Evolving Homomorphic Secret Sharing for Hierarchical Access Structures

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. Secret sharing is a cryptographic primitive that divides a secret into several shares, and allows only some combinations of shares to recover the secret. As it can also be used in secure multi-party computation protocol with outsourcing servers, several variations of secret sharing are devised for this purpose. Most of the existing protocols require the number of computing servers to be determined in advance. However, in some situations we may want the system to be "evolving". We may want to increase the number of servers and strengthen the security guarantee later in order to improve availability and security of the system. Although evolving secret sharing schemes are available, they do not support computing on shares. On the other hand, "homomorphic" secret sharing allows computing on shares with small communication, but they are not evolving. As the contribution of our work, we give the definition of "evolving homomorphic" secret sharing supporting both properties. We propose two schemes, one with hierarchical access structure supporting multiplication, and the other with partially hierarchical access structure supporting computation of low degree polynomials. Comparing to the work with similar functionality of Choudhuri et al. (IACR ePrint 2020), our schemes have smaller communication costs.

Keywords: Secure multi-party computation · Evolving secret sharing · Homomorphic secret sharing · Hierarchical secret sharing.

1 Introduction

Secret sharing is a cryptographic primitive that divides a secret into several shares, and different shares will be given to different parties. The authorized sets of parties can recover the secret from their shares, while the unauthorized sets cannot. A collection of authorized sets is called as an access structure. In one type of the access structures, called as threshold structures, a set of parties is in the collection if the size of the set is larger than a particular number.

This basic primitive can be used as a building block to construct secure multi-party computation protocols [13, 3]. We will consider the model of outsourcing servers [10, 15]. In this model, there are three roles, namely, several input clients (dealers), several computing servers, and one output client. We have several secrets from several input clients. Each input client divides its secret into shares, and distributes them to computing servers. The goal of the scheme is to let only the output client know some functions of the secret inputs. We want to use multiple servers to calculate a function of those secrets without having them know the secrets. To achieve the goal, the computing servers calculate some functions on the shares, may communicate to other servers, and send their results to the output client. The output client then reconstructs the final result using partial results from participating servers. We note again that the protocol may require several communications back and forth between computing servers.

To achieve a smaller communication cost, a variation of secret sharing called as "homomorphic" secret sharing is introduced by Boyle et al. in [6]. Functions can be calculated on shares homomorphically without any communication between computing servers. Homomorphic secret sharing has been widely studied recently with multiple constructions from different assumptions, such as decisional Diffie–Hellman assumption (DDH) [7] and learning with errors (LWE) [8]. Furthermore, evaluation of low-degree polynomials in homomorphic secret sharing setting was also considered in [22, 24]. Homomorphic secret sharing is shown in [7] to imply a useful related primitive called server-aided secure multi-party computation [16, 17].

In most of the existing multi-party computation protocols from secret sharing, including homomorphic secret sharing, number of outsourcing servers must be determined in advance, and the access structure has to be fixed. This prevents us from adding more servers in order to improve availability of the system, or changing the access structure in order to improve security. Some recent works [4, 9, 14] allow new servers to join during the protocol, but they use resharing which requires interactions and communications. In addition, these works only support threshold structures.

There are some cryptographic schemes that allow us to add more servers without resharing. We call such schemes as "evolving" schemes. Those include evolving secret sharing proposed by Komargodski et al. [20]. Some improvements in this research area were proposed in [21, 1, 2]. Although we can construct secure multi-party computation protocols for outsourcing servers from secret sharing, it is not trivial to construct an evolving version from the evolving secret sharing. The construction is stated as a future work in [20], and is still an open problem.

1.1 Our Contributions

To provide a solution for the open problem, we give the definition of "evolving homomorphic" secret sharing. For "evolving", the schemes allow us to increase the number of outsourcing servers without resharing, and for "homomorphic", the schemes support computing on shares without any communications between servers. Thus, our schemes provide protocols for evolving outsourcing servers with smaller communication cost than previous works.

Our proposed evolving homomorphic secret sharing schemes focus on hierarchical access structures, where threshold values can be changed for different

Table 1. Our contributions compared to evolving secret sharing schemes

	Homomorphic	Correctness	Security	Access structure
Evolving secret sharing [20, 21, 1, 2]	✗	Perfect	Perfect	(Dynamic) threshold or ramp
Our warm-up scheme	✓ Degree-d	Almost perfect	Perfect	Fixed threshold
Our scheme 1	✓ Multiplication	Almost perfect	Perfect	Dynamic threshold
Our scheme 2	✓ Degree-d	Perfect	Computational	Partially dynamic threshold

number of servers. These "hierarchical access structures" allow us to adjust the security level when new servers are added.

We construct our schemes from combinations of homomorphic secret sharing and cryptographic primitives, namely hash functions and pseudo-random functions. This work focuses on the schemes that support low degree polynomial computation. Our two proposed schemes support hierarchical structure and partially hierarchical structure. We relax some constraints in order to get simple schemes. The first scheme is perfectly secure, but almost perfectly correct. This scheme is quite simple, uses only one share per secret, and has flexible access structure. However, the supported function here is just multiplication. In the second scheme, the supported function is improved to degree-d polynomials, but the access structure is a little more restricted, and it requires a few more shares per secret. This scheme is perfectly correct, but computationally secure. Table 1 compares our work to the existing evolving secret sharing schemes.

There exists a concurrent and independent work by Choudhuri et al. [9], who consider a secret sharing scheme that involves dynamic sets of servers and can securely compute on shares. However, their scheme requires interactions among servers, and thus, is different from our evolving secret sharing setting, which does not require interaction among servers. We will compare their scheme with our work in Section 1.3.

1.2 Our Approach

As a warm-up, we introduce our idea here. If we allow the secret sharing scheme to be almost perfectly correct, the scheme can be simpler than the existing evolving secret sharing scheme. From the homomorphic property of the Shamir's secret sharing [25], we would like to extend it to evolving homomorphic secret sharing. Normally, each input client in the Shamir's scheme generates a polynomial over a specified prime field. However, the degree of the polynomial has to be fixed. This means the security threshold of the scheme cannot be change without resharing. In addition, the number of computing servers is limited due to the size of the underlying field.

Our idea is that, it is possible to use a collision-resistant hash function to map the ID of each server, which can be infinite, to an element in the finite prime field. Informally, the share of the original Shamir's scheme is the polynomial P of

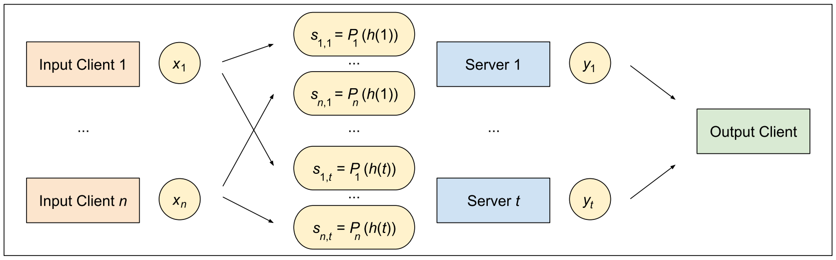

Fig. 1. The overview of our warm-up evolving homomorphic secret sharing scheme.

the ID of each server, $P(ID)$, while our idea uses $P(h(ID))$ where h is the hash function. Using this technique, the scheme can support infinite number of servers, with negligible collision probability from the hash function. The reconstructions and homomorphic property immediately follow from the Shamir's scheme. The overview of the warm-up scheme can be shown as in Fig. 1.

Shamir's scheme supports only one fixed threshold value k, i.e., any combinations with more than k servers can reconstruct the secrets. We may want to change the threshold when we add new servers to the system. To address this issue in the first proposed scheme, we elaborate the hash function idea into the hierarchical secret sharing of [26]. However, it is not straightforward to see which polynomials will be used in this case. Originally, the work of [26] used derivatives of polynomials. To realize evolving hierarchical access structure, which is equivalent to dynamic threshold structure in [21], our proposed scheme will use integrals. This scheme is correct with overwhelming probability. As shown in Table 2, share size of this scheme is $poly(\lambda)$ where λ is the security parameter.

To improve the scheme to be perfectly correct and support more general functions, we trade-off the share size, the security, and the generality of the access structure in the second proposed scheme. Based on the Shamir's scheme, instead of using only one fixed prime field, we try to expand the field during the sharing phase. We change the tool to pseudo-random function in order to maintain the consistency between several shares of different prime fields. Each computing server will get two or three values as its shares instead of one. In our second scheme, we can divide the protocol into two phases. In the setup phase, random values with size $O(\lambda)$ can be distributed in advance before secret inputs are determined. And in the online phase, when the secret inputs are ready, the t-th party that join the protocol will get a share with size $O(\log t)$. Share size of each phase is also shown in Table 2.

1.3 Related Works

We consider two types of secret sharing that involve dynamic sets of servers.

Schemes that only support storage and retrieval of secrets. Evolving secret sharing is a secret sharing scheme that supports infinite number of parties. The input clients will secretly share their inputs to the computing servers, and

Table 2. Quantitative comparison of evolving secret sharing schemes

Scheme	|Unauth. Set|	|Auth. Set|	Share Size
Komargodski et al. [20]	k	$k+1$	$O(k \log t)$
Komargodski et al. [21]	$f(t)$	$f(t)+1$	$O(t^4 \log t)$
Beimel and Othman [1]	αt	βt	$O(1)$
Beimel and Othman [2]	$\gamma t - t^\beta$	γt	$O(t^{4-\log^{-2}(1/\beta)} \log t)$
Our warm-up scheme	k	$k+1$	$O(\lambda)$
Our scheme 1	$f(t)$	$f(t)+1$	$poly(\lambda)$
Our scheme 2	See Section 5		$O(\lambda)$ for setup phase $O(\log t)$ for online phase

only authorized subsets of servers can reconstruct the secrets. The idea was firstly proposed by Komargodski et al. in [20]. Their scheme has one fixed threshold value k, i.e., subsets with more than k servers can reconstruct the secrets. The scheme is improved to dynamic threshold by Komargodski et al. in [21]. In this case, when the t-th party arrives, the threshold is changed to $f(t)$, which can be any non-decreasing function of t.

The next two schemes are based on ramp access structure. For some integers x and y, subsets with at most x parties will not be able to reconstruct the secret, and subsets with at least y parties will be able to reconstruct the secret. The key of the ramp schemes is that x and y do not have to be adjacent in order to reduce the share size, and there is no condition for subsets with size between x and y. In the work of Beimel and Othman [1], $x = \alpha t$ and $y = \beta t$ for some value $0 \leq \alpha < \beta \leq 1$. The same authors proposed closer bounds in [2], with $x = \gamma t$ and $y = \gamma t - t^\beta$ for some value $0 \leq \gamma, \beta \leq 1$. Table 2 shows the maximum size of unauthorized sets, the minimum size of authorized sets, and the share sizes.

None of the previous works claimed an application to multi-party computation. In these works, several generations of secret sharing are used, and some are additive secret sharing. If we build multi-party computation protocol from these schemes, the multiplication operation will require a lot of communication. Thus, we want to construct a better protocol which has no communication between computing servers at all.

For our proposed scheme, with security parameter λ, the access structure of the warm-up scheme is equivalent to [20], and that of the first scheme is equivalent to [21]. The access structure of the second scheme is less general, and details will be presented in the later section.

Schemes that also support computation on shares. Apart from evolving secret sharing, there exist some other works that achieve a somewhat similar evolving functionality by using blockchain. These works allow a set of participating servers to change during the protocol. The work of Goyal et al. [14] used a technique called "dynamic proactive secret sharing", and the work of Benhamouda et al. [4] used a similar idea called "evolving-committee proactive secret sharing". Although the main objective of both works is to store and retrieve secrets on blockchain, an application to MPC is also suggested. A notable work that directly focuses on MPC in the setting where dynamic sets of servers securely computing functions on shares is a recent protocol called “Fluid MPC” by Choudhuri et al. [9], which can be considered as a “fluid” version of the classic BGW protocol [3].

In these three papers, a set of participating servers is changed during the protocol by resharing the secrets from one set of servers to the other set. This point increases the numbers of communication and interaction in the protocol, and is obviously different from our work. In addition, these works only support threshold access structures, and hence all computing servers have the same role. On the other hand, our schemes support hierarchical access structures [26], where some parties can be assigned with different roles and powers in accessing the shared secret, and thus can be more flexible.

1.4 Organization

In Section 2, we review the background on secret sharing and some cryptographic primitives. Combining these definitions, we propose the definition of the evolving homomorphic secret sharing in Section 3. The first evolving protocol with hierarchical access structure is proposed in Section 4. The second proposed scheme, which improve the first one with some trade-offs, is proposed in Section 5. We compare our schemes to the work with similar functionality [9] in Section 6. Finally, Section 7 concludes the papers.

2 Preliminaries

In this section, we review the definitions of secret sharing, homomorphic secret sharing, and evolving secret sharing. We then review two cryptographic primitives, including collision-resistant hash function and pseudo-random function.

2.1 Secret Sharing

Our setting includes n input clients, t computing servers, and one output client (see Fig. 1). For now, t will be fixed. In the next subsection, t can be “evolving” or increased during the protocol.

For the i-th input client, who has secret input x_i, a pack of t shares will be generated as $s_{i,1}, \dots, s_{i,t}$, and the share $s_{i,j}$ is forwarded to the j-th computing server. To reconstruct the secret, the authorized subset of servers will send the given shares to the output client. The output client then reconstructs the desired value from these shares. We refer to the definitions on access structure and secret sharing from [20]. Let $\mathcal{P} = \{1, \dots, t\} = [t]$ be the set of t computing servers.

Definition 1. *An access structure $\mathcal{A} \subseteq 2^{\mathcal{P}}$ contains all subsets of computing servers that can reconstruct the secret. The set $\mathcal{A}$ must be monotone, i.e., if $A \in \mathcal{A}$ and $A \subseteq A' \subseteq \mathcal{P}$, then $A' \in \mathcal{A}$. Subsets in $\mathcal{A}$ are called authorized, while subsets not in $\mathcal{A}$ are called unauthorized.*

Definition 2. *Secret sharing scheme for an access structure $\mathcal{A}$ consists of two probabilistic algorithms* Share *and* Recon*. The properties are:*

1. ***Secret Sharing.*** *As in the earlier description, the i-th input client uses* $\mathsf{Share}(x_i) = (s_{i,1}, \ldots, s_{i,t})$ *to randomly generate shares of x_i. The share $s_{i,j}$ is given to the j-th computing server.*
2. ***Correctness.*** *For every authorized set $A \in \mathcal{A}$ and every secret x_i in the domain, we have* $Pr[\mathsf{Recon}((s_{i,j})_{j\in A}) = x_i] = 1$.
3. ***Security.*** *Consider the following game.*
   - *The adversary chooses two different secrets $x_i^{(0)}$ and $x_i^{(1)}$, and sends to the challenger.*
   - *The challenger randomly chooses $b \in \{0,1\}$, and generates shares from* $\mathsf{Share}(x_i^{(b)}) = (s_{i,1}^{(b)}, \ldots, s_{i,t}^{(b)})$.
   - *The adversary chooses an unauthorized subset $B \in 2^{\mathcal{P}} \setminus \mathcal{A}$, and sends to the challenger.*
   - *The adversary receives $(s_{i,j}^{(b)})_{j\in B}$, and outputs b'.*

   We say that the scheme is secure if $b' = b$ with probability $\frac{1}{2}$.

There exists several well-known secret sharing schemes. Here, we introduce the work of Shamir [25] and Tassa [26]. Shamir's secret sharing supports threshold access structures. The definition and the scheme are as follows.

Definition 3. *The access structure $\mathcal{A} = \{A \in 2^{\mathcal{P}} : |A| > k\}$ of $\mathcal{P} = [t]$ is called (k,t)-threshold access structure.*

Let the desired access structure be (k,t)-threshold. To share the secret input x_i using the Shamir's scheme, the i-th input client generates degree-k polynomial P_i over a prime field with size larger than t such that $P_i(0) = x_i$. Then, $s_{i,j} = P_i(j)$ is distributed to the j-th computing server. From Lagrange interpolation, secrets can be reconstructed by using a system of linear equations $s_{i,j} = P_i(j)$ where j is the server in the authorized set $A \in \mathcal{A}$.

Tassa's secret sharing supports hierarchical access structures. (In this paper, we only focus on disjunctive type.) The definition and the scheme are as follows.

Definition 4. *In disjunctive hierarchical access structure, the computing servers are divided into disjoint partitions $\mathcal{P}_1 \cup \ldots \cup \mathcal{P}_\ell = \mathcal{P} = [t]$. Let $k_1 \leq \ldots \leq k_\ell$ be threshold for each hierarchical level. The $(k_1, \ldots, k_\ell, \mathcal{P}_1, \ldots, \mathcal{P}_\ell)$-hierarchical access structure is defined as*

$$\mathcal{A} = \left\{ A \in 2^{\mathcal{P}} : \exists j \in [\ell], \;\; \left| A \cap \bigcup_{m=1}^{j} \mathcal{P}_m \right| > k_j \right\}.$$

In other words, the hierarchical access structure can be viewed as a disjunction of several threshold structures, namely, $(k_1, |\mathcal{P}_1|)$-threshold, $(k_2, |\mathcal{P}_1 \cup \mathcal{P}_2|)$-threshold, ..., and $(k_\ell, |\bigcup_{m=1}^{\ell} \mathcal{P}_m|)$-threshold.

Let $\mathcal{P} = [t]$, and the desired access structure be $(k_1, \ldots, k_\ell, \mathcal{P}_1, \ldots, \mathcal{P}_\ell)$-hierarchical. To share the secret input x_i, the i-th input client generates degree-k_ℓ

polynomial P_i over a prime field with size $p > t$ such that the coefficient of the term with the highest degree is x_i. Then, $s_{i,j} = P_i(j)$ is distributed to the j-th computing server in $\mathcal{P}_\ell$. For the j-th computing server in $\mathcal{P}_m$ where $m < \ell$, the share $s_{i,j} = P_i^{(k_\ell - k_m)}(j)$ is given, where $P_i^{(k_\ell - k_m)}$ is the $(k_\ell - k_m)$-th derivative of P_i. Note that $P_i^{(k_\ell - k_m)}$ has degree k_m, and the coefficient of the term with the highest degree includes x_i. From Birkhoff interpolation, secrets can be reconstructed by using a system of linear equations $s_{i,j} = P_i^{(k_\ell - k_m)}(j)$ where $j \in \mathcal{P}_m$ is the server in the authorized set. Although the equations may not have a unique solution in some settings, it is unique with overwhelming probability, as stated in the following theorem.

Proposition 1. [26], [18]. *For random allocation of participant identities, the above scheme from* [26] *realized the access structure in Definition 4 with probability at least* $1 - \varepsilon'$ *where*

$$\varepsilon' \leq \frac{\binom{t+1}{k_\ell+1} k_\ell (k_\ell - 1)}{2(p - k_\ell - 1)}.$$

2.2 Homomorphic Secret Sharing

We continue considering the same setting as in the previous subsection. In multi-party computation, we want to do more than reconstructing the secrets. The goal of multi-party computation is to let the output client learns the result of a function of secret inputs. Unauthorized subsets of servers must not learn the secret inputs or the results. Multi-party protocols can be constructed from garbled circuits [28], homomorphic encryption [11], secret sharing [13], etc.

In this paper, we focus on multi-party computation protocols that are based on homomorphic secret sharing. Each server can locally calculate some functions of the shares, but communication with other servers is not allowed. We refer to the definition of homomorphic secret sharing from [7] as follows.

Definition 5. *A degree-d homomorphic secret sharing is a secret sharing scheme with one additional algorithm* Eval*. The property for* Eval *is that, for every degree-d polynomial f of secret inputs, every secret inputs $x_1, \ldots, x_n$, and some authorized subset $A \in \mathcal{A}' \subseteq \mathcal{A}$, the j-th computing server which $j \in A$ can locally compute* $y_j = \mathsf{Eval}(A, f, j, (s_{i,j})_{i \in [n]})$ *such that* $\mathsf{Recon}((y_j)_{j \in A}) = f(x_1, \ldots, x_n)$.

In addition to the the previous subsection, Shamir's scheme has the following homomorphic property.

Proposition 2. [3]. *Shamir's scheme with (k,t)-threshold is degree-d homomorphic when $\mathcal{A}' = \{A \in 2^{\mathcal{P}} : |A| > d \cdot k\}$. The value $f(x_1, \ldots, x_n)$ where f is a degree-d polynomial can be reconstructed from $f(s_{1,j}, \ldots, s_{n,j})$ for the j-th server in an authorized set with at least $d \cdot k + 1$ servers.*

It is also proved in [18] that Tassa's scheme is 2-multiplicative for some specified settings, i.e., value $f(x_i, x_{i'}) = x_i x_{i'}$ can be reconstructed from $s_{i,j} s_{i',j}$

for the j-th server in some authorized subsets. The scheme can also be strongly 2-multiplicative in some stronger settings, i.e., value $f(x_i, x_{i'}) = x_i x_{i'}$ can be reconstructed from $s_{i,j} s_{i',j}$ where j comes from "any" authorized subsets.

2.3 Evolving Secret Sharing

In contrast to the previous subsections, evolving secret sharing allows infinite number of participated servers. We then have $\mathcal{P} = \mathbb{Z}^+$. The following definitions for evolving access structure and evolving secret sharing are from [20].

Definition 6. *An evolving access structure $\mathcal{A} \subseteq 2^{\mathcal{P}}$ is defined in the same way as Definition 1, except that $\mathcal{P}$ is infinite and $\mathcal{A}$ can be infinite. $\mathcal{A}_t = \mathcal{A} \cap 2^{[t]}$ is a finite access structure of the first t servers.*

Definition 7. *Let $\mathcal{A} = \{\mathcal{A}_t\}_{t \in \mathcal{P}}$ be an evolving access structure. An evolving secret sharing scheme for $\mathcal{A}$ contains two algorithms* Share *and* Recon *such that*

1. ***Secret Sharing.*** *To share the secret input x_i, the share $s_{i,t}$ randomly generated from $\mathsf{Share}(x_i, s_{i,1}, \ldots, s_{i,t-1})$ is given to the t-th computing server when it arrives. This share cannot be modified later after it is given.*
2. ***Correctness.*** *For every secret input x_i and $t \in \mathcal{P}$, an authorized subset $A \in \mathcal{A}_t$ can reconstruct the secret. That is $Pr[\mathsf{Recon}((s_{i,j})_{j \in A}) = x_i] = 1$.*
3. ***Security.*** *Consider the following game.*
   - *The adversary chooses two different secrets $x_i^{(0)}$ and $x_i^{(1)}$, and sends to the challenger.*
   - *The challenger randomly chooses $b \in \{0, 1\}$, and generates $s_{i,j}^{(b)}$ from $x_i^{(b)}$ using* Share *algorithm.*
   - *The adversary chooses $t \in \mathcal{P}$ and an unauthorized subset $B \in 2^{[t]} \setminus \mathcal{A}_t$, and sends to the challenger.*
   - *The adversary receives $(s_{i,j}^{(b)})_{j \in B}$, and outputs b'.*

   We say that the scheme is secure if $b' = b$ with probability $\frac{1}{2}$.

2.4 Cryptographic Primitives

We refer to the definition of collision-resistant hash function from [5, Chapter 8].

Definition 8. *Let λ be the security parameter. A collision-resistant hash function $h : S_1 \to S_2$ is a function such that for all probabilistic polynomial time algorithm Λ, the following probability is negligible in λ.*

$$Pr[x_1, x_2 \leftarrow \Lambda(h); x_1 \neq x_2 : h(x_1) = h(x_2)]$$

In the real-world implementation, one normally uses the current standard hash function, namely, SHA-3 [23]. Heuristically, it can be said that outputs from SHA-3 look almost uniformly distributed [5, Chapter 8]. We will use this latter property to attain the random allocation, as required by Proposition 1 in our first construction.[3]

The other tool that we review is pseudo-random function [12].

Definition 9. *Let λ be the security parameter, and S_1, S_2, and S_3 be collections of sets indexed by λ. A pseudo-random function $g : S_1 \times S_2 \to S_3$ is a function such that for all probabilistic polynomial time algorithm Λ, the following probability is negligible in λ.*

$$| \, Pr[s \leftarrow S_1 : \Lambda(g(s,\cdot)) = 1] - Pr[\text{a random map } g \text{ from } S_2 \text{ to } S_3 : \Lambda(g(\cdot)) = 1] \, |$$

3 Evolving Homomorphic Secret Sharing

In this section, we propose the definition of evolving homomorphic secret sharing. We combine Definition 5 and Definition 7 as follows. We also allow the scheme to be almost perfectly correct and computationally secure.

Definition 10. *An evolving degree-d homomorphic secret sharing is an evolving secret sharing scheme with three algorithms* $\mathsf{Share}, \mathsf{Recon}$, *and* Eval. *Security parameter λ can be used as necessary. Properties of the three algorithms include:*

1. ***Secret Sharing.*** *This is the same as in Definition 7.*
2. ***Correctness.*** *$Pr[\mathsf{Recon}((s_{i,j})_{j \in A}) = x_i]$ equals to 1 for perfect correctness, or equals to $1 - \varepsilon$ for almost perfect correctness where ε is negligible in λ.*
3. ***Security.*** *Consider the following game.*
   - *The adversary chooses two different secrets $x_i^{(0)}$ and $x_i^{(1)}$, and sends to the challenger.*
   - *The challenger randomly chooses $b \in \{0, 1\}$, and generates $s_{i,j}^{(b)}$ from $x_i^{(b)}$ using* Share *algorithm.*
   - *The adversary chooses $t \in \mathcal{P}$ and an unauthorized subset $B \in 2^{[t]} \setminus \mathcal{A}_t$, and sends to the challenger.*
   - *The adversary receives $(s_{i,j}^{(b)})_{j \in B}$, and outputs b'.*

   We say that the scheme is perfectly secure if $b' = b$ with probability $\frac{1}{2}$, and is computationally secure if $b' = b$ with probability $\frac{1}{2} + \varepsilon$, where the advantage ε is negligible in λ.
4. ***Homomorphism.*** *For every degree-d polynomial f, every secret inputs $x_1, \ldots, x_n$, every $t \in \mathcal{P}$, and some authorized subset of servers $A \in \mathcal{A}'_t \subseteq \mathcal{A}_t$, the j-th computing server which $j \in A$ can locally compute $y_j = \mathsf{Eval}(A, f, j, (s_{i,j})_{i \in [n]})$ such that $\mathsf{Recon}((y_j)_{j \in A}) = f(x_1, \ldots, x_n)$.*

This definition will be applied for our first scheme in Section 4 and our second scheme in Section 5.

[3] We could also go all the way by using the random oracle model. However, the random oracle model usually allows us to do more: a reduction algorithm can simulate an output of any queried input to the hash function. We do not use this property, and hence do not directly assume the random oracle.

4 Our Scheme 1: From Hierarchical Secret Sharing

In the first proposed scheme, we combine the hierarchical secret sharing from [26] and [18] with a collision-resistant hash function. The purpose of hash function here is similar to the warm-up scheme in Section 1.2.

4.1 Access Structure

The evolving disjunctive hierarchical access structure is similar to the disjunctive hierarchical access structure in Definition 4. In addition, the computing servers may be infinite, and the partition of servers may also be infinite.

Definition 11. *In evolving disjunctive hierarchical access structure, the computing servers are divided into disjoint partitions* $(\mathcal{P}_m)_{m\in\mathbb{Z}^+}$ *such that* $\bigcup_{m\in\mathbb{Z}^+}\mathcal{P}_m = \mathcal{P} = \mathbb{Z}^+$. *Let* $(k_m)_{m\in\mathbb{Z}^+}$ *be threshold for each hierarchical level where* $k_m \leq k_{m+1}$ *for all* $m \in \mathbb{Z}^+$. *The evolving* $((k_m)_{m\in\mathbb{Z}^+}, (\mathcal{P}_m)_{m\in\mathbb{Z}^+})$*-hierarchical access structure is defined as*

$$\mathcal{A} = \left\{ A \in 2^{\mathcal{P}} : \exists j \in \mathcal{P}, \ \left| A \cap \bigcup_{m=1}^{j} \mathcal{P}_m \right| > k_j \right\}.$$

Note that the number of partitions and number of thresholds are unbounded. This evolving disjunctive hierarchical access structure is equivalent to the dynamic threshold structure in [21]. We have dynamic threshold $f(t) = k_m$ for $t \in \mathcal{P}_m$. Here, the computing servers in $\mathcal{P}_m$ must come before those in $\mathcal{P}_{m+1}$.

4.2 Construction

Secret sharing. We propose the first scheme based on the idea of hierarchical secret sharing. In [26], the work realized the access structure by using the idea of derivatives. Our work will use the idea of integrals, since the number of servers can not be determined in advance. In this way, our scheme can support the evolving setting while preserving the properties of [18].

1. All input clients agree on a collision-resistant hash function h with uniformly distributed output. The domain and range of h are $\{0,1\}^*$ and $\mathbb{Z}_p\backslash\{0\}$ where p (will be specified later) is a prime number greater than 2^λ, and λ is the security parameter. We also assume that each t-th computing server has a unique random identity, ID_t, over $\{0,1\}^*$.[4]
2. To share the secret input x_i to the computing servers in $\mathcal{P}_1$, the i-th input client generates a degree-k_1 polynomial $P_{i,1}(\chi) = \sum_{j=0}^{k_1} a_j \chi^j$ over $\mathbb{Z}_p$ with random coefficients a_j, and the coefficient $a_{k_1} = x_i$. When the t-th computing server in $\mathcal{P}_1$ arrives, it gets its share as $s_{i,t} = P_{i,1}(h(ID_t))$. It can be seen that the hash function h is used in order to map from infinite set of IDs $(\{0,1\}^*)$ to the finite prime field $\mathbb{Z}_p$.

[4] A public bulletin board can be used for keeping the record and checking the uniqueness of all IDs.

3. For $m \geq 2$, when the first server in $\mathcal{P}_m$ arrives, the input client generates a degree-k_m polynomial

$$P_{i,m}(\chi) = P_{i,m-1}^{[k_m - k_{m-1}]}(\chi) + \sum_{j=0}^{k_m - k_{(m-1)} - 1} a_j \chi^j$$

with random coefficients a_j, where the term $P_{i,m-1}^{[k_m - k_{m-1}]}(\chi)$ is defined as the $(k_m - k_{m-1})$-th integral of $P_{i,m-1}(\chi)$. For $t \in \mathcal{P}_m$, when the t-th computing server arrives, it gets its share as $s_{i,t} = P_{i,m}(h(ID_t))$.

Reconstruction. Assume that a subset $A \in \mathcal{A}_t$ for some t is going to reconstruct the secret. The linear system described in Section 2.1 can be constructed from the equations $s_{i,t} = P_{i,m}(h(ID_t))$ for all $t \in A$. According to Proposition 1, the secret can be reconstructed by using Birkhoff interpolation. Recall that the coefficient of the term with the highest degree of $P_{i,m}(\chi)$ is a_{k_m}. Then, the secret is $a_{k_m} \times (k_m!/k_1!)$.

Evaluation. From the multiplicative property shown in [18], to calculate the multiplication of two secret inputs x_i and $x_{i'}$, each j-th computing server can locally compute its partial result $y_j = s_{i,j} s_{i',j}$. Suppose that we share the secrets using polynomials $P_{i,m}(\chi)$ and $P_{i',m}(\chi)$, and the coefficients of the terms with the highest degree are a_{k_m} and a'_{k_m}. We will obtain $a_{k_m} a'_{k_m}$ from the reconstruction. To obtain the multiplication result, which is $(a_{k_m} k_m!/k_1!)(a'_{k_m} k_m!/k_1!)$, we multiply the value from the reconstruction with $(k_m!/k_1!)^2$.

Example 1. Assume that the thresholds are $k_1 = 1$, $k_2 = 2$, and $k_3 = 3$. (The setting for other levels are omitted.) To share a value 12, the i-th input client randomly chooses a polynomial $P_{i,1}(\chi) = 12\chi + 2$, and uses it to generate shares for servers in the first level.

To generate shares for other levels, we calculate the integration result of $P_{i,1}(\chi)$ as $6\chi^2 + 2\chi$. We add the result with a random constant to obtain $P_{i,2}(\chi)$. Suppose that the constant is 3. We then have $P_{i,2}(\chi) = 6\chi^2 + 2\chi + 3$. Similarly, we have $P_{i,3}(\chi) = 2\chi^3 + \chi^2 + 3\chi + 4$.

In reconstruction process, after recovering $P_{i,3}(\chi)$ from Birkhoff interpolation, we know that the coefficient of the term with the highest degree (a_3) is 2. The secret is then $a_3 \times (k_3!/k_1!) = 12$.

Compare to [26], there exists a scheme using the same sequence of polynomials. To share a value 2, the input client randomly chooses $P_{i,3}(\chi) = 2\chi^3 + \chi^2 + 3\chi + 4$ for the third level. From derivatives, the polynomials for the second and the first levels are $P_{i,2}(\chi) = 6\chi^2 + 2\chi + 3$ and $P_{i,1}(\chi) = 12\chi + 2$. Thus, the correctness from [26] and [18] can be applied to ours.

4.3 Properties

In this subsection, we are interested in correctness, security, and share size of the scheme. We summarize the properties of the first scheme in the following theorem, and give a brief explanation.

Theorem 1. *If $k_m \leq poly(\lambda)$, there exists p with $poly(\lambda)$ bits such that the evolving homomorphic secret sharing scheme over $\mathbb{Z}_p$ proposed in Section 4.2 is almost perfectly correct and perfectly secure.*

Correctness. From the construction in Section 4.2, the sequence of polynomials $(P_{i,1}(\chi), P_{i,2}(\chi), \ldots, P_{i,m}(\chi))$ from integrals is the same as $(P_{i,m}(\chi), \ldots, P_{i,m}^{(k_m-k_2)}(\chi), P_{i,m}^{(k_m-k_1)}(\chi))$ from derivatives, but the order is reversed. Using the correctness of [26] and [18], the combinations according to the access structure $\mathcal{A}$ in Definition 11 can reconstruct the secret by using Birkhoff interpolation.

Although we have k_m+1 servers from the first m levels, we cannot reconstruct the polynomial only when 1) the solution of Birkhoff interpolation is not unique, or 2) some of the parties holds the same point in the polynomial.

Since ID_t is random and the output of h is uniformly distributed, Proposition 1 can be applied. The probability that the interpolation solution is not unique is no more than ε'. And because h is collision-resistant, $h(ID_t)$ and $h(ID_{t'})$ from two servers are equal with negligible probability ε''. The probability that there is at least one collision in $k_m + 1$ servers is at most $\binom{k_m+1}{2}\varepsilon''$. Therefore, the probability that we cannot have the polynomial is no more than $\varepsilon' + \binom{k_m+1}{2}\varepsilon''$.

There is p with $poly(\lambda)$ bits that makes ε' in Proposition 1 negligible. Also, if we assume that $k_m \leq poly(\lambda)$, the value of $\binom{k_m+1}{2}\varepsilon''$ will be also negligible.[5] Thus, the probability that we cannot have the polynomial is negligible.

From [18], if there exists $m \in \mathcal{P}$ such that $|\mathcal{P}_m| > 2k_m$, then the scheme is 2-multiplicative, i.e., all the servers together can calculate multiplication of two inputs. If $|\mathcal{P}_m| > 3k_m$ for some m, then the scheme is strongly 2-multiplicative, any authorized subsets can calculate multiplication of two inputs. (See Section 7 for more discussion.) The scheme that can calculate more general functions will be proposed in the next section.

Security. From Proposition 1, our scheme realizes evolving disjunctive hierarchical access structure, and is perfectly secure with overwhelming probability. We know from [26] and [18] that, even when the adversary knows exactly k_j pieces of different $P_{i,m}(h(ID_t))$ for any $m \leq j$, they cannot recover the secrets from those values. When the adversary can collect shares from at most k_j servers from $\bigcup_{m=1}^{j} \mathcal{P}_m$, they will know at most k_j pieces of different $P_{i,m}(h(ID_t))$. Therefore, they cannot recover the secrets from those values.

Share size. From the description, the t-th computing server in $\mathcal{P}_m$ will get only one share $s_{i,t} = P_{i,m}(h(ID_t))$ for each secret input x_i, which is an element in $\mathbb{Z}_p$. In this scheme, we can trade-off the share size with correctness. If we increase λ, the share size will be larger, but the probability that the hashed values will be collided is reduced.

[5] It is important to note that the correctness of our protocol does not depend on the number of all parties, but the minimum number of parties involving in the reconstruction, denoted by k_m. Therefore, although k_m must be polynomial of the security parameter, our protocol can support infinite number of parties.

5 Our Scheme 2: Multi-generation of Shamir's Scheme

From the previous section, it can be seen that the first proposed scheme has negligible error probability from the hash function, so it is not perfectly correct. The computable function is also limited. We will address these issues in this section. Here, we combine the Shamir's secret sharing [25] with pseudo-random functions, and allow the computing servers to store more than one shares. A variant of this scheme is also proposed.

5.1 Access Structure

We describe the partially hierarchical structure as follows. This structure is similar to, but less general than the hierarchical one introduced in the previous section. Parameters for the access structure are $(k_m)_{m\in\mathbb{N}}$ where each k_m is a non-negative integer. The authorized sets that can reconstruct the secret are the combination of $k_m + 1$ servers from the first k_{m+2} servers, or (k_m, k_{m+2})-threshold, for any $m \in \mathbb{N}$. The authorized sets that can compute degree-d polynomials are the combination of $d \cdot k_m + 1$ servers from the first k_{m+2} servers, or $(d \cdot k_m, k_{m+2})$-threshold, for any $m \in \mathbb{N}$. We call the set of all servers $1 \le j \le k_1$ as the 1st generation, and the set of all servers $k_{m-1} < j \le k_m$ as the m-th generation for $m \ge 2$. Note that the number of thresholds here is also unbounded.

5.2 Construction

Secret sharing. In this scheme, we use pseudo-random functions with security parameter λ. The usage is different from hash functions in the previous section.

1. When the first server arrives, the i-th input client generates a degree-k_0 polynomial $P_{i,1}$ of prime field $p_1 > k_2$, where $P_{i,1}(0) = x_i$.
2. For $1 \le t \le k_2$, the share $s_{i,t}$, which includes $P_{i,1}(t)$ and a random bit string $r_{i,t}$ of size λ, is given to the t-th computing server in the 1st and 2nd generations when it arrives. Note that the random value $r_{i,t}$ can be distributed in the setup phase before the value of x_i is known.
3. For all $j \ge 2$, when the $(k_{j-1}+1)$-th server (which is the first server of the j-th generation) arrives, the input clients agree on a pseudo-random function $g_j : \{0,1\}^\lambda \times \mathbb{Z}_{p_{j-1}} \to \mathbb{Z}_{p_j}$, where $p_j > k_{j+1}$ is a prime number. Then, generate a degree-k_{j-1} polynomial $P_{i,j}$ of prime field p_j, where $P_{i,j}(0) = x_i$ and $P_{i,j}(t) = g_j(r_{i,t}, P_{i,j-1}(t))$ for all $1 \le t \le k_{j-1}$.
4. For all $j \ge 2$ and $k_{j-1} + 1 \le t \le k_{j+1}$, the share $s_{i,t}$ which includes $P_{i,j}(t)$ and a random bit string $r_{i,t}$ of size λ is given to each server in the j-th and the $(j+1)$-th generations when it arrives. Intuitively, the pseudo-random functions in this second scheme are used to maintain the consistency of the shares from different prime fields.

Table 3 summarizes the share values related to the i-th input client. In addition to $r_{i,t}$, each server will get only bold values in the corresponding row which

Table 3. Values need for reconstruction in the second scheme.

Polynomial	$P_{i,1}$	$P_{i,2}$	$P_{i,3}$	$P_{i,4}$	...
Prime field	$p_1 > k_2$	$p_2 > k_3$	$p_3 > k_4$	$p_4 > k_5$	...
Degree	k_0	k_1	k_2	k_3	...
Threshold	(dk_0, k_2)	(dk_1, k_3)	(dk_2, k_4)	(dk_3, k_5)	...
Gen. 1: Server 1 to k_1	$\boldsymbol{P_{i,1}(t)}$	$g_2(r_{i,t}, P_{i,1}(t))$	$g_3(r_{i,t}, P_{i,2}(t))$	$g_4(r_{i,t}, P_{i,3}(t))$	...
Gen. 2: Server k_1+1 to k_2	$\boldsymbol{P_{i,1}(t)}$	$\boldsymbol{P_{i,2}(t)}$	$g_3(r_{i,t}, P_{i,2}(t))$	$g_4(r_{i,t}, P_{i,3}(t))$	...
Gen. 3: Server k_2+1 to k_3		$\boldsymbol{P_{i,2}(t)}$	$\boldsymbol{P_{i,3}(t)}$	$g_4(r_{i,t}, P_{i,3}(t))$	...
Gen. 4: Server k_3+1 to k_4			$\boldsymbol{P_{i,3}(t)}$	$\boldsymbol{P_{i,4}(t)}$	...
... ...				...	...

is at most two elements per secret input. Other values that are not given can be generated from $r_{i,t}$, $P_{i,j}(t)$, and the pseudo-random functions.

Reconstruction. According to Shamir's scheme [25], if the subset satisfies the access structure, Lagrange interpolation can be used to reconstruct the corresponding polynomial. The values in Table 3 will be used in the reconstruction process corresponding to the specified polynomial. For example, suppose $(k_0, k_1, k_2, k_3) = (1, 2, 5, 9)$. When we have information from server 1, 6, and 7, we can reconstruct $P_{i,2}$ using $P_{i,2}(1) = g_2(r_{i,1}, P_{i,1}(1))$, $P_{i,2}(6)$, and $P_{i,2}(7)$.

Evaluation. From the property of Shamir's scheme, to calculate a degree-d polynomial f of the secret inputs, each computing server can locally compute f on its given shares corresponding to the satisfied threshold level (see Table 3).

5.3 Properties

Similar to the previous section, the properties of the second scheme are summarized in the following theorem with a brief explanation.

Theorem 2. *The evolving homomorphic secret sharing scheme proposed in Section 5.2 is perfectly correct and computationally secure.*

Correctness. Each polynomial $P_{i,j}$ can be uniquely generated from pseudo-random value of the first $j-1$ generations. According to Proposition 2, a set of computing servers satisfied the conditions in the defined access structure will be able to compute degree-d polynomials. The Eval algorithm can be performed by calculating the function f on the corresponding share of each secret input.

The correctness of scheme 2 should not be confusing with scheme 1. In scheme 1, the "input" of the polynomial may be collided from the use of hash function. Thus, the scheme is almost perfectly correct. However, in scheme 2, the "output" of the polynomial may be collided from the use of pseudo-random function, but this does not affect the perfect correctness of the scheme. That is because in the original Shamir's scheme, shares for different servers can have the same value.

Security. We prove the security of the scheme with a sequence of games. It starts from the first setting with values from pseudo-random functions, and ends with the final setting with totally random values.

Game 0. This game is based on the exact construction in Section 5.2. Assume that the adversary collects at most k_j shares for all level j, and $q \leq poly(\lambda)$ of them are shares from pseudo-random functions. The adversary tries to distinguish between shares of any two secrets.

Game 1. The setting is same as Game 0 except that one share from pseudo-random function is changed to random value.

We continue changing one pseudo-random share to random value for each game. In the final game, *Game q*, all shares are totally random values.

The security of the final game follows the security of the Shamir's scheme. The advantage of the adversary to distinguish the shares is $\varepsilon_q = 0$. Let us consider the following lemma.

Lemma 1. *Assume that there is an adversary with advantages ε_g and ε_{g+1} in Game g and Game $g+1$, respectively. Then, we can construct an adversary against pseudo-random function with advantage $Adv_{PRF} = \frac{1}{2}(\varepsilon_g - \varepsilon_{g+1})$.*

Proof. We define Game g based on the secret sharing setting with the usage of $q - g$ pseudo-random functions. The difference between Game g and Game $g+1$ is only at the g-th part of the share; the former comes from pseudo-random function while the latter comes from random function. The other parts of the shares are exactly the same. Assume that we have an adversary Ψ such that the advantage to distinguish shares in Game g and Game $g+1$ are ε_g and ε_{g+1}, respectively. We will construct an adversary Φ against the security of pseudo-random function as follows.

The challenger flips coin $a \in \{0, 1\}$ which represents pseudo-random and random function, respectively. Adversary Φ wins if it can make a guess a' equals to a. Adversary Φ firstly let the adversary Ψ generate two secrets x_0 and x_1. Next, Φ flips coin $b \in \{0, 1\}$, and generates shares of x_b according to the scheme in Section 5.2, using one query to the challenger and its own $q - g - 1$ pseudo-random functions. Φ then forwards the shares to Ψ. After receiving b' in return from Ψ, if $b' = b$, Φ guesses $a' = 0$, and guesses $a' = 1$ otherwise.

If the challenger has $a = 0$, the setting is Game g, which Ψ has advantage of ε_g. On the other hand, if the challenger has $a = 1$, the setting is Game $g+1$, which Ψ has advantage of ε_{g+1}. Hence by the definition, we have

$$
\begin{aligned}
Pr[b' = b \mid a = 0] - \frac{1}{2} &= \varepsilon_g \\
Pr[b' = b \mid a = 1] - \frac{1}{2} &= \varepsilon_{g+1}.
\end{aligned}
$$

Let the advantage to break pseudo-random function is Adv_{PRF}. In the other words, $Pr[a' = a] - \frac{1}{2} = Adv_{PRF}$. And from the explanation above, $Pr[a' = 0] = Pr[b' = b]$ and $Pr[a' = 1] = 1 - Pr[b' = b]$. We will show that $\varepsilon_g - \varepsilon_{g+1} =$

$2Adv_{PRF}$. Consider the probability that $a' = a$ as follows.

$$\begin{aligned}
Pr[a' = a] &= Pr[a' = 0 \mid a = 0]Pr[a = 0] \\
&\quad + Pr[a' = 1 \mid a = 1]Pr[a = 1] \\
&= (\frac{1}{2} + \varepsilon_g)(\frac{1}{2}) + (\frac{1}{2} - \varepsilon_{g+1})(\frac{1}{2}) \\
&= \frac{1}{2} + \frac{1}{2}\varepsilon_g - \frac{1}{2}\varepsilon_{g+1}
\end{aligned}$$

Substitute this probability to the advantage of pseudo-random function, we have $Adv_{PRF} = \frac{1}{2}(\varepsilon_g - \varepsilon_{g+1})$. □

Put everything together, the advantage to break our scheme is $\varepsilon_0 = \varepsilon_0 - \varepsilon_q = (\varepsilon_0 - \varepsilon_1) + \cdots + (\varepsilon_{q-1} - \varepsilon_q) = 2qAdv_{PRF}$ which is negligible if Adv_{PRF} is negligible.

Share size. Since the random values $r_{i,t}$ are not related to the secrets, these values can be distributed in the setup phase. The communication complexity here is $O(\lambda)$ for each pair of input client and server. After the secret inputs are determined in the online phase, the t-th computing server will get at most two shares per secret input. The share size depends on the value k_j in the access structure. One possible way is to choose $k_j \approx d^{(j+1)/2}$. If $t \approx d^j$, it will receive two field elements, where the size of the field is at most d^{j+1}. The share size is then approximately $2(j+1)\log d$. Thus, the share size of the t-th server in the online phase is approximately $O(\log t)$.

5.4 Variant of the Scheme

The scheme in this section can be generalized so that each server receives at most α shares, where α is a positive integer. In this case, the combinations of computing servers that can compute degree-d polynomials are $(d \cdot k_m, k_{m+\alpha})$-threshold, for any $m \in \mathbb{N}$. Compare to Table 3, the parts with pseudo-random functions are the same, but more cells of bold shares will be added. It can be seen that these combinations are more generalized than the scheme in Section 5.2 when α is increased. This is a trade-off between the generality of the access structure and the share size.

6 Comparison to a Recent Scheme

In this section, we briefly compare communication costs of our schemes to Fluid MPC [9]. Note that the cost of our works is a result from Definition 10, and does not depend on the construction.

Assume that there are n input clients. For computing servers, at first we use m_1 servers. Later, we increase the number of servers to m_2, and then $m_3, \ldots, m_\ell$. In [9], there are nm_1 messages sent from n input clients to the first set of m_1 computing servers. Since resharing between servers is required, $m_i m_{i+1}$ messages are sent from the i-th set of servers to the $(i+1)$-th set. Thus, the total number of messages sent is $nm_1 + m_1 m_2 + \cdots + m_{\ell-1} m_\ell$. In contrast, our schemes do not use resharing, shares are only sent from the input clients to newly added servers. With some restrictions on computing functions, our schemes only require $nm_1 + n(m_2 - m_1) + \cdots + n(m_\ell - m_{\ell-1}) = nm_\ell$ communications. Furthermore, servers which already received shares do not have to be online when new servers are added.

7 Concluding Remarks

In this paper, we propose two evolving homomorphic secret sharing schemes. By relaxing the conditions to be almost perfectly correct or computationally secure, our schemes are simpler than the existing ones. Users can choose the appropriate schemes and trade-off between several parameters. We suggest some interesting issues that are left for future studies.

For the first scheme, the number of shares for each computing server is small, but the share size may be large, since the prime p has to be large. If we can increase the size of the prime field later during the protocol (similar to the second scheme), then the share size can be reduced. In order to do this, we may integrate the polynomial to a different prime field, and then solve multi-variable Chinese remainder theorem, which is studied in [19], instead of simple linear system. However, the multi-variable CRT is not thoroughly understood.

As the other issue, the paper [18] mentioned the access structure of type Q_d (union of any d sets in the access structure cannot cover all parties), but not the multiplicativity of d secret inputs when $d > 2$. This issue should be further investigated. The other work [27] can perform unlimited number of multiplications by using precomputed multiplicative triples. This requires some interactions between computing servers.

For the second scheme, the appropriate value of threshold k_m for all $m \in \mathbb{N}$ should be suggested, but these values may depend on the applications. We may try to extend the idea of this construction to more general class of access structures.

Acknowledgments. Nuttapong Attrapadung was partly supported by JST CREST Grant Number JPMJCR19F6, and by JSPS KAKENHI Kiban-A Grant Number 19H01109. Kanta Matsuura was partially supported by JSPS KAKENHI Grant Number 17KT0081.

References

1. Beimel, A., Othman, H.: Evolving ramp secret-sharing schemes. In: International Conference on Security and Cryptography for Networks. pp. 313–332 (2018)
2. Beimel, A., Othman, H.: Evolving ramp secret sharing with a small gap. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 529–555 (2020)

3. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. pp. 1–10 (1988)
4. Benhamouda, F., Gentry, C., Gorbunov, S., Halevi, S., Krawczyk, H., Lin, C., Rabin, T., Reyzin, L.: Can a public blockchain keep a secret? In: Theory of Cryptography Conference. pp. 260–290. Springer (2020)
5. Boneh, D., Shoup, V.: A graduate course in applied cryptography (2020), https://toc.cryptobook.us/book.pdf
6. 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)
7. Boyle, E., Gilboa, N., Ishai, Y., Lin, H., Tessaro, S.: Foundations of homomorphic secret sharing. In: 9th Innovations in Theoretical Computer Science Conference (2018)
8. Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without fhe. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 3–33. Springer (2019)
9. Choudhuri, A.R., Goel, A., Green, M., Jain, A., Kaptchuk, G.: Fluid MPC: Secure multiparty computation with dynamic participants. IACR Cryptol. ePrint Arch **754**, 2020 (2020)
10. Feige, U., Killian, J., Naor, M.: A minimal model for secure computation. In: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing. pp. 554–563 (1994)
11. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing. pp. 169–178 (2009)
12. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM (JACM) **33**(4), 792–807 (1986)
13. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game, or a completeness theorem for protocols with honest majority. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. pp. 218–229 (1987)
14. Goyal, V., Kothapalli, A., Masserova, E., Parno, B., Song, Y.: Storing and retrieving secrets on a blockchain. IACR Cryptol. ePrint Arch. **2020**, 504 (2020)
15. Halevi, S., Lindell, Y., Pinkas, B.: Secure computation on the web: Computing without simultaneous interaction. In: Annual Cryptology Conference. pp. 132–150. Springer (2011)
16. Kamara, S., Mohassel, P., Raykova, M.: Outsourcing multi-party computation. IACR Cryptol. Eprint Arch. (2011)
17. Kamara, S., Mohassel, P., Riva, B.: Salus: a system for server-aided secure function evaluation. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 797–808 (2012)
18. Käsper, E., Nikov, V., Nikova, S.: Strongly multiplicative hierarchical threshold secret sharing. In: International Conference on Information Theoretic Security. pp. 148–168 (2007)
19. Knill, O.: A multivariable chinese remainder theorem. arXiv preprint arXiv:1206.5114 (2012)
20. Komargodski, I., Naor, M., Yogev, E.: How to share a secret, infinitely. In: Theory of Cryptography Conference. pp. 485–514 (2016)
21. Komargodski, I., Paskin-Cherniavsky, A.: Evolving secret sharing: Dynamic thresholds and robustness. In: Theory of Cryptography Conference. pp. 379–393 (2017)

22. Lai, R.W., 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. Springer (2018)
23. NIST: SHA-3 standard: Permutation-based hash and extendable-output functions. Federal Information Processing Standards Publication 202 (2015)
24. Phalakarn, K., Suppakitpaisarn, V., Attrapadung, N., Matsuura, K.: Constructive t-secure homomorphic secret sharing for low degree polynomials. In: International Conference on Cryptology in India. pp. 763–785. Springer (2020)
25. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
26. Tassa, T.: Hierarchical threshold secret sharing. In: Theory of Cryptography Conference. pp. 473–490 (2004)
27. Traverso, G., Demirel, D., Buchmann, J.: Performing computations on hierarchically shared secrets. In: International Conference on Cryptology in Africa. pp. 141–161 (2018)
28. Yao, A.C.: Protocols for secure computations. In: 23rd Annual Symposium on Foundations of Computer Science. pp. 160–164 (1982)