusions

In this paper, a novel double non-interactive multi-party computation (BeeHive) is proposed. Specifically, it realized that shareholders can help dealer to calculate any-degree polynomial of secret numbers in a non-interactive way, and the dealer can verify the correctness of responses sent by shareholders in the same way. Moreover, a detailed performance evaluation is presented. Finally, we presented a security proof of BeeHive, which proved that shareholders cannot get any information if the number of malicious shareholders is less than the threshold number.

[1] Andrew Chi-Chih Yao: Protocols for Secure Computations (Extended Abstract). FOCS 1982: 160-164

[2] Genkin, D., Ishai, Y., Polychroniadou, A.: Efficient multi-party computation: from passive to active security via secure SIMD circuits. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 721-741. Springer, Heidelberg (2015)

[3] Elette Boyle, Niv Gilboa, Yuval Ishai: Group-Based Secure Computation: Optimizing Rounds, Communication, and Computation. EUROCRYPT (2) 2017: 163-193

[4] Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System. White Paper. https://bitcoin.org/bitcoin.pdf. (2008)

[5] Wood, G. (2014) Ethereum: a secure decentralised generalised transaction ledger. White Paper. Available online: http://gavwood.com/Paper.pdf.

[6] Stanley, A. (2017) EOS: Unpacking the Big Promises Behind a Possible Blockchain Contender. CoinDesk (June 25, 2017). Available online: https://www.coindesk.com/eosunpacking-the-big-promises-behind-a-possible-blockchain-contender/.

[7] Prabhanjan Ananth, Arka Rai Choudhuri, Abhishek Jain: A New Approach to Round-Optimal Secure Multiparty Computation. CRYPTO (1) 2017: 468-499

[8] Sanjam Garg, Susumu Kiyoshima, Omkant Pandey: On the Exact Round Complexity of Self-composable Two-Party Computation. EUROCRYPT (2) 2017: 194-224

[9] Adi Akavia, Rio LaVigne, Tal Moran: Topology-Hiding Computation on All Graphs. CRYPTO (1) 2017: 447-467

[10] Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, Lior Zichron: Secure Arithmetic Computation with Constant Computational Overhead. CRYPTO (1) 2017: 223-254

[11] Carmit Hazay, Muthuramakrishnan Venkitasubramaniam: On the Power of Secure Two-Party Computation. CRYPTO (2) 2016: 397-429

[12] Rafail Ostrovsky, Silas Richelson, Alessandra Scafuro: Round-Optimal Black-Box Two-Party Computation. CRYPTO (2) 2015: 339-358

[13] Payman Mohassel, Mike Rosulek: Non-interactive Secure 2PC in the Offline/Online and Batch Settings. EUROCRYPT (3) 2017: 425-455

[14] Juan A. Garay, Yuval Ishai, Rafail Ostrovsky, Vassilis Zikas: The Price of Low Communication in Secure Multi-party Computation. CRYPTO (1) 2017: 420-446

[15] Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, Anat Paskin-Cherniavsky: Secure Computation with Minimal Interaction, Revisited. CRYPTO (2) 2015: 359-378

[16] Jonathan Katz, Rafail Ostrovsky: Round-Optimal Secure Two-Party Computation. CRYPTO 2004: 335-354

[17] Ivan Damgård, Jesper Buus Nielsen, Antigoni Polychroniadou, Michael A. Raskin: On the Communication Required for Unconditionally Secure Multiplication. CRYPTO (2) 2016: 459-488

[18] Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, Baltimore, Maryland, USA, 13-17 May 1990, pp. 503-513 (1990)

[19] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova: Two-Round Secure MPC from Indistinguishability Obfuscation. TCC 2014: 74-94

[20] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, Daniel Wichs: Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. EUROCRYPT 2012: 483-501

[21] Zvika Brakerski, Shai Halevi, Antigoni Polychroniadou: Four Round Secure Computation Without Setup. TCC (1) 2017: 645-677

[22] Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, Antigoni Polychroniadou: The Exact Round Complexity of Secure Computation. EUROCRYPT (2) 2016: 448-476

[23] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and fasttrack multiparty computations with applications to threshold cryptography. In Proceedings of PODC 1997, pages 101-111, 1998.

[24] Ronald Cramer, Ivan Damgård, Robbert de Haan: Atomic Secure Multiparty Multiplication with Low Communication. EUROCRYPT 2007: 329-346

[25] Dorri, A., Kanhere, S. S., and Jurdak, R. (2017, April). Towards an optimized blockchain for IoT. In Proceedings of the Second International Conference on Internet-of-Things Design and Implementation (pp. 173-178). ACM.

[26] Lijing Zhou, Licheng Wang, Yiru Sun and Pin Lv: BeeKeeper: A Blockchain-based IoT System with Secure Storage and Homomorphic Computation. In: IEEE Access 2018. DOI:10.1109/ACCESS.2018.2847632.

[27] Pratyay Mukherjee, Daniel Wichs: Two Round Multiparty Computation via Multi-key FHE. EUROCRYPT (2) 2016: 735-763

[28] Adi Shamir: How to Share a Secret. Commun. ACM 22(11): 612-613 (1979)

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

[30] https://github.com/debatem1/pypbc.

[31] https://github.com/aleaxit/gmpy.

[32] Zheng, Zibin, et al. "Blockchain Challenges and Opportunities: A Survey." International Journal of Web & Grid Services (2017).