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

In this paper, we focus on the research of non-interactive secure multi-party computation (MPC). At first, we propose a fully homomorphic non-interactive verifiable secret sharing (FHNVSS) scheme. In this scheme, shareholders can generate any-degree polynomials of shared numbers without interaction, and the dealer can verify whether shareholders are honest without interaction. We implemented the FHNVSS scheme in Python with a detailed performance evaluation. 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. Besides, we make a extension on the FHNVSS scheme to obtain a double non-interactive secure multi-party computation, called BeeHive. In the BeeHive scheme, distrustful players can jointly calculate a any-degree negotiated function, the input of which are inputs of all players, without interaction, and each player can verify whether other players calculate honestly without interaction. To the best of our knowledge, it is the first work to realize that players can jointly calculate any-degree function, the input of which are inputs of all players, without interaction.

Keywords: MPC, verifiable secret sharing, non-interactive, homomorphism.

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

1. Introduction

Secure multi-party computation (MPC) [1] is a significant technology, where 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.

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 this paper, we mainly focus on non-interactive secure MPC, where players can jointly calculate a negotiated function, the input of which are inputs of all players, without interaction.

1.1. Our Results

Our contributions are summarized as follows:

- We present a fully homomorphic non-interactive verifiable secret sharing (FHNVSS) scheme. In the scheme, shareholders can generate any-degree polynomial of shared numbers without interaction, and the dealer can verify whether shareholders are honest without interaction. A security analysis of FHNVSS schem is presented.

- We present detailed performance evaluation of FHNVSS scheme by deploying it on a Ubuntu 16.04 environment laptop. Specifically, the proposed FHNVSS 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 FHNVSS is satisfactory. For instance, when the request is a 10-degree polynomial of shared numbers, generating a response takes about 0.0017263 s; verifying a response takes about 0.1221394 s; recovering a result takes about 0.0003862 s.

- We propose a *Double Non-interactive Secure Multi-party Computation*, called BeeHive. In this BeeHive scheme, distrustful players can jointly calculate an any-degree negotiated function, the input of which are inputs of all players, without interaction, they can verify correctness of responses sent by other players without interaction. A security analysis of BeeHive is given.

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]. Recently, in Ref.[26], we proposed a secure multi-party computation scheme, where shareholders can generate shares of two-degree polynomials of secret numbers without interaction. Temporarily, the secure MPC scheme proposed in [26] is called Pre-Scheme, and it 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.

Compared with the Pre-Scheme, improvements of BeeHive are as follows:

- Theoretically, distrustful players can jointly calculate any-degree negotiated function, the input of which are inputs of all players, without interaction.
- Each player can verify other players compute honestly. In this verification process, BeeHive does not use pairing to verify responses of players, 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 FHNVSS and BeeHive is shown in Sec.2. Sec.3 briefly presents preliminaries. We introduce the FHNVSS scheme without verifiability and the verifiability of FHNVSS in Sec.4.1 and Sec.4.2, respectively. A detailed performance evaluation is shown in Sec.5. A security analysis is presented in Sec.6. Construction and security analysis of BeeHive are studied in Sec.7. Finally, a short conclusion is presented in Sect.8.

2. An Overview of fully homomorphic non-interactive verifiable secret sharing and BeeHive

In a fully homomorphic non-interactive verifiable secret sharing (FHNVSS) scheme, components include a dealer and a certain number of shareholders (servers). A (t, n) FHNVSS scheme works as follows:

- Step 1: The dealer generates n 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 core-shares 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.

The FHNVSS scheme 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 are always confidential as long as less than t servers are malicious.
- **Verifiability.** Verification key, core-shares and responses are verifiable.
  - *Verification key.* When the verification key (VK) is opened, anyone can verify its validity.
  - *Core-shares.* When a server receives a core-share, he can verify whether this core-share is correctly computed by the dealer. 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.

By making an extension on the (n, n) FHNVSS scheme, we obtain a (n, n) BeeHive scheme, where n players can jointly calculate a negotiated function, the input of which are numbers shared by all players. Each player independently works as a dealer of (n, n) FHNVSS scheme to share his input among the n players, and he also works as a server of (n, n) FHNVSS scheme to jointly compute the negotiated function. The work process of a (n, n) BeeHive scheme is as follows:

- Step 1: Each player executes a (n, n) FHNVSS scheme independently. He generates n core-shares and a verification key (VK). In these n core-shares, one belongs to this player, and other $n-1$ will be sent to other $n-1$ players respectively in the next step. Each player opens his VK. Anyone (including other players) can verify whether the VK is correctly computed by its generator. If a VK is invalid, its generator has to regenerate his VK. Once all VKs are valid, all players join in the next step.
- Step 2: Each player secretly sends his $n-1$ core-shares (except his own core-share) to other $n-1$ servers respectively. After receiving a core-share,

each player can verify whether this core-share is correctly computed by the sender via sender's VK. If a core-share is invalid, the receiver can ignore it and request corresponding sender to re-send it. Once all core-shares are valid, all players join in the next step.

- Step 3: Each player encrypts his input into encrypted numbers, then he broadcasts these encrypted numbers.
- Step 4: Players negotiate a function, which will be jointly calculated by players. Inputs of the negotiated function are inputs of all players.
- Step 5: According to the negotiated function, each player can generate a response with his core-shares and encrypted numbers shared by players, then he broadcasts his response.
- Step 6: After receiving a response, each player can verify whether this response is correctly computed via this sender's VK. This verification process does not need interaction. If a player receives an invalid response, then he can ignore it or request the corresponding player to re-send it.
- Once a player collects n valid responses, he will recover the correct result of negotiated function.

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:

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

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 responses.

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.*

3.3. Adversary Model

In this subsection, we present the adversary models of FHNVSS and BeeHive.

For a (t, n) FHNVSS scheme, which includes a dealer and n shareholders, we have the following assumptions:

- The dealer could generate the verification key (VK) and core-shares dishonestly, but he does not reveal any secret data to servers.
- A server could generate a response dishonestly, but the number of dishonest players is less than t.

For a (n, n) BeeHive scheme, which includes n players, we have the following assumptions:

- A player could generate the verification key (VK) and core-shares dishonestly, but he does not reveal any secret data to other players.
- A player could generate responses dishonestly, but he does not reveal his input to other players.

4. Construction of Fully Homomorphic Non-interactive Verifiable Secret Sharing Scheme

In this section, we will present a fully homomorphic non-interactive verifiable secret sharing (FHNVSS) scheme. To clearly present the work process of FHNVSS, we will present the FHNVSS scheme without verifiability at first. Then we will give out the verifiability of FHNVSS. Finally, basic applications of FHNVSS combined with blockchain will be illustrated.

4.1. FHNVSS without verifiability

In this subsection, we will present the FHNVSS 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 FHNVSS, where the dealer and servers could be dishonest, and a (t, n) FHNVSS 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
$$\begin{aligned} H(x) = &\quad S(x) - \textstyle\sum_{j=2}^{k} b_j h_j(x) \\ = &\quad \textstyle\sum_{i=1}^{k} b_i f(x)^i + b_0 - \sum_{j=2}^{k} b_j h_j(x) \\ = &\quad \textstyle\sum_{j=2}^{k} b_j (c_j(x) + s^j) + b_1 f(x) + b_0. \end{aligned} \tag{1}$$

- $H(x)$ 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 FHNVSS 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. FHNVSS with Sharing Encrypted Numbers

In this subsection, we will add a feature of sharing encrypted numbers on FHNVSS. 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 $v_t \leq k$ 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 FHNVSS. 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 FHNVSS in Sec. 6.*

4.2. Verifiability of FHNVSS

In Sec. 4.1, we described the FHNVSS 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 FHNVSS without verifiability mentioned in Sec.4.1, the full FHNVSS scheme 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 $(2, 3)$ FHNVSS as an example (The (t, n) FHNVSS can be similarly constructed since it is similar to $(2, 3)$ FHNVSS.). The FHNVSS 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 the FHNVSS scheme, 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 FHNVSS, 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.

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

5. Performance Evaluation of FHNVSS

In this section, we will present a performance evaluation of FHNVSS by deploying it on a Ubuntu 16.04 environment laptop. Specifically, the FHNVSS 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, FHNVSS 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) FHNVSS (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) FHNVSS 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.

6. Security Analysis of FHNVSS

In this section, we will take the (t, n) FHNVSS as an example to discuss the confidentiality of the proposed FHNVSS, and the highest degree of polynomial that the dealer can query is k. Because the FHNVSS is a threshold cryptography scheme, its confidentiality means that $t - 1$ malicious servers cannot jointly

Table 2: Performance of algorithms of FHNVSS 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) FHNVSS 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.

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 polynomials 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 $c_j(0) =$

Table 3: Performance of algorithms of FHNVSS 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) FHNVSS 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.

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 do not know the coefficients of the following equations.

$$\begin{array}{rcl} f(x) & = & b^f_{t-1}x^{t-1} + b^f_{t-2}x^{t-2} + ... + b^f_1 x + s, \\ c_2(x) & = & b^{c_2}_{t-1}x^{t-1} + b^{c_2}_{t-2}x^{t-2} + ... + b^{c_2}_1 x, \\ c_3(x) & = & b^{c_3}_{t-1}x^{t-1} + b^{c_3}_{t-2}x^{t-2} + ... + b^{c_3}_1 x, \\ ...... & .. & .... \\ c_k(x) & = & b^{c_k}_{t-1}x^{t-1} + b^{c_k}_{t-2}x^{t-2} + ... + b^{c_k}_1 x. \end{array}$$

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.

In order to solve coefficients of $f(x), c_2(x), c_3(x), ..., c_k(x)$, the $t-1$ malicious

Table 4: Core-shares of servers

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

servers can construct linear equations by using their core-shares as follows:

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

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

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

....

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

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. BeeHive

In this section, we will make an extension on (n, n) $(n > 2)$ FHNVSS to obtain a double non-interactive multi-party computation scheme, called BeeHive. In a BeeHive scheme, players jointly compute a negotiated function, the input of which are inputs of all players, but each player does not reveal his own input. Because the MPC scheme BeeHive is based on the FHNVSS, properties of non-interactive and verifiable are similar to FHNVSS. That is to say, players can jointly compute with inputs of all players without interaction; players can verify VK, core-shares and responses without interaction. The verification process of BeeHive can be simply obtained from FHNVSS. The construction of BeeHive will be presented first, then we will discuss the security of BeeHive.

7.1. Construction of BeeHive

A BeeHive scheme includes n $(n > 2)$ players. Each player works as a dealer of (n, n) FHNVSS to confidentially share his data with other players; he also works as a server of FHNVSS to jointly compute a specified function, the input of which are data shared by all players. In the following content, we will take a (n, n) BeeHive with n players as an example to present the work process of the scheme. Let these n players be $P_1, P_2, ..., P_n$.

- Step 1: i from 1 to n, P_i executes a (n, n) FUNVSS scheme among the n players. He generates a verification key (VK_i) and three core-shares $(CS_{i,1}, CS_{i,2}, ..., CS_{i,n})$. He opens the VK_i and securely sends $CS_{i,j}$ $(j = 1, 2, ..., n,\ j \neq i)$ to P_j, and he securely keeps the $CS_{i,i}$. The VK_i can be verified by other players. If VK_i is invalid, P_i has to re-generate this VK_i. $CS_{i,j}$ can be verified by P_j. If $CS_{i,j}$ is invalid, P_j can request P_i

to re-send a $CS_{i,j}$ until a valid core-share is received. Once each player opens a valid verification key and sends valid core-shares to other players, they join in the next step.

- Step 2: i from 1 to n, P_i use his secret numbers ($n_{i,1}$, $n_{i,2}$, ..., n_{i,m_i}) to generates his encrypted numbers ($encn_{i,1}$, $encn_{i,2}$, ..., $encn_{i,m_i}$) and sends them to other players. These numbers are as P_i's input of the function negotiated by players in the next step.

- Step 3: These n players negotiate a function, which will be jointly calculated by them. The input this function are encrypted numbers shared by them. The function is a sum of n polynomials as follow:

$$\sum_{t=1}^{n} f_t(n_{i,1}, n_{i,2}, ..., n_{i,m_i}),$$

  where f_t is a polynomial.

- Step 4: According to the negotiated function, i from 1 to n, P_i generates a response $resp_i$ with his core-shares $(CS_{1,i}, CS_{2,i}, ..., CS_{n,i})$ and encrypted numbers shared by all players. Then he broadcasts his response to other players.

- Step 5: After receiving a response, a player verifies it with $VK_1, VK_2, ..., VK_n$. If the response is invalid, the player can request the corresponding player re-send a response until a valid response is received.

- Step 6: If a player collects n valid responses, he can use Lagrangian interpolation to recover the correct result of negotiated function.

7.2. Security analysis of BeeHive

In this subsection, we will discuss security of (n, n) BeeHive.

- i from 1 to n, P_i's inputs are always confidential as long as he does not reveal $CS_{i,i}$. P_i's inputs are shared among $P_1, P_2, ..., P_n$ via a (n, n) FH-NVSS. A player can recover P_i's inputs iff he can get all core-shares of P_i

($CS_{i,1}$, $CS_{i,2}$, ..., $CS_{i,n}$). However, this player cannot get ($CS_{i,1}$, $CS_{i,2}$, ..., $CS_{i,n}$) as long as P_i does not reveal his $CS_{i,i}$. Consequently, P_i's data are always confidential as long as he does not reveal his own $CS_{i,i}$.

- A player cannot obtain other players' input from the result of specified function. The number of players is at least three and the input of function negotiated by players includes inputs of all players. Therefore, a result of negotiated function is a combination of inputs of more than three players. Each player does not reveal his input to others. Because a player only has his own input, the result of negotiated function includes at least two undetermined inputs for this player. Consequently, a player cannot obtain other players' inputs from the result of negotiated function.

8. Conclusions

In this paper, we propose a fully homomorphic non-interactive verifiable secret sharing (FHNVSS) scheme. In this scheme, shareholders can generate any-degree polynomials of shared numbers without interaction, and the dealer can verify whether shareholders are honest without interaction. We implemented the FHNVSS scheme in Python with a detailed performance evaluation. Besides, we make a extension on the FHNVSS scheme to obtain a double non-interactive secure multi-party computation, called BeeHive, where distrustful players can jointly calculate a any-degree negotiated function, the input of which are inputs of all players, without interaction, and each player can verify whether other players calculate honestly without interaction.

[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).