Efficient identity-based threshold decryption scheme from bilinear pairings ☆

Wei Gao[a,b], Guilin Wang[c], Kefei Chen[a], Xueli Wang[d], Guoyan Zhang[e]

[a] *Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China*
[b] *School of Mathematics and Information, Ludong University, Yantai 264025, China*
[c] *School of Computer Science & Software Engineering, University of Wollongong, NSW 2522, Australia*
[d] *School of Mathematics, South China Normal University, Guangzhou 510631, China*
[e] *School of Computer Science and Technology, Shandong University, Jinan 250100, China*

Abstract

Taking advantage of a technique that allows to safely distribute a private key among decryption servers we introduce a new identity-based threshold scheme, proven secure in the random oracle model. This new paring-based scheme features a lot of improvements compared to other schemes that can be found in the literature. Among them the two most noticeable ones are, the efficiency, by drasticaly reducing the number of pairing computations, and the ability for a user to generate and share a private key without requiring any access to a PKG.

Keywords:
Identity-based cryptography, Threshold cryptography, Provable security, Random oracle model, Bilinear pairing, Identity-based threshold decryption

1. Introduction

Identity based (ID-based) cryptography was proposed by Shamir in 1984 [1] to simplify key management and remove the public key certificates. In ID-based cryptography, the identity of a user, such as his/her e-mail address, is taken as the public key and so the certificate for certifying the public key is not needed. The secret key, calculated from the public key (identity of a user), is issued by a trusted authority called private key generator (PKG). The first practical ID-based encryption (IBE) scheme was proposed in 2001 by Boneh and Franklin [2], which was proved to be secure against adaptive chosen ciphertext attack in random oracle model. Since then, in the so-called identity-based cryptography field, many ID-based cryptographic schemes have been proposed [3]. Bilinear pairing [2] is the most popular tool to construct identity-based cryptographic primitives. Due to the various applications of bilinear pairings, the so-called pairing-based cryptography is becoming one of hot topics in cryptography.

Threshold cryptography [4, 6, 7], mainly including signature and encryption, increases the availability of the cryptographic operations which need involvement of secret keys, and at the

☆This work is partially supported by National Natural Science Foundation of China (No. 60970111, No. 60973135).
Email addresses: sdgaowei@gmail.com (Wei Gao), guilin@uow.edu.au (Guilin Wang), kfchen@sjtu.edu.cn (Kefei Chen), wangxuyuyan@gmail.com (Xueli Wang), guoyanzhang@sdu.edu.cn (Guoyan Zhang)

same time enhances the protection against the attacker which wants to compromise secret keys. In a threshold public-key decryption system [4, 7], the decryption key corresponding to an identity is shared among a set of n users (or servers). In such a system, a ciphertext can be decrypted only if at least t users cooperate, where t is the threshold value. However, the cooperation of less than t users cannot leak any information about the plaintext or signature. This is crucial in all the applications where one cannot fully trust a unique person, but possibly a pool of individuals.

A combination of these two concepts will result the concept of ID-based threshold decryption (IBTD) schemes. It is a very useful cryptographic primitive in practice [8]. One possible application of such a IBTD scheme can be considered in a situation where an identity denotes the name of the group whose decryption key is shared by its members. Another application of the ID-based threshold decryption scheme is to use it as a building block to construct a mediated ID-based encryption scheme[9, 8]. The underlying idea is to split the private key associated with the receiver's ID into two parts. One is given to him and the other to a Security Mediator (SEM). Then as the receiver requiers the help of the SEM in order to decrypher a given encrypted message, instantaneous revocation of his privilege to perform decryption is possible only by instructing the SEM not to help him any more.

In tradtional IBC the PKG is the holder of all the private keys, implying some risk if its security is compromised. In order to prevent such a scenario to occur, Boneh and Franklin [2] suggested to introduce the threshold method, where the master key is distributed among n PKGs. Then a user can get its private key by obtaining more than t shares from different distributed PKGs. Following this idea, Boneh et al. [10] extended the IBE scheme [11] by adding the key sharing algorithm and the key recovering algorithm. As further extensions, in [12, 13, 14], the threshold decryption function was considered. However, in these schemes, the user can not share his private key by himself, but has to ask the PKG to complete this task, since the algorithm to share the private key depends on the master key. In 2004, Baek and Zheng [8] considered the threshold decryption scheme where the shares of the identity-based private key D_{ID} can be generated by the user himself. However, the Baek-Zheng method heavily relies on a suit of secret sharing tools which involve much more pairing computation than other IBTD schemes.

In the field of pairing-based cryptography, it is known that the computation of bilinear pairing is the bottleneck of the pairing application in practice [15]. To solve this problem, we should try to use as less as possible pairings in designing pairing-based cryptographic primitives. However, all previous threshold identity-based decryption schemes involve many bilinear pairings. On the other hand, although the user can ask the PKG to share the identity-based private key for him, it is preferable for him to share it by himself. In fact, only the Baek-Zheng method enjoys this functionality. So it is an interesting issue to design an IBTD scheme which enjoys this functionality and involves only a little computation for pairings. Ideally, it is the best if the number of pairing computations in all component algorithms are constants, i.e., not variables of either n or t.

With this motivation, by introducing a new technique that indirectly shares a private key in the bilinear group through simply sharing an element in the finite field using the famous secret sharing scheme due to Shamir [16], this paper proposes a new identity-based threshold decryption scheme from bilinear pairings that enjoys the following advantages. In terms of efficiency, compared with previous works, it uses much less bilinear pairings among all the algorithms. More specifically, all algorithms but **Setup**, which involves no pairing, need only one pairing computation. In terms of functionality, it is the holder of the private key associated with an identity rather than the Private Key Generator (PKG) to share the private key. Finally, the formal proof of security is provided in the random oracle model. In a short word, we construct the first

provably secure IBTD scheme with constant pairing computations.

The rest of the paper is organized as follows. We first review some preliminaries mainly including the basic property of pairings, the computational assumption on which our scheme is (indirectly) based, and the security notion of IBTD schemes in Section 2. We then present our IBTD scheme in Section 3, prove its security in Section 4 and compare it with other IBTD schemes in Section 5. Finally, Section 6 concludes the paper.

2. Preliminaries

2.1. Bilinear Pairing and Complexity Assumption

This section briefly reviews the definition of bilinear pairings and the related complexity assumption.

Definition 1. *Let $\mathbb{G}$ and $\mathbb{G}_T$ be two groups of prime order q and let P be a generator of $\mathbb{G}$, where $\mathbb{G}$ is additively represented and $\mathbb{G}_T$ is multiplicatively. A map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is said to be a bilinear pairing and the group $\mathbb{G}$ is called a bilinear group, if the following three conditions hold:*

(1) e is bilinear, i.e. $e(aP, bP) = e(P, P)^{ab}$ for all $a, b \in \mathbb{Z}_q^*$;

(2) e is non-degenerate, i.e. $e(P, P) \neq 1$, where 1 is the identity of $\mathbb{G}_T$;

(3) e is efficiently computable.

Next, we review the Bilinear Diffie-Hellman (BDH) problem, which was introduced by Boneh and Franklin [2].

Definition 2. *Let $\mathbb{G}$ and $\mathbb{G}_T$ be two groups of prime order q, P be a generator of $\mathbb{G}$ and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an bilinear pairing. Let A^{BDH} be an attacker modelled as a probabilistic Turing machine taking the security parameter k as input. Suppose that a, b, and c are uniformly chosen at random from $\mathbb{Z}_q$ and aP, bP, and cP are computed. Given $(\mathbb{G}, \mathbb{G}_T, q, e, P, aP, bP, cP)$, A^{BDH} is required to compute $e(P, P)^{abc}$. We define A^{BDH}'s success probability by*

$$Succ_{A^{BDH}}^{BDH}(k) = Pr\,[A^{BDH} \text{ outputs } e(P, P)^{abc}]$$

The attacker A^{BDH} is said to be a (t, ϵ)-solver for the BDH problem if the success probability $Succ_{A^{BDH}}^{BDH}$ is lowerly bounded by ϵ (i.e. $\geq \epsilon$) and the running time is bounded by t. The BDH problem is said to be (t, ϵ)-intractable if there is no (t, ϵ)-solver for it.

2.2. Security Model

As in [8], we review the syntax and security definitions of IBTD schemes below with slight modification.

Definition 3. *The identity-based (t, n)-threshold decryption scheme consists of the following algorithms.*

- *System initialization algorithm* **Setup** (1^k)*: Given a security parameter 1^k, this algorithm outputs the system common parameters cp and the master secret key s. cp is made public, while s is kept secret.*

- *Private key extraction algorithm* $\mathbf{EX}$ (cp, s, ID)*: Given identity ID and master key s, this algorithm computes the private key* D_{ID}*. It is then secretly sent to the corresponding entity.*

- *Private key distribution algorithm* $\mathbf{DK}(cp, ID, D_{ID}, t, n)$*: Given a private key* D_{ID}*, the number of decryption servers n and a threshold parameter t, this algorithm generates n shares* $\{\overline{s}_i\}_{i=1}^n$ *of* D_{ID} *and some public parameters, mainly including a set of verification keys* $\{S_i\}_{i=1}^n$*. The key share* $\overline{s}_1, \overline{s}_2, \ldots, \overline{s}_n$ *are sent to the decryption server* $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$ *respectively. These public parameters are used to check the validity of each private key share and decryption share.*

- *Encryption scheme* $\mathbf{E}(cp, ID, M)$*: Given a plaintext* $M \in \{0,1\}^l$ *and an identity ID, this algorithm generates a ciphertext C. Here note that, usually, the validity of the ciphertext can be publicly verified and the invalid ciphertexts will be rejected in the following algorithms.*

- *Decryption share generation algorithm* $\mathbf{D}(cp, \overline{s}_i, C)$*: Given a ciphertext C and a private key share* $\overline{s}_i$*, this algorithm first checks the validity of the ciphertext C. If C is invalid, it outputs "Invalid Ciphertext". Otherwise, it outputs the decryption share* δ_i*.*

- *Decryption share verification algorithm* $\mathbf{SV}(cp, \{S_i\}_{i=1}^n, C, \delta_i)$*: Given a ciphertext C, a set of verification keys* $\{S_i\}_{i=1}^n$*, and a decryption share* δ_i*, this algorithm first checks the validity of the ciphertext C. If C is invalid, it outputs "Invalid Ciphertext". Otherwise, it outputs "Valid Share" or "Invalid Share".*

- *Share combining algorithm* $\mathbf{SC}$ $(cp, C, \{\delta_i\}_{i\in\Phi})$*: Given a ciphertext C and a set of decryption shares* $\{\delta_i\}_{i\in\Phi}$ *(without loss of generality, we assume that there are at least t valid ciphertext sahres in* $\{\delta_i\}_{i\in\Phi}$*) , this algorithm first checks the validity of the ciphertext C. If C is invalid, it outputs "Invalid Ciphertext". Otherwise, it then chooses t valid decryption shares by using decryption share verification algorithm* $\mathbf{SV}$ *and last outputs the plaintext M.*

Following the work by Baek and Zheng, We now present the security notion for the IBTD scheme against chosen-ciphertext attack [8], which we call "IBTD-IND-CCA".

Definition 4. *Let* A^{IBTD} *be an attacker assumed to be a probabilistic Turing machine and consider the following game* G^{IBTD} *involving* A^{IBTD} *and its challenger* C^{IBTD}*.*

Phase 1. A^{IBTD} *chooses to corrupt a fixed set of* $t-1$ *out of n decryption servers.*

Phase 2. *The challenger* C^{IBTD} *runs the PKG's key/common parameter generation algorithm taking a security parameter k as input, gives* A^{IBTD} *the resulting common parameter cp and keeps the master key s secret.*

Phase 3. A^{IBTD} *issues a number of private key extraction queries, each of which is denoted by ID. On receiving the identity query ID, the challenger* C^{IBTD} *runs the private key extraction algorithm to compute the corresponding private key* D_{ID} *for* A^{IBTD}*.*

Phase 4. A^{IBTD} *issues a target identity query* ID^**. On receiving* ID^**,* C^{IBTD} *runs the private key extraction algorithm to obtain a private key* D_{ID^*} *associated with* ID^**.* C^{IBTD} *then runs the private key distribution algorithm on input* D_{ID^*} *with parameter* (t, n) *and obtains a set of private/verification key pairs* $\{\overline{s}_i, S_i\}$*. Next,* C^{IBTD} *gives* A^{IBTD} *the* $t-1$ *private keys of corrupted decryption servers and the verifications keys of all the decryption servers. However, the* $n-t+1$ *private keys of uncorrupted servers are kept secret from* A^{IBTD}*.*

Phase 5. A^{IBTD} *issues arbitrary private key extraction queries and arbitrary decryption share generation queries with respect to any uncorrupted decryption server. We denote each of these*

queries by ID and (C, k) respectively. On receiving ID, the challenger C^{IBTD} runs the private key extraction algorithm to obtain a private key associated with ID and returns it to A^{IBTD}. The only restriction here is that A^{IBTD} is not allowed to query the target identity ID^* to the private key extraction algorithm. On receiving (C, k), the challenger C^{IBTD} runs the decryption share generation algorithm to get the decryption share δ_k with respect to the ciphertext C, the target identity ID^* and the uncorrupted servers Γ_k and sends it to A^{IBTD}.

Phase 6. *A^{IBTD} outputs two equal-length plaintexts (M_0, M_1). Then the challenger C^{IBTD} chooses a random bit β and computes a target ciphertext $C^* = \mathbf{E}(cp, ID^*, M_\beta)$ for A^{IBTD}.*

Phase 7. *As in* **Phase 5**, *A^{IBTD} issues arbitrary private key extraction queries and arbitrary decryption share generation queries and the challenger C^{IBTD} deals with these queries with the additional restriction that the target ciphertext C^* is not allowed to query in this phase.*

Phase 8. *A^{IBTD} outputs a guess $\overline{\beta} \in \{0, 1\}$.*

We define the attacker A^{IBTD}'s advantage by

$$Adv^{IBTD}_{A^{IBTD}}(k) = |\Pr[\overline{\beta} = \beta] - \tfrac{1}{2}|.$$

A^{IBTD} is said to be the (t, q_e, q_d, ϵ)-attacker if $Adv^{IBTD}_{A^{IBTD}}(k)$ is at least ϵ when the running time, the times of private key extraction queries and decryption share generation queries are at most t, q_e, q_d respectively. The IBTD scheme is said to be IBTD-IND-CCA (t, q_e, q_d, ϵ)-secure if there is no (t, q_e, q_d, ϵ)-attacker against it.

2.3. Non-Interactive Zero Knowledge Proof for the Equality of Two Discrete Logarithms

To make the validity of the ciphertexts and the decryption shares checkable, we shall use a non-interactive zero-knowledge proof system for the equality of two discrete logarithm [17, 7].

Let $\mathbb{G}$ be a group of order q with generators $g, \overline{g}$. Let $\mathsf{EDLog}_{g,\overline{g}}$ be the language of pairs $(u, \overline{u}) \in \mathbb{G}^2$ such that $\log_g u = \log_{\overline{g}} \overline{u}$. Let $(u, \overline{u}) \in \mathsf{EDLog}_{g,\overline{g}}$ be given, so there exists $r \in \mathbb{Z}_q$ such that $u = g^r, \overline{u} = \overline{g}^r$. To construct the non-interactive zero knowledge proof (c, d) for the equality $\log_g u = \log_{\overline{g}} \overline{u}$, the prover randomly selects $t \in \mathbb{Z}_q$, and computes $w = g^t, \overline{w} = \overline{g}^t, c = H(u, \overline{u}, w, \overline{w}), d = t - rc$, where H is a cryptographic hash function taken as a random oracle. The verifier accepts (c, d), if and only if $c = H(u, \overline{u}, g^d u^c, \overline{g}^d \overline{u}^c)$. In the random oracle model, this protocol is a non-interactive zero-knowledge proof for the language $\mathsf{EDLog}_{g,\overline{g}}$.

3. Construction

We now describe our ID-based threshold decryption (IBTD) scheme. We claim that our IBTD scheme is based on the ID-based encryption scheme due to Boneh and Franklin [2]. We call our IBTD scheme "IdThdBm", meaning "ID-based threshold decryption scheme from the bilinear map". IdThdBm consists of the following algorithms.

- **Setup** (k): Given a security parameter k, this algorithm performs as follows.
  (1) It generates two groups $\mathbb{G}$ and $\mathbb{G}_T$ of the same prime order $q \geq 2^k$, chooses a generator P of $\mathbb{G}$ and specifies the bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$.
  (2) It specifies the hash functions $H_1 : \{0, 1\}^* \to \mathbb{G}, H_2 : \mathbb{G}_T \to \{0, 1\}^l, H_3 : \mathbb{G} \times \{0, 1\}^l \to \mathbb{G}, H_4 : \mathbb{G}^4 \to \mathbb{Z}_q$ and $H_5 : \mathbb{G}_T^4 \to \mathbb{Z}_q$, where l denotes the length of a plaintext, which is a funciton of k.

(3) It chooses the PKG's master key s uniformly at random from $\mathbb{Z}_q$, computes the PKG's public key $P_{pub} = sP$.

(4) It returns the common parameters $cp = (\mathbb{G}, \mathbb{G}_T, q, P, e, P_{pub}, H_1, H_2, H_3, H_4, H_5)$ and keeps the master key s secret.

- **EX** (cp, s, ID): Given an identity ID, this algorithm computes $Q_{ID} = H_1(ID)$ and returns the private key $D_{ID} = sQ_{ID}$ associated with ID.

- **DK**(cp, ID, D_{ID}, t, n) where $1 \leq t \leq n < q$: Given a private key D_{ID}, the number of decryption servers n and a threshold parameter t, the algorithm shares D_{ID} as follows.

  (1) It randomly chooses $a_0, a_1, \ldots, a_{t-1} \in \mathbb{Z}_q$, constructs the polynomial over $\mathbb{Z}_q$

  $$F(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}$$

  and sets $\overline{s} = a_0$.

  (2) It computes two public parameters $S = e(P, P)$, $\overline{D}_{ID} = D_{ID} - \overline{s}Q_{ID}$.

  (3) It computes each decryption server Γ_i's private key share $\overline{s}_i = F(i)$, and verification key $S_i = S^{\overline{s}_i}$ for $1 \leq i \leq n$.

  (4) It secretly sends the private key share $\overline{s}_i$ to server Γ_i for $1 \leq i \leq n$ and publishes $S_1, S_2, \ldots, S_n, S, \overline{D}_{ID}$ among the n decryption servers (not system-widely). Here note that it can be easily seen that Γ_i is able to check the validity of all these parameters.

Remark 1. *Here note that, as one of our techniques to avoid using pairing computation, $e(P, P)$ is added in the public parameters. As a result, at many places in the following algorithms* **DK**, **D**, **SV**, **SC***, the complex computation for the bilinear pairing can be replaced by one simple exponentiation.*

Remark 2. *Here note that the private key $D_{ID} \in \mathbb{G}$ is indirectly shared through sharing the element $\overline{s} \in \mathbb{Z}_p$ by the conventional Shamir sharing method. In other words, D_{ID} can be recovered by any t shares $\overline{s}_i$ and the public parameter $\overline{D}_{ID}$. Obviously, our method is much more efficient than the Baek and Zheng's direct sharing method which heavily involves pairing computation. The more interesting feature is that this indirect key sharing method will result in much less pairings in the following algorithms. For more details, please refer to Section 5.*

- **E** (cp, ID, M): Given a plaintext $M \in \{0, 1\}^l$ and an identity ID, this algorithm does as follows.

  (1) It chooses a random integer r from $\mathbb{Z}_q$, and computes $Q_{ID} = H_1(ID)$, $K = e(rQ_{ID}, P_{pub})$, $V = H_2(K) \oplus M$, where K plays the role of a temporary encryption key.

  (2) It computes $U = rP, \overline{P} = H_3(U, V)$, $\overline{U} = r\overline{P}$.

  (3) To prove $\log_P U = \log_{\overline{P}} \overline{U}$, it randomly chooses $t \in \mathbb{Z}_q$ and presents the non-interactive proof (c, d):

  $$W = tP, \overline{W} = t\overline{P}, c = H_4(U, \overline{U}, W, \overline{W}), d = t - rc.$$

  (4) It returns the ciphertext $C = (U, V, \overline{U}, c, d)$.

Here note the following points.

(i) The validity of the ciphertext $C = (U, V, \overline{U}, c, d)$ can publicly verified by checking the equation:

$$c = H_4(U, \overline{U}, dP + cU, d\overline{P} + c\overline{U}), \text{ where } \overline{P} = H_3(U, V).$$

(ii) Informally speaking, $\overline{U}$ and the above zero-knowledge proof can be seen as the digital signature of the message V under the public key U;

- **D** $(cp, \overline{s}_i, C)$: Given the private key $\overline{s}_i$ of a decryption server Γ_i and a ciphertext $C = (U, V, Z, \overline{U}, c, d)$, the decryption server Γ_i uses his private key s_i to generate a decryption share δ_i as follows.

  (1) It verifies the validity of the ciphertext C by checking the equation $c = H_4(U, \overline{U}, dP + cU, d\overline{P} + c\overline{U})$, where $\overline{P} = H_3(U, V)$. If C cannot pass this test, it outputs "Invalid Ciphertext".

  (2) Otherwise, Γ_i computes $Z = e(Q_{ID}, U)$, $Z_i = Z^{\overline{s}_i}$.

  (3) To show the equality of the two discrete logarithms $\log_Z Z_i = \log_S S_i$, it randomly chooses $t_i \in \mathbb{Z}_q$ and computes the zero knowledge proof (c_i, d_i):

$$\overline{Z}_i = Z^{t_i}, \overline{S}_i = S^{t_i}, c_i = H_5(Z_i, S_i, \overline{Z}_i, \overline{S}_i), d_i = t_i - \overline{s}_i c_i, .$$

  (4) At last, it outputs the decryption share $\delta_i = (Z_i, c_i, d_i)$.

- **SV**$(C, \{S_i\}_{i=1}^n, S, \delta_i)$: Given a ciphertext $C = (U, V, \overline{U}, c, d)$, a set of verification keys $\{S_1, \ldots, S_n\}$, the additional parameter S (recall that $S = e(P, P)$) and a decryption share δ_i, this algorithm does as follows.

  (1) It checks whether $c = H_4(U, \overline{U}, dP + cU, d\overline{P} + c\overline{U})$, where $\overline{P} = H_3(U, V)$. If C does not pass the above test, then this algorithm returns "Invalid Ciphertext".

  (2) Otherwise, it parses δ_i as (Z_i, c_i, d_i) and checks the equality

$$c_i = H_5(Z_i, S_i, Z^{d_i} Z_i^{c_i}, S^{d_i} S_i^{c_i}), \text{ where } Z = e(Q_{ID}, U).$$

  If δ_i does not pass this test, returns "Invalid Shares". Otherwise, it accept δ_i as valid share.

- **SC** $(cp, C, \{\delta_i\}_{i\in\Phi})$: Given a ciphertext $C = (U, V, \overline{U}, c, d)$ and a set of decryption shares $\{\delta_i\}_{i\in\Phi}$ where $|\Phi| \geq t$ and $\delta_i = (Z_i, c_i, d_i)$, this algorithm does as follows.

  (1) It checks whether $c = H_4(U, \overline{U}, dP + cU, d\overline{P} + c\overline{U})$, where $\overline{P} = H_3(U, V)$. If C has not passed the above test, this algorithm returns "Invalid Ciphertext".

  (2) For $i \in \Phi$, it checks the validity of each decryption share $\delta_i \in \Phi$ by the equation $c_i = H_5(Z_i, S_i, Z^{d_i} Z_i^{c_i}, S^{d_i} S_i^{c_i})$, where $Z = e(Q_{ID}, U)$, until it finds the t-th valid one. Without loss of generality, we assume that there exist at least t valid ciphertext shares and $\overline{\Phi}$ denotes the subset of t valid ones.

  (3) It computes the temporary key $K = e(\overline{D}_{ID}, U) \prod_{i\in\overline{\Phi}} Z_i^{c_{0,i}^{\overline{\Phi}}}$ and returns the plaintext $M = H_2(K) \oplus V$, where the Lagrange coefficients $c_{0,i}^{\overline{\Phi}} = \prod_{j\in\overline{\Phi}, j\neq i} \frac{-j}{i-j}$ satisfies $\overline{s} = \sum_{i\in\overline{\Phi}} c_{0,i}^{\overline{\Phi}} \overline{s}_i$.

Here note the correctness of the above IBTD scheme, since

$$\begin{aligned} K &= e(\overline{D}_{ID}, U) \prod_{i \in \overline{\Phi}} Z_i^{c_{0,i}^{\overline{\Phi}}} \\ &= e(D_{ID} - \overline{s}Q_{ID}, rP) \prod_{i \in \overline{\Phi}} e(Q_{ID}, rP)^{\overline{s}_i c_{0,i}^{\overline{\Phi}}} \\ &= e(sQ_{ID}, rP) e(Q_{ID}, rP)^{-\overline{s}} e(Q_{ID}, rP)^{\overline{s}} \\ &= e(sQ_{ID}, rP) \\ &= e(rQ_{ID}, P_{pub}). \end{aligned}$$

4. Security Proof

Following the proof method in [8], to prove the security of the proposed IBTD scheme IdThdBm, we derive a non-ID-based threshold decryption scheme called "ThdBm" from IdThdBm, which will be described shortly. We then show that the TD-IND-CCA security of the scheme ThdBm, which will be defined after the description of ThdBm, implies the IBTD-IND-CCA security of the scheme IdThdBm. Next, we show that the intractability of the BDH problem implies the TD-IND-CCA security of the ThdBm scheme. At last, we obtain the provable security result.

Due to the similarity to **Definition** 3 for IBTD schemes (Essentially, A TD scheme is just a IBTD scheme without key extraction), here we shortly states the syntax of TD (threshold decryption) schemes as follows. A TD scheme in the non-ID-based setting consists of a key/common parameter generation algorithm $\overline{\mathbf{GK}}$, an encryption algorithm $\overline{\mathbf{E}}$, a decryption share generation algorithm $\overline{\mathbf{D}}$, a decryption share verification algorithm $\overline{\mathbf{SV}}$, and a share combining algorithm $\overline{\mathbf{SC}}$. We first simply present the following TD scheme ThdBm derived from the IBTD scheme IdThdBm as follows.

- $\overline{\mathbf{GK}}(k, t, n)$: Given a security parameter k, the number of decryption servers n and a threshold parameter t, the dealer does as follows.

  (1) It generates two groups $\mathbb{G}$ and $\mathbb{G}_T$ of the same prime order $q \geq 2^k$, chooses a generator P of $\mathbb{G}$ and specifies the bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$.

  (2) It specifies the hash functions $H_2 : \mathbb{G}_T \to \{0,1\}^l$, $H_3 : \mathbb{G} \times \{0,1\}^l \to \mathbb{G}$, $H_4 : \mathbb{G}^4 \to \mathbb{Z}_q$ and $H_5 : \mathbb{G}_T^4 \to \mathbb{Z}_q$, where l denotes the length of a plaintext.

  (3) It randomly chooses s from $\mathbb{Z}_q$ and Q from $\mathbb{G}$, and then sets the private key $D = sQ$, the public key $P_{pub} = sP$ and the common parameters $cp = (\mathbb{G}, \mathbb{G}_T, q, P, e, P_{pub}, Q, H_2, H_3, H_4, H_5)$.

  (4) It randomly chooses $a_0, a_1, \ldots, a_{t-1} \in \mathbb{Z}_q$ and constructs the polynomial over $\mathbb{Z}_q$

  $$F(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}$$

  and sets $\overline{s} = a_0$.

  (5) It computes two public parameters $S = e(P, P)$, $\overline{D}_{ID} = D_{ID} - \overline{s}Q$.

  (6) It computes each decryption server Γ_i's private key share $\overline{s}_i = F(i)$, verification key $S_i = S^{\overline{s}_i}$ for $1 \leq i \leq n$.

(7) It secretly sends the distributed private key $\overline{s}_i$ to server Γ_i for $1 \le i \le n$ and publishes $S_1, S_2, \ldots, S_n$, S, $\overline{D}$ and the public parameters cp.

- $\overline{\mathbf{E}}(cp, M)$: Given a plaintext $M \in \{0,1\}^l$, this algorithm chooses r, t uniformly at random from $\mathbb{Z}_q$, and subsequently computes $K = e(rQ, P_{pub})$. It then computes $U = rP, V = H_2(K) \oplus M, \overline{P} = H_3(U, V), \overline{U} = r\overline{P}, c = H_4(U, \overline{U}, tP, t\overline{P}), d = t - rc$ and returns a ciphertext $C = (U, V, \overline{U}, c, d)$.

- $\overline{\mathbf{D}}$ $(cp, \overline{s}_i, C)$: Given a private key $\overline{s}_i$ of one decryption server and a ciphertext $C = (U, V, \overline{U}, c, d)$, the decryption server Γ_i uses his private key s_i to generate a decryption share δ_i as follows. It verifies the validity of the ciphertext C by checking the equation $c = H_4(U, \overline{U}, dP + cU, d\overline{P} + c\overline{U})$, where $\overline{P} = H_3(U, V)$. If C is valid, then Γ_i computes $Z_i = Z^{\overline{s}_i}$ and generates the zero knowledge proof $(c_i, d_i, \overline{Z}_i, \overline{S}_i)$ for the equality of the two discrete logarithms $\log_Z Z_i = \log_S S_i$, where $Z = e(Q, U)$, $\overline{Z}_i = Z^{t_i}$,$\overline{S}_i = S^{t_i}$, $c_i = H_5(Z_i, S_i, Z^{t_i}, S^{t_i})$, $d_i = t_i - \overline{s}_i c_i$, and t_i is randomly chosen from $\mathbb{Z}_q$. At last, it outputs the decryption share $\delta_i = (Z_i, c_i, d_i)$.

- $\overline{\mathbf{SV}}(C, \{S_i\}_{i=1}^n, S, \delta_i)$: Given a ciphertext $C = (U, V, \overline{U}, c, d)$, a set of verification keys $\{S_1, \ldots, S_n\}$, the parameter S and a decryption share δ_i, this algorithm first verifies the validity of C by checking $c = H_4(U, \overline{U}, dP + cU, d\overline{P} + c\overline{U})$, where $\overline{P} = H_3(U, V)$, and then the validity of $\delta_i = (Z_i, c_i, d_i)$ by checking $c_i = H_5(Z_i, S_i, Z^{d_i} Z_i^{c_i}, S^{d_i} S_i^{c_i})$, where $Z = e(Q_{ID}, U)$.

- $\overline{\mathbf{SC}}(cp, C, \{\delta_i\}_{i \in \Phi})$: Given a ciphertext $C = (U, V, \overline{U}, c, d)$ and a set of decryption shares $\{\delta_i\}_{i \in \Phi}$ where $|\Phi| \ge t$ and $\delta_i = (Z_i, c_i, d_i)$, this algorithm first checks the validity of C by checking $c = H_4(U, \overline{U}, dP + cU, d\overline{P} + c\overline{U})$, where $\overline{P} = H_3(U, V)$. It then constructs the subset $\overline{\Phi}$ of t valid ciphertext shares through the checking equation $c_i = H_5(Z_i, S_i, Z^{d_i} Z_i^{c_i}, S^{d_i} S_i^{c_i})$, where $Z = e(Q_{ID}, U)$. At last, it computes the session key $K = e(\overline{D}, U) \prod_{i \in \overline{\Phi}} Z_i^{c_{0,i}^{\overline{\Phi}}}$ and returns the plaintext $M = H_2(K) \oplus V$, where the Lagrange coefficients $c_{0,i}^{\overline{\Phi}} = \prod_{j \in \overline{\Phi}, j \ne i} \frac{-j}{i-j}$ satisfies $\overline{s} = c_{0,i}^{\overline{\Phi}} \overline{s}_i$.

Next, we review the security notion for the threshold decryption scheme against chosen-ciphertext attack [7, 8], which we call "TD-IND-CCA".

Definition 5. *Let A^{TD} be an attacker assumed to be a probabilistic Turing machine. Suppose that a security parameter k is given to A^{TD} as input. Now, consider the following game G^{TD} in which the attacker A^{TD} interacts with the challenger C^{TD}.*

Phase 1*: A^{TD} chooses to corrupt a fixed subset of $t-1$ servers.*

Phase 2*: C^{TD} runs the key/common parameter generation algorithm $\overline{\mathbf{GK}}$, and gives A^{TD} the resulting private key shares of the corrupted servers, the verification keys of all servers, the public key and other common parameters.*

Phase 3*: A^{TD} adaptively interacts with the uncorrupted decryption servers, submitting ciphertexts and obtaining decryption shares.*

Phase 4*: A^{TD} chooses two equal-length plaintexts (M_0, M_1). Then C^{TD} chooses a random bit β and returns a target ciphertext $C^* = \overline{\mathbf{E}}(cp, M_\beta)$ to A^{TD}.*

Phase 5*: A^{TD} adaptively interacts with the uncorrupted decryption servers, submitting ciphertexts and obtaining decryption shares. However, the target ciphertext C^* is not allowed to query to the decryption servers.*

Phase 6*: A^{TD} outputs a guess $\overline{\beta} \in \{0, 1\}$.*
We define the attacker A^{TD}'s advantage by

$$Adv^{TD}_{A^{TD}}(k) = |\Pr[\overline{\beta} = \beta] - \tfrac{1}{2}|.$$

A^{TD} is said to be the (t, q_d, ϵ)-attacker if the advantage $Adv^{TD}_{A^{TD}}(k)$ is at most ϵ when the running time and the times of decryption share generation queries are at most t, q_d respectively. The TD scheme is said to be TD-IND-CCA (t, q_d, ϵ)-secure if there is no (t, q_d, ϵ)-attacker against it.

Lemma 1. *Suppose that A^{IBTD} is the (t, q_e, q_d, ϵ)-attacker against the IBTD-IND-CCA security of the IBTD scheme IdThdBm in the random oracle model, where q_{H_1} queries to the random oracles H_1 are issued. Then there exists a (t', q_d, ϵ')-attacker A^{TD} against the TD-IND-CCA security of the TD scheme ThdBm, where $t' = t + (q_e + q_{H_1})O(T_1)$, $\epsilon' > \frac{1}{q_{H_1}}\epsilon$, and T_1 denotes the time for computing one scalar multiplication in $\mathbb{G}$.*

Proof. To construct A^{TD} using A^{IBTD} as a subroutine, we will simulate the challenger C^{IBTD} of A^{IBTD} as in the game G^{IBTD} and make use of A^{IBTD}'s output as follows.

Phase 1. If A^{IBTD} chooses to corrupt the $t-1$ decryption servers during the attack, then A^{TD} correspondingly chooses to corrupt $t-1$ decryption servers for $ThmBm$.

Phase 2. A^{TD} sets cp_{IBTD}=$(\mathbb{G}, \mathbb{G}_T, q, P, e, P_{pub}, H_1, H_2, H_3, H_4, H_5)$ as the common parameters of IdThdBm and sends them to A^{IBTD}, where $(\mathbb{G}, \mathbb{G}_T, q, P, e, P_{pub}, Q, H_2, H_3, H_4, H_5)$ is the ThdBm's common parameters cp_{TD} obtained from the challenger C^{TD}, and the hash function H_1 is taken as the random oracle which is controlled by A^{TD} as follows.

A^{TD} randomly chooses an index μ from the range $\{1, 2, \ldots, q_{H_1}\}$. By ID_μ, we denote the μ-th query of A^{IBTD} to the random oracle H_1. We hope ID_μ would be a target identity ID^* that A^{IBTD} outputs in Phase 4. Whenever H_1 is queried at ID, A^{TD} performs the following:

- If the query ID exists in the entry $< (ID, \tau), Q_{ID} > \in L_1$, return Q_{ID} to A^{IBTD}. (Note that the initially empty list L_1 is the "input-output" table for the simulation of H_1.)
- Otherwise, do the following.
  - If $ID = ID_\mu$, set $Q_{ID} = Q$, append $< (ID_\mu, \perp), Q_{ID_\mu} >$ to L_1 and return Q_{ID} to A^{IBTD}. (Note that Q is from A^{TD}'s common parameter cp_{TD}.)
  - Else $ID \neq ID_\mu$, do the following.
    - Choose τ uniformly at random from $\mathbb{Z}_q$.
    - Append $< (ID, \tau), Q_{ID} >$ to L_1.
    - Compute $Q_{ID} = \tau P$ and return Q_{ID} to A^{IBTD}.

Phase 3. If A^{IBTD} issues ID as a private key extraction query, A^{ID} performs the following:

- If the query ID exists in the entry $< (ID, \tau), Q_{ID} > \in L_1$, extract τ from it, compute $D_{ID} = \tau P_{pub}$, and return D_{ID} to A^{IBTD}.
- Otherwise, do the following.
  - If $ID = ID_\mu$, terminate the whole game.
  - Else $(ID \neq ID_\mu)$, do the following.
    - Choose τ uniformly at random from $\mathbb{Z}_q$.

* Compute $Q_{ID} = \tau P$ and save $< (ID, \tau), Q_{ID} >$ into L_1.
* Compute $D_{ID} = \tau P_{pub}$ and return D_{ID} to A^{IBTD}.

Phase 4. A^{IBTD} issues a target identity query ID^*. On receiving ID^*, A^{TD} obtains from its challenger C^{TD} the private key shares (without loss of generality, denoted by $\{\overline{s}_i\}_{i=1}^{t-1}$) of corrupted decryption servers, all verification keys $\{\overline{S}_i\}_{i=1}^{n}$ and the parameter $S, \overline{D}$, and then sends them to A^{IBTD} as the corresponding information for the identity ID^*. In other words, Q is taken as $H_1(ID^*)$.

Phase 5. A^{IBTD} issues arbitrary private key extraction queries and arbitrary decryption share generation queries to the uncorrupted decryption servers. A^{TD} answers the key extraction queries as in **Phase 3**. A^{TD} uses its own decryption share generating oracle, provided by C^{TD}, to answer those decryption share generation queries.

Phase 6. If A^{IBTD} submits a pair of two equal-length plaintexts (M_0, M_1), then A^{TD} provides (M_0, M_1) to its challenger C^{ID}, and obtains a target ciphertext C^* such that

$$C^* = (U, V, \overline{U}, c, d),$$

where $U = rP, V = H_2(e(rQ_{ID^*}, P_{pub})) \oplus M_\beta$, and r and β are chosen uniformly at random from $\mathbb{Z}_q$ and $\{0, 1\}$ respectively. A^{TD} simply returns the C^* to A^{IBTD} as a target ciphertext.

Phase 7. A^{IBTD} issues arbitrary private key extraction queries and arbitrary decryption share generation queries to the uncorrupted decryption servers. A^{TD} answers these queries as in **Phase 5** with additional restriction that A^{IBTD} is not allowed to query C^* to any of the uncorrupted decryption servers.

Phase 8. Finally, if A^{IBTD} submits its guess $\bar{\beta}$, then A^{TD} returns $\bar{\beta}$ to C^{TD}.

Now we complete the description of the attacker A^{TD}. Next, we show that the above constructed attacker A^{TD} perfectly simulates the challenger C^{IBTD}, when it does not terminates the game. Firstly, due to the randomness of $\tau \in \mathbb{Z}_q$ and Q, the above simulation of the random oracle H_1 is perfect. Secondly, for $ID \neq ID_\mu$, D_{ID} is rightly simulated, since $D_{ID} = \tau P_{pub} = \tau sP = sQ_{ID}$. Thirdly, the decryption shares and the target ciphertext are also perfectly simulated, since there is no difference between the IBTD scheme IdThdBm and the TD scheme ThdBm when their public keys, ID and Q respectively, satisfy the equation $H_1(ID) = Q$.

Since A^{TD} perfectly simulates the challenger C^{IBTD}, when it does not terminates the game. So we have that the advantage of A^{TD} against the TD scheme ThdBm is equal to the advantage of A^{IBTD} against the IBTD scheme IdThdBm, when A^{TD} does not abort. On the other hand, the probability that the attacker A^{TD} does not terminate is $\frac{1}{q_{H_1}}$, since μ has been uniformly chosen from $[1, q_{H_1}]$. So we have

$$\epsilon' \geq \tfrac{1}{q_{H_1}}\epsilon.$$

Finally, we analyze the running time t' of the TD-IND-CCA attacker A^{TD}. Note that t' mainly consists of the time for simulation of the random oracle H_1 and the private key extraction oracle **EX**, as well as that for the attacker A^{IBTD}. Since the simulation of H_1 and **Ex** mainly consists of 1 or 2 scalar multiplications in $\mathbb{G}$, so t' is lower-bounded by $t_{IDCCA} + (q_e + q_{H_1})O(T)$, where T denotes the running time for computing one scalar multiplication. The proof is complete. □.

Lemma 2. *Suppose that there exists a (t', q_d, ϵ')-attacker A^{TD} against the TD-IND-CCA security of the scheme ThdBm. Then there exists a (t'', ϵ'')-solver A^{BDH} for the BDH problem, where*

$$t'' = t' + max\{q_{H_2}, q_{H_3}, q_{H_4}, q_{H_5}, q_d\}O(T_2),\ \epsilon'' \geq \frac{2\epsilon}{q_{H_2}}.$$

The above T_2 is the biggest one among the running time for pairing, exponentiation in $\mathbb{G}$ and $\mathbb{G}_T$.

Proof. To construct A^{BDH} using A^{TD} in the game G^{TD} as a subroutine, we will simulate the view of A^{TD} and make use of its answers as follows. Assume that A^{BDH} obtains the parameters $\mathbb{G}, \mathbb{G}_T, q, P, e$, and problem denoted by aP, bP, cP, and is required to compute $e(P,P)^{abc}$.

Phase 1. A^{TD} chooses a fixed set Φ' of $t-1$ decryption servers it wants to corrupt. Without loss of generality, assume $\Phi' = \{1, 2, \ldots, t-1\}$.

Phase 2. In this phase, A^{BDH} does as follows.

- A^{BDH} randomly selects $\tilde{s} \in \mathbb{Z}_q$ sets

  $$P_{pub} = sP = bP, Q = cP, \overline{D} = \tilde{s}Q, S_0 = e(P_{pub} - \tilde{s}P, P).$$

  Here note that if the unknown value $s - \tilde{s}$ is denoted by $\overline{s}$, then we have

  $$\overline{D} = sQ - \overline{s}Q, S_0 = e(P,P)^{\overline{s}}.$$

- Next, for $1 \leq i \leq t-1$, A^{BDH} randomly picks an element $\overline{s}_i \in \mathbb{Z}_q$ as the key share of Γ_i, and sets the verification key $S_i = S^{\overline{s}_i}$, where $S = e(P,P)$. Let $F(x) \in \mathbb{Z}_q[x]$ be the polynomial of degree $(t-1)$, which is implicitly defined by

  $$F(0) = \overline{s},\ F(i) = \overline{s}_i \text{ for } 1 \leq i \leq t-1.$$

  Hence, we have

  $$S_i = e(P,P)^{F(i)},\ i = 0, 1, 2, \ldots, t-1.$$

- For every integer $k \in [t, n]$, A^{BDH} simulates the k-th verification key $S_k = S^{F(k)} = e(P,P)^{F(k)}$ as follows.

  - A^{BDH} sets $\Phi' = \{0, 1, \ldots, t-1\}$ and then computes the Lagrange coefficients

    $$c_{k,j}^{\Phi'} = \prod_{s \neq j, 0 \leq s \leq t-1} \frac{k-s}{j-s},\ j = 0, 1, \ldots, t-1.$$

  - According to Lagrange interpolation formula $F(k) = \sum_{j=0}^{t-1} c_{k,j}^{\Phi'} F(j)$, A^{BDH} then computes

    $$S_k = S_0^{c_{k,0}^{\Phi'}} S_1^{c_{k,1}^{\Phi'}} \ldots S_{t-1}^{c_{k,t-1}^{\Phi'}}.$$

- At last, A^{BDH} sends $\overline{D}, S, S_1, S_2, \ldots, S_n, \overline{s}_1, \ldots, \overline{s}_{t-1}$ and cp to A^{TD}, where $cp = (\mathbb{G}, \mathbb{G}_T, q, P, e, P_{pub}, Q, H_2, H_3, H_4, H_5)$ and H_2, H_3, H_4, H_5 are random oracles controlled by A^{BDH} as follows.

  - H_2-queries. A^{TD} can issue H_2 queries at any time. A^{BDH} maintains an initially empty list L_2 of tuples $< K_i, R_i >$. For the the query K_i, A^{TD} does as follows.

* If the query K_i already appears on the L_2 list in a tuple $< K_i, R_i >$, then A^{BDH} responds with $H_2(K_i) = R_i$.
* Otherwise, A^{BDH} picks a random string $R_i \in \{0,1\}^l$, adds the tuple $< K_i, R_i >$ to L_2 and responds to A^{TD} with $H_2(K_i) = R_i$.

– H_3-queries. Initially, A^{BDH} sets

$$U^* = aP,\ V^* \overset{R}{\leftarrow} \{0,1\}^l,\ H_3(U^*, V^*) = t^*P,$$

where "$\overset{R}{\leftarrow}$" means "is randomly chosen from", and then adds $< t^*, (U^*, V^*) >$ to the empty list L_3. For a query $H_3(U_i, V_i)$, A^{BDH} does as follows.

* If the query U_i, V_i already appears on the L_3 list in a tuple $< t_i, (U_i, V_i) >$, then A^{BDH} responds with $t_i P_{pub}$ in the case $(U_i, V_i) \neq (U^*, V^*)$, or $t_i P$ in the other case $(U_i, V_i) = (U^*, V^*)$.
* If (U_i, V_i) does not appears in L_3, A^{BDH} randomly selects $t_i \in \mathbb{Z}_q$, adds $< t_i, (U_i, V_i) >$ to the L_3, and returns the answer sets $H_3(U_i, V_i) = t_i P_{pub}$.

– H_4-queries. This random oracle H_4 is simulated in the same way as simulating the random oracle H_2. Roughly speaking, when A^{TD} queries the random oracle H_4 at a distinct point $(U_i, \overline{U}_i, W_i, \overline{W}_i)$, A^{BDH} choose c_i at random as the answer and maintains an initially empty list L_4 of tuples $< c_i, (U_i, \overline{U}_i, W_i, \overline{W}_i) >$.

– H_5-queries. This random oracle H_5 is simulated in the same way as simulating the random oracles H_2, H_4. Roughly speaking, when A^{TD} queries the random oracle H_5 at a distinct point $(Z_i, S_i, \overline{Z}_i, \overline{S}_i)$, A^{BDH} choose c_i at random as the answer and maintains an initially empty list L_5 of tuples $< c_i, (Z_i, S_i, \overline{Z}_i, \overline{S}_i)) >$.

Phase 3. Assume that A^{TD} asks for the k-th decryption share of the ciphertext $C = (U, V, \overline{U}, c, d)$ due to the uncorrupted server Γ_k, where $t \leq k \leq n$.

- A^{BDH} searches the list L_3 for a tuple $< t', (U, V) >$ containing (U, V). If it is nonexistent, A^{BDH} returns "Invalid Ciphertext".
- A^{BDH} searches the list L_4 for a tuple $< c, (U, \overline{U}, W, \overline{W}) >$ containing $(U, \overline{U}, dP + cU, d\overline{P} + c\overline{U})$, where $\overline{P} = H_3(U, V)$. If it does not appears in L_4, A^{BDH} returns "Invalid Ciphertext".
- Although A^{BDH} cannot get r from $U = rP$, he can assume $\overline{U} = r\overline{P}$ and simulates the decryption of C. In fact, A^{BDH} can computes the temporary key $K = e(Q, \overline{U})^{\frac{1}{t'}}$, since

$$K = e(D, U) = e(sQ, rP) = e(Q, rsP) = e(Q, r(t' P_{pub}))^{\frac{1}{t'}} = e(Q, r\overline{P})^{\frac{1}{t'}} = e(Q, \overline{U})^{\frac{1}{t'}}.$$

- Just like simulating the verification keys S_k in **Phase 2**, $k \in [t, n]$, A^{BDH} simulates the k-th decryption share δ_k as follows.

  – A^{BDH} computes

$$Z = e(Q, U), Z_0 = KZ^{-\tilde{s}}, Z_i = Z^{\overline{s}_i} = e(\overline{s}_i Q, U),\ i = 0, 1, 2 \ldots, t-1.$$

  Here note that $Z_0 = Z^{\overline{s}}$, since

$$Z_0 = KZ^{-\tilde{s}} = Z^s Z^{-\tilde{s}} = Z^{s-\tilde{s}} = Z^{\overline{s}}.$$

– A^{BDH} computes $Z_k = Z^{F(k)}$ through

$$Z_k = Z_0^{c_{k,0}^{\Phi'}} Z_1^{c_{k,1}^{\Phi'}} \cdots Z_{t-1}^{c_{k,t-1}^{\Phi'}},$$

where

$$\begin{aligned} c_{k,j}^{\Phi'} &= \prod_{s \neq j, 0 \leq s \leq t-1} \frac{k-s}{j-s}, j = 0, 1, 2, \ldots, t-1, \\ F(k) &= \sum_{j=0}^{t-1} c_{k,j}^{\Phi'} F(j). \end{aligned}$$

– A^{BDH} randomly selects $c_k, d_k \in \mathbb{Z}_q$, and sets the k-th decryption share $\delta_k = (Z_k, c_k, d_k)$. At last, A^{BDH} appends $< c_k, (Z_k, S_k, Z^{d_k} Z_k^{c_k}, S^{d_k} S_k^{c_k}) >$ to the list L_5, where $Z = e(Q, U)$. Now, it can be seen that this simulated ciphertext C can pass the validity checking.

Phase 4. A^{TD} chooses two equal-length plaintexts (M_0, M_1) and provides them to A^{BDH}. A^{BDH} provides to A^{TD} the target ciphertext $C^* = (U^*, V^*, \overline{U}^*, c^*, d^*)$ as below.

- U^*, V^* are defined in the initial step of simulating the random oracle H_3 and we have

$$\overline{P}^* = H_3(U^*, V^*) = t^* P.$$

- The other parts $\overline{U}^*, c^*, d^*$ is defined as:

$$\overline{U}^* = t^* U^*, c^* \overset{R}{\leftarrow} \mathbb{Z}_q, d^* \overset{R}{\leftarrow} \mathbb{Z}_q.$$

- At last, A^{BDH} appends $< c^*, (U^*, \overline{U}^*, d^* P + c^* U^*, d^* \overline{P}^* + c^* \overline{U}^*) >$ to the list L_4.

Here note that $\overline{U}^* = a\overline{P}^*$, since

$$\overline{U}^* = t^* U^* = a(t^* P) = a H_3(U^*, V^*).$$

Phase 5: A^{BDH} simulates the decryption share oracle for A^{TD} as in **Phase 3**. However, the target ciphertext C^* is not allowed to query to the decryption servers.

Phase 6: A^{TD} outputs a guess $\overline{\beta} \in \{0, 1\}$. A^{BDH} ignores $\overline{\beta}$, picks a tuple $< K_i, R_i >$ from L_2 randomly, and then output K_i as the solution to the given instance aP, bP, cP of the BDH problem.

Now we complete the description of the attacker A^{BDH}. Then we show that A^{TD}'s view in the above simulation is identical to its view in the real attack, as long as A^{TD} does not issue a query for $H_2(e(P, P)^{abc})$.

- In **Phase 2**, we show the following points.

  – The four random oracles H_2, H_3, H_4, H_5 are perfectly simulated, since all the responses are uniform and independent in the respective ranges.

  – The parameters $\overline{s}, F(x)$ are distributed as in the real attack. Then the other parameters $\overline{D}, S_1, S_2, \ldots, S_n, \overline{s}_1, \ldots, \overline{s}_{t-1}$ are also perfectly simulated, since they are defined by $\overline{s}, F(x)$.

– Obviously, the other parameters, such as P_{pub}, Q, S, are generated as in the real attack.

- In **Phase 3**, observe that for the queried ciphertext $C = (U, V, \overline{U}, c, d)$, if A^{TD} does not queries $H_3(U, V)$ or $H_4(U, \overline{U}, dP + cU, d\overline{P} + c\overline{U})$, he can not get these values by himself and so can not make the ciphertext C valid (Here we ignore the negligeable probability that A^{TD} obtains these values through purely random guess). So, if C is valid, then these queries have been made and hence the simulation in **Phase 3** is obviously successful.

- In **Phase 4**, observe that if A^{BDH} does not issue a query for $H_2(e(P,P)^{abc})$, he does not knows any information of $H_2(e(P,P)^{abc})$. In this case, the target ciphertext C^* is distributed as in the real attack.

Then we analyze the success probability of A^{BDH}. Let $\mathcal{H}$ be the event that algorithm A^{TD} issues a query for $H_2(e(P,P)^{abc})$ at some point during the simulation above (this implies that at the end of the simulation $e(P,P)^{abc}$ appears in some tuple on the L_2). We also study event $\mathcal{H}$ in the real attack game, namely the event that A^{TD} issues a query for $H_2(e(P,P)^{abc})$ when communicating with a real challenger and a real random oracle for H_2. In the real attack, if the event $\mathcal{H}$ does not occur, the decryption of C^* is independent of A^{TD}'s view and hence $\Pr[\beta = \beta' | \mathcal{H} \textit{ does not occur }] = \frac{1}{2}$. By definition of A^{TD}, we know that in the real attack $\Pr[\beta = \beta'] - \frac{1}{2} \geq \epsilon$. We show that these two facts imply that $\Pr[\mathcal{H}] \geq 2\epsilon$. To do so we first derive simple upper and lower bounds on $\Pr[\beta = \beta']$:

$$\begin{aligned}
\Pr[\beta = \beta'] &= \Pr[\beta = \beta' | \neg\mathcal{H}] \Pr[\neg\mathcal{H}] + \Pr[\beta = \beta' | \mathcal{H}] \Pr[\mathcal{H}] \\
&\leq \frac{1}{2} \Pr[\neg\mathcal{H}] + \Pr[\mathcal{H}] = \frac{1}{2} + \frac{1}{2} \Pr[\mathcal{H}], \\
\Pr[\beta = \beta'] &\geq \Pr[\beta = \beta' | \neg\mathcal{H}] \Pr[\neg\mathcal{H}] = \frac{1}{2} - \frac{1}{2} \Pr[\mathcal{H}].
\end{aligned}$$

It follows that $\epsilon \leq |\Pr[\beta = \beta'] - \frac{1}{2}| \leq \frac{1}{2} \Pr[\mathcal{H}]$. Therefore, in the real attack $\Pr[\mathcal{H}] \geq 2\epsilon$. On the hand, observe that, we have showed that A^{TD}'s view in the above simulation is identical to its view in the real attack before $\mathcal{H}$ occurs. Hence, we can claim that $\Pr[\mathcal{H}]$ in the simulation above is equal to $\Pr[\mathcal{H}]$ in the real attack. Furthermore, at the end of the simulation $e(P,P)^{abc}$ appears in some tuple on the L_2 with probability greater than 2ϵ. So we get the last probability

$$\Pr[A^{BDH} \textit{ outputs } e(P,P)^{abc}] \geq \frac{2\epsilon}{q_{H_2}}.$$

Finally, we roughly analyze the running time t'' of the algorithm A^{BDH}. The running time t'' mainly consists of the time t' of the attacker A^{TD} and that for simulating random oracle queries and decryption share queries. The computation for each query, random oracle query or decryption share query, is a certain number of exponentiations in $\mathbb{G}$ or $\mathbb{G}_T$ or pairings. So we have $t'' = t' + max\{q_{H_2}, q_{H_3}, q_{H_4}, q_{H_5}, q_d\}O(T)$, where T is the biggest one among the running time for pairing, exponentiation in $\mathbb{G}$ and $\mathbb{G}_T$. The proof is complete. □

From Lemma 1,2, we obtain the following the theorem

Theorem 1. *Under the BDH assumption, the proposed IBTD scheme is TD-IND-CCA secure in the random oracle. Concretely, suppose that A^{IBTD} is the (t, q_e, q_d, ϵ)-attacker against the IBTD-IND-CCA security of the IBTD scheme IdThdBm in the random oracle model, where q_{H_1} queries to the random oracles H_1 are issued. Then there exists a (t'', ϵ'')-solver A^{BDH} for the BDH problem, where*

$$t'' = t + (q_e + q_{H_1})O(T_1) + max\{q_{H_2}, q_{H_3}, q_{H_4}, q_{H_5}, q_d\}O(T_2),\ \epsilon'' \geq \frac{2\epsilon}{q_{H_2} q_{H_1}}.$$

The parameters $T_1, T_2, q_{H_1}, q_{H_2}, q_{H_3}, q_{H_4}, q_{H_5}, q_e, q_d$ are defined as in lemma 1,2.

5. Comparison

At first, we introduce some aspects which will be considered during comparing ID-based threshold decryption schemes.

- Number of pairings in all relative algorithms. There are many kinds of concrete constructions of bilinear pairings, using Tate pairing, Weil Pairing or their variants, on supersingular elliptic curves or ordinary elliptic curves such as MNT curves, Freedman curves and Barreto-Naehrig curves. Furthermore, there are appearing various new techniques on pairing implementation with all kinds of optimizations. Hence, it is impossible to simply conclude how much the speed difference is between the scalar multiplication in the group G (or the exponentiation in G_T) and the bilinear pairing, since the speed difference will get changed with the difference of the type of pairing, the security parameters, the type of underlying curves, the parameters of curves such as embedding degrees and so on. All in all, it remains reasonable to say that the scalar multiplication in G (or the exponentiation in G_T) is remarkably faster than the pairing operation for most pairing-based cryptographic schemes. So we take the number of pairings as the main efficiency index, and try to use as less pairings as possible .

- Who (PKG or the user himself) is the dealer to distribute the private key of the identity. First, note that, in many applications of IBTD schems, the object of splitting the private key of the identity is to help the user to distribute his power for decryption. In such cases, it is obviously preferable to adopt the IBTD scheme which can help the user to distribute his private key without bothering PKG. Second, note that it is trivial to transform the IBTD scheme supporting the user himeslf to distribute his private key to the one supporting the PKG to distribute the user's private key. All in all, it is more flexible, if we can provide the function for user to distribute his private key by himself in the IBTD scheme.

- Random oracle Model or Standard model. From the purely theoretical viewpoint, it is preferable to adopt the scheme secure in the standard model. Now, the only IBTD scheme in the standard model is the one proposed by Kiltz and Galindo [18]. However, at the price of standard model, in their IBTD scheme, the public key involves k elements in the group G, where k is the security parameter, and the underlying mathematical assumption (BDDH) is a stronger than BDH. Additionally, it also involves more pairings and can not support the user to distribtue his private keys. Hence, it is not appropriate to simply conclude that the IBTD scheme in the standard model is better than that in random oracle model. By the way, we mention that we tried but failed to find the way for applying our technique to the identity based encryption in the standard model [19].

In terms of security and functionality, we compare our IBTD scheme with other schemes as follows. Like the schemes in [8, 14], our scheme is provably secure in the random oracle model (ROM) under the BDH assumption, while the IBTD scheme proposed in [18] is provably secure in the standard model (SM) under the BDDH assumption (Bilinear Decision Diffie-Hellman).

Like the scheme in [8], the user himself in our scheme can share the private key, while the user in the schemes of[14, 18] has to depend on the PKG to share the private key.

In terms of efficiency, the comparison between our IBTD scheme and other ones is as follows. In the field of pairing based cryptography, the number of involved pairings in algorithms is the most important efficiency index. As explained in Remark 1, 2, we make use of some techniques to avoid using pairings. Among these techniques, the indirect method to share the private key in group is the main one . As a result, our identity-based (t, n)-threshold decryption scheme is much more efficient than others as showed in the above table. Especially, observe that the number of pairings in the algorithms **DK**, **SC** in [8, 18, 14] linearly increases with the parameter t or n, while the number of pairings in our scheme is a constant, more exactly, 0 or 1.

At last, we present Table 1 for comparison of different IBTD schemes in terms of number of pairings, length of decryption shares, security model (Random oracle model or standard model), assumption and dealer (the one who generates the key shares, user or PKG).

Table 1: Comparison of different IBTD schemes

Scheme	Number of Pairings					Dec. Share	Model	Assume	Dealer
	E	**DK**	**D**	**SV**	**SC**				
BZ04	1	n	5	4	$\geq 2+2t$	$\mathbb{G}_T^3 \times \mathbb{Z}_q \times \mathbb{G}$	ROM	BDH	User
LCL07	1	n	3	2	$\geq 2t$	$\mathbb{G}_T \times \mathbb{Z}_q \times \mathbb{G}$	ROM	BDH	PKG
GK06	0	0	2	6	$\geq 5+4t$	$\mathbb{G}^3$	SM	BDDH	PKG
Ours	1	1	1	1	1	$\mathbb{G}_T \times \mathbb{Z}_q^2$	ROM	BDH	User

6. Application and Extension

First, our IBTD scheme is very suitable, when the user wants to share his private key out among a number of decryption servers in such a way that any committee member can successfully decrypt the ciphertext if, and only if, the committee member obtains a certain number of decryption shares from the decryption servers. In this case, unlike the IBTD schemes [14, 18], the user can perform this process without asking the PKG to do so.

Second, our IBTD scheme can be used as a building block to construct a mediated ID-based encryption scheme [8]. The idea is to split a private key associated with the receiver Bob's ID into two parts, and give one share to Bob and the other to the Security Mediator (SEM). Accordingly, Bob can decrypt a ciphertext only with the help of the SEM. As a result, instantaneous revocation of Bob's privilege to perform decryption is possible by instructing the SEM not to help him any more. Using Beak's method for contructing mediated ID-based encryption scheme, a more efficient one can be similarly constructed based on our IBTD scheme.

Third, since the main compuation is the common group operations in the group G or G_T and only very few pairings are involved during the decrypting procedure, our IBTD scheme is very efficeint and hence is suitable for some resource-restricted applications.

7. Conclusion

In this paper, we proposed a new identity-based threshold decryption (IBTD) scheme from bilinear pairings. With this scheme, the user can by himeself distribute the private key among

decryption servers, without bothering PKG in the sharing procedure. It uses much less bilinear pairings than other IBTD schemes in the involved algorithms. The advantage in efficiency make it suitable for some applications where the computation resource is limited, such as wireless networks and smart cards. All these properties are due to our new basic technique by which the private key in the bilinear group is indirectly shared through simply sharing an element in the finite field. As the future work, it is interesting to explore further applications of this new technique in threshold cryptography, such as ID-based threshold decryption in the standard model and hierarchical threshold decryption schemes.

References

[1] A. Shamir, Identity-based cryptosystems and signature schemes, in: Proceedins of Crypto 84, LNCS 196, Springer-Verlag, 1984, pp. 47-53.

[2] D. Boneh, M. Franklin, Identity-Based encryption from the Weil pairing, in: Proceedings Crypto 2001, LNCS 2139, Springer-Verlag, 2001, pp. 213-229.

[3] M. C. Gorantla, R. Gangishetti, A. Saxena, A Survey on ID-Based Cryptographic Primitives, http:// eprint.iacr.org /2005 /094.pdf

[4] Y. Desmedt and Y. Frankel. Threshold cryptosystems. in: Proceedings of CRYPTO'89, LNCS 435, Springer-Verlag, 1990, pp. 307-315.

[5] Y. Desmedt and T. Lange. Pairing Based Threshold Cryptography Improving on Libert-Quisquater and Baek-Zheng. In: Proceedings of Financial Cryptography and Data Security 2006, LNCS 4107, Springer-Verlag, pp.154-159.

[6] A. Santis, Y. Desmedt, Y. Frankel, and M. Yung, How to share a function securely, in: Proceedings of 26th ACM STOC, ACM Press, 1994, pp. 522-533.

[7] V. Shoup, R. Gennaro, Securing Threshold Cryptosystems against Chosen Ciphertext Attack, Journal of Cryptology, Vol. 15, Springer-Verlag, 2002, pp. 75-96.

[8] J. Baek, Y. Zheng, Identity-based threshold decryption, in: Proceedings of PKC04, LNCS 2947, Springer, 2004, pp. 262-76.

[9] D. Boneh, X. Ding, G. Tsudik, C. Wong, A Method for Fast Revocation of Public Key Certicates and Security Capabilities, in: Proceedings of the 10th USENIX Security Symposium, USENIX, 2001, pp.297-310.

[10] D. Boneh, X.Boyen, S.Halevi, Chosen ciphertext secure public key threshold encryption without random oracles, in: Proceedings of RSA-CT06, LNCS 3860, Springer-Verlag, 2006, pp. 226-43.

[11] D. Boneh, X. Boyen, Efficient selective-ID identity based encryption without random oracles, in: Proceedings of Eurocrypt 2004, volume 3027 of LNCS, Springer-Verlag, 2004, pp. 522-533.

[12] B. Libert, J. Quisquater, Efficient Revocation and Threshold Pairing Based Cryptosystems, in: Proceedings of the twenty-second annual symposium on Principles of distributed computing, ACM Press, 2003, pp.163-171.

[13] Z. Chai, Z. Cao, R. Lu, ID-based threshold decryption without random oracles and its application in key escrow, in: Proceedings of the 3rd international conference on Information security, ACM Press, 2004, pp.119-124.

[14] L. Long, K. Chen, S. Liu, ID-based threshold decryption secure against adaptive chosen-ciphertext attack, Computers and Electrical Engineering, 2007, 33 (3) 166-176.

[15] C. Zhao, F. Zhang, Research and Development on Efficient Pairing Computations, Journal of Software, 2009, 20 (11) 3001-3009.

[16] A. Shamir, How to Share a Secret, Communications of the ACM, 1979, 22(11) 612-613.

[17] D. Chaum, T. Pedersen, Wallet databases with observers, in: Proceedings of Advances in Cryptology-Crypto 92, LNCS 740, Springer-Verlag,1992, pp.89-105.

[18] D. Galindo and E. Kiltz, Chosen-Ciphertext Secure Threshold Identity-Based Key Encapsulation Without Random Oracles, in: Proceedinsg of Security and Cryptography for Networks, LNCS 4116, Springer-Verlag, 2006, pp.173-185.

[19] Brent Waters. Efficient identity-based encryption without random oracles. In: Proceedings of EUROCRYPT 2005, LNCS 3494, Springer-Verlag, pp.114-127.