Constructive t-secure Homomorphic Secret Sharing for Low Degree Polynomials

Kittiphop Phalakarn[1], Vorapong Suppakitpaisarn[1], Nuttapong Attrapadung[2], and Kanta Matsuura[1]

[1] The University of Tokyo, Tokyo, Japan
{kittipop,kanta}@iis.u-tokyo.ac.jp, vorapong@is.s.u-tokyo.ac.jp
[2] National Institute of Advanced Industrial Science and Technology, Tokyo, Japan
n.attrapadung@aist.go.jp

Abstract. This paper proposes t-secure homomorphic secret sharing schemes for low degree polynomials. Homomorphic secret sharing is a cryptographic technique to outsource the computation to a set of servers while restricting some subsets of servers from learning the secret inputs. Prior to our work, at Asiacrypt 2018, Lai, Malavolta, and Schröder proposed a 1-secure scheme for computing polynomial functions. They also alluded to t-secure schemes without giving explicit constructions; constructing such schemes would require solving set cover problems, which are generally NP-hard. Moreover, the resulting implicit schemes would require a large number of servers. In this paper, we provide a constructive solution for threshold-t structures by combining homomorphic encryption with the classic secret sharing scheme for general access structure by Ito, Saito, and Nishizeki. Our scheme also quantitatively improves the number of required servers from $O(t^2)$ to $O(t)$, compared to the implicit scheme of Lai et al. We also suggest several ideas for future research directions.

Keywords: Homomorphic secret sharing · Homomorphic encryption · Threshold non-access structure.

1 Introduction

Homomorphic secret sharing is one of the interesting techniques in cryptography which introduces a way to outsource the computation to a set of servers while restricting some subsets of servers from learning about the secret inputs. There are several homomorphic secret sharing schemes that support polynomial computation, including the scheme of Shamir [36], Catalano and Fiore [15], and Lai, Malavolta, and Schröder [31].

The schemes of Catalano and Fiore [15], and Lai et al. [31] both used degree-k homomorphic encryption, which supports computation of degree-k polynomial of ciphertexts, as a building block. The scheme of Catalano and Fiore [15] outsources the secret inputs to two servers, and the maximum computable degree of the scheme is $2k$. Obviously, the security of the scheme is 1-secure, meaning that the two servers cannot collude. The work of Lai et al. [31] improved the results of Catalano and Fiore. The secret inputs are outsourced to m servers, and the maximum computable degree of the scheme is $(k+1)m-1$.

The scheme of Lai et al. [31] has many applications. It can support outsourced computation of low-degree polynomials on private data, which appears in private information retrieval system, non-linear function approximation in neural networks [16, 24], and many other applications. It is also shown in [1] that any P/poly functions can be evaluated using polynomials of degree 3 with the assumption that there exists a pseudo-random generator in NC1.

However, the main scheme of Lai et al. [31] is still 1-secure. Any subsets of m servers, including any pairs of servers, cannot collude, which is not a realistic situation. The authors also discussed in the paper that the scheme can be extended to t-secure, where a collusion of t servers does not violate the security of the scheme. Their proposed construction requires solving set cover problems, which are NP-complete. Since there is no efficient and accurate algorithms for set cover problems, their scheme gives an implicit construction which requires a large number of servers.

If we consider the secret sharing scheme of Shamir [36] instead, which is information theoretic, the maximum computable degree of the scheme is equal to the upper bound proved in [4]. Trying to further increase the maximum computable degree of the scheme by using computational setting is not straightforward, since the threshold of the scheme changes after performing computations.

All in all, finding a constructive t-secure homomorphic secret sharing that can compute polynomials of degree higher than the limit set by [4] can be considered as an open problem.

1.1 Our Contributions

In this paper, we propose t-secure homomorphic secret sharing scheme for low degree polynomials. Our scheme improves the scheme of Lai et al. [31] so that it can constructively support threshold security. The proposed scheme also inherits the applications of [31], but with t-security for all possible $t \geq 1$. Here, we claim four contributions: (1) increasing the maximum degree of homomorphic secret sharing from Shamir [36], (2) adding threshold structures to Lai et al., (3) decreasing the number of required servers in t-secure scheme of Lai et al., and (4) extending the capability of homomorphic encryption.

Before going into details, we briefly explain the overview of the proposed scheme as shown in Fig. 1. There are three roles, namely n input clients (each has secret input x_i), m servers, and an output client. The goal is to let the output client learn the value $f(x_1, \ldots, x_n)$ for some function f, while the unauthorized sets of servers are restricted from knowing this value. In addition, both output client and servers should not know the secret inputs. We scope the function f in this paper to be a polynomial of the secret inputs.

In our proposed scheme, any homomorphic encryption scheme can be used as the underlying scheme. Each input client divides its secret input into b shares, $x_i = x_{i,1} + \ldots + x_{i,b}$. These b shares are given to each computing server with p shares as plaintexts (represented as yellow circles without border) and $b-p$

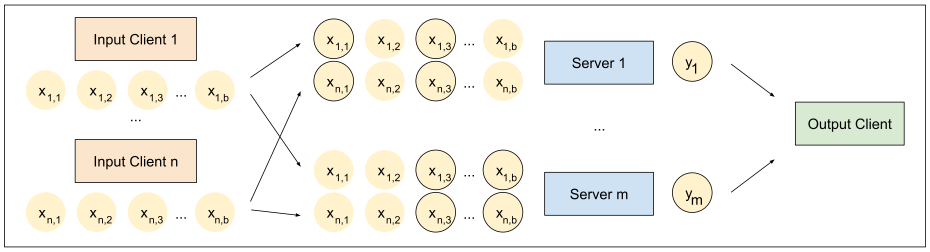
Input Client 1
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ ... $x_{1,b}$
Input Client n
$x_{n,1}$ $x_{n,2}$ $x_{n,3}$ ... $x_{n,b}$
Server 1
Server m
y_1
y_m
Output Client

Fig. 1. The overview of our homomorphic secret sharing scheme.

shares as ciphertexts of the homomorphic encryption scheme (represented as yellow circles with border). Each server then locally performs some computations on the given shares, and send the result to the output client. The output client combines all the shares into the final result.

We consider these parameters in this paper, including

- λ: The security parameter of the underlying homomorphic encryption scheme
- k: The maximum computable degree of the underlying homomorphic encryption scheme
- m: Number of servers
- t: The maximum number of colluding servers allowed
- b: Number of shares per input for each server
- p: Number of plaintext shares per input for each server
- d: The maximum computable degree of the homomorphic secret sharing scheme

The results of previous works and our work are shown in Table 1. Note that for t-secure scheme of Lai et al. [31], the values of k, t, p, and d are given as a scheme specification, while in the other schemes, the values of k, m, and t are given. Next, we claim four contributions of our work.

Increase the Maximum Degree of Homomorphic Secret Sharing from Shamir [36]. It is well known that Shamir's secret sharing scheme supports threshold security. However, the threshold of the scheme changes after local multiplication. If we consider Shamir's secret sharing as a homomorphic secret sharing scheme, the maximum polynomial degree of the homomorphic secret sharing is $d = \lfloor \frac{m-1}{t} \rfloor$, which is equal to the upper bound proved in [4] for information theoretic setting. When a polynomial of degree higher than d is computed, the computation result will not be able to be reconstructed. In our proposed scheme, combining secret sharing of general access structure from Ito, Saito, and Nishizeki [27] and homomorphic encryption allows us to construct t-secure homomorphic secret sharing scheme, and increase the maximum degree d to $\lfloor \frac{(k+1)m-1}{t} \rfloor$ which is approximately $k+1$ times increasing comparing to [36] with the same values of m and t.

Add Threshold Structures to Lai et al. [31]. The main scheme proposed in [31] is 1-secure, which is secure only if no servers are colluding. This security

Table 1. Comparing homomorphic secret sharing schemes for polynomials.

Scheme	k	m	t	b	p	d
Shamir [36]	-	m	t	1	1	$\lfloor \frac{m-1}{t} \rfloor$
Catalano et al. [15]	k	2	1	$1, 2$	1	$2k$
1-secure Lai et al. [31]	k	m	1	m	$m-1$	$(k+1)m-1$
t-secure Lai et al. [31][3]	1	t^2	t	$2t+1$	2	3
This work	k	m	t	$\binom{m}{t}$	$\binom{m-1}{t}$	$\lfloor \frac{(k+1)m-1}{t} \rfloor$

model may not suffice for real-life applications. The authors of [31] also proposed a t-secure scheme, but it is not so constructive: one needs to solve the set cover problem in order to know which plaintext shares are distributed to which server. In our work, we use the idea of secret sharing from Ito et al. [27] to realize threshold non-access structures. Our scheme is constructive, and allows us to use any values of threshold t. Although our scheme has more number of shares, possible improvements are also mentioned in this paper.

Decrease the Number of Required Servers in t-secure Scheme of Lai et al. [31]. For t-secure scheme of [31], they set the parameters as $k = 1$, $d = 3$, $p = 2$, and $b = 2t + 1$ as shown in Table 1, and they claimed that the number of required servers is $m = t^2$, which is not efficient. With the same setting, our scheme requires $\lceil \frac{3t+1}{2} \rceil$ servers, which is linear in threshold t. Our scheme also requires $O(t)$ servers for any values of k, d, and t.

Extend the Capability of Homomorphic Encryption. The purpose of this work is not to outperform homomorphic encryption, but to combine it with secret sharing. It is known that computing complex polynomials using homomorphic encryption is quite inefficient. Our proposed combination enables higher computable degrees, from k to $\lfloor \frac{(k+1)m-1}{t} \rfloor$. Moreover, it can be seen that our setting adds the capability in mitigating a single point of failure to the homomorphic encryption scheme. Although we are not aiming to improve homomorphic encryption, our scheme could be better than them for higher degree computation.

1.2 Overview of Our Techniques

We use the idea of secret sharing from Ito, Saito, and Nishizeki [27] to add threshold structures to the work of Lai et al. [31]. Here, we explain the concept of our scheme by using a simple example.

Example 1. Suppose we have two input clients. The first input client has a secret input x_1, and the second input client has a secret input x_2. The goal of the scheme is to let the output client learn the multiplication of the secret inputs x_1x_2, but not the secret inputs x_1 or x_2. The input clients decide to outsource the computation to three servers, and any collusion of two servers are not allowed to learn the secret inputs. Suppose that the ID of the servers are 1, 2, and 3, we can

[3] The construction is not explicitly provided.

write all the largest forbidden subsets of servers or "the maximum non-access structure" as $\Gamma^* = \{\{1,2\},\{1,3\},\{2,3\}\}$.

To secretly share the secret input x_1 to the three servers, we randomly generate a share for each member of Γ^* according to Ito et al. [27] with the condition that $x_{1,\{1,2\}} + x_{1,\{1,3\}} + x_{1,\{2,3\}} = x_1$. Suppose further that we use degree-1 homomorphic encryption scheme, which supports additions between ciphertexts, additions between ciphertexts and plaintexts, and multiplications between a ciphertext and plaintexts. For each share $x_{1,\gamma}$, the j-th server will get the share as a ciphertext if $j \in \gamma$, and as a plaintext otherwise. This process is also applied to the secret input x_2. Packages of shares S_1, S_2 and S_3 of the secret inputs x_1 and x_2 for the three servers are

$$S_1 = \begin{bmatrix} \boxed{x_{1,\{1,2\}}} & \boxed{x_{1,\{1,3\}}} & x_{1,\{2,3\}} \\ \boxed{x_{2,\{1,2\}}} & \boxed{x_{2,\{1,3\}}} & x_{2,\{2,3\}} \end{bmatrix} \qquad S_2 = \begin{bmatrix} \boxed{x_{1,\{1,2\}}} & x_{1,\{1,3\}} & \boxed{x_{1,\{2,3\}}} \\ \boxed{x_{2,\{1,2\}}} & x_{2,\{1,3\}} & \boxed{x_{2,\{2,3\}}} \end{bmatrix}$$

$$S_3 = \begin{bmatrix} x_{1,\{1,2\}} & \boxed{x_{1,\{1,3\}}} & \boxed{x_{1,\{2,3\}}} \\ x_{2,\{1,2\}} & \boxed{x_{2,\{1,3\}}} & \boxed{x_{2,\{2,3\}}} \end{bmatrix}$$

where $\boxed{x_{i,\gamma}}$ denotes the ciphertext of $x_{i,\gamma}$. Note that the plaintexts from any pairs of servers are not enough to reconstruct the secret inputs. In contrast, in the 1-secure scheme of Lai et al. [31], the number of shares per input is equal to the number of servers. The j-th server will get only one share as a ciphertext, the j-th share, and get the other shares as plaintexts. This makes the scheme of Lai et al. only 1-secure.

To compute the multiplication $x_1 x_2$, we have $x_1 x_2 = \sum_{\gamma_1,\gamma_2 \in \Gamma^*} x_{1,\gamma_1} x_{2,\gamma_2}$. Using the properties of homomorphic encryption scheme, servers 1, 2, and 3 can locally compute the ciphertexts

$$\begin{aligned} \boxed{y_1} &= x_{1,\{2,3\}} \boxed{x_{2,\{1,2\}}} + x_{1,\{2,3\}} \boxed{x_{2,\{1,3\}}} + x_{1,\{2,3\}} x_{2,\{2,3\}} \\ \boxed{y_2} &= x_{1,\{1,3\}} \boxed{x_{2,\{1,2\}}} + x_{1,\{1,3\}} x_{2,\{1,3\}} + x_{1,\{1,3\}} \boxed{x_{2,\{2,3\}}} \\ \boxed{y_3} &= x_{1,\{1,2\}} x_{2,\{1,2\}} + x_{1,\{1,2\}} \boxed{x_{2,\{1,3\}}} + x_{1,\{1,2\}} \boxed{x_{2,\{2,3\}}} \end{aligned}$$

respectively. Note that for each term, there is at most one ciphertext, so computing with degree-1 homomorphic encryption is possible. Each server then forwards the result ciphertext $\boxed{y_j}$ to the output client who sums all the ciphertexts and then decrypts to see the final result $x_1 x_2$.

From the formula in Table 1 which we will prove in this paper, the maximum polynomial degree that our homomorphic secret sharing scheme can compute is $\lfloor \frac{(1+1)3-1}{2} \rfloor = 2$. Other degree-2 polynomials can be computed in the same way. This concept can also be generalized to any numbers of servers and any values of threshold.

1.3 Related Works

One of the first secret sharing techniques is additive secret sharing [25], where the j-th server gets a share $s_{i,j}$ with the condition that the sum of all shares is equal to the secret input $\sum_{j\in[m]} s_{i,j} = x_i$. To perform the multiplication on this scheme, a lot of communication between the servers are required. Optimization techniques such as Beaver's multiplication triple [5] are proposed in order to reduce the communication costs.

Homomorphic secret sharing was first defined by Boyle, Gilboa, and Ishai [11]. The scheme is designed in the way that the communication between the servers are not required. Homomorphic secret sharing is shown in [12] to imply a useful related primitive called server-aided secure multi-party computation [29, 30]. In the scheme of Boyle et al. [11], a branching program is evaluated on secret inputs using two computing servers, based on the decisional Diffie-Hellman (DDH) assumption. However, the result from the scheme will be correct only with probability $\frac{1}{\mathsf{poly}(\lambda)}$ where λ is the security parameter.

Homomorphic encryption can also be used as a private outsourced computation scheme. Secret inputs are encrypted and are sent to only one server to perform the computations, so the communication is minimized. The spooky encryption of [17] is similar to multi-key homomorphic encryption and can also be used for secure computation. However, using homomorphic encryption to compute a polynomial with degree greater than 1 is not so efficient. Catalano and Fiore [19] proposed a generic method to increase the degree of the homomorphic encryption from k to $2k$. They also proposed two-server homomorphic secret sharing scheme from homomorphic encryption. The work of Lai et al. [31] further generalized this by sending plaintexts and ciphertexts to m servers. As mentioned in [31], this classic technique to split the evaluation of polynomials came from [3] and [7].

There are also other kinds of private outsourced computation. Function secret sharing [10] and non-interactive secure multiparty computation [6] are different from our scheme. The function secret sharing focused on the sharing of secret functions which will be computed on public inputs, while our work focuses on the sharing of secret inputs which will be computed on public functions. For non-interactive secure multiparty computation, the work focused on the secret sharing of public functions which will be computed on secret inputs. Note that the secret inputs of this scheme are not shared.

2 Preliminaries

In this section, we first review the definition of non-access structure. Since our homomorphic secret sharing scheme is based on homomorphic encryption, we then introduce the definitions of homomorphic secret sharing together with homomorphic encryption. We refer to some notations in [31].

Notations. We denote a set of positive integers $\{1, \ldots, n\}$ as $[n]$. We denote the uniform sampling of an element x from a set S as $x \leftarrow S$. We denote any function that is a polynomial in λ as $\mathsf{poly}(\lambda)$, and denote any function that is negligible in λ as $\mathsf{negl}(\lambda)$.

2.1 Non-access Structure

Suppose there is a set of m servers defined as $[m]$. We define a non-access (adversarial) set, a non-access (adversarial) structure, and the maximum non-access (adversarial) structure according to [27] as follows.

Definition 1. *A non-access set $\gamma \subset [m]$ is a subset of servers that is restricted from obtaining the secret. A non-access structure $\Gamma \subset 2^{[m]}$ is a set that contains all selected non-access sets. The non-access structure should follow the monotone property that if $\gamma \in \Gamma$ and $\gamma' \subset \gamma$, then $\gamma' \in \Gamma$. The maximum non-access structure Γ^* of Γ is defined as $\Gamma^* = \{\gamma \in \Gamma : \nexists \gamma' \in \Gamma, \gamma \subset \gamma'\}$.*

In the case that $\Gamma = \{\gamma \subset [m] : |\gamma| \leq t\}$, the non-access structure Γ is called as (m, t)-threshold non-access structure.

The (maximum) non-access structure can be represented as a hyper-graph $H = (V, E)$ where the set of vertices is $V = [m]$ and the set of hyper-edges is $E = \Gamma$ (or $E = \Gamma^*$). Each vertex in V represents a server in $[m]$, and each hyper-edge in E represents a non-access set γ in Γ (or Γ^*).

2.2 Homomorphic Secret Sharing (HSS)

Our setting is already described in Fig. 1. The scheme starts when the output client generates a pair of a public key and a secret key, and distributes the public key to all parties. Each input client then generates shares from its secret input using the public key, and forwards the shares to each corresponding server. Each server gathers the shares from input clients, and performs the computation on the shares using the public key. The result from each server is then forwarded to the output client. Finally, the output client combines all the results using the secret key. We refer to the definitions in [31] as follows. More details can be found in [9, 12].

Definition 2. *Suppose there are n inputs and m servers, a homomorphic secret sharing scheme consists of four algorithms $\mathsf{HSS} = (\mathsf{KGen}, \mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$.*

Key Generation. $(pk, sk) \leftarrow \mathsf{HSS.KGen}(1^\lambda)$: *The algorithm receives the security parameter λ as an input, and then generates a pair of a public key and a secret key (pk, sk).*

Secret Sharing. $(s_{i,1}, \ldots, s_{i,m}) \leftarrow \mathsf{HSS.Share}(pk, i, x_i)$: *The algorithm receives the public key pk, the index i of the input, and a secret input x_i in a specified input space, and then generates shares $s_{i,j}$ for the j-th server.*

Evaluation. $y_j \leftarrow \mathsf{HSS.Eval}(pk, j, f, (s_{1,j}, \ldots, s_{n,j}))$: *The algorithm receives the public key pk, the index j of the server, a function f, and the shares of the j-th server $s_{1,j}, \ldots, s_{n,j}$ as inputs, and then returns the output of the evaluation y_j.*

Decoding. $y \leftarrow \mathsf{HSS.Dec}(sk, (y_1, \ldots, y_m))$ *: The algorithm receives the secret key* sk*, and the evaluation results from all servers* $(y_1, \ldots, y_m)$ *as inputs, and then combines the results into* y*.*

Correctness. The homomorphic secret sharing for degree-d polynomials with n inputs and m servers is correct if, for any $\lambda \in \mathbb{N}$, any $n, m \in \mathsf{poly}(\lambda)$, any key pair $(pk, sk) \leftarrow \mathsf{HSS.KGen}(1^\lambda)$, any degree-$d$ polynomial f, any secret inputs $(x_1, \ldots, x_n)$, any shares $(s_{i,1}, \ldots, s_{i,m}) \leftarrow \mathsf{HSS.Share}(pk, i, x_i)$ for all $i \in [n]$, and any $y_j \leftarrow \mathsf{HSS.Eval}(pk, j, f, (s_{1,j}, \ldots, s_{n,j}))$ for all $j \in [m]$, it holds that

$$\Pr[\mathsf{HSS.Dec}(sk, (y_1, \ldots, y_m)) = f(x_1, \ldots, x_n)] \geq 1 - \mathsf{negl}(\lambda).$$

Security. The homomorphic secret sharing for degree-d polynomials with n inputs and m servers is Γ-secure if, for any $\lambda \in \mathbb{N}$, any $n, m \in \mathsf{poly}(\lambda)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for any probabilistic polynomial time adversary $A = (A_0, A_1)$,

$$|\Pr[\mathsf{Security}^0_{A,\mathsf{HSS}} = 1] - \Pr[\mathsf{Security}^1_{A,\mathsf{HSS}} = 1]| < \mathsf{negl}(\lambda)$$

where $\mathsf{Security}^b_{A,\mathsf{HSS}}$ is defined in Fig. 2 (left) for $b \in \{0, 1\}$. If Γ is (m, t)-threshold non-access structure, we call Γ-security as t-security.

To explain in more details, adversaries A_0 together with A_1 are trying to guess the bit b. After receiving the public key pk, adversary A_0 generates two secret inputs x_0 and x_1, chooses a non-access set γ from the non-access structure Γ, and creates state to pass some messages to A_1. Adversary A_1 receives state and all shares of the parties in the chosen non-access set γ, and then guesses which input x_0 or x_1 that the shares are generated from.

Context Hiding. The output client should learn nothing from the shares except the result of the evaluation. This means the distribution of shares $(y_1, \ldots, y_m)$ from the actual evaluation function and the distribution of shares $(s_{i,1}, \ldots, s_{i,m})$ $\leftarrow \mathsf{HSS.Share}(pk, i, f(x_1, \ldots, x_n))$ from secret sharing of the result $f(x_1, \ldots, x_n)$ should be indistinguishable.

The homomorphic secret sharing for degree-d polynomials with n inputs and m servers is context-hiding if, for any $\lambda \in \mathbb{N}$, any $n, m \in \mathsf{poly}(\lambda)$, there exists a probabilistic polynomial time algorithm S_{HSS} and a negligible function $\mathsf{negl}(\lambda)$ such that for any probabilistic polynomial time adversary $A = (A_0, A_1)$,

$$|\Pr[\mathsf{ContextHiding}^0_{A,S_{\mathsf{HSS}},\mathsf{HSS}} = 1] - \Pr[\mathsf{ContextHiding}^1_{A,S_{\mathsf{HSS}},\mathsf{HSS}} = 1]| < \mathsf{negl}(\lambda)$$

where $\mathsf{ContextHiding}^b_{A,S_{\mathsf{HSS}},\mathsf{HSS}}$ is defined in Fig. 2 (right) for $b \in \{0, 1\}$.

The explanation is similar to the definition of the security. Adversaries A_0 together with A_1 are trying to guess the bit b. After receiving the public key pk, adversary A_0 generates a function f, secret inputs $x_1, \ldots, x_n$, and creates state to pass some messages to A_1. Adversary A_1 receives state and the computation results $y_1, \ldots, y_m$. It has to guess whether the results are generated from the actual homomorphic secret sharing, or are simulated by the algorithm S_{HSS}.

```
$\mathsf{Security}^b_{A,\mathsf{HSS}}(1^\lambda)$ :
$(pk, sk) \leftarrow \mathsf{HSS.KGen}(1^\lambda)$
$(x_0, x_1, \gamma \in \Gamma, \mathsf{state}) \leftarrow A_0(pk)$
$(s_1, \ldots, s_m) \leftarrow \mathsf{HSS.Share}(pk, b, x_b)$
$b' \leftarrow A_1(\mathsf{state}, \{s_j : j \in \gamma\})$
return $b = b'$
```

```
$\mathsf{ContextHiding}^b_{A,S_{\mathsf{HSS}},\mathsf{HSS}}(1^\lambda)$ :
$(pk, sk) \leftarrow \mathsf{HSS.KGen}(1^\lambda)$
$(f, x_1, \ldots, x_n, \mathsf{state}) \leftarrow A_0(pk)$
if $b = 0$ then
    $(s_{i,1}, \ldots, s_{i,m}) \leftarrow \mathsf{HSS.Share}(pk, i, x_i)\ \forall i \in [n]$
    $y_j \leftarrow \mathsf{HSS.Eval}(pk, j, f, (s_{i,j})_{i \in [n]})\ \forall j \in [m]$
else
    $(y_1, \ldots, y_m) \leftarrow S_{\mathsf{HSS}}(1^\lambda, pk, f(x_1, \ldots, x_n))$
endif
$b' \leftarrow A_1(\mathsf{state}, (y_1, \ldots, y_m))$
return $b = b'$
```

Fig. 2. Security and context hiding for homomorphic secret sharing scheme

One of the most famous homomorphic secret sharing schemes is Shamir's secret sharing scheme [36]. To share a secret input in the (m, t)-threshold setting, a degree-t polynomial $P(x)$ is randomly generated with the constant term equals to the secret input. The j-th server gets the share of the secret input as $P(j)$. If the servers locally add two shares, the results, which is also degree-t, will represent the addition of the two corresponding inputs. However, the results of locally multiplying two shares will correspond to a degree-$2t$ polynomial, which represents $(m, 2t)$-threshold setting. Thus, the maximum computable polynomial degree d of Shamir's scheme must satisfy $dt < m$ in order to reconstruct the computation result, and we have $d \leq \lfloor \frac{m-1}{t} \rfloor$.

Shamir's scheme only supports threshold non-access structure. To realize any general non-access structure Γ, the secret sharing technique of Ito et al. [27] can be used. To share a secret input x_i, generate a share $x_{i,\gamma}$ for each non-access set γ in the maximum non-access structure Γ^* such that $\sum_{\gamma \in \Gamma^*} x_{i,\gamma} = x_i$. The share $x_{i,\gamma}$ is given to the j-th server if $j \notin \gamma$. From this technique, each server can get more than one shares per one secret input.

2.3 Homomorphic Encryption (HE)

Homomorphic encryption schemes let users perform operations on ciphertext in the same way as performing on plaintext while keeping the plaintext secret. The definition of homomorphic encryption scheme is as follows. More details can be found in [33].

Definition 3. *A homomorphic encryption scheme consists of four algorithms* $\mathsf{HE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$.

Key Generation. $(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$: *The algorithm receives the security parameter* λ *as an input, and then generates a pair of a public key and a secret key* (pk, sk).

Encryption. $c \leftarrow \mathsf{Enc}(pk, x)$: *The algorithm receives the public key* pk *and a plaintext* x *in a specified plaintext space as inputs, and then generates the corresponding ciphertext* c.

Evaluation. $c \leftarrow \mathsf{Eval}(pk, f, (c_1, \ldots, c_n))$: *The algorithm receives the public key pk, a function f, and ciphertexts $c_1, \ldots, c_n$ as inputs, and then generates the ciphertext of the evaluation c.*

Decryption. $x \leftarrow \mathsf{Dec}(sk, c)$: *The algorithm receives the secret key sk and a ciphertext c as inputs, and then decrypts into the corresponding plaintext x.*

Compactness. The homomorphic encryption scheme is compact if the size of the output from the evaluation algorithm $\mathsf{HE.Eval}$ is a polynomial in the security parameter λ, independent to the size of the description of the function f.

Correctness. The homomorphic encryption scheme for degree-d polynomials with n inputs is correct if, for any $\lambda \in \mathbb{N}$, any $n \in \mathsf{poly}(\lambda)$, any key pair $(pk, sk) \leftarrow \mathsf{HE.KGen}(1^\lambda)$, any degree-$d$ polynomial f, any inputs $(x_1, \ldots, x_n)$, any ciphertexts $c_i \leftarrow \mathsf{HE.Enc}(pk, x_i)$ for all $i \in [n]$, and any ciphertext $c \leftarrow \mathsf{HE.Eval}(pk, f, (c_1, \ldots, c_n))$, it holds that

$$\mathsf{Pr}[\mathsf{HE.Dec}(sk, c) = f(x_1, \ldots, x_n)] \geq 1 - \mathsf{negl}(\lambda).$$

Security. The homomorphic encryption scheme for degree-d polynomials with n inputs is semantically secure [26] if, for any $\lambda \in \mathbb{N}$, any $n \in \mathsf{poly}(\lambda)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for any probabilistic polynomial time adversary $A = (A_0, A_1)$,

$$\mathsf{Pr}[\mathsf{Security}_{A,\mathsf{HE}} = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathsf{Security}_{A,\mathsf{HE}}$ is defined in Fig. 3. The explanation of adversaries A_0 and A_1 is similar to the explanation in Section 2.2.

Circuit Privacy. Similar to homomorphic secret sharing, the output client should learn nothing from the ciphertexts except the result of the evaluation. This means the distribution of the actual encrypted result $c \leftarrow \mathsf{HE.Eval}(pk, f, (c_1, \ldots, c_n))$ and the distribution of newly encrypted result $c \leftarrow \mathsf{HE.Enc}(pk, f(x_1, \ldots, x_n))$ should be indistinguishable.

The homomorphic encryption scheme for degree-d polynomials with n inputs is circuit-private if there exists a probabilistic polynomial time algorithm S_{HE} and a negligible function $\mathsf{negl}(\lambda)$ such that for any $\lambda \in \mathbb{N}$, any $n \in \mathsf{poly}(\lambda)$, any key pair $(pk, sk) \leftarrow \mathsf{HE.KGen}(1^\lambda)$, any degree-$d$ polynomial f, any inputs $(x_1, \ldots, x_n)$, any ciphertext $c_i \leftarrow \mathsf{HE.Enc}(pk, x_i)$ for all $i \in [n]$, it holds that

$$\mathbb{SD}[\mathsf{HE.Eval}(pk, f, (c_1, \ldots, c_n)), S_{\mathsf{HE}}(1^\lambda, pk, f(x_1, \ldots, x_n))] < \mathsf{negl}(\lambda)$$

where the statistical distance $\mathbb{SD}$ between random variables X and Y over a finite set U is defined as

$$\mathbb{SD}[X, Y] = \frac{1}{2} \sum_{u \in U} \mid \Pr[X = u] - \Pr[Y = u] \mid .$$

```
$\mathsf{Security}_{A,\mathsf{HE}}(1^\lambda)$ :
$(pk, sk) \leftarrow \mathsf{HE.KGen}(1^\lambda)$
$(x_0, x_1, \mathsf{state}) \leftarrow A_0(pk)$
$b \leftarrow \{0, 1\}$
$c \leftarrow \mathsf{HE.Enc}(pk, x_b)$
$b' \leftarrow A_1(\mathsf{state}, pk, c)$
return $b = b'$
```

Fig. 3. Security for homomorphic encryption scheme

In the literature, there exists homomorphic encryption schemes for degree-1 polynomials [18, 34] which are the most efficient, for degree-2 polynomials [2, 8, 15, 20] which are somewhat efficient, and for degree-k polynomials [13, 22, 37] which are not efficient enough to be used in real applications when k is large.

3 Proposed Scheme

In this section, we show our construction of a t-secure homomorphic secret sharing scheme for degree-d polynomials with n inputs and m servers from degree-k homomorphic encryption scheme. Note again that the maximum polynomial degree of our proposed scheme is $d = \lfloor \frac{(k+1)m-1}{t} \rfloor$. After that, we prove the formula for the maximum polynomial degree d and the maximum number of servers m. We also analyze correctness and security of the scheme, and suggest several variants.

3.1 Construction

Given the number of inputs n, the number of servers m, and the threshold t, our homomorphic secret sharing scheme consists of four algorithms as follows. We also refer to some details of Lai et al. [31] for the sake of completeness of this paper. Similar to [31], a specific type of ring mentioned in [15] is sufficient to be our message space.

Key Generation. Output client generates a pair of a public key and a secret key of the underlying degree-k homomorphic encryption $(pk, sk) \leftarrow \mathsf{HE.KGen}(1^\lambda)$, and publishes the public key pk to all parties.

Secret Sharing. Consider the maximum non-access structure Γ^* of the (m, t)-threshold non-access structure. To secretly share an input x_i from the message space of the underlying homomorphic encryption scheme, randomly generate $|\Gamma^*| = \binom{m}{t}$ shares $(x_{i,\gamma})_{\gamma \in \Gamma^*}$ corresponding to each $\gamma \in \Gamma^*$ such that $\sum_{\gamma \in \Gamma^*} x_{i,\gamma} = x_i$ as in Ito et al. [27]. In addition, randomly generate shares of zero $(z_{i,j})_{j \in [m]}$ such that $\sum_{j \in [m]} z_{i,j} = 0$. For the j-th server, define $x_{i,\gamma,j}$ as $\mathsf{HE.Enc}(pk, x_{i,\gamma})$ if $j \in \gamma$, and define as $x_{i,\gamma}$ otherwise. Finally, the j-th server gets $s_{i,j} = ((x_{i,\gamma,j})_{\gamma \in \Gamma^*}, z_{i,j})$ as a share of x_i. The shares $s_{i,j}$ of the secret input x_i for the j-th server and all relevant values can be summarized as follows.

$$\begin{array}{lll} 0 & \xrightarrow{share} & (z_{i,j})_{j\in[m]} \\ x_i & \xrightarrow{share} & (x_{i,\gamma})_{\gamma\in\Gamma^*} \\ & \xrightarrow{separate} & (x_{i,\gamma})_{\gamma\in\Gamma^*:j\notin\gamma}, (x_{i,\gamma})_{\gamma\in\Gamma^*:j\in\gamma} \text{ when consider the } j\text{-th server} \\ & \xrightarrow{encrypt} & (x_{i,\gamma})_{\gamma\in\Gamma^*:j\notin\gamma}, (\boxed{x_{i,\gamma}})_{\gamma\in\Gamma^*:j\in\gamma} \\ s_{i,j} := & & (x_{i,\gamma})_{\gamma\in\Gamma^*:j\notin\gamma}, (\boxed{x_{i,\gamma}})_{\gamma\in\Gamma^*:j\in\gamma}, (z_{i,j}) \end{array}$$

Evaluation. To calculate a degree-d polynomial f with n variables $(x_1, \ldots, x_n)$, we split the function f into $(f_1, \ldots, f_m)$ and assign f_j to the j-th server. The first condition for the split is $\sum_{j\in[m]} f_j(s_{1,j}, \ldots, s_{n,j}) = f(x_1, \ldots, x_n)$. The second condition can be explained as follows.

Similar to [31], we write the polynomial f as a sum of monomials $f(x_1, \ldots, x_n) = \sum_w a_w M_w(x_1, \ldots, x_n)$ where each w is a monomial of degree $c \le d$ with coefficient a_w. Consider each monomial w of degree c, say $a_w M_w(x_1, \ldots, x_n) = a_w x_{w_1} x_{w_2} \cdots x_{w_c}$ for some indices $w_1, \ldots, w_c \in [n]$. From the secret sharing algorithm above, $x_i = \sum_{\gamma\in\Gamma^*} x_{i,\gamma}$. Thus, we have

$$\begin{aligned} a_w M_w(x_1, \ldots, x_n) = a_w x_{w_1} \cdots x_{w_c} &= a_w \prod_{i\in\{w_1,\ldots,w_c\}} \left(\sum_{\gamma\in\Gamma^*} x_{i,\gamma} \right) \\ &= \sum_{\gamma_1,\ldots,\gamma_c\in\Gamma^*} a_w x_{w_1,\gamma_1} \cdots x_{w_c,\gamma_c} \end{aligned}$$

We prove in Section 3.2 that to calculate any monomial $a_w x_{w_1,\gamma_1} \cdots x_{w_c,\gamma_c}$ with $c \le d$, there must be at least one j-th server such that from all c variables, $a_w x_{w_1,\gamma_1,j} \cdots x_{w_c,\gamma_c,j}$, at most k of them are encrypted. Because the underlying homomorphic encryption supports degree-k polynomials, the j-th server can compute this monomial.

Each monomial will be assigned to one of the servers that can compute that monomial. We define the homomorphic summation of all monomials assigned to the j-th server as $g_j(x_1, \ldots, x_n)$. Finally, we define $f_j(x_1, \ldots, x_n) = g_j(x_1, \ldots, x_n) + \sum_{i\in[n]} z_{i,j}$. The j-th server then computes $y_j = f_j(x_1, \ldots, x_n)$ using the operations of the underlying homomorphic encryption scheme, and forwards y_j to the output client.

Decoding. The output client homomorphically sums the results from all servers $y = y_1 + \ldots + y_m$. The final result of $f(x_1, \ldots, x_n)$ is obtained from $\mathsf{HE.Dec}(sk, y)$.

3.2 The Maximum Degree d of the Proposed Scheme

We prove the upper bound of the maximum degree d of the proposed homomorphic secret sharing scheme in this section. We show that each monomial $a_w x_{w_1,\gamma_1} \cdots x_{w_c,\gamma_c}$ with degree $c \le d$ can be assigned to one of the servers.

Theorem 1. *The upper bound of the maximum degree d of the construction in Section 3.1 is $d \le \frac{(k+1)m-1}{t}$.*

Proof. We prove the theorem using hyper-graph representation of the non-access structure. Do not confuse between the degree of the hyper-graph and the degree of the polynomial.

Given the maximum non-access structure Γ^*, we construct a hyper-graph $H = (V, E)$ with $V = [m]$ and $E = \Gamma^*$. A hyper-edge $\gamma \in \Gamma^*$ represents a non-access set, and is incident to a vertex j if and only if $j \in \gamma$. This also means that the j-th server gets the γ-part of the share as a ciphertext.

To be able to calculate the degree-d monomials $a_w x_{w_1,\gamma_1} \cdots x_{w_d,\gamma_d}$, there should be at least one servers that the number of encrypted variables is at most k (the degree of the underlying homomorphic encryption scheme). Note that a hyper-edge is corresponding to a non-access set and also corresponding to one part of the shares. In the same way, if we choose any d hyper-edges, we want that there exists at least one vertex that has at most k incident hyper-edges. The degree of each hyper-edge is t. Thus, choosing d hyper-edges is equivalent to dt incidences. From pigeonhole principle and the fact that H is a regular hyper-graph, if there is at least one vertex that has at most k incident hyper-edges, the maximum number of all incidences must not exceed

$$\underbrace{(k+1) + \cdots + (k+1)}_{m-1 \text{ times}} + k = (k+1)m - 1$$

Thus, we have $dt \leq (k+1)m - 1$, and this concludes the proof. □

3.3 Computational Complexity

The computational complexity of all parties must be a polynomial in the security parameter λ. The output client only performs m homomorphic additions and one decryption. If the underlying homomorphic encryption scheme is compact, it is obvious that the computational complexity of the output client is $\mathsf{poly}(\lambda)$.

For the input clients, each one generates m-choose-t shares and encrypts $(m-1)$-choose-$(t-1)$ out of them per server. We know that $\binom{m}{t} \leq (em)^t$ where e is Euler's constant. Because $t < m$, the computational complexity of each input client is $O(m^m)$. It is sufficient to set $m = O(\frac{\log \lambda}{\log \log \lambda})$ to let this bound be a polynomial in λ. This bound of m is equal to 1-secure scheme of Lai et al. [31].

The computational complexity of each server must also be a polynomial in the security parameter λ. From Section 3.1, we define that each monomial $a_w M_w(x_1, \ldots, x_n) = a_w x_{w_1} \cdots x_{w_c}$ in $f(x_1, \ldots, x_n)$ is expanded to $\sum a_w x_{w_1,\gamma_1} \cdots x_{w_c,\gamma_c}$. We now show that the number of monomials $a_w x_{w_1,\gamma_1} \cdots x_{w_c,\gamma_c}$ assigned to each server for each monomial $a_w M_w(x_1, \ldots, x_n)$ can be $\mathsf{poly}(\lambda)$.

The maximum number of degree-d monomials $a_w x_{w_1,\gamma_1} \cdots x_{w_d,\gamma_d}$ that a server can calculate is equal to $\sum_{\ell=0}^{k} \binom{d}{\ell} (b-p)^\ell p^{d-\ell} = \sum_{\ell=0}^{k} \binom{d}{\ell} \binom{m-1}{t-1}^\ell \binom{m-1}{t}^{d-\ell}$. The idea is that a server can calculate all monomials with $\ell \leq k$ encrypted shares. For a monomial with ℓ encrypted shares, we can choose ℓ out of d positions to be the encrypted shares. For each encrypted position, there are $b - p = \binom{m-1}{t-1}$ choices of ciphertexts to choose from. For each plaintext position, there are $p = \binom{m-1}{t}$ choices of plaintexts to choose from. Thus the total combinations are

$\sum_{\ell=0}^{k}\binom{d}{\ell}\binom{m-1}{t-1}^{\ell}\binom{m-1}{t}^{d-\ell}$. We show that this value can be a polynomial in λ if we choose an appropriate value of m.

$$\begin{aligned}
\sum_{\ell=0}^{k}\binom{d}{\ell}\binom{m-1}{t-1}^{\ell}\binom{m-1}{t}^{d-\ell} &\leq \sum_{\ell=0}^{k}\binom{d}{\ell}(em)^{\ell t}(em)^{(d-\ell)t} \\
&= (em)^{dt}\sum_{\ell=0}^{k}\binom{d}{\ell} \\
&\leq (em)^{dt}2^{d} \\
&\leq e^{(k+1)m}2^{\frac{(k+1)m}{t}}m^{(k+1)m} = m^{O(m)}
\end{aligned}$$

It is sufficient to set $m = O(\frac{\log\lambda}{\log\log\lambda})$ to let this bound be a polynomial in λ. In addition, assume that the number of all monomials $a_w M_w(x_1,\ldots,x_n)$ in $f(x_1,\ldots,x_n)$ is also a polynomial in λ. Thus, the total number of monomials $a_w x_{w_1,\gamma_1}\cdots x_{w_c,\gamma_c}$ that a server has to calculate is a polynomial in λ. In case that we want a larger value of threshold t, the maximum degree d of our homomorphic secret sharing scheme will be smaller, and this will not affect the bound.

3.4 Correctness and Security

We have shown that the bound of the maximum degree d is correct, and the number of servers can be a polynomial in the security parameter λ. In this section, we extend the arguments of Lai et al. [31] to show correctness and security of the proposed t-secure scheme. We first show that the proposed scheme is correct.

Theorem 2. *If the underlying homomorphic encryption scheme is correct, then the proposed homomorphic secret sharing scheme is correct.*

Proof. From Theorem 1, each monomial $a_w x_{w_1,\gamma_1}\cdots x_{w_c,\gamma_c}$ with degree $c \leq d$ can be assigned to one of the servers. Thus, according to the description of the evaluation algorithm in Section 3.1, we have $\sum_{j\in[m]}\mathsf{HE.Dec}(sk, g_j(x_1,\ldots,x_n)) = f(x_1,\ldots,x_n)$. If the underlying homomorphic encryption scheme is correct, then

$$\begin{aligned}
\mathsf{HE.Dec}(sk, y) &= \mathsf{HE.Dec}(sk, \sum_{j\in[m]} y_j) \\
&= \mathsf{HE.Dec}(sk, \sum_{j\in[m]} f_j(x_1,\ldots,x_n)) \\
&= \mathsf{HE.Dec}(sk, \sum_{j\in[m]} (g_j(x_1,\ldots,x_n) + \sum_{i\in[n]} z_{i,j})) \\
&= \mathsf{HE.Dec}(sk, \sum_{j\in[m]} g_j(x_1,\ldots,x_n)) + \sum_{i\in[n]}\sum_{j\in[m]} z_{i,j} \\
&= \sum_{j\in[m]}\mathsf{HE.Dec}(sk, g_j(x_1,\ldots,x_n)) + 0 = f(x_1,\ldots,x_n)
\end{aligned}$$

which means the proposed scheme is also correct. □

Next, we show that the proposed scheme is a t-secure homomorphic secret sharing scheme.

Theorem 3. *If the underlying homomorphic encryption scheme is semantically secure, then the proposed homomorphic secret sharing scheme is t-secure.*

Proof. We will show that if there is an adversary A that can guess the input of our homomorphic secret sharing with a non-negligible advantage, we can construct an adversary A' to break the semantic security of the underlying homomorphic encryption scheme. The process is shown in Fig. 4.

Consider (m,t)-threshold non-access structure Γ. The process starts when the oracle generates a public key pk of the homomorphic encryption scheme, and forwards it to the adversary A', which also forwards to the adversary A. Adversary A then generates two inputs x_0^*, x_1^*, and a non-access set $\gamma^* \in \Gamma^*$, and forwards to A'. (If the homomorphic secret sharing scheme is secure for Γ^*, then it is also secure for Γ.) A' then generates a set of random shares S, and let x_0 and x_1 be γ^*-part of the shares of x_0^* and x_1^*, respectively. Adversary A' sends x_0 and x_1 to the oracle, and gets c_b as a response. Adversary A' forwards (S, c_b) as the shares of x_b^* with γ^*-part encrypted. If the adversary A can correctly guess the input x_b^* with a non-negligible advantage, this guess also has the same non-negligible advantage for adversary A' to guess x_b. $\square$

Finally, we state the theorem for context hiding as follows. The proof below is similar to the proof in [31].

Theorem 4. *If the underlying homomorphic encryption scheme is circuit-private, then the proposed homomorphic secret sharing scheme is context-hiding.*

Proof. According to the definition of context hiding, we must show that there exists a probabilistic polynomial time algorithm S_{HSS}. To process $S_{\mathsf{HSS}}(1^\lambda, pk, f(x_1, \ldots, x_n))$, the algorithm S_{HSS} generates random shares $r_1, \ldots, r_m$ such that $\sum_{j \in [m]} r_j = f(x_1, \ldots, x_n)$. The algorithm then generates $y_j \leftarrow \mathsf{HE.Enc}(pk, r_j)$ for all $j \in [m]$, and outputs $(y_1, \ldots, y_m)$.

We compare the distributions of the output from S_{HSS} and the output from $\mathsf{HSS.Eval}$. Because the random shares of zeroes $z_{i,j}$ are added in the evaluation algorithm $\mathsf{HSS.Eval}$, the output from $\mathsf{HSS.Eval}$ can be any encrypted tuple that adds up to $f(x_1, \ldots, x_n)$. In the same way, the output from the algorithm S_{HSS} is randomly generated such that the encrypted tuple adds up to $f(x_1, \ldots, x_n)$. If the underlying homomorphic encryption is semantically secure, then these two distributions must be indistinguishable. Thus, there is no adversary that can distinguish these two distributions with non-negligible advantage. $\square$

3.5 Variants of the Scheme

Extending from Lai et al. [31], we present two variants of our homomorphic secret sharing scheme.

Oracle		Adversary A'		Adversary A
$pk \leftarrow \mathsf{HE.KGen}(1^\lambda)$	$\Rightarrow$	pk	$\Rightarrow$	pk
		x_0^*, x_1^*, γ^*	$\Leftarrow$	Gen x_0^*, x_1^*, γ^*
		Gen $S = (s_\gamma)_{\gamma \in \Gamma^* \wedge \gamma \neq \gamma^*}$		
x_0	$\Leftarrow$	$x_0 = x_0^* - \sum_{s \in S} s$		
x_1	$\Leftarrow$	$x_1 = x_1^* - \sum_{s \in S} s$		
Random b				
$c_b \leftarrow \mathsf{HE.Enc}(pk, x_b)$	$\Rightarrow$	c_b		
		(S, c_b)	$\Rightarrow$	(S, c_b)
b'	$\Leftarrow$	b'	$\Leftarrow$	Guess b'

Fig. 4. Security proof of Theorem 3

Multi-key Homomorphic Encryption and Plain Model. There exists several multi-key homomorphic encryption schemes in the literature [32]. We describe it as follows.

Definition 4. *A multi-key homomorphic encryption scheme consists of four algorithms* $\mathsf{MKHE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$*.*

Key Generation and Encryption. The algorithms behave exactly the same as simple homomorphic encryption scheme in Definition 3.

Evaluation. $c \leftarrow \mathsf{Eval}(f, ((pk_1, c_1), \ldots, (pk_n, c_n)))$ *: The algorithm receives a function* f*, and the pairs of public key and the corresponding ciphertext* (pk_1, c_1)*,* $\ldots, (pk_n, c_n)$ *as inputs, and then generates the ciphertext of the evaluation* c *under all the public keys* $pk_1, \ldots, pk_n$*.*

Decryption. $x \leftarrow \mathsf{Dec}((sk_1, \ldots, sk_n), c)$ *: The algorithm receives the secret keys* $sk_1, \ldots, sk_n$ *and a ciphertext* c *encrypted under all the corresponding public keys as inputs, and then decrypt into the plaintext* x*.*

Similar to [31], our proposed scheme can be naturally extended to multi-key plain model. In this setting, the output client does not have to generate any key pairs. The i-th input client will generate its own key pair (pk_i, sk_i), and publish pk_i to all parties. The secret input x_i will be shared as $s_{i,j}$ using the public key pk_i. In addition, the shares of secret key sk_i are generated such that $\sum_{j \in [m]} sk_{i,j} = sk_i$. Both $s_{i,j}$ and $sk_{i,j}$ are forwarded to the corresponding j-th server. The j-th server then calculates f_j using the multi-key evaluation algorithm, and forwards the result together with the secret key shares to the output client. Finally, the output client reconstructs the secret keys from all shares, and uses the multi-key decryption algorithm to see the final result. It can be proved similar to [31] that this variant is t-secure.

***r*-robustness.** Suppose that only r servers can forward the results to the output client, the scheme is not completely failed. Calculating some polynomials is still possible in this setting. This property is called as r-robustness in [31]. The maximum degree d of the scheme will be decreased to $\lfloor \frac{(k+1)r-1}{t} \rfloor$. This bound can be proved in the same way as in Theorem 1.

There are some points to consider. Notice that in Section 3.1 and in the plain model described in this section, the shares of zeroes and the shares of secret keys can be reconstructed if all the servers are presented (m-out-of-m secret sharing). These prevent the possibility of r-robustness. The solution for the shares of zeroes is to use the secret sharing of the (m, t)-threshold non-access structure. The secret sharing of secret keys can also be done in the same way. In addition, threshold homomorphic encryption [28] may be used for this purpose.

4 Discussions

Compare to Lai et al. We consider these topics comparing to t-secure homomorphic secret sharing scheme of Lai et al. [31].

Number of Servers. For the t-secure scheme in [31], the authors fixed the degree of homomorphic encryption $k = 1$, the degree of homomorphic secret sharing $d = 3$, number of plaintext shares per input for each server $p = 2$, and number of all shares per input for each server $b = 2t + 1$. Then, the number of required servers is $m = t^2$ which is not efficient. In contrast, our scheme has $m \geq \frac{3t+1}{2}$ which is linear in t. This can be generalized to $m \geq \frac{dt+1}{k+1}$ for all values of k, d, and t, which is also linear in t.

Number of Shares. In 1-secure scheme of [31], the number of shares per input for each server is equal to the number of servers, $b = m$, and the number of plaintext shares per input for each server is $p = m - 1$. For the t-secure scheme, the authors fixed the degree of homomorphic encryption $k = 1$ and the degree of homomorphic secret sharing $d = 3$, then the suggested number of shares per input for each server is $b = 2t + 1$, and the number of plaintext shares per input for each server is $p = 2$.

In our scheme, $b = \binom{m}{t}$ and $p = \binom{m-1}{t}$. Increasing share size can be viewed as a trade-off with security. Since [31] is the special case of ours, the share size is the same when $t = 1$. We believe that the size is acceptable when t is small or close to m. For $t \approx m/2$, possible improvements are mentioned in Section 5. Although, our scheme has more number of shares, the computational complexity of the scheme is still a polynomial in the security parameter λ if the number of servers m is $O(\frac{\log \lambda}{\log \log \lambda})$. This bound is equal to the 1-secure scheme of [31].

Constructiveness. The concrete construction of the t-secure homomorphic secret sharing scheme is not proposed in [31]. We have to solve set cover problem in order to find the combination of encrypted shares for each server. Since the set cover problem is NP-complete, their construction is not constructive. In our scheme, the construction is concrete and clear. In addition, it is claimed in [31] that the number of plaintext shares from t servers must not exceed the number of all shares per input for each server, which can be written as $t \cdot p < b$, in order to keep the scheme secure. However, this is not mandatory. The plaintext shares from different servers can be overlapped.

Compare to Homomorphic Encryption. Let m, t, and d be fixed constants. Roughly comparing our scheme (with the BGN scheme [8] as the base scheme) to the GSW scheme [23], with the security parameter λ, our share size and computation time for each server are $O(\lambda)$ and $O(\lambda^3)$, while the ciphertext size and computation time of GSW are $O(\lambda^2 \log \lambda)$ and $O(\lambda^3 \log \lambda)$, respectively. Although, our scheme has hidden constant $\binom{m}{t}$ for share size, it can be smaller than $c\lambda \log \lambda$ for small m and t, and c depends on the GSW implementation. We mention again that the purpose of our work is not to outperform homomorphic encryption, but to combine it with secret sharing. The results are higher computable degree and single-point-of-failure mitigation.

5 Possible Improvements from Non-threshold Structure

Until the previous section, we only focus on threshold non-access structures. In fact, our proposed scheme also supports any non-access structures, including non-threshold non-access structures. Some observations about the benefits from non-threshold non-access structures are found, but they are not thoroughly understood. We give an example here how we can optimize the scheme using a non-threshold non-access structure and a relaxed constraint.

Example 2. Consider a setting with $m = 4$ servers and threshold $t = 2$. We can construct the maximum $(4, 2)$-threshold non-access structure $\Gamma_1^* = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$. With this maximum threshold non-access structure, each server gets $b_1 = |\Gamma_1^*| = 6$ shares, and the maximum degree of the homomorphic secret sharing scheme is $d_1 = 3$. However, if we relax the constraint that any three servers can reconstruct the secret, we can have the other maximum non-threshold non-access structure $\Gamma_2^* = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3, 4\}\}$. Any two servers still cannot reconstruct the secret inputs. Using this non-threshold structure, the number of shares for each server is reduced to $b_2 = |\Gamma_2^*| = 4$ while the maximum computable degree is still $d_2 = 3$. In this case, the numbers of plaintext and ciphertext shares are different for each server.

In addition, we also find an improvement in 6-party non-threshold cases that can reduce the share size from 20 to 10 shares. The setting is $m = 6$, $t = 3$, and $\Gamma_3^* = \{\{1, 2, 3, 4\}, \{1, 2, 5\}, \{1, 2, 6\}, \{1, 3, 5, 6\}, \{1, 4, 5\}, \{1, 4, 6\}, \{2, 3, 5, 6\}, \{2, 4, 5\}, \{2, 4, 6\}, \{3, 4, 5, 6\}\}$.

We have tried to find the relationship between non-threshold non-access structures and the other parameters, but it is still unclear. We present four possible improvements as follows.

Non-threshold Non-access Structure and the Maximum Degree d. To find the maximum degree d of the homomorphic secret sharing scheme realized any non-access structure, the exact formula is still unknown. However, we can construct an optimization model from the corresponding hyper-graph. This optimization model can be solved using integer program (IP) as in Fig. 5. The idea is to select the smallest number of hyper-edges which is adjacent to each vertex more than k times.

Input	Positive integers m, Non-negative integer k, hyper-graph $H = (V, E) = ([m], \Gamma^*)$
Output	Non-negative integer d
Objective function	Minimize d
Constraints	Let $x_\gamma \in \mathbb{Z}_{\geq 0}$ be a variable for each $\gamma \in \Gamma^*$ $\sum_{\gamma \in \Gamma^*} x_\gamma = d + 1$ $\sum_{\gamma \in \Gamma^* : j \in \gamma} x_\gamma > k$ for all $j \in [m]$

Fig. 5. Integer program to find the maximum degree d of any non-access structure.

In addition to the optimization model, the multiplicative property of the non-access structure in [4] may also be considered. In the paper, the maximum degree of computable polynomial is described for the case when homomorphic encryption is not used (or the degree of the homomorphic encryption is $k = 0$).

Non-threshold Non-access Structure and Number of Shares b. It is possible to reduce the number of shares by using non-threshold non-access structures as in Example 2 above. The process may be considered as to union some non-access sets in the threshold structure. However, we still do not know which non-threshold structure should be used instead of the threshold one. Generalizing threshold structure to non-threshold structure can sometimes reduce the maximum degree d of the homomorphic secret sharing scheme.

Extension of r-robustness. It may be possible to generalize the definition of r-robustness to $\bar{\Gamma}$-robustness, where $\bar{\Gamma}$ is the "access" structure that contains the "access" sets. In $\bar{\Gamma}$-robustness, combining the result from an access set of server $\bar{\gamma} \in \bar{\Gamma}$ can reconstruct the secret. The relationship between $\bar{\Gamma}$ and the maximum degree d of the $\bar{\Gamma}$-robustness homomorphic secret sharing is also unknown. It may be possible to use $\bar{\Gamma}$ such that $\Gamma \cap \bar{\Gamma} = \emptyset$ but $\Gamma \cup \bar{\Gamma} \neq 2^{[m]}$.

Multi-use Context Hiding. For the sake of randomness, the shares of zeroes must be used for one time only. To evaluate several polynomials of the same shares, Lai et al. [31] proposed a technique using random values from a pseudo-random function. The key k_j and k_{j+1} are given to the j-th server, and $\mathsf{PRF}(k_j, f) - \mathsf{PRF}(k_{j+1}, f)$ is used instead of the shares of zeroes. However, their security threshold is only 1. If at least $m/2$ servers are colluding, they can learn all the keys. The construction of multi-use context-hiding homomorphic secret sharing scheme for threshold and non-threshold non-access structure Γ from a pseudo-random function has not been proposed. Extending the idea to r-robust and $\bar{\Gamma}$-robust multi-use context-hiding is also interesting.

6 Concluding Remarks

In this paper, we propose the constructive t-secure homomorphic secret sharing scheme from homomorphic encryption. The maximum computable degree of the scheme is proved using hyper-graph, and the number of required servers is analyzed. We also proposed several variants and possible improvements from non-threshold structure. To conclude the paper, two interesting settings are suggested as future works.

Malicious Security. In the construction presented in Section 3, we only consider semi-honest security where each party follows the protocol correctly, but may try to collude and learn the secret inputs. However, a party may deviate from the protocol arbitrarily. The input client may send different value of shares to each server. The ciphertext and plaintext may represent different values. The sum of all shares of zeroes may not be equal to zero. For the server, it may send any value as the evaluation result to the output client.

We may have to change the security model to real-ideal model [14] to handle the malicious security. The solution to these problems can be proposed in two levels. The simpler solution is to detect the malicious behaviour, and abort if some errors are detected. The better solution is to correct the errors, and then continue to follow the protocol. The scheme with r-robustness or $\bar{\Gamma}$-robustness can be useful. The idea of verifiable homomorphic encryption [19] and verifiable secret sharing [21, 35] may also be used for this purpose.

Generalize to any Inner/Outer Scheme. In our construction, it can be seen that we use homomorphic encryption as the inner scheme, and secret sharing of Ito et al. as the outer scheme. It may be possible to generalize this idea of homomorphic secret sharing construction to any combinations of inner and outer schemes. Thus, we should have a better understanding of each secure computation protocol, and how the combinations should be.

Acknowledgments. Nuttapong Attrapadung was partly supported by JST CREST Grant Number JPMJCR19F6, and by JSPS KAKENHI Kiban-A Grant Number 19H01109.

References

1. Applebaum, B., Ishai, Y., Kushilevitz, E.: Computationally private randomizing polynomials and their applications. Computational Complexity **15**(2), 115–162 (2006)
2. Attrapadung, N., Hanaoka, G., Mitsunari, S., Sakai, Y., Shimizu, K., Teruya, T.: Efficient two-level homomorphic encryption in prime-order bilinear groups and A fast implementation in WebAssembly. In: Asia Conference on Computer and Communications Security. pp. 685–697 (2018)
3. Babai, L., Kimmel, P.G., Lokam, S.V.: Simultaneous messages vs. communication. In: Annual Symposium on Theoretical Aspects of Computer Science. pp. 361–372 (1995)
4. Barkol, O., Ishai, Y., Weinreb, E.: On d-multiplicative secret sharing. Journal of cryptology **23**(4), 580–593 (2010)
5. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Annual International Cryptology Conference. pp. 420–432 (1991)
6. Beimel, A., Gabizon, A., Ishai, Y., Kushilevitz, E., Meldgaard, S., Paskin-Cherniavsky, A.: Non-interactive secure multiparty computation. In: Annual Cryptology Conference. pp. 387–404 (2014)

7. Beimel, A., Ishai, Y.: Information-theoretic private information retrieval: A unified construction. In: International Colloquium on Automata, Languages, and Programming. pp. 912–926 (2001)
8. Boneh, D., Goh, E.J., Nissim, K.: Evaluating 2-DNF formulas on ciphertexts. In: Theory of Cryptography Conference. pp. 325–341 (2005)
9. Boyle, E.: Recent advances in function and homomorphic secret sharing - (invited talk). In: International Conference on Cryptology in India. pp. 1–26 (2017)
10. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 337–367 (2015)
11. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Annual International Cryptology Conference. pp. 509–539 (2016)
12. Boyle, E., Gilboa, N., Ishai, Y., Lin, H., Tessaro, S.: Foundations of homomorphic secret sharing. In: Innovations in Theoretical Computer Science Conference (2018)
13. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: IEEE Symposium on Foundations of Computer Science. pp. 97–106 (2011)
14. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: IEEE Symposium on Foundations of Computer Science. pp. 136–145 (2001)
15. Catalano, D., Fiore, D.: Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data. In: ACM SIGSAC Conference on Computer and Communications Security. pp. 1518–1529 (2015)
16. Chabanne, H., de Wargny, A., Milgram, J., Morel, C., Prouff, E.: Privacy-preserving classification on deep neural network. IACR Cryptology ePrint Archive (2017)
17. Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: Annual International Cryptology Conference. pp. 93–122 (2016)
18. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31**, 469–472 (1985)
19. Fiore, D., Gennaro, R., Pastro, V.: Efficiently verifiable computation on encrypted data. In: ACM SIGSAC Conference on Computer and Communications Security. pp. 844–855 (2014)
20. Freeman, D.M.: Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 44–61 (2010)
21. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In: ACM Symposium on Principles of Distributed Computing. pp. 101–111 (1998)
22. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: ACM Symposium on Theory of Computing. pp. 169–178 (2009)
23. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Annual Cryptology Conference. pp. 75–92 (2013)
24. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: International Conference on Machine Learning. pp. 201–210 (2016)
25. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: ACM Symposium on Theory of Computing. pp. 218–229 (1987)

26. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences **28**(2), 270–299 (1984)
27. Ito, M., Saito, A., Nishizeki, T.: Secret sharing schemes realizing general access structure. In: IEEE Global Telecommunication Conference. pp. 99–102 (1987)
28. Jain, A., Rasmussen, P.M.R., Sahai, A.: Threshold fully homomorphic encryption. IACR Cryptology ePrint Archive (2017)
29. Kamara, S., Mohassel, P., Raykova, M.: Outsourcing multi-party computation. IACR Cryptology ePrint Archive (2011)
30. Kamara, S., Mohassel, P., Riva, B.: Salus: A system for server-aided secure function evaluation. In: ACM Conference on Computer and Communications Security. pp. 797–808 (2012)
31. Lai, R.W.F., Malavolta, G., Schröder, D.: Homomorphic secret sharing for low degree polynomials. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 279–309 (2018)
32. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: ACM Symposium on Theory of Computing. pp. 1219–1234 (2012)
33. Martins, P., Sousa, L., Mariano, A.: A survey on fully homomorphic encryption: An engineering perspective. ACM Computing Surveys **50**(6), 1–33 (2017)
34. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 223–238 (1999)
35. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. pp. 129–140 (1991)
36. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
37. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 24–43 (2010)