Continual Leakage-Resilient Dynamic Secret Sharing in the Split-State Model

Hao Xiong[1], Cong Zhang[1], Tsz Hon Yuen[1,⋆], Echo P. Zhang[1], Siu Ming Yiu[1], and Sihan Qing[2,⋆⋆]

[1] The University of Hong Kong, Hong Kong
{hxiong,czhang2,thyuen,pzhang2,smyiu}@cs.hku.hk
[2] Chinese Academy of Sciences and Peking University, China
qsihan@ss.pku.edu.cn

Abstract. Traditional secret sharing assume the absolute secrecy of the private shares of the uncorrupted users. It may not hold in the real world due to the side-channel attacks. Leakage-resilient cryptography is proposed to capture this situation. In the continual leakage model, the attacker can continuously leak the private value owned by the user with the constraint that the information leaked should be less than ℓ between updates. We propose continual leakage-resilient dynamic secret sharing under split-state model in this paper. After a preprocessing stage, the dealer is able to dynamically choose a set of n users and to allow a threshold of t users to reconstruct different secrets in different time instants, by using the same broadcast message. The secrets are protected even if an adversary corrupts up to $t-1$ users and obtains continual leakage from the rest of them. Our scheme can provide the security for secret sharing under continual leakage model while at the same time allowing the users to join and quit the scheme dynamically.

1 Introduction

Secret sharing is an important cryptographic primitive that a dealer shares a secret value between a group of users $\mathcal{P}$. Any set of t (threshold) users from $\mathcal{P}$ can jointly recover the secret value. However, the collusion of any $t-1$ or less users cannot obtain any information about the secret value. Secret sharing was firstly proposed by Shamir [13]. This type of secret sharing is sometimes referred as *threshold secret sharing*, in contrast with the secret sharing with general access structure. In this paper, we will focus on the threshold secret sharing. Secret sharing is useful in applications that require a collaboration like the management of cryptographic keys and secure multi-party computation.

Dynamic secret sharing has the feature that the dealer enables different sets of users (based on different access structures) to recover different secrets in different

⋆ Supported by the HKU Small Project Funding No. 201109176192.
⋆⋆ Supported by the National Natural Science Foundation of China Grant No. 60970135 and 61170282.

T.W. Chim and T.H. Yuen (Eds.): ICICS 2012, LNCS 7618, pp. 119–130, 2012.

time instants simply by sending the same broadcast message to all of them. It was firstly formalized by Blundo *et al.* [3]. In the sense of threshold secret sharing, the dealer can choose different sets of users to recover different secrets, but a threshold of them is sufficient to recover a secret. It is fully dynamic since anyone can join or leave the system at any time, and the authorized set (to recover the secret) can be chosen at the time of generating the broadcast message (instead of the user share generating phase). In practical application, consider the example of the board of directors of a company. A threshold of the directors can sign a contract on behalf of the company. However, the members of the board of directors may change from time to time. Furthermore, it is favorable that only the directors of the related departments is capable to sign on certain projects. Therefore, dynamic (threshold) secret sharing is useful in this kind of situation.

Traditional secret sharing schemes assume the absolute secrecy of the user shares. Like many cryptosystems, such assumption may not hold in the real world due to the side-channel attacks, such as the timing attack, power analysis, etc. These attacks capture partial information about the private user shares and their internal states through various physical attributes of a computation device. Therefore, traditional security model can no longer provide security guarantee under the side-channel attack. Hence, leakage-resilient cryptography is proposed to capture this scenario in recent years. The relative leakage model was firstly introduced by Akavia et al. [1] which allows the attacker to learn at most ℓ-bits information about the internal state of the system. Later on, the continual leakage model [9] describes that the secret key is updated periodically and the attacker can learn information continually, with the constraint that at most ℓ-bits leakage of internal state is allowed between update.

Our Contribution. Our main result is to construct a dynamic secret sharing scheme under the continual leakage model (CLM-DSS). In our scheme, the attacker can continually learn the share owned by each user with the constraint that the total information leaked is less than ℓ-bits between update. The random number used to update the share can also be leaked. Our scheme will guarantee the privacy of the secret value even up to $t-1$ parties are corrupted (where t is threshold) and information owned by other parties are partially leaked. Moreover, the user can dynamically join and leave the secret sharing scheme. For different secret values, we can set different authorized set and threshold for each of them.

We model the leak function in our security model as the split-state model in [7]. Davì *et al.* [7] assumed that the memory of a system can be divided into two parts, and each of them is accessed independently by different process and different time interval. Therefore, when there is side-channel attack on the memory, the attacker can only obtain leakage from one part only. Therefore in the security model, the leak oracle is modeled in a way that the leak function can only leak on one part of the memory at one time, but not leak on both parts simultaneously.

Our Techniques. Our construction is based on the dynamic threshold encryption by Delerablée and Pointcheval [8]. Our construction begins with generating

the master key and the public key as in [8]. Instead of sending a share directly to each user as in [8], the dealer encrypts it and sends the corresponding ciphertext and decryption key to the user. We use the Continual-Leakage-Resilient Sharing (CLRS)-friendly encryption scheme in [6] to provide the leakage-resilient property as well as the update protocol. Therefore, we can update the ciphertext and the secret key asynchronously to protect the secret value under the continual leakage model.

We have a few restrictions imposed on our security model due to the building blocks we used. Firstly, we use the split-state model for leakage since the ciphertext and the secret key of the CLRS-friendly encryption scheme is leaked independently in [6]. Secondly, we use the non-adaptive adversary and corruption model to handle the corruption of users which is inherited from the dynamic threshold encryption in [8]. Finally, we assume that the secret value to be shared and the master key of the dealer is leak-free, or else we can continuously leak on them and thus get the exact value of them. This is a common assumption in leakage-resilient cryptography, such as [5,11][1].

Related Work. It is useful to compare our scheme with other primitives from the literature. Firstly, standard secret sharing scheme [13] provides security when some subset of the shares are fully compromised which others are fully secure. Laih *et al.* [10] gave the first dynamic threshold secret sharing such that different secrets can be shared by different broadcast messages. Blakley *et al.* [2] introduced the user disenrollment to the dynamic threshold secret sharing. Their security models also assume full security of the uncorrupted users. In our security model, the shares that are not fully compromised can be partially leaked.

In leakage-resilient secret sharing, Boyle *et al.* [5] introduced a (non-dynamic) secret sharing which guarantees the privacy of the secret value only under the bounded leakage model; while we provide the security under the continual leakage model. The continual leakage-resilient secret sharing scheme for general access structure proposed by Dodis *et al.* [9] mainly focus on two parties and when it extends to more than two parties, the user cannot dynamically join and leave the scheme.

2 Security Model

Our goal is to give the definition of dynamic threshold secret sharing scheme under continual leakage model. We require that any user can dynamically join the secret sharing system. For each secret, the dealer can dynamically choose the share group $\mathcal{P}$ and the threshold value t. Finally, the secret share stored in each device may be continuously leaked by the attacker.

Our security is modified from the model of dynamic secret sharing in [3], by changing from monotone access structure to threshold access structure, and adding the update and leak oracle for the continual leakage model.

[1] There exist cryptosystems that allow leakage of the master secret key, such as identity-based encryption in [14].

2.1 Security Notion

A dynamic threshold secret sharing scheme is a tuple of algorithms $\mathsf{CLR\_DSS} = (\mathsf{Setup}, \mathsf{SharePreprocess}, \mathsf{Update}, \mathsf{MsgGen}, \mathsf{Reconstruct})$ as follows:

- $\mathsf{Setup}(1^\lambda)$: On input a security parameters 1^λ, it outputs $(\mathsf{MK}, \mathsf{PK})$, where MK is the master secret key of the dealer and PK is the corresponding public key. PK may include the maximal size of an authorized set m, and a list of participants $\mathtt{L}$.
- $\mathsf{SharePreprocess}(\mathsf{MK},\mathsf{PK},S,\mathcal{P})$: On input MK, PK, the space of possible secrets S and a set of participants $\mathcal{P} = (ID_1, \ldots, ID_n)$, it outputs the corresponding shares $a_1, \ldots, a_n$ for each participant, and may update $\mathtt{L}$.
- $\mathsf{MsgGen}(\mathsf{PK},s,t,\mathcal{P}',(a_1,\ldots,a_n))$: On input PK, a secret $s \in S$, a threshold t, a set of participants $\mathcal{P}'$ with corresponding shares $(a_1, \ldots, a_n)$, it outputs the broadcast message M.
- $\mathsf{Update}(\mathsf{PK}, a_i)$: On input PK, each user updates his share a_i to protect the privacy of the secret share against continual leakage from the attacker. It outputs the updated share a_i'.
  In this model, we also assume that the share a_i is split into two parts and they are updated and accessed independently. Denote them as sh_1 and sh_2, and define Update_b $(b = 1, 2)$ as follows (we omit the input PK for simplicity): $\mathsf{Update}_b(sh_b) \to sh_b'$: The randomized update algorithm takes the index b and the current version of the share sh_b and outputs an update version sh_b'. The notation $\mathsf{Update}_b^k(sh_b)$ denote the operation of updating the share sh_b successively k times in a row so that $\mathsf{Update}_b^0(sh_b) = sh_b$, $\mathsf{Update}_b^{k+1}(sh_b) = \mathsf{Update}_b(\mathsf{Update}_b^k(sh_b))$.
- $\mathsf{Reconstruct}(\mathsf{PK}, (a_1, \ldots, a_t), M)$: On input PK, the secret shares $(a_1, \ldots, a_t)$ and a broadcast message M, it outputs the secret value s.

2.2 Security Model

First we will define what information can be leaked and what the leakage function should be in the split-state model. As described before, user i will have a secret share a_i which is stored in two parts sh_1 and sh_2. The leak function f (which is an input to the Leak oracle) is a function of a_i and the random value used. We denote $state_b = (sh_b, R_b)$ for $b \in \{1, 2\}$, where R_b is the randomness used for sh_b. We can see that the attacker only leaks on the information on each user's share and randomness since other information are either leak-free or published. There is one restriction to the leak function f as specified in [7]. For each user i, the leak oracle can only be applied on either $state_1$ or $state_2$, but it cannot be applied on both of them at the same time.

Correctness: It consists of two parts:

- *Update Correctness:* Notice that the update of sh_1 and sh_2 can be asynchronous, which means that for any sequence of $i \geq 0$, $j \geq 0$, $sh_1' \leftarrow \mathsf{Update}_1^i(sh_1)$, $sh_2' \leftarrow \mathsf{Update}_2^j(sh_2)$, the updated shares (sh_1', sh_2') can be viewed as a valid share a used for reconstruction.

– *Reconstruct Correctness*: If $M \leftarrow \mathsf{MsgGen}(\mathsf{PK}, s, t, \mathcal{P}', (a_1, \ldots, a_n))$ and $(a'_1, \ldots, a'_t)$ is a (updated) subset of the shares $(a_1, \ldots, a_n)$, then $s \leftarrow \mathsf{Reconstruct}(\mathsf{PK}, (a'_1, \ldots, a'_t), M)$.

Privacy: For any secret s shared by a set $\mathcal{P}$ of registered users with a threshold t, any collusion that contains less than t users from this authorized set and the bounded leakage from the uncorrupted users cannot learn any information about the secret s.

We formally define the above privacy notion, under the classical semantic security notion and under various attacks, using a game between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$. Both the adversary and the challenger are given as input a security parameter 1^λ. The restriction we impose are:

– $\mathcal{A}$ cannot get any leakage about the master secret key and secret value.
– $\mathcal{A}$ has to decide the challenge set $\mathcal{P}^*$ of users, the challenge threshold t^*, and the identities set $\mathcal{I}$ that he will corrupt at the beginning of the game. This restriction is called non-adaptive adversary and corruption model(NAA-NAC). We restrict that $|\mathcal{P}^* \bigcap \mathcal{I}| \leq t^* - 1$.
– $\mathcal{A}$ cannot get leakage of more than ℓ bits between updates.

We now define our game $\mathsf{CorruptLeak}$ which consists of the following phases:

1. **Setup:** The adversary $\mathcal{A}$ sends to the challenger $\mathcal{C}$ the challenge set $\mathcal{P}^*$ of users, the challenge threshold t^*, and the identities set $\mathcal{I}$ that he will corrupt. $\mathcal{C}$ runs $\mathsf{Setup}(1^\lambda)$ to obtain the set of parameters $\mathsf{param} = (\mathsf{MK}, \mathsf{PK})$. The public key PK is given to $\mathcal{A}$. Denote the secret space as S. $\mathcal{C}$ stores an initially empty list $\mathcal{L}$ of the form $(ID, b, sh_b, \ell_b, rand_b)$, which stores a part of the share sh_b (b=1/2) for a user ID, the leaked bits, and the randomness used for the next update.
2. **Query Phase 1:** The adversary $\mathcal{A}$ adaptively issues queries:
   – **Join** query: on input an identity ID, $\mathcal{C}$ runs the $a = (sh_1, sh_2) \leftarrow \mathsf{SharePreprocess}(\mathsf{MK}, \mathsf{PK}, S, ID)$ to create a new user in the system. $\mathcal{C}$ chooses the randomness $rand_1$ and $rand_2$ used for the next update for sh_1 and sh_2 respectively. $\mathcal{C}$ stores $(ID, 1, sh_1, 0, rand_1)$ and $(ID, 2, sh_2, 0, rand_2)$ in the list $\mathcal{L}$.
   – **Corrupt** query: on input an identity $ID \in \mathcal{I}$, $\mathcal{C}$ retrieves $(ID, 1, sh_1, \cdot, \cdot)$ and $(ID, 2, sh_2, \cdot, \cdot) \in \mathcal{L}$ and returns $a = (sh_1, sh_2)$ to $\mathcal{A}$.
   – **Leak** query: on input an identity $ID \in \mathcal{I} \cup \mathcal{P}^*$, a leakage function $f : \{0,1\}^* \rightarrow \{0,1\}$ and $b = 1/2$, $\mathcal{C}$ retrieves $(ID, b, sh_b, \ell_b, rand_b) \in \mathcal{L}$. If $\ell_b \leq \ell$, then $\mathcal{C}$ responds with $f(sh_b, rand_b)$ and increases the counter $\ell_b = \ell_b + 1$ in the list $\mathcal{L}$. Otherwise returns $\perp$.
   – **Update** query: on input an identify ID and $b = 1/2$, $\mathcal{C}$ retrieves $(ID, b, sh_b, \ell_b, rand_b) \in \mathcal{L}$ and computes $sh'_b = \mathsf{Update}_b(sh_b; rand_b)$. It samples fresh random number $rand'_b$ and updates $(ID, b, sh'_b, 0, rand'_b) \in \mathcal{L}$.
3. **Challenge:** $\mathcal{A}$ submits a secret value $s^* \in S$ to $\mathcal{C}$. $\mathcal{C}$ sets $s^*_0 = s^*$ and s^*_1 to be a random number. $\mathcal{C}$ retrieves $(ID^*_j, b, sh^*_{j,b}, \cdot, \cdot) \in \mathcal{L}$ for all $ID_i \in \mathcal{P}^*$ where $b = 1/2$. Denote $a^*_j = (sh^*_{j,1}, sh^*_{j,2})$. $\mathcal{C}$ flips a uniform coins $b' \xleftarrow{\$} \{0,1\}$, generates $M^* = \mathsf{MsgGen}(\mathsf{MK}, s^*_{b'}, t^*, \mathcal{P}^*, \{a^*_j\})$ and sends it to $\mathcal{A}$.

4. **Query Phase 2:** $\mathcal{A}$ adaptively issues **Join, Corrupt** and **Leak** queries as in phase 1 , but the constraint is that the total number of identities $ID \in \mathcal{P}^*$ asked in **Corrupt** queries is less that $t^* - 1$ (which is $\mathcal{P}^* \cap \mathcal{I} \leq t^* - 1$) and the **Leak** queries in phase 2 cannot be executed in the target set of users $\mathcal{P}^* - \mathcal{P}^* \cap \mathcal{I}$.
5. **Guess:** Finally, $\mathcal{A}$ outputs a bit $b^* \leftarrow \{0, 1\}$ and wins the game if $b^* = b'$.

The advantage of $\mathcal{A}$ is defined as $Adv_{\mathcal{A}}(\lambda) = |\mathbf{Pr}[b^{'} = b] - \frac{1}{2}|$.

Definition 1. *A dynamic threshold secret sharing scheme is ℓ-continual leakage resilient if for all PPT adversaries $\mathcal{A}$, $Adv_{\mathcal{A}}(\lambda) \leq negl(\lambda)$.*

3 Notation and Preliminaries

Bilinear Maps. Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be three cyclic groups of prime order p. A map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is bilinear if for any generators $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, $\hat{e}(g_1^a, g_2^b) = \hat{e}(g_1, g_2)^{ab}$. Let $\mathcal{G}$ be a pairing generation algorithm which takes as input a security parameter 1^λ and outputs $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}) \leftarrow \mathcal{G}(1^\lambda)$. The generators of the groups may also be given. All group operations as well as the bilinear map $\hat{e}$ are efficiently computable.

The Symmetric External Diffie-Hellman Assumption (SXDH) [12]. The SXDH assumption is that the DDH assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$.

The Multi-Sequence of Exponents Diffie-Hellman Assumption (MSE-DDH) [8]. We now give the following decisional problem (ℓ, m, t)-MSE-DDH introduced by Delerablée and Pointcheval [8]. Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}) \leftarrow \mathcal{G}(1^\lambda)$ be a bilinear map generator and let ℓ, m and t be three integers. Let g_0 be a generator of $\mathbb{G}_1$ and h_0 a generator of $\mathbb{G}_2$. Given two random coprime polynomials f and g, of respective orders ℓ and m, with pairwise distinct roots $-x_1, \ldots, -x_\ell$ and $-y_1, \ldots, -y_m$ respectively, as well as several sequences of exponentiations of some random and hidden $\alpha, \gamma \in \mathbb{Z}_p$:

$$g_0, g_0^{\gamma}, \ldots, g_0^{\gamma^{\ell+t-2}}, g_0^{\alpha\cdot\gamma}, \ldots, g_0^{\alpha\cdot\gamma^{\ell+t}}, g_0^{k\cdot\gamma\cdot f(\gamma)}$$
$$h_0, h_0^{\gamma}, \ldots, h_0^{\gamma^{m-2}}, h_0^{\alpha\cdot\gamma}, \ldots, h_0^{\alpha\cdot\gamma^{2m-1}}, h_0^{k\cdot g(\gamma)},$$

and $T \in \mathbb{G}_T$, decide whether T is equal to $e(g_0, h_0)^{k\cdot f(\gamma)}$.

Delerablée and Pointcheval [8] showed that this problem belongs to the general Diffie-Hellman exponent problem due to Boneh *et al.* [4]. Therefore, the generic security of the assumption follows the result from [4]. We emphasize on the fact that, whereas the assumption has several parameters, it is non-interactive, and thus falsifiable.

Continual Leakage-Resilient Sharing (CLRS) and CLRS-Friendly Encryption. We review the two user continual leakage-resilient sharing (2-CLRS) in [6]. It is constructed from an updatable encryption. An updatable encryption [6] is a standard public key encryption scheme (KeyGen, Encrypt, Decrypt) with two additional procedures:

– $\mathsf{SKUpdate}$: On input a secret key sk, it outputs an updated key sk'.
– $\mathsf{CTUpdate}$: On input a ciphertext ct, it outputs an updated ciphertext ct'.

The 2-CLRS in [6] is constructed as follows (without the share preprocess phase):

– Setup: On input a security parameter 1^λ, it samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and outputs the public key pk.
– MsgGen: On input a secret s, it calculates $\mathsf{ct} \leftarrow \mathsf{Encrypt}_{\mathsf{pk}}(s)$ and outputs two shares $sh_1 = \mathsf{sk}$ and $sh_2 = \mathsf{ct}$.
– Update: On input a share sh_1 or sh_2, it outputs $\mathsf{SKUpdate}(\mathsf{sk})$ or $\mathsf{CTUpdate}(\mathsf{ct})$.
– $\mathsf{Reconstruct}$: On input sh_1 and sh_2, it outputs $s \leftarrow \mathsf{Decrypt}_{\mathsf{sk}}(\mathsf{ct})$.

We say that an updatable encryption scheme is an ℓ-CLRS-friendly encryption if the corresponding 2-CLRS is correct and ℓ-continual leakage resilient. The details of the construction of the ℓ-CLRS-friendly encryption is in [6].

4 Our Construction

In this section, we will describe how to construct dynamic secret sharing scheme under continual leakage model (CLM-DSS).

Our construction is inspired from the threshold encryption scheme by Delerablée and Pointcheval [8]. In order to allow leakage of secret shares of different users, we adopt the CLRS-friendly encryption scheme described in [6]. It is composed of (KeyGen, $\mathsf{Encrypt}$, $\mathsf{Decrypt}$, $\mathsf{SKUpdate}$, $\mathsf{CTUpdate}$). Then the secret share of each user is composed of two parts: (1) the secret key of the CLRS-friendly encryption [6], and (2) the ciphertext which is the encryption of the secret key of the threshold encryption [8]. By [6], both parts can be updated and leaked independently in the security proof.

4.1 CLM-DSS Scheme

$\mathsf{Setup}(1^\lambda)$: Given the security parameter 1^λ, it runs $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}) \leftarrow \mathcal{G}(1^\lambda)$. Also, two generators $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$ are randomly selected as well as two secret values $\gamma, \alpha \in \mathbb{Z}_p^*$. Finally, a set $\mathcal{D} = \{d_i\}_{i=1}^{m-1}$ of values in $\mathbb{Z}_p^*$ is randomly selected, where m is the maximal size of an authorized set. This corresponds to a set of dummy users, that will be used to complete a set of authorized users. It computes $u = g^{\alpha\gamma}$, $v = \hat{e}(g,h)^\alpha$. Denote $H : \mathbb{G}_1 \to \mathbb{G}_T$ be an efficiently computable function and its inverse H^{-1} is also efficiently computable. The list of users $\mathtt{L}$ is initially empty. It sets

$$\mathsf{PK} = (m, u, v, h, h^\alpha, \{h^{\gamma^i}\}_{i=1}^{m-2}, \{h^{\alpha\gamma^i}\}_{i=1}^{2m-1}, \mathcal{D}, H, H^{-1}, \mathtt{L}), \quad \mathsf{MK} = (g, \gamma, \alpha).$$

$\mathsf{SharePreprocess}(\mathsf{MK}, \mathsf{PK}, S, \mathcal{P})$: Given MK,PK and a set of users $\mathcal{P}$ (our scheme supports the secret space $S = \mathbb{G}_T$), for each identity $ID_i \in \mathcal{P}$, it randomly chooses a new and distinct $x_i \in \mathbb{Z}_p^*$ such that x_i should be different from all previous one, including the dummy users in $\mathcal{D}$. It runs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $\mathsf{ct}_i \leftarrow \mathsf{Encrypt}_{\mathsf{pk}_i}(H(g^{\frac{1}{\gamma+x_i}}))$ of the CLRS-friendly encryption of [6]. It generates the share $a_i = (\mathsf{sk}_i, \mathsf{ct}_i)$ for user ID_i. The value (x_i, ID_i) is published by putting it into $\mathtt{L}$.

$\mathsf{MsgGen}(\mathsf{PK}, s, t, \mathcal{P}, (a_1, \ldots, a_n))$: Given PK, the secret s, a threshold t and a set of users $\mathcal{P} = (ID_1, \ldots, ID_n)$, it first finds the corresponding x_i for ID_i from $\mathtt{L}$. Denote the set $\bar{P}$ be the set of all x_i. It randomly picks $k \in \mathbb{Z}_p^*$ and a random subset $\bar{\mathcal{D}} \subseteq \mathcal{D}$ of size $m+t-n-1$. It outputs the broadcast message $M = (C_1, C_2, C_3, \bar{\mathcal{D}})$, where:

$$C_1 = u^{-k}, \quad C_2 = h^{k \cdot \alpha \cdot \prod_{x_i \in \bar{P}}(\gamma + x_i) \cdot \prod_{x \in \bar{\mathcal{D}}}(\gamma + x)}, \quad C_3 = s \cdot v^k.$$

$\mathsf{Update}(\mathsf{PK}, a_i)$: For a share $a_i = (\mathsf{sk}_i, \mathsf{ct}_i)$, it can either update the first or the second part. For $\mathsf{Update}_1(\mathsf{sk}_i)$, it calls the function $\mathsf{SKUpdate}(\mathsf{sk}_i)$. For $\mathsf{Update}_2(\mathsf{ct}_i)$, it calls the function $\mathsf{CTUpdate}(\mathsf{ct}_i)$.

$\mathsf{Reconstruct}(\mathsf{PK}, (a_1, \ldots, a_t), M)$: Given PK, a broadcast message $M = (C_1, C_2, C_3, \bar{\mathcal{D}})$, and shares $a_i = (\mathsf{sk}_i, \mathsf{ct}_i)$ for $i = 1 \ldots, t$, denote $T = (x_1, \ldots x_t)$ be the corresponding value in $\mathtt{L}$. It first calls the function $usk_i \leftarrow H^{-1}(\mathsf{Decrypt}_{\mathsf{sk}_i}(\mathsf{ct}_i))$, where we can get $usk_i = g^{\frac{1}{\gamma + x_i}}$. We can compute

$$\sigma_i = \hat{e}(usk_i, C_2) = e(g, h)^{\frac{k \cdot \alpha \cdot \prod_{x_i \in \bar{P} \bigcup \bar{\mathcal{D}}}(\gamma + x_i)}{\gamma + x}}.$$

Denote Ω as the set of $\{\sigma_1, \ldots \sigma_t\}$. Finally, it can recover the secret s:

$$s = \frac{C_3}{(\hat{e}(C_1, h^{p(T,\bar{P})(\gamma)}) \cdot \mathbf{Aggregate}(\mathbb{G}_T, \Omega))^{\frac{1}{c(T,\bar{P})}}},$$

where $h^{p(T,\bar{P})(\gamma)}$ is computed from PK since

$$\begin{aligned}
p(T, \bar{P})(\gamma) &= \frac{1}{\gamma} \cdot \left(\prod_{x \in \bar{P} \bigcup \bar{\mathcal{D}} - T} (\gamma + x) - c(T, \bar{P}) \right), \\
c(T, \bar{P}) &= \prod_{x \in \bar{P} \bigcup \bar{\mathcal{D}} - T} x, \\
\mathbf{Aggregate}(\mathbb{G}_T, \Omega) &= \mathbf{Aggregate}(\mathbb{G}_T, \{\hat{e}(g, C_2)^{\frac{1}{\gamma + x}}\}_{x \in T}) \\
&= \hat{e}(g, C_2)^{\frac{1}{\prod_{x \in T}(\gamma + x)}} = \hat{e}(g, h)^{k \cdot \alpha \cdot \prod_{x_i \in \bar{P} \bigcup \bar{\mathcal{D}} - T}(\gamma + x_i)}.
\end{aligned}$$

The **Aggregate** algorithm computes $\hat{e}(g, C_2)^{\frac{1}{(\gamma + x_1) \ldots (\gamma + x_t)}}$ given $\Omega = \{\sigma_j = \hat{e}(g, C_2)^{\frac{1}{\gamma + x_j}}\}_{j=1}^t$. The detail of **Aggregate** can be found in [8].

Correctness. Assuming that Ω is correct, we have

$$\begin{aligned}
&\hat{e}(C_1, h^{p(T,\bar{P})(\gamma)}) \cdot \mathbf{Aggregate}(\mathbb{G}_T, \Omega) \\
=&\hat{e}(g^{-k \cdot \alpha \cdot \gamma}, h^{p(T,\bar{P})(\gamma)}) \cdot \hat{e}(g, C_2)^{\frac{1}{\prod_{x \in T}(\gamma + x)}} \\
=&\hat{e}(g, h)^{-k \cdot \alpha \cdot \gamma \cdot p(T,\bar{P})(\gamma)} \cdot \hat{e}(g, h)^{k \cdot \alpha \cdot \prod_{x \in \bar{P} \bigcup \mathcal{D}_{m+t-s-1} - T}(\gamma + x)} \\
=&\hat{e}(g, h)^{k \cdot \alpha \cdot c(T,\bar{P})} = (v^k)^{c(T,\bar{P})}.
\end{aligned}$$

4.2 Security

Hybrid Game Definitions. We first define several hybrid games. The main part of the proof is to show statistical indistinguishability between these games. The output of each game consists of the view of the adversary $\mathcal{A}$ as well as a bit b chosen by the challenger $\mathcal{C}$ representing its choice of which challenge secret s_0^*, s_1^* to be shared. Denote Q as $|\mathcal{P}^*| - t^* + 2$, where $|\mathcal{P}^*|$ is the size of the challenge set of users and t^* is the threshold. We describe the games in details below.

- **GameReal:** This is the original "$\mathsf{CorruptLeak}$" game.
- **Game** i**:** In Game i, for Leak queries on the first i uncorrupted identities from $\mathcal{P}^*$, the challenger $\mathcal{C}$ will answer the query using a random number, and for the rest of the identities, $\mathcal{C}$ answers it using the valid key.
- **GameFinal:** It is the same as **Game** Q, except that instead of sharing the challenge secret value s_b^*, we just share a random number. More specifically, in **GameFinal**, the secret shares used in the $\mathsf{LeakOracle}$ are all random and the broadcast message is just a random number. Thus, the view of the adversary in **GameFinal** is independent of the challenger's bit b.

Theorem 1. *For $z \geq 6$, $n \geq 3z-6$, our secret sharing scheme is ℓ_{SK}-continual leakage-resilient under the SXDH assumption and the (l, m, t)-MSE-DDH assumption, for $\ell_{SK} \leq \min(z/6-1, n-3z+6)\log(p) - \omega(\log(\lambda))$, $\mathcal{P}^* \cap \mathcal{I} \leq t-1$, $l = \mathcal{I} - \mathcal{P}^* \cap \mathcal{I}$ and $m = \max|\mathcal{P}^*|$.*

Proof. In order to prove that our secret sharing scheme is ℓ_{SK}-continual leakage secure, we will build several games to simulate the different cases and show that those games are indistinguishable under the SXDH, MSE-DDH assumption and the following lemma.

Lemma 1. *Let $z \geq 6$, $n \geq 3z-6$. Given $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e})$, an efficiently computable function $H : \mathbb{G}_1 \to \mathbb{G}_T$, the CLRS-friendly encryption [6] algorithm $\mathsf{Encrypt}$ with corresponding public key pk, secret key sk and the sequences:*

$$x_1, x_2, \ldots, x_{l+1};\ g^{\frac{1}{\gamma+x_1}}, g^{\frac{1}{\gamma+x_2}}, \ldots, g^{\frac{1}{\gamma+x_l}}; \{h^{\alpha\cdot\gamma^i}\}_{i=0}^{2m-1}, \{h^{\gamma^i}\}_{i=0}^{m-2}, g^{\alpha\cdot\gamma}, \hat{e}(g,h)^{\alpha}.$$

For simplicity, we denote the above mentioned sequences as D. Let $(sh_1, sh_2) = (\mathsf{sk}, \mathsf{Encrypt}_{\mathsf{pk}}(H(g^{\frac{1}{\gamma+x_{l+1}}})))$ and $(sh_1', sh_2') = (\mathsf{sk}, \mathsf{Encrypt}_{\mathsf{pk}}(R))$ where R is randomly chosen from the message space. Then under the SXDH assumption:

$$(D, f_1(sh_1), f_2(sh_2)) \overset{stat}{\approx} (D, f_1(sh_1'), f_2(sh_2')),$$

as long as the function f_1 and f_2 have the output size $|f_1|, |f_2| \leq \min(z/6-1, n-3z+6)\log(p) - \omega(\log(\lambda))$.

Proof. According to the security of the CLRS-Friendly Encryption in [6], no PPT adversary can distinguish the ciphertext of $H(g^{\frac{1}{\gamma+x_{l+1}}})$ from a random number, as long as $|f_1|, |f_2| \leq \min(z/6-1, n-3z+6)\log(p) - \omega(\log(\lambda))$, even given the value $H(g^{\frac{1}{\gamma+x_{l+1}}})$. Therefore, given the sequence D, no PPT adversary can distinguish these two distributions under the SXDH assumption. □

Lemma 2. ***GameReal*** $\overset{stat}{\approx}$ ***Game*** 0.

It is easy to see that **GameReal** is the same as **Game** 0.

Lemma 3. *For* $i = 0, \ldots, Q-1$, *under the SXDH assumption with* $z \geq 6, n \geq 3z-6, \ell_{SK} \leq \min(z/6-1, n-3z+6)\log(p)-\omega(\log(\lambda))$, ***Game*** $i \overset{stat}{\approx}$ ***Game*** $i+1$.

Proof. If there exists an adversary $\mathcal{A}$ who has a non-negligible advantage in distinguishing **Game** i and **Game** $i+1$, we will create a PPT algorithm $\mathcal{B}$ which will distinguish between $(D, f_1(sh_1), f_2(sh_2))$ and $(D, f_1(sh_1'), f_2(sh_2'))$ from Lemma 1 with non-negligible probability. This will yield a contradiction, since these distributions have a negligible statistical distance if the CLRS-friendly encryption scheme [6] is secure.

$\mathcal{B}$ simulates either **Game** i or **Game** $i+1$ with $\mathcal{A}$ as follows. It starts by running the Setup algorithm by itself and giving $\mathcal{A}$ the public parameters by using the **Lemma** 1's instances. By the notion of the **Lemma** 1, $\mathcal{B}$ can create enough valid keys and responds to all the $\mathcal{A}$'s queries.

For the Join queries on the $(i+1)$-th uncorrupted identity from $\mathcal{P}^*$, $\mathcal{B}$ just responds by putting x_{l+1} in L. When $\mathcal{A}$ issues the Leak query on the $(i+1)$-th identity, $\mathcal{B}$ will encode the leakage $\mathcal{A}$ asks for this key in **Phase 1** as a single polynomial time computable function f. It can do this by fixing the values of all other keys and fixing all other variables involved in the challenge key. $\mathcal{B}$ then receives a sample $\langle D, f_1(SH_1), f_2(SH_2)\rangle$(where SH_i is either distributed as sh_i or sh_i' using the notation of the **Lemma** 1). $\mathcal{B}$ will use $f_1(SH_1)$, $f_2(SH_2)$ to answer $\mathcal{A}$'s Leak query for the $(i+1)$-th identity by implicitly defining the challenge key as (SH_1, SH_2).

At some point, $\mathcal{A}$ declares the challenge secret value, and $\mathcal{B}$ flips a coin b and produces the challenge broadcast message M^*. Since M^* is independent of the queries above, the ciphertext is well-distributed. The **Phase 2** is the same as the **Phase 1** except the forbiddance of the Leak queries.

If SH_2 is the ciphertext of $H(g^{\frac{1}{\gamma+x_{l+1}}})$, then $\mathcal{B}$ responds with the correct leak query, so $\mathcal{B}$ properly simulates **Game** i. If SH_2 is the ciphertext of a random number, it correctly simulates **Game** $i+1$. □

Lemma 4. *If the* (l, m, t)*-MSE-DDH assumption holds, where* $t-1 \geq \mathcal{P}^* \cap \mathcal{I}$, $l = \mathcal{I} - \mathcal{P}^* \cap \mathcal{I}$, $m = \max|\mathcal{P}^*|$, *then* ***Game*** $Q \overset{stat}{\approx}$ ***GameFinal***.

Proof. Initially, the attacker output a target set $\mathcal{P}^* = \{ID_1^*, \ldots, ID_{|\mathcal{P}^*|}^*\}$ of identities that he wants to attack (the target authorized set), and a set $\mathcal{I}$ of identities that he wants to corrupt, with $|\mathcal{I}| \leqslant l+t-1$ and $|\mathcal{P}^* \cap \mathcal{I}| \leqslant t-1$. Given the (l, m, t)-MSE-DDH instance with the bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e})$, two coprime polynomials f and g of respective orders l and m with their pairwise distinct roots $(-x_1, \ldots, -x_l)$ and $(-x_{l+t}, \ldots, -x_{l+t+m-1})$, and given:

$$g_0, g_0^{\gamma}, \ldots, g_0^{\gamma^{\ell+t-2}}, g_0^{\alpha\cdot\gamma}, \ldots, g_0^{\alpha\cdot\gamma^{\ell+t}}, g_0^{k\cdot\gamma\cdot f(\gamma)}$$
$$h_0, h_0^{\gamma}, \ldots, h_0^{\gamma^{m-2}}, h_0^{\alpha\cdot\gamma}, \ldots, h_0^{\alpha\cdot\gamma^{2m-1}}, h_0^{k\cdot g(\gamma)}, T,$$

where T is either equal to $\hat{e}(g_0, h_0)^{k \cdot f(\gamma)}$ or a random elements from $\mathbb{G}_T$. We can randomly choose $x_{l+1}, \ldots, x_{l+t-1}$ from $\mathbb{Z}_p^*$ (different from other x_i's) and write f, g and an additional function q as:

$$f(X) = \prod_{i=1}^{l}(X + x_i), \quad q(X) = \prod_{i=l+1}^{l+t-1}(X + x_i), \quad g(X) = \prod_{i=l+t}^{l+t+m-1}(X + x_i).$$

The polynomial f corresponds to a set of l users not in the target set that can be corrupted. The polynomial q corresponds to a set of $t-1$ users of the target set that can be corrupted. The polynomial g corresponds to the $|\mathcal{P}^*| - t + 1$ users of the target set and the rest dummy keys used in the challenge phase, and the users of the target set can be leaked by the leakage function specified by the attacker. For $i \in [1, l+t-1]$, we set $f_i(\gamma) = \frac{f(\gamma) \cdot q(\gamma)}{\gamma + x_i}$.

Setup: To generate the system parameters, the simulator $\mathcal{B}$ calculates $g = g_0^{f(\gamma) \cdot q(\gamma)}$ and sets

$$h = h_0, \quad u = g_0^{\alpha \cdot \gamma \cdot f(\gamma) \cdot q(\gamma)} = g^{\alpha \cdot \gamma}, \quad v = e(g_0, h_0)^{\alpha \cdot f(\gamma) \cdot q(\gamma)} = e(g, h)^{\alpha}.$$

The two latter formula can be computed from the MSE-DDH instance input, since $f \cdot q$ is a polynomial of degree $l + t - 1$. $\mathcal{B}$ then sets the set $\mathcal{D} = \{d_i\}_{i=1}^{m-1}$ corresponding to dummy users:

- $\mathcal{D}^* = \{d_i\}_{i=1}^{m+t-|\mathcal{P}^*|-1}$ is a subset of $\{x_i\}_{l+t}^{l+t+m-1}$. This subset corresponds to the dummy users included to complete the target set in the challenge phase as mentioned above.
- $\{d_i\}_{m+t-|\mathcal{P}^*|}^{m-1}$ is a set of random elements in $\mathbb{Z}_p^*$.

Finally, $\mathcal{B}$ sets PK as the definition in the previous section.

Query: For each Join query with input ID_i, $\mathcal{B}$ puts $(x, ID_i) \in \mathsf{L}$ where:

- if $ID_i \in \mathcal{I} \cap \mathcal{P}^*$, $\mathcal{B}$ sets x as an "unused" element in $\{x_i\}_{l+1}^{l+t-1}$.
- if $ID_i \in \mathcal{I} - \mathcal{I} \cap \mathcal{P}^*$, $\mathcal{B}$ sets x as an "unused" element in $\{x_i\}_1^l$.
- if $ID_i \in \mathcal{P}^* - \mathcal{I} \cap \mathcal{P}^*$, $\mathcal{B}$ sets x as an "unused" element in $\{x_i\}_{l+t}^{l+t+m-1}$.
- if $ID_i \notin \mathcal{I} \cup \mathcal{P}^*$, $\mathcal{B}$ randomly picks a $x \notin \{x_i\}_{i=1}^{l+t+m-1}$ in $\mathbb{Z}_p$.

When comes to the Leak query, $\mathcal{B}$ answers it using a random number. When receiving the $\mathsf{Corrupt}$ query for x_i, $\mathcal{B}$ computes the valid key $g^{\frac{1}{\gamma + x_i}} = g_0^{f_i(\gamma)}$ using the $\mathsf{MSE\text{-}DDH}$ problem instance and gives it to $\mathcal{A}$.

Challenge: $\mathcal{B}$ picks a random bit b and generates a broadcast message for the secret s_b^* for a dummy set $\mathcal{D}^*$:

$$C_1^* = g_0^{-k \cdot \gamma \cdot f(\gamma)} = u^{-k'}, \quad C_2^* = h_0^{k \cdot g(\gamma)} = h^{k' \cdot \alpha \cdot \prod_{x_i \in \mathcal{P}^* \cup \mathcal{D}^*}(\gamma + x)}, \quad C_3^* = T \cdot s_b^*,$$

where $k' = \frac{k}{\alpha \cdot q(\gamma)}$. $\mathcal{B}$ returns $(C_1^*, C_2^*, C_3^*, \mathcal{D}^*)$ to $\mathcal{A}$.

The definition of **GameFinal** is the same as **Game** Q except for that instead of sending $v^{k'} \cdot s_b^*$ to $\mathcal{A}$, $\mathcal{B}$ sends $v^{k'} \cdot R$ to the attacker, where R is a random number. If $T = e(g_0, h_0)^{k \cdot f(\gamma)}$, then $C_3^* = v^{k'} \cdot s_b^*$ which is the simulation of **Game** Q. If T is a random number in $\mathbb{G}_T$, then C_3^* is random in $\mathbb{G}_T$. Therefore $\mathcal{B}$ simulates the **GameFinal**. □

Putting Them All Together. Following the above lemmas, we can easily get **GameReal** $\overset{stat}{\approx}$ **GameFinal**. Recall that the output of each game includes the view of $\mathcal{A}$ at the end of experiment along with the challenger's choice bit b, since $\mathcal{A}$'s guess b' at the end of the game can be efficiently computed from the output of each game. In the **GameFinal**, the view of $\mathcal{A}$ is independent of the random bit b. Hence we have $\Pr[\mathcal{A} \text{ wins}] = \frac{1}{2}$ since the two games are indistinguishable under the SXDH assumption the MSE-DDH assumption. □

References

1. Akavia, A., Goldwasser, S., Vaikuntanathan, V.: Simultaneous Hardcore Bits and Cryptography against Memory Attacks. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 474–495. Springer, Heidelberg (2009)
2. Blakley, B., Blakley, G.R., Chan, A.H., Massey, J.L.: Threshold Schemes with Disenrollment. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 540–548. Springer, Heidelberg (1993)
3. Blundo, C., Cresti, A., De Santis, A., Vaccaro, U.: Fully Dynamic Secret Sharing Schemes. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 110–125. Springer, Heidelberg (1994)
4. Boneh, D., Boyen, X., Goh, E.-J.: Hierarchical Identity Based Encryption with Constant Size Ciphertext. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (2005)
5. Boyle, E., Goldwasser, S., Kalai, Y.T.: Leakage-Resilient Coin Tossing. In: Peleg, D. (ed.) Distributed Computing. LNCS, vol. 6950, pp. 181–196. Springer, Heidelberg (2011)
6. Brakerski, Z., Kalai, Y.T., Katz, J., Vaikuntanathan, V.: Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In: FOCS, pp. 501–510. IEEE Computer Society (2010)
7. Davì, F., Dziembowski, S., Venturi, D.: Leakage-Resilient Storage. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 121–137. Springer, Heidelberg (2010)
8. Delerablée, C., Pointcheval, D.: Dynamic Threshold Public-Key Encryption. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 317–334. Springer, Heidelberg (2008)
9. Dodis, Y., Lewko, A.B., Waters, B., Wichs, D.: Storing secrets on continually leaky devices. In: Ostrovsky, R. (ed.) FOCS, pp. 688–697. IEEE (2011)
10. Laih, C.-S., Harn, L., Lee, J.-Y., Hwang, T.: Dynamic Threshold Scheme Based on the Definition of Cross-Product in an N-dimensional Linear Space. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 286–298. Springer, Heidelberg (1990)
11. Lewko, A.B., Rouselakis, Y., Waters, B.: Achieving Leakage Resilience through Dual System Encryption. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 70–88. Springer, Heidelberg (2011)
12. Scott, M.: Authenticated id-based key exchange and remote log-in with simple token and pin number. Cryptology ePrint Archive, Report 2002/164 (2002), `http://eprint.iacr.org/`
13. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)
14. Yuen, T.H., Chow, S.S.M., Zhang, Y., Yiu, S.M.: Identity-Based Encryption Resilient to Continual Auxiliary Leakage. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 117–134. Springer, Heidelberg (2012)