Annales Univ. Sci. Budapest., Sect. Comp. 57 (2024) 69–99

UPDATABLE THRESHOLD ENCRYPTION FROM ISOGENIES

Chenfeng He, Fatna Kouider and Péter Kutas

(Budapest, Hungary)

Communicated by Péter Burcsi

(Received November 8, 2023; accepted May 21, 2024)

Abstract. In this paper, we propose a new primitive called updatable threshold encryption (UTE) which is motivated by real-world applications. Namely one would like to encrypt extremely sensitive data, handle post-compromise and forward security and distribute trust amongst many parties for decryption. As one is interested in long-term security we also would like the scheme to be quantum-resistant.

UTE can be seen as a variant of updatable encryption (UE) with certain threshold properties. We introduce algorithms and security definitions for UTE and provide an instantiation with cryptographic group actions. In order to handle shares being revoked and new parties being added we introduce the first post-quantum dynamic secret sharing scheme based on group actions.

1. Introduction

Once large-scale quantum computers are built, currently deployed protocols will become insecure. This is particularly important in certain use cases where

Key words and phrases: Updatable encyption, threshold encryption, isogenies, post quantum, secret sharing.

2010 Mathematics Subject Classification: 14K02, 58E40, 68P25, 94A60, 94A62.

Péter Kutas is supported by the Hungarian Ministry of Innovation and Technology NRDI Office within the framework of the Quantum Information National Laboratory Program, the János Bolyai Research Scholarship of the Hungarian Academy of Sciences and by the UNKP-23-5 New National Excellence Program and partly supported by EPSRC through grant number EP/V011324/1.

https://doi.org/10.71352/ac.57.069

secrets need to stay secret for long periods of time. For instance one might want to encrypt classified data to avoid leakage of important information even if quantum computers become a reality in only 20–30 years.

In the case of these documents one might have extra requirements. Namely, one would like to handle a situation where secret keys get compromised and one would like to distribute trust amongst several parties. More precisely, one needs to satisfy these conditions:

1. The encryption should be quantum-safe.

2. The scheme should have threshold properties, i.e., only a set of authorised parties can decrypt encrypted data.

3. In case a secret key gets leaked, it should be easy to update ciphertexts to ensure forward and post-compromise security.

4. One should be able to handle changes in access structure, i.e., if authorizations are added or revoked.

Our starting point is an existing primitive called updatable encryption (UE) which was defined by Boneh, Levi, Montgomery and Raghunathan [3]. The motivation for (UE) comes from secure storage. Clients want to store encrypted data on a server and be able to issue updates without decrypting the ciphertexts. The only post-quantum instantiation of UE is [18] which uses cryptographic group actions coming from isogeny-based cryptography. UE has strong security properties that fit our framework but clients hold entire secret keys.

1.1. Our contribution

In this paper, we make the following contributions:

1. We introduce a new primitive called updatable threshold encryption (UTE) which (informally) is an updatable encryption scheme which is simultaneously a threshold encryption scheme.

2. We provide a precise definition of this new primitive (algorithms and security properties) and provide an instantiation using isogenies.

3. In order to handle accesses being revoked and added one could just issue a new update every time this happens. This might be enough for certain applications but in certain cases this is quite costly. We provide a more efficient way to handle changes in the access structure. Namely we integrate a dynamic secret sharing scheme into our UTE construction.

4. We introduce the first post-quantum dynamic secret sharing scheme (again using group actions) which might be of independent interest. To the best of our knowledge, there does not exist a quantum-resistant dynamic secret sharing scheme in the literature.

The paper is structured as follows. In Section 2 we introduce the basic properties of updatable encryption and cryptographic group actions. In Section 3 we provide a formal definition of updatable threshold encryption and in Section 4 we propose a quantum-resistant instantiation of UTE. A major part of Section 4 is devoted to our dynamic secret sharing scheme.

2. Preliminaries

In this section we describe most of the necessary background needed for later sections, such as hard homogeneous spaces, updatable encryption and instantiations of crytographic group actions from isogenies.

2.1. Notations

λ denotes the security parameter for updatable encryption and $\leftarrow\!\!\$\ S$ denotes sampling uniformly from a set S. We use $\mathcal{A}^{\mathcal{O}}$ to denote that an algorithm $\mathcal{A}$ has access to the oracle $\mathcal{O}$.

2.2. Hard homogeneous spaces

Hard homogeneous spaces (HHS) were introduced by Couveignes [10].

Definition 2.1. If G is a group with identity element e, and X is a set, then a (left) group action α of G on X is a function

$$\alpha\colon G \times X \to X,$$

that satisfies the following two axioms:

(i) Identity: $\alpha(e, x) = x$.

(ii) Compatibility: $\alpha\left(g, \alpha\left(h, x\right)\right) = \alpha\left(gh, x\right).$

For simplicity, we denote action $\alpha(g, h)$ by $g * h$. A principal homogeneous space for a group G is a non-empty set X on which G acts freely and transitively (meaning that, for any $x, y \in X$, there exists a unique $g \in G$ such that $g{*}x = y$).

Couveignes defines a HHS as a finite principal homogeneous space with some additional algorithmic properties. He requires that the following problems can be solved efficiently:

(i) (Group Operations) Given strings g_1 and g_2 decide if they represent elements in G and if these elements are equal or not. Given $g_1, g_2 \in G$, compute g^{-1}, $g_1 g_2$ and decide if $g_1 = g_2$.

(ii) (Random Element) Find a random element in G with uniform probability.

(iii) (Membership) Given a string h_0 decide if h_0 represents an element in X.

(iv) (Equality) Given $h_1, h_2 \in X$ decide if $h_1 = h_2$.

(v) (Action) Given $g \in G$ and $h \in X$ compute $g * h$.

Furthermore, the following problems should be hard (e.g., not known to be solvable in polynomial time):

(i) Vectorization: Given $h, h' \in X$ find $g \in G$ such that $g * h = h'$.

(ii) Parallelization: Given $h, h', F \in X$, such that $h' = g * h$, find $F' = g * F$.

As a simple example, let X be a group of prime order q, then $G = \mathbb{Z}/q\mathbb{Z}^\times$ acts on $X \backslash \{1\}$ by $a * g = g^a$. In this case, the Vectorization problem is the discrete logarithm problem in X, and the Parallelization problem is the Computational Diffie–Hellman problem. Hence any discrete logarithm group is a HHS. However, the discrete logarithm problem carries more structure (instead of an abelian group action one has $\mathbb{Z}/q\mathbb{Z}$-module as one can multiply and add exponents) which is the reason it is vulnerable to Shor's algorithm. CSIDH [26] is an example of a cryptographic group action which satisfies most properties of HHS (uniform sampling is conjectured as in many CSIDH instantiations it is hard to compute the structure of the group). We will expand on this in Section 2.6.

2.3. Updatable encryption

An updatable encryption scheme is a symmetric encryption scheme defined by Boneh, Lewi, Montgomery and Raghunathan in 2013 [3] to periodically update ciphertexts stored in the cloud using a new key.

The main goal behind this scheme is to grant the organizations an extra level of security for their encrypted data stored in the cloud against unauthorized use and access by their former employees who are keeping the corresponding secret key[3].

The updatable encryption scheme consists of the following PPT (probabilistic polynomial-time) set of algorithms(UE.Setup, UE.KeyGen, UE.Enc, UE.Dec UE.TokenGen, UE.Upd) that operate in epoch times. The Figure 1 illustrates the Updatable Encryption scheme.

Algorithm	Input	Output	Syntax
Setup (UE.Setup)	Security parameter λ	Public parameter pp	$pp \leftarrow UE.Setup(1^{\lambda})$
Key generation (UE.KeyGen)	Public parameter pp	Epoch key k_e	$K_e \leftarrow UE.KeyGen(pp)$
Encryption (UE.Enc)	m, k_e	C_e	$C_e \leftarrow\$ UE.Enc(k_e, m)$
Decryption (UE.Dec)	C_e, k_e	message m or $\perp$	$\{m', \perp\} \leftarrow UE.Dec(k_e, C_e)$
Token generation (UE.TokenGen)	k_e, k_{e+1}	Δ_{e+1}	$\Delta_{e+1} \leftarrow UE.TokenGen(k_e, k_{e+1})$
Update (UE.Upd)	C_e, Δ_{e+1}	C_{e+1}	$C_{e+1} \leftarrow UE.Upd(\Delta_{e+1}, C_e)$

Figure 1: Updatable Encryption scheme's algorithms

Definition 2.2. Correctness [3, 6, 17]: For all messages $m \in M$, the updatable encryption scheme is correct if it always holds that $Pr[UE.Dec(k_e, C_e) = = m] = 1$. For all $C_e = UE.Enc(k_e, m)$ and $C_e = UE.upd(\Delta_{e-1}, C_{e-1})$.
In other words, the correctness notion for updatable encryption means that decryption of both fresh and updated ciphertexts leads always to the correct message m.

2.4. Security of UE

We follow the previous works on updatable encryption schemes [6, 17, 18], where the security notions are game-based and described as challenges (experiments) run between a challenger and an adversary $\mathcal{A}$.

During each experiment and depending on the desired security notion, the Adversary $\mathcal{A}$ is allowed to issue queries to some specific oracles $\mathcal{O}$ described in the Figure 2.

All the oracles in updatable encryption security games are controlled by the challenger. We summarize the functionality of each oracle as follows:

Initialize(1^{λ}) : initialize the challenger's state $CS = \{\mathcal{L}, \tilde{\mathcal{L}}, \mathcal{C}, \mathcal{K}, \mathcal{T}\}$ and the system's state $\{pp, k_0, \Delta_0, e = 0, CS\}$ s.t:

$\mathcal{L}$: list of honestly generated ciphertexts, i.e non-challenge ciphertexts of the form (c, C, e) which are generated through a call to $\mathcal{O}$.Enc or $\mathcal{O}$.Upd, c is a counter that increments with each call to $\mathcal{O}$.Enc.

$\tilde{\mathcal{L}}$: list of updated versions of challenge ciphertexts of the form $(\tilde{C}, e)$ automatically updated via a call to $\mathcal{O}$. Next,an adversary $\mathcal{A}$ gets the $\tilde{C}_e$ via a call to $\mathcal{O}$.Upd$\tilde{C}$ oracle.

Initialize(1^λ): $pp \leftarrow$ UE.Setup(1^λ) $k_0 \leftarrow$ UE.KeyGen(pp) $e, c, ph, twf \leftarrow 0$ $\Delta_0 \leftarrow\perp$ $\mathcal{L}, \tilde{\mathcal{L}}, \mathcal{C}, \mathcal{K}, \mathcal{T} \leftarrow \emptyset$	**$\mathcal{O}$.Enc(M)** C $\leftarrow$ UE.Enc(k_e,M) c $\leftarrow c + 1$ $\mathcal{L} \leftarrow \mathcal{L} \cup \{(c, C, e)\}$ return C	**$\mathcal{O}$.Dec(C)** if ($ph = 1$ and $C \in \tilde{\mathcal{L}}$) then $twf \leftarrow 1$ M or $\perp\leftarrow$ UE.Dec(k_e,C) return M or $\perp$

$\mathcal{O}$.Next()
$e \leftarrow e + 1$
K$_e \leftarrow$ UE.KeyGen(pp)
$\Delta_e \leftarrow$ UE.TokenGen
(k_{e-1}, k_e)
if ($ph = 1$) then
$\tilde{C}_e \leftarrow$
UE.Upd($\Delta_e, \tilde{C}_{e-1}$)
	$\mathcal{O}$.Upd(C$_{e-1}$)
	if (($j, C_{e-1}, e-1) \notin \mathcal{L}$)
	then
	return $\perp$
	$C_e \leftarrow$UE.Upd
	(Δ_e, C_{e-1})
	$\mathcal{L} \leftarrow \mathcal{L} \cup \{(j, C_e, e)\}$
	return C_e
		$\mathcal{O}$.Corr(input,$\hat{e}$)
		if ($\hat{e} > e$) then
		return $\perp$
		if (input=key) then
		$\mathcal{K} \leftarrow \mathcal{K} \cup \{\hat{e}\}$
		return K$_{\hat{e}}$
		if (input=token) then
		$\mathcal{T} \leftarrow \mathcal{T} \cup \{\hat{e}\}$
		return $\Delta_{\hat{e}}$

$\mathcal{O}$.Upd$\tilde{C}$
if ($ph \neq 1$) then
return $\perp$
$\mathcal{C} \leftarrow \mathcal{C} \cup$ {e}
$\tilde{\mathcal{L}} \leftarrow \tilde{\mathcal{L}} \cup \{(\tilde{C}_e, e)\ \}$
return $\tilde{C}_e$
	$\mathcal{O}$.Try$\tilde{C}$
	if ($ph = 1$) then
	return $\perp$
	$ph \leftarrow 1$
	if ($e \in \mathcal{K}^*$ or $\tilde{C} \in \mathcal{L}^*$)
	then
	$twf \leftarrow 1$
	M or
	$\perp\leftarrow$UE.Dec($K_e, \tilde{C}$)
	if ($M \neq\perp$) then
	winForgery $\leftarrow 1$
		$\mathcal{O}$.Chall($\bar{M}$,$\bar{C}$)
		if ($ph \neq 1$) then
		return $\perp$
		$ph \leftarrow 1$
		$\tilde{e} \leftarrow e$
		if ((. ,$\bar{C}$,$e-1$)$\notin \mathcal{L}$)
		then
		return $\perp$
		if ($b = 0$) then
		$\tilde{C}_e \leftarrow$UE.Enc
		(k_e,$\bar{M}$)
		else
		$\tilde{C}_e \leftarrow$UE.Upd
		($\Delta_e, \bar{C}$)
		$\mathcal{C} \leftarrow \mathcal{C} \cup \{e\}$
		$\tilde{\mathcal{L}} \leftarrow \tilde{\mathcal{L}} \cup \{(\tilde{C}_e, e)\}$
		return $\tilde{C}_e$

Figure 2: Overview of oracles used in security experiments for updatable encryption

$\mathcal{C}$: list of epochs e in which an adversary $\mathcal{A}$ learned the updated version of the challenge ciphertexts by calling $\mathcal{O}$.chall$\tilde{C}$ or $\mathcal{O}$.Upd$\tilde{C}$.

$\mathcal{K}$: list of all epochs e in which an $\mathcal{A}$ corrupted the key of certain epoch e.

$\mathcal{T}$: list of all epochs e where an adversary corrupted the underlying token Δ_e.

$\mathcal{O}$**.Enc**: this oracle is used to encrypt a message M using the current epoch key k_e, stores its ciphertext C and the underlying epoch to the list $\mathcal{L}$ and returns the resulting C.

$\mathcal{O}$**.Dec**: this oracle is used to decrypt a ciphertext C using the current epoch key k_e and returns the decryption result (plaintext M, or $\perp$ for invalid result).

$\mathcal{O}$**.Upd**: this oracle is used to re-encrypt the ciphertext C of certain epoch e. C_e must be honestly generated(C_e contained in $\mathcal{L}$) i.e generated by $\mathcal{O}$.Enc or $\mathcal{O}$.Upd otherwise it returns $\perp$.

$\mathcal{O}$**.Corr**: this oracle is used to corrupt epoch keys or tokens depends on the input, if the Adversary $\mathcal{A}$ call the Corruption oracle and set:

(i) $input = Key$ and $\hat{e}$, $\forall\, \hat{e} \leq e$, the oracle will update the list $\mathcal{K} : \mathcal{K} \leftarrow \mathcal{K} \cup \{\hat{e}\}$ and return the corrupted key to the adversary $\mathcal{A}$.

(ii) $input = Token$ and $\hat{e}$,$\forall\ \hat{e} \leq e$, the oracle will update the list $\mathcal{T} : \mathcal{T} \leftarrow$ $\leftarrow \mathcal{T} \cup \{\hat{e}\}$ and return the corrupted token to the adversary $\mathcal{A}$.

$\mathcal{O}$**.Next**: this oracle is used to move to the next epoch $e + 1$ and hence generating a new epoch key and new update token.

$\mathcal{O}$**.Chall**: the challenge oracle that challenges the adversary with either the encryption of $\bar{M}$ or the update of $\bar{C}$.it returns $\perp$ if the ciphertext $\bar{C}$ corresponds to a plaintext of length $\neq |\bar{M}|$.

$\mathcal{O}$**.Upd**$\tilde{C}$: this oracle is used to acquire the challenge ciphertext $\tilde{C}$ corresponding to the current epoch e from the updated version list $\tilde{\mathcal{L}}$.

$\mathcal{O}$**.Try**$\tilde{C}$: a special oracle which called only during an INT-CTXT challenge by an adversary $\mathcal{A}$ who will win the challenge if his forgery attempt $\tilde{C}$ is valid.

2.4.1. Security properties

We recall the required security properties for updatable encryption from [17]:

(i) **Token Security:** No additional benefit is gained for breaking the UE scheme if the ciphertexts or all the tokens are exposed.

(ii) **Forward Security:** No additional advantage is gained by an adversary who may compromise a secret key k_e of certain epoch e to decrypt the ciphertexts of previous epochs $e' < e$ already obtained before.

(iii) **Post-Compromise Security:** No additional advantage is gained by an adversary who may compromise a secret key k_e of certain epoch e to decrypt the ciphertexts of next epochs $e' > e$ that he will obtained after this compromise.

(iv) **IND-CPA Security(IND-CPA [6, 18])**: An updatable encryption scheme is called secure under IND-CPA if for any PPT adversary $\mathcal{A}$ the following advantage is negligible for the security parameter λ while $exp^{IND-CPA}$ represents the experiment between the adversary and the challenger.

$$Adv_{UE,\mathcal{A}}^{IND-CPA} = \left| Pr[Exp_{UE,\mathcal{A}}^{IND-CPA-1} = 1] - Pr[Exp_{UE,\mathcal{A}}^{IND-CPA-0} = 1] \right|.$$

(v) **IND-CCA Security(IND-CCA [6, 18])**: An updatable encryption scheme is called secure under IND-CCA if for any PPT adversary $\mathcal{A}$ the following advantage is negligible for the security parameter λ while $exp^{IND-CCA}$ represents the experiment between the adversary and the challenger.

$$Adv_{UE,\mathcal{A}}^{IND-CCA} = \left| Pr[Exp_{UE,\mathcal{A}}^{IND-CCA-1} = 1] - Pr[Exp_{UE,\mathcal{A}}^{IND-CCA-0} = 1] \right|.$$

(vi) **Plaintext Integrity**: Requires that it be computationally infeasible to create a ciphertext decrypting to a new plaintext (never encrypted before by the sender).More formally, an updatable encryption scheme is called secure under (INT-PTXT) if for any PPT adversary $\mathcal{A}$ the following advantage is negligible for the security parameter λ.

$$Adv_{UE,\mathcal{A}}^{INT-PTXT} = Pr[Exp_{UE,\mathcal{A}}^{INT-PTXT} = 1].$$

(vii) **Ciphertext Integrity**: Requires that it be computationally infeasible to create a fresh ciphertext (not created before by the sender) regardless of whether the corresponding plaintext is new or not. In more detail, a scheme achieves the INT-CTXT security if no adversary with access to the encryption oracles ($\mathcal{O}$.Enc, $\mathcal{O}$.Next, $\mathcal{O}$.Upd and $\mathcal{O}$.Corr) can produce a valid ciphertext (its decryption outputs a message) and differ from all the ciphertexts which he obtained from the oracles. This produced ciphertext is usually called a ciphertext forgery. More formally, an updatable encryption scheme is called secure under (INT-CTXT) if for any PPT adversary $\mathcal{A}$ the following advantage is negligible for the security parameter λ.

$$Adv_{UE,\mathcal{A}}^{INT-CTXT} = Pr[Exp_{UE,\mathcal{A}}^{INT-CTXT} = 1].$$

2.5. UE from group actions

In this subsection we recall constructions of UE from group actions. In [18], they introduced two methods for updatable encryption. One is called GAINE, and another is ETOGA (Efficient **TOGA**). Since our work is based on TOGA (Triple Orbital Group Action), we introduce it here.

TOGA consists of three group actions, for more details of this setting, we refer to [18].

As a starting point we have a main action $(A, S, \star_A)$ for an abelian group A and a set S. Group A has a congruence relation $\sim_A$. We could use this relation and the group action $\star_A$ to induce a relation $\sim_S$ in S. Namely,

$$s_1 \sim_S s_2, \iff \exists a_1, a_2 \in A, \text{ with } a_1 \sim_A a_2 \text{ such that } a_1 \star_A s_1 = a_2 \star_A s_2.$$

It's not hard to see that this is indeed a relation in S. For the transitivity of $\sim_S$, assume $s_1 \sim_S s_2$, and $s_2 \sim_S s_3$, we have $a_1 \star_A s_1 = a_2 \star_A s_2$, and $b_2 \star_A s_2 = b_3 \star_A s_3$, therefore $a_1 b_2 \star_A s_1 = a_2 b_3 \star_A s_3$ and $a_1 b_2 \sim_A a_2 b_3$ since A is an abelian group and $\sim_A$ is a congruence relation.

We also have a second group action: $A/\sim_A$ acts on $S/\sim_S$. To see this is a group action, the only non-trivial thing is to show that this is a well-defined group action. In other words, to prove when $a_1 \sim_A a_2$, and $s_1 \sim_S s_2$, we have $a_1 \star_A s_1 \sim_S a_2 \star_A s_2$. Because we have $b_1 \star_A s_1 = b_2 \star_A s_2$ for some $b_1 \sim_A b_2$, hence $(a_2 b_1) \star_A (a_1 \star_A s_1) = (a_1 b_2) \star_A (a_2 \star_A s_2)$ and here $a_2 b_1 \sim_A a_1 b_2$.

Last but not least, we need to find the third group action: $(H, S, \star_H)$ for which we can treat our messages as group elements of H. For decryption to be possible, we also need this action to be efficiently invertible. Specifically, for any $s \in S$ and $h \in H$, we have $h \star_H s \sim_S s$ and for all $s' \sim_S s$, there exists h such that $s' = h \star_H s$.

Moreover, we need the third group action to be free in order to make this action invertible and $\star_H, \star_A$ commute. Finally, for any $a_1, a_2 \in A$ with $a_1 a_2 \sim_A \sim_A 1_A$ there exists a unique element $h(a_1 a_2) \in H$ such that

$$a_1 a_2 \star_A s = h(a_1 a_2) \star_H s$$

for all $s \in S$.

When $a_1 a_2 \sim_A 1_A$, we have $h(a_1, a_2) = Invert_H((a_1 a_2) \star_A s, s)$. We fix a starting element $s_0 \in S$, and also assume the existence of an invertible map $\psi : \mathcal{M} \to H$ where $\mathcal{M}$ is the space of the messages. We will use this function ψ to map the messages to the group H and encrypt them with group action $\star_H$. Decryption will rely on $Invert_H$. We sometimes apply the action $\star_A$ on the elements of $S/\sim_S$ to obtain the canonical representative. Our secret key will consist of elements in $A \times H$. The updatable encryption from TOGA is described in Figure 3.

2.6. Class group action and orientations

The goal of this section is to introduce some preliminaries which are needed for instantiating TOGA.

```
Setup($1^\lambda$)
    $(A, H, S, \star_A, \sim_A, star_H) \leftarrow \mathcal{TOGA}(\lambda)$
    choose $\psi, s_0$
    pp$\leftarrow (A, H, S, \star_A, \sim_A, \star_H, \psi, s_0)$
    return pp

KeyGen(pp)
    $a \leftarrow\$ A$
    $h \leftarrow\$ H$
    return $Reduce_A(a), h$

TokenGen($k_e, k_{e+1}$)
    $(a_e, h_e) \leftarrow k_{e+1}$
    $(a_{e+1}, h_{e+1}) \leftarrow k_{e+1}$
    $c_e \leftarrow Reduce_A(a_e^{-1} a_{e+1})$
    compute $h = h(a_e^{-1} a_{e+1}, c_e^{-1})$
    return $c_e, h h_{e+1} h_e^{-1}$

Upd($\Delta_{e+1}, C_e$)
    $a, h \leftarrow \Delta_{e+1}$
    return $h \star_H (a \star_A C_e)$

Enc($k_e, M$)
    $r' \leftarrow\$ A$
    $r \leftarrow Reduce_A(r')$
    $s = Reduce_S(r \star_A s_0)$
    $(a_e, h_e) \leftarrow k_e$
    return $(\psi(M) h_e) \star_H (a_e \star_A s)$

Dec($k_e, C_e$)
    $(a_e, h_e) \leftarrow k_e$
    $b_e \leftarrow Reduce_A(a_e^{-1})$
    $h' \leftarrow h(a_e, b_e)$
    $s' \leftarrow (h_e h)^{-1} \star_H (b_e \star_A C_e)$
    $s \leftarrow Reduce_S(s')$
    $M' \leftarrow \psi^{-1}(Invert_H(s', s))$
    return $M'$
```

Figure 3: TOGA-UE

Definition 2.3. A K-orientation of E is an embedding: $\iota : K \hookrightarrow End(E) \otimes_{\mathbb{Z}} \mathbb{Q}$. (E, ι) is an $\mathcal{O}$-orientation if $\iota(\mathcal{O}) \subseteq End(E)$. It is primitive if $\iota(\mathcal{O}) = End(E) \cap$ $\cap \iota(K)$. In that case, the couple (E, ι) is called an $\mathcal{O}$-oriented curve and E is an $\mathcal{O}$-orientable curve.

If E is supersingular, then $End(E)$ is a maximal order in a quaternion algebra. Let $\varphi : E \to F$ be an isogeny. Then ϕ induces a K-orientation $\varphi_\star(\iota)$ on F.

$$\varphi_\star(\iota) = \frac{1}{deg(\varphi)} \varphi \circ \iota(\alpha) \circ \hat{\varphi}$$

If F already possesses a K-orientation, then the above isogeny is K-oriented if this orientation coincides with the orientation induced by φ A K-oriented isogeny $\psi : (E, \iota_E) \to (F, \iota_F)$ is a K-oriented isomorphism if it has an inverse isogeny $F \to E$ that is also K-oriented $(F, \iota_F) \to (E, \iota_E)$. With all the above knowledge, we can now build our TOGA.

Our main group will be the group of all invertible $\mathcal{O}$-ideals. Before introducing the set S, we need the following definition:

Definition 2.4. $S_{\mathcal{O}}(p)$ is the set of $\mathcal{O}$-oriented curves (E, ι) up to isomorphisms and Galois conjugacy.

When we consider invertible $\mathcal{O}$-ideals, we get an abelian group (for the multiplication operation). This group acts on the elements of $S_{\mathcal{O}}(p)$ transitively by an operation that we denote as $\star_A$. This action is computed concretely using isogenies. Given an integral ideal $\mathfrak{a}$, and $E, \iota \in S_{\mathcal{O}}$, the kernel of $\mathfrak{a}$ is

$$E[\mathfrak{a}] := \underset{\alpha \in \mathfrak{a}}{\cap} ker(\iota(\alpha)).$$

There exists a unique(up to isomorphism) separable isogeny with this kernel i.e., $\psi_{\mathfrak{a}} : E \longrightarrow E_{\mathfrak{a}} = E/E[\mathfrak{a}]$ and this also maps ι to $\iota_{\mathfrak{a}}$ which satisfying $\iota_{\mathfrak{a}}(x) = \frac{1}{N(\mathfrak{a})}\psi_{\mathfrak{a}} \circ \iota(x) \circ \hat{\psi_{\mathfrak{a}}}$. We can write the action of $\mathfrak{a}$ as:

$$\mathfrak{a} \star (E, \iota) := (F, (\psi_{\mathfrak{a}})_*(\iota)).$$

Since principle ideals act on the set trivially, we get an action of the ideal class group

$$\mathrm{cl}(\mathcal{O}) \times S_{\mathcal{O}}(p) \longrightarrow S_{\mathcal{O}}(p).$$

So here we have our first group A which is the invertible $\mathcal{O}$-ideals, and relation $\sim_A$ means that ideals differing by principal ideals are equivalent. Hence, the second group G is $\mathrm{cl}(\mathcal{O})$. In order to compute this group action efficiently, usually we choose a basis which consists of prime ideals $\mathfrak{l}_1, \ldots, \mathfrak{l}_n$. They are all split in $\mathcal{O}$, i.e., for any prime number l_i, $l_i\mathcal{O} = \mathfrak{l}_l\bar{\mathfrak{l}_i}$. Here $\bar{\mathfrak{l}_i}$ is the inverse ideal of $\mathfrak{l}_i$ in $\mathrm{cl}(\mathcal{O})$ which denoted as $\mathfrak{l}_i^{-1}$. The elements of this basis have small norm and represent elements of $\mathrm{cl}(\mathcal{O})$ as:

$$\mathfrak{a} = \prod_{i=1}^{n} \mathfrak{l}^{e_i}$$

where $(e_1, \ldots, e_n) \in \mathbb{Z}^n$. To get the third group action [18] uses the group of points of the orientable elliptic curves. Take N a split integer in $\mathcal{O}$ coprime with all the l_i for $1 \leq i \leq n$. Assuming $\mathcal{O} = \mathbb{Z}[\theta]$ and ν is an eigenvalue of θ, we can consider elements of the form (E, ι, P) where $P \in E[N] \cap ker(\iota(\theta) - \nu)$ is a generator of this cyclic group. Since N is split, ν exists. Let us take the set

$$S = \{(E, \iota, P) | (E, \iota) \in S_{\mathcal{O}}(p) \text{ and } \langle P \rangle = E[N] \cap ker(\iota(\theta) - \nu)\}$$

The group of fractional ideals of norm coprime to N acts on this set in the following way: any fractional ideal can be written as $(a/b)\mathfrak{a}$ where $\mathfrak{a}$ is an integral ideal, we have the action $(a/b)\mathfrak{a} \star (E, \iota, P) = (E_{\mathfrak{a}}, \iota_{\mathfrak{a}}, \psi^E_{\mathfrak{a}}(P))$. Indeed, if there are two equivalent fractional ideals $\mathfrak{a}_1, \mathfrak{a}_2$ act on $(E, \iota, P) \in S$, the result will be (E, ι, P') where P' is another generator of $E[N] \cap ker(\iota(\theta) - \nu)$. This means there is a $\mu \in H = \mathbb{Z}/N\mathbb{Z}^\times$ such that $P = \mu P'$. This μ depends only on two equivalent ideals $\mathfrak{a}_1, \mathfrak{a}_2$(by the principal ideal between them), and μ will be our $h(\mathfrak{a}_1, \mathfrak{a}_2)$ in the definition of TOGA.

In summary, for any scalar $h \in H$, we define $h \star_H (E, \iota, P)$ as $(E, \iota, [h]P)$ and one can show that $(A, H, S, \star_A, \sim_A, \star_H)$ is a TOGA.

Note that here our basis $\mathfrak{l}_i$ may not cover generate all the elements in $\mathrm{cl}(\mathcal{O})$, and there might be non trivial relations between them. In general, it is hard to compute the exact structure of the ideal class group $\mathrm{cl}(\mathcal{O})$ (even its cardinality is rather hard to compute). However, heuristically one may assume that the $\mathfrak{l}_i$ generate the class group, for further details see [26]. Hence, technically, this TOGA is not ETOGA if one uses a generic CSIDH instantiation. Here we used the group $A' = \langle \mathfrak{l}_1, \ldots, \mathfrak{l}_n \rangle$ is a subgroup of our main group A, but when restricting the exponents of $\mathfrak{l}_i$ to $[-B, B]$, such that$(2B + 1)^n \geq \#\mathrm{cl}(\mathcal{O})$, it's most likely they are evenly represented the elements of $cl(\mathcal{O})$. Alternatively, we can use the structure of $\mathrm{cl}(\mathcal{O})$ in CSI-FISH[2]. For the prime of CSIDH-512, they managed to compute the exact structure of $cl(\mathbb{Z}[\pi])$ which happens to be cyclic.

3. Updatable threshold encryption

In this section, we propose a new primitive called *updatable threshold encryption* (UTE) by combining updatable encryption and threshold encryption. We replace the secret key K_e in the updatable encryption scheme with a threshold secret key (distributed across multiple parties) and only K out of n parties can recover it, and therefore they can launch the encryption and decryption process that will be performed by a trusted party which is the same entity responsible for distributing the secret key K_e to the parties as shown in the Figure 4.

3.1. Motivation

The main motivation behind UTE scheme is that one might want to decrypt sensitive data now that stays secure for a long period of time. A potential (but not only) situation that would fit this scenario would be some military secret. In such scenarios a key compromise is a dangerous issue and updatable encryption seems like a viable candidate. However, if one requires long-term security, one might want to distribute trust amongst many parties as access structures might change throughout long periods of time. In a military scenario high ranking officers who have some sort access to classified data might get replaced

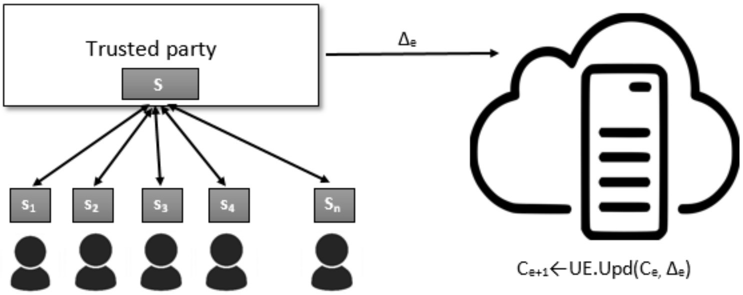

Figure 4: Threshold updatable encryption scheme

throughout time and it is paramount that unauthorised people do not leak classified data.

Other motivations could be to improve the security of the existing updatable encryption by using the threshold encryption or to capture the spirit of updatable encryption and key rotation schemes without performing the decryption. Nowadays it is a common requirement in some contexts, such as the payment card industry data security standard (PCI DSS) as mentioned by [15].

Suppose we have stored our data in some cloud, and we want to make sure it's secure for a long time. We need to update our secret key in the server and the encryption data in the cloud. After we updated the key, the server can distribute the secret key to n parties. We can use (k, n) threshold here to make sure at least k parties can recover the key and access the data.

3.2. Construction

We begin by formalizing the notion of our UTE scheme. In a UTE scheme, epoch e defines the timeframe during which a specific key remains valid and it is represented as an index incremented with each key update. A UTE scheme is the following tuple of algorithms:

1. **UTE.Setup**: Initiate the State of the System and setup the environment (Choose appropriate message space, appropriate map function,. . .).

2. **UTE.KeyGen**: A probabilistic algorithm to generate the secret key, say k_e for epoch e.

$$K_e \leftarrow\$ \, UTE.KeyGen() \tag{3.1}$$

3. **UTE.Enc**: The encryption algorithm uses k_e to encrypt the message M in some epoch e.

$$C \leftarrow UTE.Enc(k_e, M) \tag{3.2}$$

4. **UTE.Dis**: The distribution algorithm distributes this secret key k_e to n parties in a way that at least k parties are needed to recover the secret.

   The distribution algorithm of the secret key k_e to those n parties, returning a set of n shares $\{k_{e_1}, k_{e_2}, \ldots, k_{e_n}\}$, such that :

   $$k_{e_i} \leftarrow f(i), \forall i \in \{1, \ldots, n\} \tag{3.3}$$

   Where $f(i)$ is a secret polynomial of degree $k-1$ represented by the following Shamir's Secret Sharing equation:

   $$f(i) = \alpha + \sum_{j=1}^{k-1} c_j i^j \tag{3.4}$$

   With:

   $$\alpha \leftarrow \phi(k_e) \tag{3.5}$$

   Here ϕ is an invertible function chosen a priori to maps the secret key k_e to some $\mathbb{Z}/q\mathbb{Z}$ secret α, and clearly $f(0) = \alpha$.

5. **UTE.Rec**:The recovery algorithm enables the dealer (server) to recover the secret k_e once he receive the shares $\{k_{e_1}, k_{e_2}, \ldots, k_{e_n}\}$ from n parties with $n \geq k$(The threshold),by evaluating $f(0) = \alpha$ using Lagrange interpolation as follows:

   $$\alpha = \sum_{i \in S} k_{e_i} \cdot \prod_{j \in S \& j \neq i} \frac{j}{j-i} \tag{3.6}$$

   Once the dealer get the secret α,he easily can recover the secret key k_e from it by computing the inverse of $\phi(\alpha)$:

   $$k_e \leftarrow \phi^{-1}(\alpha) \tag{3.7}$$

6. **UTE.Dec**: A deterministic algorithm use the recovered key k_e of some epoch e to decrypt the ciphertext C_e and outputs some message M or $\perp$ (non valid message).

   $$M \leftarrow UTE.Dec(k_e, C) \tag{3.8}$$

7. **UTE.TokenGen**: Generating a token Δ_{e+1} for updating the ciphertexts.

   $$\Delta_{e+1} \leftarrow UTE.TokenGen(k_e, k_{e+1}) \tag{3.9}$$

8. **UTE.Update**: A deterministic algorithm used to update the ciphertext C_e to the next epoch using the generated token.

   $$C_{e+1} \leftarrow UTE.Update(\Delta_{e+1}, C_e) \tag{3.10}$$

3.3. Security of UTE

In this subsection, we will discuss the security notions of our scheme. Some notions come from updatable encryption and were already defined in Section 2. However, to make the section more self-contained we define them here as well.

(i) **Post-compromise Security:** No additional advantage is gained by an adversary who may compromise a secret key k_e of certain time epoch e to decrypt the ciphertexts of next epochs $e' > e$ that he will obtained after this compromise.

(ii) **UTE Forward security:** No additional advantage is gained by an adversary who may compromise a secret key k_e of certain epoch e to decrypt the ciphertexts of previous epochs $e' < e$ already obtained before.

(iii) **UTE detIND-atk:** For an UTE encryption scheme {UTE.Setup, UTE.KeyGen , UTE.Enc, UTE.Dis, UTE.Rec, UTE.Dec, UTE.TokenGen, UTE.Update} the advantage of an adversary, who has access to any set of unauthorised shares (i.e., a set of shares that does not have access to the secret), $\mathcal{A}$ for atk $\in \{CPA, CCA\}$ is

(3.11)
$$Adv_{UTE,\mathcal{A}}^{detIND-atk} = \left| Pr[Exp_{UTE,\mathcal{A}}^{detIND-atk-1} = 1] - Pr[Exp_{UTE,\mathcal{A}}^{detIND-atk-0} = 1] \right|.$$

(iv) **INT-CTXT:** Let UTE=UTE.Setup, UTE.KeyGen, UTE.Enc, UTE.Dec, UTE.Dis, UTE.Rec, UTE.TokenGen, UTE.Upd be an Updatable threshold encryption scheme, we define ciphertext integrity INT-CTXT of any adversary, who has access to any set of unauthorised shares (i.e., a set of shares that does not have access to the secret), $\mathcal{A}$ as:

$$Adv_{UTE,\mathcal{A}}^{INT-CTXT}(\lambda) = Pr[Exp_{UTE,\mathcal{A}}^{INT-CTXT} = 1] \tag{3.12}$$

Remark 3.1. Ciphertext integrity is strictly stronger than INT-PTXT,The INT-CTXT experiment for UTE scheme without share leakage is same as the one described for updatable encryption UE.

Remark 3.2. Note that the main difference between security definitions for UE and UTE is that an adversary has generally more information in a UTE scheme (as they can access a set of unauthorised shares).

4. UTE from isogenies

In this section we instantiate our updatable threshold encryption scheme based on CSIDH. We draw our main inspirations from [18] and [12]. First we instantiate our UTE that naturally fulfills all the security properties but requires a new update every time a share gets revoked. Then we propose a post-quantum dynamic secret sharing scheme and apply it in the UTE context to construct a scheme which can support revoking shares/providing shares without performing update operations.

4.1. Simple UTE construction

In order to get a quantum-resistant instantiation based on isogenies, we could use CSIDH [26] on GAINE [18, Section 3.1.] and it really makes our secret key distribution much easier. However, we face the challenge of encrypting messages, since it's hard to hash messages into a j value of some elliptic curve [5]. To circumvent this issue, we will use TOGA. Instead of hashing messages into elliptic curves, we can encrypt our messages by mapping them into a group, which then acts on the set of elliptic curves to achieve encryption.

To get our three group actions, we consider orientations on elliptic curves. We have already introduced some concepts of it in section 2.6. In CSIDH, the $\mathcal{O} = \mathbb{Z}[\sqrt{-p}] = \mathbb{Z}[\theta]$, and it can be treated as a special case of orientated elliptic curve, in which the orientation ι maps θ to the Frobenius morphism π: $(x, y) \mapsto (x^p, y^p)$.

For instantiating the updatable part, we need to fulfill the following steps.

1. Membership test: Verifing three things: First, E is supersingular and its j-invariant is in $\mathbb{F}_p$. These can all be accomplished efficiently (and are well-known in the context of CSIDH). Second ι is a correct orientation, this can be checked by verifying the norm and trace of $\iota(\theta)$. Third P is in $ker(\iota(\theta) - \mu)$ and has order N. What we need here is to perform efficient operations on the N-torsion and find an efficient way to compute a canonical point P_E of order N in $E[N] \cap ker(\iota(\theta) - \nu)$ from E, ι. Good news is all these operations are efficient and well studied.

2. Parameters finding: Choose a prime p has a form $c \prod_{i=1}^{n} l_i - 1$, with all the primes l_i being split in $\mathcal{O}$ and with efficient l_i-isogeny computations. Those l_i divide the norm of ideals which are the generators of our main group A. We also need to chose a smooth integer N which must be coprime with all the l_i in order to get our third group action. Usually this N is chosen to be a smooth divisor of $\#E(\mathbb{F}_{p^k})$ for a small value k for which can increasing the size of N torsion points.

3. Computing class group actions: This step may involve the computation of the lattice of relation which will be the major time consuming process. In general, it needs subexponential classical complexity due to it's equivalent to compute the structure of class group of $\mathcal{O}$. Fortunately, for the prime of CSIDH-512, this structure and the corresponding lattice of relations

was computed in CSI-fish[2]. One can find explicit instantiation of TOGA in the section 5 of [18].

From section 2.6, after we generate the secret key $(\mathfrak{a}, h)$ i.e., we should write it as $(e_1, \ldots, e_n, h)$. Each e_i has some range $[-B, B]$ for some small integer B. In CSI-FiSh [2], they managed to compute the structure of the class group of size CSIDH-512, which happens to be cyclic. In particular, they found out that the ideal $\mathfrak{l}_1 = \langle 3, \pi - 1 \rangle$ generates $\mathrm{cl}(Z[\pi])$. Once we have a generator of the class group, we can represent the secret ideal as an exponent which can be treated as an element in some finite field. Because the third group $H = \mathbb{Z}/N\mathbb{Z}^*$, we can concatenate the exponent of the secret ideal and $h \in H$, then map this concatenation to a finite field, say $\mathbb{Z}/q\mathbb{Z}$, where q need to fulfill some security level and the smallest factor of q should be lager than the number of parties(see [1] for details).

Then using Shamir's secret sharing scheme to distribute this number to m parties. If at least k of them want to recover the key and access the data, they could combine their share to recover the secret key k_e and use it to decrypt data.

Remark 4.1. The perfect security of Shamir's secret sharing scheme relies on $\mathbb{Z}/q\mathbb{Z}$ being a field, but this restriction can be loosed to a ring, as long as the denominator in Lagrange's formula is not "0". In order to allow more participants, one should choose $\mathfrak{l}_1^3$ or $\mathfrak{l}_1^{111}$. More details about this can be found in [12].

Remark 4.2. As SHINE scheme[6] and GAINE[18] did, here we can also use some nonce space $\mathcal{N}$ and concatenate $N \in \mathcal{N}$ with the secret key to make it random. This can make our secret key more secure.

If we only give the decryption permission to the server(i.e., a trusted party), then these parties have to send their share to the server, and the server will recover the key and decrypt data for them. If some party's share gets revoked, then the server would generate a new key k_{e+1} and update the ciphertext, and redistribute k_{e+1} to new n parties. We summary our scheme in the figure 5.

With above instantiation which is named as TOGA-UTE, we can achieve almost all the security properties defined in Section 3 (except detIND-CCA). The correctness of TOGA-UTE follows directly from the correctness of TOGA and Shamir's secret sharing scheme.

4.1.1. TOGA-UTE security

1. TOGA-UTE is **post compromise** share secure. Based on our construction, in epoch e, the secret key $(\mathfrak{a}_e, h_e)$ has no relation with the key in the future.

```
UTE.Setup($1^\lambda$)
    $(A, H, S, \star_A, \sim_A, star_H) \leftarrow \mathcal{TOGA}(\lambda)$
    choose $\psi, s_0, \phi$
    $\mathbb{F}_q \leftarrow \mathbb{Z}/q\mathbb{Z}$
    pp$\leftarrow (A, H, S, \star_A, \sim_A, \star_H, \psi, s_0, \pi)$
    return pp
```

```
UTE.TokenGen($k_e, k_{e+1}$)
    $(a_e, h_e) \leftarrow k_{e+1}$
    $(a_{e+1}, h_{e+1}) \leftarrow k_{e+1}$
    $c_e \leftarrow Reduce_A(a_e^{-1} a_{e+1})$
    compute $h = h(a_e^{-1} a_{e+1}, c_e^{-1})$
    return $c_e, h h_{e+1} h_e^{-1}$
```

```
UTE.Enc($k_e, M$)
    $r' \leftarrow\$ A$
    $r \leftarrow Reduce_A(r')$
    $s = Reduce_S(r \star_A s_0)$
    $(a_e, h_e) \leftarrow k_e$
    return $(\psi(M) h_e) \star_H (a_e \star_A s)$
```

```
UTE.Rec($\{\alpha_i\}$)
    $\alpha = \sum_{i \in S} \alpha_i \cdot \prod_{j \in S \& j \neq i} \frac{j}{j-i}$
    $k_e \leftarrow \phi^{-1}(\alpha)$
    Parse $k_e$ as $(a_e \| h_e)$
    return $(a_e, h_e)$
```

```
UTE.KeyGen(pp)
    $a \leftarrow\$ A$
    $h \leftarrow\$ H$
    return $Reduce_A(a), h$
```

```
UTE.Upd($\Delta_{e+1}, C_e$)
    $a, h \leftarrow \Delta_{e+1}$
    return $h \star_H (a \star_A C_e)$
```

```
UTE.Dis($k_e$)
    $c_1, \ldots, c_{k-1} \leftarrow\$ \mathbb{F}_q$
    $\alpha \leftarrow \phi(k_e)$
    $f(x) = \alpha + \sum_{i=1}^{k-1} c_i x^i$
    $\alpha_1, \ldots, \alpha_n = f(1), \ldots, f(n)$
    return $\alpha_1, \ldots, \alpha_n$
```

```
UTE.Dec($k_e, C_e$)
    $(a_e, h_e) \leftarrow k_e$
    $b_e \leftarrow Reduce_A(a_e^{-1})$
    $h' \leftarrow h(a_e, b_e)$
    $s' \leftarrow (h_e h)^{-1} \star_H (b_e \star_A C_e)$
    $s \leftarrow Reduce_S(s')$
    $M' \leftarrow \psi^{-1}(Invert_H(s', s)$
    return $M'$
```

Figure 5: TOGA UTE Scheme

2. TOGA-UTE has **forward security**. UTE ensure the forward security directly. An adversary $\mathcal{A}$ who compromises (only) a secret key of some epoch e cannot decrypt the ciphertexts which he obtained in a previous epoch e' s.t. $e' < e$.

3. **detIND-CPA**

Theorem 4.1. ***TOGA-UTE is detIND-CPA.*** *For the UTE discussed in the Figure 5. Let $\mathcal{GA}$ group action family where $\mathcal{GA}(1^\lambda)$ is $(G, T, \star_G)$ induced by $A \in \mathcal{TOGA}(1^\lambda)$. $\forall$ Adversary $\mathcal{A}$, there exist a reduction $\mathcal{B}$ s.t.*

$$Adv^{detIND-CPA}_{TOGA-UTE,\mathcal{A}}(\lambda) \leq \mathcal{O}(1)(n+1)^3 \cdot Adv^{Weak-PR}_{\mathcal{GA},\mathcal{B}}(\lambda) \tag{4.1}$$

$Adv^{Weak-PR}$ represent the advantage of weak pseudorandom group action (a group action with a specific property that an adversary cannot distinguish tuples $(x_i, g \star x_i)$ from those of the form (x_i, v_i) where $g \in$ $\in G$ is a secret group element chosen randomly and x_i and v_i are sampled uniformly from X.) as defined by Navid Alamati et al. in [1].

Proof. Because in the TOGA-UTE, the updatable part is similar to updatable encryption in the [18], we follow the proof strategy of it. ■

4. **detIND-CCA**: As the definition on section 3, it seems hard that TOGA-UTE can achieve detIND-CCA. We leave this as future work.

5. **Shares leakage security** As our UTE scheme is for threshold (m, k), therefore even $k - 1$ shares got leaked, they can't reconstruct the secret key from it.

There are two additional scenarios that require consideration: first, there is the possibility that an individual's share could be disclosed to unauthorized parties, who may then collude with others to reconstruct the secret key. Second, should an individual's share be lost, the (m, k) threshold would effectively become an $(m - 1, k)$ threshold. These situations can be remedied by having a server update the secret key and redistribute new shares to the participants. Nonetheless, there exists the potential issue that the server may not perform the update promptly. In such instances, consider the verification function which will be in the dynamic secret share.

4.2. Dynamic secret sharing scheme

From the above one can see that we need to redistribute the key whenever some share gets revoked. However, this may be considered as system resource consuming and sometimes impractical. To eliminate these weaknesses, we may use a dynamic secret sharing scheme which allows some participants to join (exit) dynamically without changing the shares of the other old participants. With the development of secret sharing, we have a lot of options, for example [7] and [21]. In 2015, Mashhadi et al. [20] constructed two verifiable multi-secret sharing schemes, which can realize that the number of participants can be dynamically changed, and multiple secrets can be shared. Also in 2019, Yuan et al. [28] presented a fully dynamic secret sharing scheme, which can adapt to the changed access structure and the shared secrets. Recently in [19] they claim that they build a fully dynamic multi-secret sharing scheme with redundant authorization, which has a general access structure. All the schemes we mentioned above have their own advantages and disadvantages. However, they are not quantum-resistant.

In CSIDH, the secret key consists of multiple values, more specifically, $k_e = (e_1, \ldots, e_n, h)$ which can be treated as $n+1$ secrets. We should use multi-use and multi-secret sharing. A secret sharing scheme(SSS) is said to be multi-use if even after a secret is reconstructed by some participants, the combiner cannot misuse their submitted information to reconstruct some other secrets. In making a scheme multi-use, the participants provide the combiner not with the original share but with a shadow or image of that share, which is actually an entity that depends on the original share. This image or shadow is known as the pseudo-secret share. Moreover, we will use group actions on elliptic curve based on CSIDH to make this SSS quantum resistant.

Note that users should choose these two secret sharing schemes according to the balance of updating and secret sharing. For example, if they don't want to update frequently, and they should use dynamically secret share, on the other hand, if they want to update the data every month, so to speak, then it's more convenient to treat the secret key just as one share by using Shamir's secret sharing.

Inspired by [19], although it has some mistakes, we construct the first post-quantum fully dynamical secret sharing scheme. We will present our SSS in three main stages: initialization, construction, and recovery. After this, the correctness, verifiability, and securities of our SSS are analyzed.

Before going into the details, we will highlight the mistakes on [19].

1. In Section 3.1 of [19] "Each participant $P_i (i = 1, 2, ..., n)$ selects an integer $r_i \in \mathbb{Z}_p^*$ as his (her) own share and calculates $R_i = r_i \cdot S, B_i = r_i' \cdot S$, where $r_i' = R_i - r_i$." Here R_i is a point of elliptic curve, yet r_i is an integer in $\mathbb{Z}_p^*$.The operation "$-$" is undefined in this context, and thus B_i is nonsensical.

2. In Section 3.2 "D calculates $t_i = j \cdot R_i = j \cdot r_i S$ by combining j with R_i for $i = 1, 2, \ldots, n$, where t_i is recorded as the pseudo-share of the participant P_i". Again t_i is supposed to be a point on elliptic curve, but the authors incorrectly treat them as integers in order to do the $\oplus$ operation with integers M_i and C in step 3.

We will fix all the mistakes in [19] and present a more generalized version. Furthermore we will make a post-quantum secret sharing scheme. Before that, let's provide some comparison and short description of our dynamic scheme which should make the scheme more understandable.

Secret sharing schemes

name	Shamir's sharing scheme	Dynamic Secret sharing scheme
number of sceres	one(S)	multiple($S_1, S_2, \dots, S_m$)
number of polynamials	one: $S+a_1x^1+\cdots+a_{k-1}x^{k-1}$	multiple: $(S \oplus C) + a_1x^1+\cdots+a_{k-1}x^{k-1}$ and some other polynamials
shares	dealer chooses the shares for parties	parties can choose their own shares which are called pseudo-share.
dynamic	no	yes
public bulletin board	no	yes

Figure 6: Comparison table of Shamir and Dynamic secret sharing schemes

The following conditions are needed for the dynamic secret sharing scheme:

1. m shared secrets: $S_1, S_2, \dots, S_m$.

2. n participants: $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$.

3. $H(\cdot)$ is some suitable hash function.

4. We requires a public bulletin board for all public content, only the secret dealer D can store and update the contents and others can only view or download them.

5. We denote the access structure AS $= \{\gamma_1, \gamma_2, \dots, \gamma_d\}$, where d is represented as the number of qualified subsets in AS.

6. For any qualified subset $\gamma_h = \{\mathcal{P}_{h1}, \dots, \mathcal{P}_{h|\gamma_h|}\}$ in AS, where $1 \leq h \leq d$, $|\gamma_h|$ represents the number of participants in the qualified subset γ_h, and $k \leq |\gamma_h| \leq n$(k is represented as the threshold).

4.2.1. Initialization

1. Dealer(D) chooses a supersingular elliptic curve E_0 over a finite filed $\mathbb{Z}/q\mathbb{Z}$, and randomly chooses a secret ideal $\mathfrak{a}_0$.

2. Each participant $\mathcal{P}_i (i = 1, 2, \dots, n)$ selects an ideal $\mathfrak{a}_i$ as his (her) own secret and calculates $E_i = \mathfrak{a}_i * E_0$, Then $\mathcal{P}_i$ saves $\mathfrak{a}_i$ in privacy and sends E_i to D.

3. D needs to ensure the uniqueness of E_i for $i = 1, 2, \dots, n$. In other words, D needs to determine whether $E_1, E_2, \dots, E_n$, are different with

each other. If not, D has to ask the corresponding participants to re-select and resend E_i until they are different with each other.

4. D publishes (id_i, E_i) and $g_0(x)$, where $id_i = i$ for $i = 1, 2, \ldots, n$ and $g_0(x)$ is a polynomial which is constructed by using the n number pairs (id_i, E_i), here E_i are elliptic curves, but we could use Montgomery form, i.e., $y^2 = x^3 + Ax^2 + x$ and the number A can uniquely determine a curve up to $\mathbb{F}_p$-isomorphism. id_i is the unique identity information of the participant $\mathcal{P}_i$.

4.2.2. Construction

1. D randomly pick an ideal $\mathfrak{a}_0$ as his or her private key, and publishes $E_D = \mathfrak{a}_0 * E_0$ and $T = H(S_1 \| S_2 \| \cdots \| S_m)$, where H is a hash function and E needs to be ensured that $E_D \neq E_i$ for $i = 1, 2, \ldots, n$.

2. D calculates $t_i = \mathfrak{a}_0 * E_i = (\mathfrak{a}_0 \cdot \mathfrak{a}_i) * E_0$ for $i = 1, 2, \ldots, n$, where t_i is recorded as the pseudo-share of the participant $\mathcal{P}_i$.

3. D calculates $T_i = x_i \oplus t_i \oplus C$. $x_i = M_i * E_0$ for $i = 1, 2, \ldots, n$, where the selections of M_i, which is an ideal class in $\mathrm{cl}(\mathcal{O})$ and C is an Elliptic curve over $\mathbb{F}_p$ which equals to $c * E_D$ for some random ideal c. We need to ensure that $x_1, x_2, \ldots, x_n, E_D, E_1, E_2, \ldots, E_n$, are different with each other and C, t_i are also different with each other, "$\oplus$" is denoted as XOR operation(Note we can do this because we use Montgomery form of elliptic curves). Subsequently, D saves C in privacy and publishes T_i for $i = 1, 2, \ldots, n$, where C is represented as the redundancy.

4. D constructs a polynomial $g(x)$ of degree $k-1$ by using S_1, C and randomly selected $k-1$ integers $a_1, \ldots, a_{k-1} \in \mathbb{Z}/q\mathbb{Z}$, where $a_{k-1} \neq 0$.

$$g(x) = (S_1 \oplus C) + a_1 x + \cdots + a_{k-1} x^{k-1}. \tag{4.2}$$

5. D calculates the n number pairs $v_i = (x_i, y_i)$, where $y_i = g(x_i)$ for $i = = 1, 2, \ldots, n$.

6. D publishes two polynomials $g_1(x)$ and $g_2(x)$, where $g_1(x)$ is constructed by using the n number pairs $v'_i = (t_i, x_i)$ and $g_2(x)$ is constructed by using the n number pairs $v''_i = (t_i, y_i)$ for $i = 1, 2, \ldots, n$.

7. D constructs a new polynomial $f_1(x)$ by using S_1 and $a_1, \ldots, a_{k-1}$ in step 4:

$$f_1(x) = S_1 + a_1 x + \cdots + a_{k-1} x^{k-1}. \tag{4.3}$$

8. D calculates $f_1(2), f_1(3), \ldots, f_1(m)$ by using (4.3) and $\{2, 3, \ldots, m\}$, and constructs the following $m-1$ polynomials of degree 1 containing the remaining $m-1$ secrets $S_2, \ldots, S_m$:

$$
\begin{aligned}
f_2(x) &= S_2 + f_1(2)x, \\
f_3(x) &= S_3 + f_1(3)x, \\
&\vdots \\
f_m(x) &= S_m + f_1(m)x.
\end{aligned}
\tag{4.4}
$$

9. D calculates the $m-1$ number pairs $w_j = (j, f_j(j))$ by using (4.4), where $j = 2, 3, \ldots, m$, and constructs a polynomial $g_3(x)$ by using the $m-1$ number pairs $w_2, w_3, \ldots, w_m$. Then, D publishes the polynomial $g_3(x)$.

4.2.3. Recovery

Without loss of generality, we assume that the k participants $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \ldots \mathcal{P}_k\} = \gamma \subset \gamma_h$ want to recover the m shared secrets, then they need to obtain the redundant C from the secret dealer D and exchange (id_i, t_i) with each other, where $i \in \{1, 2, \ldots, k\}$. The specific recovery steps are as follows:

1. Each participant in the set γ will send a request to D for recovering the shared secrets. If the request is passed, D will generate $T_i^1 = c*(\mathfrak{a}_0 * E_i) = = (c \cdot \mathfrak{a}_0 \cdot \mathfrak{a}_i) * E_0$ and send it to the corresponding participant $\mathcal{P}_i$ for $1 \leq i \leq k$. The participant $\mathcal{P}_i$ can obtain the redundancy C by using their own secret ideal $\mathfrak{a}_i$ i.e., $C = \mathfrak{a}_i^{-1} * T_i^1 = c * E_D$.

2. The participant $\mathcal{P}_i$ will exchange pseudo-share with other participants in the set γ, thus obtaining the pseudo-shares t_a of other participants $\mathcal{P}_a$, where $i, a \in \{1, 2, \ldots, k\}$, and $a \neq i$.

3. Do the verification procedure(see bellow 4.2.4). After this, we assume that the (id_i, t_i) provided by these k participants are correct and valid, $i = 1, 2, \ldots, k$, then each participant will use the published polynomials $g_1(x)$ and $g_2(x)$ to generate the k number pairs $v_i = (x_i, y_i)$ for $i = = 1, 2, \ldots, k$. Secondly, the polynomial $g(x)$ can be reconstructed by using the k number pairs v_i , and then the polynomial $f_1(x)$ can be obtained by using $g(x)$ and C. Finally, the shared $S_1 = f_1(0)$ is recovered.

4. Participants compute $f_1(2), f_1(3), \ldots, f_1(m)$ by using the polynomial $f_1(x)$ in step 3, and $f_j(j) = g_3(j)$ can be obtained by using the published polynomial $g_3(x)$, and then the $m-1$ secrets $S_2, \ldots, S_m$ can be reconstructed by using $f_1(j)$ and $(j, f_j(j))$ in 4.4.

5. The m shared secrets $S_1, S_2, \ldots, S_m$ can be verified with published $T = = H(S_1 \| S_2 \| \cdots \| S_m)$.

4.2.4. Verification

Each participant $\mathcal{P}_i$ can verify the honesty of other participants $\mathcal{P}_a$, where $1 \leq a, i \leq |\gamma_h| \leq n$, and $a \neq i$. For example, $\mathcal{P}_a$ sends his or her pseudo-share t'_a to $\mathcal{P}_i$, then $\mathcal{P}_i$ determines the honesty of the participant $\mathcal{P}_a$ by judging whether the equation

$$T_a \oplus C \oplus t'_a = x_a = g_1(t'_a) \tag{4.5}$$

holds. If (4.5) holds, $\mathcal{P}_i$ determines $\mathcal{P}_a$ is honest, otherwise $\mathcal{P}_a$ is dishonest.

4.3. Dynamic properties

With the above scheme, we can add new participants, delete old participants, change shared secrets, add a new qualified subset, and delete a qualified subset without redistributing the shares every time.

4.3.1. Add a participant

1. D choose which access set should add $\mathcal{P}_{n+1}$.
2. Then $\mathcal{P}_{n+1}$ picks his or her secret $\mathfrak{a}_{n+1}$, calculates $\mathfrak{a}_{n+1} * E_0 = E_{n+1}$ and sends E_{n+1} to D. They both confirm the pseudo-share of $\mathcal{P}_{n+1}$ is $t_{n+1} = \mathfrak{a}_0 * E_{n+1} = \mathfrak{a}_{n+1} * E_D$.
3. D calculates and publishes $T_{n+1} = x_{n+1} \oplus t_{n+1} \oplus C$, the $x_{n+1} = M_{n+1} * E_0$.
4. D calculates $y_{n+1} = g(x_{n+1})$ and gets the new number pair $v_{n+1} = = (x_{n+1}, y_{n+1})$.
5. D updates the polynomials $g_1(x)$ and $g_2(x)$ to $g'_1(x)$ and $g'_2(x)$ by using these new $n+1$ pairs $v_i, i = 1, 2, \ldots n+1$. Moreover, D constructs a new polynomial $g_0(x)$ by using $n+1$ pairs (id_i, E_i). This polynomial will be used for the judgement of the recovery request.

Note that when new participants are added, the shares of the old participants remain the same.

4.3.2. Delete a participant

If we want to delete $\mathcal{P}_u$ from the access set γ_h, for convenience, we assume AS be a minimal access structure. Then we only need to delete the γ_h from the access structure AS and delete (id_u, E_u) of the participant $\mathcal{P}_u$ from the public bulletin board.

4.3.3. Update some secrets

We denote S_i^e as the epoch e updates of the i-th secret. The Dealer only needs to change any information related to the old secret S_i, some polynomials and T to $T^e = H(S_1\| \cdots \|S_i^e \cdots \|S_m)$. One important update is the redundancy C, because we used "$\oplus$" to hide our secret. We summarize the updates here: $C, g(x), T_i^1, g_j(x)$(for $j = 1, 2, 3$) and $f_i(x)$ to $C^e, g^e(x), T_i^e, g_j^e(x)$ and $f_i^e(x)$.

Note that no matter how many times the secrets are changed, D only needs to save and update C^e in privacy, the shares of the participants remain the same.

Adding and deleting an access structure are similar to above adding and deleting participants, we left this to interested readers.

4.4. Correctness and verifiability of the secret sharing scheme

4.4.1. Correctness

If both participants and Dealer honestly implemented the scheme, then any qualified set can recover the shared secrets.

Indeed, without loss of generality, we may assume k participants $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k\}$ of a qualified set $\gamma_h = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n\}$,where n is the cardinality of γ_h, want to recover the secrets.

Step 1 These participants obtain the E_D from the public bulletin board and calculate their own pseudo-shares $t_i = \mathfrak{a}_i * E_D$ for $1 \le i \le k$.

Step 2 D sends $T_i^1 = c * (\mathfrak{a}_0 * E_i) = (c \cdot \mathfrak{a}_0 \cdot \mathfrak{a}_i) * E_0$ to the corresponding participant $\mathcal{P}_i$ for $1 \le i \le k$. so that the redundancy $C = \mathfrak{a}_i^{-1} * T_i^1 = c * E_D$ can be obtained. And then, the participant $\mathcal{P}_i$ exchanges pseudo-share with other participants $\mathcal{P}_a$ for $a \in \{1, 2, \ldots k\}$, and $i \ne a$.

Step 3 When the corresponding participants have obtained the redundancy C and the k correct pseudo-shares $t_1, t_2, \ldots, t_k, 1 \le k \le n$, they will generate x_i and y_i by using the published polynomials $g_1(x)$ and $g_2(x)$, respectively, where $x_i = g_1(t_i), y_i = g_2(t_i)$. That is to say, these participants got the k pairs (x_i, y_i).

Step 4 They can use these k pairs to recover the polynomial $g(x)$ using Lagrange interpolation formula:

$$(4.6)\quad g(x) = \sum_{i=1}^{k} y_i \prod_{j=1, j\ne i}^{k} \frac{x - x_j}{x_i - x_j} = (S_1 \oplus C) + a_1 x + \cdots + a_{k-1} x^{k-1} \ (mod\ p)$$

After this, based on the information of C and $g(0)$, they can recover the secret S_1 and get

$$f_1(x) = S_1 + a_1 x + \cdots + a_{k-1} x^{k-1} \tag{4.7}$$

Subsequently, calculate $f_1(2), f_1(3), \dots, f_1(m)$, and combine the $m-1$ number pairs $w_i = (i, g_3(i)) = (i, f_i(i))$ to recover the other secrets $S_2, \dots, S_m$ by using those linear functions $f_i(x)$.

4.4.2. Verifiability of the dealer D

For verifying the honesty of D, participants can check whether

$$T_a \oplus C \oplus t_a = x_a = g_1(t_a)$$

holds or not, where $t_a = \mathfrak{a}_a * E_D$ and $T_a = x_a \oplus t_a \oplus C$.

4.4.3. Verifiability of participants

In order to check the honesty of the participant $\mathcal{P}_u$, others need to calculate whether the formula

$$(T_u \oplus C) \oplus t'_u = x_u = g_1(t'_u)$$

holds, where t'_u is a pseudo-share of the participant $\mathcal{P}_u$ which sent to others, T_u is a public value and C is the redundancy value computed by other participants. If it holds, then $\mathcal{P}_u$ is honesty, otherwise not.

4.5. Security analysis of the secret sharing scheme

4.5.1. External attacks

The external attackers want to use some public information or intercept the communication of participants, so that they can reconstruct the shared secrets. However, this can be prevented (Even for a quantum computer) due to the following reasons:

(i) The isogeny problem through group action on the elliptic curve is quantum resistance. The attacker can't get the secret ideal of Dealer from a started curve E_0 and the ended curve E_D.

(ii) Even the attackers can listen the communication and get t_i and T_i^1, they can only reconstruct polynomial $g(x)$ by using t_is and the public polynomials $g_1(x)$ and $g_2(x)$. But they cannot reconstruct $f_1(x)$ since it's quantum resistant for getting the secret ideal $\mathfrak{a}_i$ of $\mathcal{P}_i$ from $t_i = \mathfrak{a}_i * E_D$. Meanwhile, they can't get c from $T_i^1 = c * (\mathfrak{a}_0 * E_i)$.

4.5.2. Internal attacks

There might be some dishonest parties exist during the secret recovery, and they provide false pseudo-shares to other honest parties. Then they can obtain the shared secrets, but honest parties couldn't. Our verification process 4.2.4 can prevent this kind of attacks. However, in this setting, we need a trusted combiner to verify the honesty of the parties and reconstruct the secrets.

4.6. Applying dynamic secret sharing to UTE

With the above new secret sharing scheme, we can make our UTE more flexible, for which we don't need the concatenation and update the secret shares frequently. Recall that our secret key is a vector (e, h), then we can treat one secret key as 2 secrets, i.e. $S_1 = e, S_2 = h$, and apply our dynamical secret sharing scheme(section 4.2).

Note:

(i) In the above dynamical SSS, we used the idea of CSIDH to make our SSS quantum resistant. However, one can of course use any other group actions.

(ii) We publish the hash value $T = H(S_1 \| S_2)$, in order to make sure the integrity of the secrets. Because sometimes the shared secrets can be huge. But here we can omit that, since we only have two secrets. Alternatively, one can represent the concatenation of e and h as an element S in a finite field, then randomly generated S_i, where $i = 1, 2, \ldots, m$, whose sum is S.

(iii) If one just want to use CSIDH, the secret exponents e_i are really small, usually $-5 \leq e_i \leq 5$. Although n is large($n = 74$ in CSIDH), but evaluating hash function is much faster than evaluating group action. We leave this secret sharing without concatenation as future work.

For simplicity and clearness, we make the following table 7 for all the information of secret sharing. In conclusion our scheme has the following properties:

(i) The share of each party is selected by their own, and after the shared secrets are recovered, the share of each participant can be reused;

(ii) The scheme is verifiable secret sharing.

(iii) The scheme is fully dynamic, which allows parties to add (delete) dynamically, and also allows the access structure and the shared secrets to be changed dynamically, while the shares of the participants do not need to be changed.

E_0	starting supersingular elliptic curve over $\mathbb{F}_p$	published
$\mathfrak{a}_0$	Dealer's secret ideal	private
E_D	elliptic curve $= \mathfrak{a}_0 * E_0$	published
T	hashed value $= H(e_1 \| e_2 \| \cdots \| h)$	published
$\mathfrak{a}_i$	secret ideal of i-th party	private
E_i	elliptic curve $= \mathfrak{a}_i * E_0$	send to the Dealer
t_i	$\mathcal{P}_i$'s pseudo-share $= \mathfrak{a}_i * E_D$	used for recovery
$g_0(x)$	polynomial generated by the Dealer	published(for verifying identities)
C	elliptic curve(redundancy) $= c * E_D$	private
$g(x)$	polynomial $= (S_1 \oplus C) + a_1x + \cdots + a_{k-1}x^{k-1}$	private
x_i	elliptic curve $= M_i * E_0$	private
y_i	value $= g(x_i)$	private
T_i	public value $= x_i \oplus t_i \oplus C$	send to $\mathcal{P}_i$
$g_1(x)$	generated using pairs (t_i, x_i)	published
$g_2(x)$	generated using pairs (t_i, y_i)	published
$f_1(x)$	polynomial $= S_1 + a_1x + \cdots + a_{k-1}x^{k-1}$	private
$f_i(x)$	polynomial $= S_i + f_1(i)x$	private
$g_3(x)$	polynomial generated using pairs $(i, f_i(i))$	public
T_i^1	first epoch value $= c * (\mathfrak{a}_0 * E_i) = (c \cdot \mathfrak{a}_0 \cdot \mathfrak{a}_i) * E_0$	send to $\mathcal{P}_i$ for getting value C

Figure 7: Dynamic secret sharing scheme

Note: If some access parties want to get the data, they need to communicate each other to recover all the secrets which later will be used as decrypt key. Either they can decrypt the data by themselves, or they have to talk to the trust server and the server will decrypt the data for them. In any case, once the key has been recovered, the server should generate a new key and update the ciphertexts. This time it doesn't need to redistribute the new key.

5. Conclusion

In this paper, we have introduced an updatable Threshold Encryption (UTE) that combines the merits of updateable encryption and threshold encryption. We have also instantiated a quantum-resistant UTE using TOGA and secret sharing schemes. It is worth mentioning that our primitive can also be implemented using conventional group actions, such as the discrete logarithm in elliptic curves, along with Shamir's secret sharing. Additionally, we have introduced a post-quantum dynamic secret sharing scheme, which adds flexibility and applicability to the UTE.Finally, we believe our UTE can be utilized in numerous real-world scenarios. For future work, researchers can explore more quantum-resistant instantiations.

References

[1] **Alamati, N., L. De Feo, H. Montgomery and S. Patranabis,** Cryptographic group actions and applications, *Advances in Cryptology - ASIACRYPT 2020, Springer International Publishing*, (2020), 411–493.

[2] **Beullens, W., T. Kleinjung and F. Vercauteren,** CSI-FiSh: efficient isogeny based signatures through class group computations, *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, (2019), 227–247.

[3] **Boneh, D., K. Lewi, H. Montgomery and A. Raghunathan,** Key homomorphic PRFs and their applications, *Advances in Cryptology - CRYPTO 2013*, Springer, Berlin, Heidelberg, (2013), 410–428.

[4] **Boneh, D., X. Boyen and S. Halevi,** Chosen ciphertext secure public key threshold encryption without random oracles, *Topics in Cryptology - CT-RSA 2006*, Springer, Berlin, Heidelberg, (2006), 226–243.

[5] **Booher, J., R. Bowden, J. Doliskani, T. Fouotsa, S.D. Galbraith, S. Kunzweiler, S.P. Merz, C. Petit, B. Smith, K.E. Stange and others**, Failing to hash into supersingular isogeny graphs, *arXiv preprint arXiv:2205.00135*, (2022).
https://eprint.iacr.org/2022/518.pdf

[6] **Boyd, C., G.T. Davies, K. Gjøsteen and Y. Jiang,** Fast and secure updatable encryption, *Advances in Cryptology - CRYPTO 2020*, Springer International Publishing, (2020), 464–493.

[7] **Cachin, C.,** On-line secret sharing, *IMA International Conference on Cryptography and Coding*, Springer, (1995), 190–198.

[8] **Chen, L., Y. Li and Q. Tang,** CCA Updatable encryption against malicious re-encryption attacks, *Advances in Cryptology - ASIACRYPT 2020*, Springer International Publishing, (2020), 590–620.

[9] **Colò, L. and D. Kohel,** Orienting supersingular isogeny graphs, *Cryptology ePrint Archive*, Paper 2020/985, (2020).
https://eprint.iacr.org/2020/985.

[10] **Couveignes, J.M.,** Hard homogeneous spaces, *Cryptology ePrint Archive*, (2006).

[11] **Das, A. and A. Adhikari,** An efficient multi-use multi-secret sharing scheme based on hash function, *Applied Mathematics Letters*, **23** (2010), 993–996.

[12] **De Feo, L. and M. Meyer,** Threshold schemes from isogeny assumptions, *Public-Key Cryptography-PKC 2020: 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography*, Edinburgh, UK, Springer, (2020), 187–212.

[13] **Devevey, J., B. Libert, K. Nguyen, T. Peters and M. Yung,** Non-interactive CCA2-secure threshold cryptosystems: Achieving adaptive security in the standard model without pairings, *Cryptology ePrint Archive*, Paper 2021/630, (2021).
https://eprint.iacr.org/2021/630.

[14] **Erwig, A., S. Faust and S. Riahi,** Large-scale non-interactive threshold cryptosystems in the YOSO model, *Cryptology ePrint Archive*, Paper 2021/1290, (2021).
https://eprint.iacr.org/2021/1290.

[15] **Everspaugh, K. Paterson, T. Ristenpart and S. Scott,** Key rotation for authenticated encryption, *Advances in Cryptology - CRYPTO 2017*, Springer International Publishing, (2017), 98–129.

[16] **Klooß, M., A. Lehmann and A. Rupp,** (R)CCA secure updatable encryption with integrity protection, *Cryptology ePrint Archive*, Paper 2019/222.
https://eprint.iacr.org/2019/222.

[17] **Lehmann, A. and B. Tackmann,** Updatable encryption with post-compromise security, *EUROCRYPT*, Springer, **3** (2018), 685–716.

[18] **Leroux, A. and M. Roméas,** Updatable encryption from group actions, *Cryptology ePrint Archive*, Paper 2022/739, (2022).
https://eprint.iacr.org/2022/739.

[19] **Li, F., H. Hu, S. Zhu and J. Yan,** A fully dynamic multi-secret sharing scheme with redundant authorization, *Cryptography and Communications*, Springer, (2022), 1–18.

[20] **Mashhadi, S. and M.H. Dehkordi,** Two verifiable multi secret sharing schemes based on nonhomogeneous linear recursion and LFSR public-key cryptosystem, *Information Sciences*, Elsevier, **294** (2015), 31–40.

[21] **Pinch, R.,** On-line multiple secret sharing, *Electronics Letters*, Citeseer, **32** (1996), 1087–1088.

[22] **Sandhya Sarma, K.N., H.S. Lamkuche and S. Umamaheswari,** A review of secret sharing schemes, *Research Journal of Information Technology*, **5** (2013), 67–72.

[23] **Shamir, A.,** How to share a secret, *Communications of the ACM*, ACm New York, NY, USA, **22** (1979), 612–613.

[24] **Shor, P.W.,** Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM Review*, **41** (1999), 303–332.

[25] **Tjell, K. and R. Wisniewski,** Privacy in distributed computations based on real number secret sharing, *arXiv*, (2021).

[26] **Wouter, C., L. Tanja, M. Chloe, P. Lorenz and R. Joost,** CSIDH: an efficient post-quantum commutative group action, *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, (2018), 395–427.

[27] **Yong-Jun, G., F. Xiao-Hong and H. Fan,** A new multi-secret sharing scheme with multi-policy, *The 9th International Conference on Advanced Communication Technology, IEEE*, **3** (2007), 1515–1517.

[28] **Yuan, J. and L. Li,** A fully dynamic secret sharing scheme, *Information Sciences*, Elsevier, **496** (2019), 42–52.

C. He and F. Kouider
Eötvös Loránd University (ELTE)
Faculty of Informatics
Budapest
Hungary
chenfenghe@inf.elte.hu
fatnakouider@inf.elte.hu

P. Kutas
Eötvös Loránd University (ELTE) and University of Birmingham
Budapest and Birmingham
Hungary and England
kutasp@gmail.com