Identity-Based Threshold Decryption

Joonsang Baek[1] and Yuliang Zheng[2]
[1] School of Network Computing, Monash University, Frankston, VIC 3199, Australia
joonsang.baek@infotech.monash.edu.au
[2] Dept. of Software and Info. Systems, University of North Carolina at Charlotte, Charlotte, NC 28223, USA
yzheng@uncc.edu

Abstract

In this paper, we examine issues related to the construction of identity-based threshold decryption schemes and argue that it is important in practice to design an identity-based threshold decryption scheme in which a private key associated with an identity is shared. A major contribution of this paper is to construct the first identity-based threshold decryption scheme secure against chosen-ciphertext attack. A formal proof of security of the scheme is provided in the random oracle model, assuming the Bilinear Diffie-Hellman problem is computationally hard. Another contribution of this paper is, by extending the proposed identity-based threshold decryption scheme, to construct a mediated identity-based encryption scheme secure against more powerful attacks than those considered previously.

1 Introduction

Threshold decryption is particularly useful where the centralization of the power to decrypt is a concern. And the motivation for identity (ID)-based encryption originally proposed by Shamir [17] is to provide confidentiality without the need of exchanging public keys or keeping public key directories. A major advantage of ID-based encryption is that it allows one to encrypt a message by using a recipient's identifiers such as an email address.

A combination of these two concepts will allow one to build an "ID-based threshold decryption" scheme. One possible application of such a scheme can be considered in a situation where an identity denotes the name of the group sharing a decryption key. As an example, suppose that Alice wishes to send a confidential message to a committee in an organization. Alice can first encrypt the message using the identity (name) of the committee and then send over the ciphertext. Let us assume that Bob who is the committee's president has created the identity and hence has obtained a matching private decryption key from the Private Key Generator (PKG). Preparing for the time when Bob is away, he can 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.

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 [7]. 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.

In this paper, we deal with the problem of constructing an ID-based threshold decryption scheme which is efficient and practical while meets a strong security requirement. We also treat the problem of applying the ID-based threshold decryption scheme to design a mediated ID-based encryption scheme secure against more powerful attacks than those considered previously in the literature.

2 Preliminaries

We first review the "admissible bilinear map", which is the mathematical primitive that plays on central role in Boneh and Franklin's ID-based encryption scheme [4].

Bilinear Map. The admissible bilinear map $\hat{e}$ [4] is defined over two groups of the same prime-order q denoted by $\mathcal{G}$ and $\mathcal{F}$ in which the Computational Diffie-Hellman problem is hard. (By $\mathcal{G}^*$ and $\mathbb{Z}_q^*$, we denote $\mathcal{G} \setminus \{O\}$ where O is the identity element of $\mathcal{G}$, and $\mathbb{Z}_q \setminus \{0\}$ respectively.) We will use an additive notation to describe the operation in $\mathcal{G}$ while we will use a multiplicative notation for the operation in $\mathcal{F}$. In practice, the group $\mathcal{G}$ is implemented using a group of points on certain elliptic curves, each of which has a small MOV exponent [15], and the group $\mathcal{F}$ will be implemented using a subgroup of the multiplicative group of a finite field. The admissible bilinear map, denoted by $\hat{e} : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{F}$, has the following properties.

- Bilinear: $\hat{e}(aR_1, bR_2) = \hat{e}(R_1, R_2)^{ab}$, where $R_1, R_2 \in \mathcal{G}$ and $a, b \in \mathbb{Z}_q^*$.
- Non-degenerate: $\hat{e}$ does not send all pairs of points in $\mathcal{G} \times \mathcal{G}$ to the identity in $\mathcal{F}$. (Hence, if R is a generator of $\mathcal{G}$ then $\hat{e}(R, R)$ is a generator of $\mathcal{F}$.)
- Computable: For all $R_1, R_2 \in \mathcal{G}$, the map $\hat{e}(R_1, R_2)$ is efficiently computable.

Throughout this paper, we will simply use the term "Bilinear map" to refer to the admissible bilinear map defined above.

The "$\mathsf{BasicIdent}$*" Scheme.* We now describe Boneh and Franklin's basic version of ID-based encryption scheme called "$\mathsf{BasicIdent}$" which only gives semantic security (that is, indistinguishability under chosen plaintext attack).

In the setup stage, the PKG specifies a group $\mathcal{G}$ generated by $P \in \mathcal{G}^*$ and the Bilinear map $\hat{e} : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{F}$. It also specifies two hash functions $\mathsf{H}_1 : \{0,1\}^* \rightarrow \mathcal{G}^*$ and $\mathsf{H}_2 : \mathcal{F} \rightarrow \{0,1\}^l$, where l denotes the length of a plaintext.The PKG then picks a master key x uniformly at random from $\mathbb{Z}_q^*$ and computes a public key $Y_{\mathrm{PKG}} = xP$. The PKG publishes descriptions of the group $\mathcal{G}$ and $\mathcal{F}$ and the hash functions H_1 and H_2. Bob, the receiver, then contacts the PKG to get his private key $D_{\mathtt{ID}} = xQ_{\mathtt{ID}}$ where $Q_{\mathtt{ID}} = \mathsf{H}_1(\mathtt{ID})$. Alice, the sender, can now encrypt her message $M \in \{0,1\}^l$ using Bob's identity $\mathtt{ID}$ by computing $U = rP$ and $V = \mathsf{H}_2(\hat{e}(Q_{\mathtt{ID}}, Y_{\mathrm{PKG}})^r) \oplus M$, where r is chosen at random from $\mathbb{Z}_q^*$ and $Q_{\mathtt{ID}} = \mathsf{H}_1(\mathtt{ID})$. The resulting ciphertext $C = (U, V)$ is sent to Bob. Bob decrypts C by computing $M = V \oplus \mathsf{H}_2(\hat{e}(D_{\mathtt{ID}}, U))$.

3 Related Work and Discussion

Boneh and Franklin's "Distributed PKG". In order to prevent a single PKG from full possession of the master key in ID-based encryption, Boneh and Franklin [4] suggested that the PKG's master key should be shared among a number of PKGs using the techniques of threshold cryptography, which they call "Distributed PKG". More precisely, the PKG's master key x is distributed into a number of PKGs in such a way that each of the PKG holds a share $x_i \in \mathbb{Z}_q^*$ of a Shamir's (t, n) secret-sharing [16] of $x \in \mathbb{Z}_q^*$ and responds to a user's private key extraction request with $D_{\mathtt{ID}}^i = x_i Q_{\mathtt{ID}}$, where $Q_{\mathtt{ID}} = \mathsf{H}_1(\mathtt{ID})$. If the technique of [11] is used, one can ensure that the master key is jointly generated by PKGs so that the master key is not stored or computed in any single location.

As an extension of the above technique, Boneh and Franklin suggested that the distributed PKGs should function as decryption servers for threshold decryption. That is, each PKG responds to a decryption query $C = (U, V)$ in $\mathsf{BasicIdent}$ with $\hat{e}(x_i Q_{\mathtt{ID}}, U)$. However, we argue that this method is not quite practical in practice since it requires each PKG to be involved *at all times* (that is, *on-line*) in the generation of decryption shares because the value "U" changes whenever a new ciphertext is created. Obviously, this creates a bottleneck on the PKGs and also violates one of the basic requirements of an ID-based encryption scheme, "the PKG can be closed after key generation", which was envisioned by Shamir in his original proposal of ID-based cryptography [17]. Moreover, there is a scalability problem when the number of available distributed PKGs is not matched against the number of decryption servers required, say, there are only 3 available PKGs while a certain application requires 5 decryption servers.

Therefore, a better approach would be *sharing a private key associated with an identity* rather than sharing a master key of the PKG. In addition to its easy adaptability to the situation where an identity denotes a group sharing a decryption key as described in Section 1, an advantage of this approach is that one can fully utilize Boneh and Franklin's Distributed PKG method without the above-mentioned scalability problem, dividing the role of "distributed PKGs" from that of "decryption servers". That is, an authorized dealer (a representative of group, such as "Bob" described in Section 1, or a single PKG) may ask an identity to each of the "distributed PKGs" for a *partial* private key associated the identity. Having obtained enough partial private keys, the dealer can construct the whole private key and distribute it into the "decryption servers" in his domain at will while the master key remains secret from any parties.

Other Related Work on ID-Based Threshold Decryption. To our knowledge, other papers that have treated "threshold decryption" in the context of ID-based cryptography are [8] and [13]. Dodis and Yung [8] observed how threshold decryption can be realized in Gentry and Silverberg [12]'s "hierarchical ID-based encryption" setting. Interestingly, their approach is to share a private key (not the master key of the PKG) obtained from a user at a higher level. Although this was inevitable in the hierarchical ID-based encryption setting and its advantage in general ID-based cryptography was not mentioned in [8], it is more sound approach than sharing the master key of the PKG as we discussed above. However, their threshold decryption scheme is very-sketched and chosen-ciphertext security for the scheme was not considered in [8]. More recently, Libert and Quisquater [13] also constructed an ID-based threshold decryption scheme. However, their approach was to share a master key of the PKG, which is different from ours. Moreover, our scheme gives chosen-ciphertext security while Libert and Quisquater's scheme does not.

4 Security Notion for ID-based Threshold Decryption

4.1 Description of Generic ID-Based Threshold Decryption

A generic ID-based threshold decryption scheme, which we denote by "$\mathcal{IDTHD}$", consists of algorithms GK, EX, DK, E, D, SV, and SC. Below, we describe each of the algorithms.

Like other ID-based cryptographic schemes, we assume the existence of a trusted PKG. The PKG runs the key/common parameter generation algorithm GK to generate its master/public key pair and all the necessary common parameters. The PKG's public key and the common parameters are given to every interested party.

On receiving a user's private key extraction request which consists of an identity, the PKG then runs the private key extraction algorithm EX to generate the private key associated with the requested identity.

An authorized dealer who possesses the private key associated with an identity can run the private key distribution algorithm DK to distribute the private key into n decryption servers. DK makes use of an appropriate secret-sharing technique to generate shares of the private key as well as verification keys that will be used for checking the validity of decryption shares. Each share of the private key and its corresponding verification key are sent to an appropriate decryption server. The decryption servers then keep their private key shares secret but publish the verification keys. It is important to note here that the entity that runs DK can vary flexibly depending on the cryptographic services that the PKG can offer. For example, if the PKG has an only functionality of issuing private keys, the authorized dealer that runs DK would be a normal user (such as Bob in the example given in Section 1) other than the PKG. However, if the PKG has other functionalities, for example, organizing threshold decryption, the PKG can run DK.

Given a user's identity, any user that wants to encrypt a plaintext can run the encryption algorithm E to obtain a ciphertext. A *legitimate* user that wants to decrypt a ciphertext gives it to the decryption servers requesting decryption shares. The decryption servers then run the decryption share generation algorithm D taking the ciphertext as input and send the resulting decryption shares to the user. Note that the validity of the shares can be checked by running the decryption share verification algorithm SV. When the user collects valid decryption shares from at least t servers, the plaintext can be reconstructed by running the share combining algorithm SC.

Below, we formally define $\mathcal{IDTHD}$.

Definition 1 (ID-Based Threshold Decryption) The $\mathcal{IDTHD}$ scheme consists of the following algorithms.

- A randomized key/common parameter generation algorithm $\mathsf{GC}(k)$: Given a security parameter $k \in \mathbb{N}$, this algorithm computes the PKG's master/public key pair $(sk_{\mathrm{PKG}}, pk_{\mathrm{PKG}})$. Then, it generates necessary common parameters, e.g., descriptions of hash functions and mathematical groups. The output of this algorithm denoted by cp includes such parameters and the PKG's public key pk_{PKG}. Note that cp is given to all interested entities while the matching master key sk_{PKG} of pk_{PKG} is kept secret.

- A private key extraction algorithm $\mathsf{EX}(cp, \mathtt{ID})$: Given an identity $\mathtt{ID}$, this algorithm generates a private key associated with $\mathtt{ID}$, denoted by $sk_{\mathtt{ID}}$.

- A randomized private key distribution algorithm $\mathsf{DK}(cp, sk_{\mathtt{ID}}, n, t)$: Given a private key $sk_{\mathtt{ID}}$ associated with an identity $\mathtt{ID}$, a number of decryption servers n and a threshold parameter t, this algorithm generates n shares of $sk_{\mathtt{ID}}$ and provides each one to decryption servers $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$. It also generates a set of verification keys that can be used to check the validity of each shared private key. We denote the shared private keys and the matching verification keys by $\{sk_i\}_{1 \le i \le n}$ and $\{vk_i\}_{1 \le i \le n}$, respectively. Note that for each $1 \le i \le n$, the pair (sk_i, vk_i) is sent to the decryption server Γ_i, then Γ_i publishes vk_i but keeps sk_i as secret.

- A randomized encryption algorithm $\mathsf{E}(cp, \mathtt{ID}, M)$: Given a public identity $\mathtt{ID}$ and a plaintext M, this algorithm generates a ciphertext denoted by C.

- A decryption share generation algorithm $\mathsf{D}(cp, sk_i, C)$: Given a ciphertext C and a shared private key sk_i of a decryption server Γ_i, this algorithm generates a decryption share $\delta_{i,C}$. Note that the value of $\delta_{i,C}$ can be a special symbol "*Invalid Ciphertext*".

- A decryption share verification algorithm $\mathsf{SV}(cp, \{vk_i\}_{1 \le i \le n}, C, \delta_{i,C})$: Given a ciphertext C, a set of verification keys $\{vk_i\}_{1 \le i \le n}$, and a decryption share $\delta_{i,C}$, this algorithm checks the validity of $\delta_{i,C}$. The output of this algorithm is either "*Valid Share*" or "*Invalid Share*".

- A share combining algorithm $\mathsf{SC}(cp, C, \{\delta_{i,C}\}_{i \in \Phi})$: Given a ciphertext C and a set of decryption shares $\{\delta_{i,C}\}$ where $\Phi \subset \{1, \ldots, n\}$ such that $|\Phi| \ge t$ ($|\cdot|$ denotes the cardinality), this algorithm outputs a plaintext M. Note that the combining algorithm is allowed to output a special symbol "*Invalid Ciphertext*", which is distinct from all possible plaintexts.

4.2 Chosen Ciphertext Security for ID-Based Threshold Decryption

We now define a security notion for the $\mathcal{IDTHD}$ scheme against chosen-ciphertext attack, which we call "IND-IDTHD-CCA".

Definition 2 (IND-IDTHD-CCA) Let $\mathsf{A}^{\mathsf{CCA}}$ be an attacker assumed to be a probabilistic Turing machine. Suppose that a security parameter k is given to $\mathsf{A}^{\mathsf{CCA}}$ as input. Now, consider the following game in which the attacker $\mathsf{A}^{\mathsf{CCA}}$ interacts with the "Challenger".

Phase 1: The Challenger runs the PKG's key/common parameter generation algorithm taking a security parameter k as input. The Challenger gives $\mathsf{A}^{\mathsf{CCA}}$ the resulting common parameter cp which includes the PKG's public key pk_{PKG}. However, the Challenger keeps the master key sk_{PKG} secret from $\mathsf{A}^{\mathsf{CCA}}$.

Phase 2: $\mathsf{A}^{\mathsf{CCA}}$ issues a number of private key extraction queries. We denote each of these queries by $\mathtt{ID}$. On receiving the identity query $\mathtt{ID}$, the Challenger runs the private key extraction algorithm on input $\mathtt{ID}$ and obtains a corresponding private key $sk_{\mathtt{ID}}$. Then, the Challenger returns $sk_{\mathtt{ID}}$ to $\mathsf{A}^{\mathsf{CCA}}$.

Phase 3: $\mathsf{A}^{\mathsf{CCA}}$ corrupts $t-1$ out of n decryption servers.

Phase 4: $\mathsf{A}^{\mathsf{CCA}}$ issues a target identity query $\mathtt{ID}^*$. On receiving $\mathtt{ID}^*$, the Challenger runs the private key extraction algorithm to obtain a private key $sk_{\mathtt{ID}^*}$ associated with the target identity. The Challenger then runs the private key distribution algorithm on input $sk_{\mathtt{ID}^*}$ with parameter (t, n) and obtains a set of private/verification key pairs $\{(sk_{\mathtt{ID}^*_i}, vk_{\mathtt{ID}^*_i})\}$, where $1 \leq i \leq n$. Next, the Challenger gives $\mathsf{A}^{\mathsf{CCA}}$ the private keys of corrupted decryption servers and the verifications keys of all the decryption servers. However, the private keys of uncorrupted servers are kept secret from $\mathsf{A}^{\mathsf{CCA}}$.

Phase 5: $\mathsf{A}^{\mathsf{CCA}}$ issues arbitrary private key extraction queries and arbitrary decryption share generation queries to the uncorrupted decryption servers. We denote each of these queries by $\mathtt{ID}$ and C respectively. On receiving $\mathtt{ID}$, the Challenger runs the private key extraction algorithm to obtain a private key associated with $\mathtt{ID}$ and returns it to $\mathsf{A}^{\mathsf{CCA}}$. The only restriction here is that $\mathsf{A}^{\mathsf{CCA}}$ is not allowed to query the target identity $\mathtt{ID}^*$ to the private key extraction algorithm. On receiving C, the Challenger runs the decryption share generation algorithm taking C and the target identity $\mathtt{ID}^*$ as input to obtain a corresponding decryption share and returns it to $\mathsf{A}^{\mathsf{CCA}}$.

Phase 6: $\mathsf{A}^{\mathsf{CCA}}$ outputs two equal-length plaintexts (M_0, M_1). Then the Challenger chooses a bit β uniformly at random and runs the encryption algorithm on input cp, M_β and $\mathtt{ID}^*$ to obtain a target ciphertext $C^* = \mathsf{E}(cp, \mathtt{ID}^*, M_\beta)$. Finally, the Challenger gives $(C^*, \mathtt{ID}^*)$ to $\mathsf{A}^{\mathsf{CCA}}$.

Phase 7: $\mathsf{A}^{\mathsf{CCA}}$ issues arbitrary private key extraction queries and arbitrary decryption share generation queries. We denote each of these queries by $\mathtt{ID}$ and C respectively. On receiving $\mathtt{ID}$, the Challenger runs the private key extraction algorithm to obtain a private key associated with $\mathtt{ID}$ and returns it to $\mathsf{A}^{\mathsf{CCA}}$. As Phase 5, the only restriction here is that $\mathsf{A}^{\mathsf{CCA}}$ is not allowed to query the target identity $\mathtt{ID}^*$ to the private key extraction algorithm. On receiving C, the Challenger runs the decryption share generation algorithm on input C to obtain a corresponding decryption share and returns it to $\mathsf{A}^{\mathsf{CCA}}$. Differently from Phase 5, the target ciphertext C^* is not allowed to query in this phase.

Phase 8: $\mathsf{A}^{\mathsf{CCA}}$ outputs a guess $\tilde{\beta} \in \{0, 1\}$.

We define the attacker $\mathsf{A}^{\mathsf{CCA}}$'s success by

$$\mathbf{Succ}^{\mathrm{IND-IDTHD-CCA}}_{\mathcal{IDTHD},\mathsf{A}^{\mathsf{CCA}}}(k) = 2 \cdot \Pr[\tilde{\beta} = \beta] - 1.$$

We denote by $\mathbf{Succ}^{\mathrm{IND-IDTHD-CCA}}_{\mathcal{IDTHD}}(t_{IDCCA}, q_E, q_D)$ the maximum of the attacker $\mathsf{A}^{\mathsf{CCA}}$'s success over all attackers $\mathsf{A}^{\mathsf{CCA}}$ having running time t_{IDCCA} and making at most q_E private key extraction queries and q_D decryption share generation queries. Note that the running time and the number of queries are all polynomial in the security parameter k.

The ID-based threshold decryption scheme $\mathcal{IDTHD}$ is said to be IND-IDTHD-CCA secure if $\mathbf{Succ}^{\mathrm{IND-IDTHD-CCA}}_{\mathcal{IDTHD}}(t_{IDCCA}, q_E, q_D)$ is negligible in k.

5 Our ID-Based Threshold Decryption Scheme

5.1 Building Blocks

First, we present necessary building blocks that will be used to construct our ID-based threshold decryption scheme. We remark that since our ID-based threshold decryption scheme is also of the Diffie-Hellman (DH)-type, it follows Shoup and Gennaro [18]'s framework for the design of

DH-based threshold decryption schemes to some extent. However, our scheme has a number of features that distinguishes itself from the schemes in [18] due to the special property of the underlying group $\mathcal{G}$.

Publicly Checkable Encryption. Publicly checkable encryption is a particularly important tool for building threshold decryption schemes secure against chosen-ciphertext attack as discussed by Lim and Lee [14]. The main reason is that in the threshold decryption, the attacker has decryption shares as additional information as well as a ciphertext, hence there is a big chance for the attacker to get enough decryption shares to recover the plaintext before the validity of the ciphertext is checked. (Readers are referred to [14] and [18] for more detailed discussions on this issue.)

The public checkability of ciphertexts in threshold decryption schemes is usually given by non-interactive zero-knowledge (NIZK) proofs, e.g., [18, 10]. However, we emphasize that in our scheme, this can be done *without* a NIZK proof, by simply creating a tag on the ElGamal [9] ciphertext in a similar way as was done in the signature scheme of Boneh et al. [5].

Let $M \in \{0,1\}^l$ be a message. Then, encrypt M by creating a ciphertext $C = (U, V, W) = (rP, \mathsf{H}_2(\kappa) \oplus M, r\mathsf{H}_3(U, V))$ where $\kappa = \hat{e}(\mathsf{H}_1(\mathtt{ID}), Y_{\mathrm{PKG}})^r$ for hash functions $\mathsf{H}_1 : \{0,1\}^* \to \mathcal{G}^*$, $\mathsf{H}_2 : \mathcal{F} \to \{0,1\}^l$, and $\mathsf{H}_3 : \mathcal{G}^* \times \{0,1\}^l \to \mathcal{G}^*$. Without recovering M during the decryption process (that is, leaving the ciphertext C intact), the validity of C can be checked by testing if $\hat{e}(P, W) = \hat{e}(U, H_3)$, where $H_3 = \mathsf{H}_3(U, V) \in \mathcal{G}^*$. Note that this validity test exploits the fact that the Decisional Diffie-Hellman (DDH) problem can be solved in polynomial time in the group $\mathcal{G}$, and passing the test implies that (P, U, H_3, W) is a Diffie-Hellman tuple since $(P, U, H_3, W) = (P, rP, sP, rsP)$ assuming that $H_3 = sP \in_R \mathcal{G}^*$ for some $s \in \mathbb{Z}_q^*$.

Sharing a Point on $\mathcal{G}$. In order to share a private key $D_{\mathtt{ID}} \in \mathcal{G}$, we need some trick. In what follows, we present a Shamir's (t, n)-secret-sharing over $\mathcal{G}$.

Let q be a prime order of a group $\mathcal{G}$ (of points on elliptic curve). Let $S \in \mathcal{G}^*$ be a point to share. Suppose that we have chosen integers t (a threshold) and n satisfying $1 \leq t \leq n < q$. First, we pick $R_1, R_2, \ldots, R_{t-1}$ at random from $\mathcal{G}^*$. Then, we define a function $F : \mathbb{N} \cup \{0\} \to \mathcal{G}$ such that $F(u) = S + \sum_{l=1}^{t-1} u^l R_l$. (Note that in practice, "picking R_l at random from $\mathcal{G}^*$" can be implemented by computing $r_l P$ for randomly chosen $r_l \in \mathbb{Z}_q^*$, where $P \in \mathcal{G}^*$ is a generator of $\mathcal{G}$.) We then compute $S_i = F(i) \in \mathcal{G}$ for $1 \leq i \leq n$ and send (i, S_i) to the i-th member of the group of cardinality n. When the number of shares reaches the threshold t, the function $F(u)$ can be reconstructed by computing $F(u) = \sum_{j \in \Phi} c_{uj}^{\Phi} S_j$ where $c_{uj}^{\Phi} = \prod_{\iota \in \Phi, \iota \neq j} \frac{u - \iota}{j - \iota} \in \mathbb{Z}_q$ is the Lagrange coefficient for a set $\Phi \subset \{1, \ldots, n\}$ such that $|\Phi| \geq t$.

Zero Knowledge Proof for the Equality of Two Discrete Logarithms Based on the Bilinear Map. To ensure that all decryption shares are consistent, that is, to give robustness to threshold decryption, we need a certain checking procedure. In contrast to the ciphertext validity checking mechanism of in our publicly checkable encryption presented above, we need a non-interactive zero-knowledge proof system since the share of the key κ is the element of the group $\mathcal{F}$, where the DDH problem is believed to be hard.

Motivated by [6] and [18], we construct a zero-knowledge proof of membership system for the language $L_{\mathsf{EDLog}_{P,\tilde{P}}^{\mathcal{F}}} \stackrel{\mathrm{def}}{=} \{(\mu, \tilde{\mu}) \in \mathcal{F} \times \mathcal{F} | \log_g \mu = \log_{\tilde{g}} \tilde{\mu}\}$ where $g = \hat{e}(P, P)$ and $\tilde{g} = \hat{e}(P, \tilde{P})$ for generators P and $\tilde{P}$ of $\mathcal{G}$ (the groups $\mathcal{G}$ and $\mathcal{F}$ and the Bilinear map $\hat{e}$ are as defined in Section 2) as follows.

Suppose that $(P, \tilde{P}, g, \tilde{g})$ and $(\kappa, \tilde{\kappa}) \in L_{\mathsf{EDLog}_{P,\tilde{P}}^{\mathcal{F}}}$ are given to the Prover and the Verifier, and the Prover knows a secret $S \in \mathcal{G}^*$. The proof system which we call "$\mathtt{ZKBm}$" works as follows.

- The Prover chooses a non-identity element T uniformly at random from $\mathcal{G}$ and computes $\gamma = \hat{e}(T, P)$ and $\tilde{\gamma} = \hat{e}(T, \tilde{P})$. The Prover sends γ and $\tilde{\gamma}$ to the Verifier.
- The Verifier chooses h uniformly at random from $\mathbb{Z}_q^*$ and sends it to the Prover.
- On receiving h, the Prover computes $L = T + hS \in \mathcal{G}$ and sends it to the Verifier. The Verifier checks if $\hat{e}(L, P) = \gamma\kappa^h$ and $\hat{e}(L, \tilde{P}) = \tilde{\gamma}\tilde{\kappa}^h$. If the equality holds then the Verifier returns "*Accept*", otherwise, returns "*Reject*".

The above protocol actually satisfies completeness, soundness and zero-knowledge against the honest Verifier We state the following lemma regarding the security of `ZKBm`.

Lemma 1 *The* `ZKBm` *protocol satisfies completeness, soundness and zero-knowledge against the honest Verifier.*

Proof. As preliminaries, we first prove the following two claims.

Claim 1 *Let P and $\tilde{P}$ be generators of $\mathcal{G}$. Then $\hat{e}(P, \tilde{P})$ is a generator of $\mathcal{F}$.*

Proof. The proof will use the basic fact from the elementary abstract algebra that if a is a generator of a finite cyclic group G of order n, then the other generators of G are the elements of the form a^r, where $gcd(r, n) = 1$.

First, note that the two groups $\mathcal{G}$ and $\mathcal{F}$ are cyclic because their order q is a prime. Since $\tilde{P}$ is another generator of $\mathcal{G}$ by assumption, we can write $\tilde{P} = uP$, where $gcd(u, q) = 1$. Then, by the bilinear property of $\hat{e}$, we have $\hat{e}(P, \tilde{P}) = \hat{e}(P, uP) = \hat{e}(P, P)^u$. Also, by the non-degenerate property of $\hat{e}$, $\hat{e}(P, P)$ is a generator of $\mathcal{F}$. Hence, $\hat{e}(P, \tilde{P})$ is also a generator of $\mathcal{F}$ since $\hat{e}(P, \tilde{P}) = \hat{e}(P, P)^u$ and $gcd(u, q)$=1. □

Claim 2 *Let P and $\tilde{P}$ be generators of $\mathcal{G}$. Then, $(\kappa, \tilde{\kappa}) \in L_{\mathsf{EDLog}^{\mathcal{F}}_{P,\tilde{P}}}$ if and only if there exists a non-identity element $S \in \mathcal{G}$ such that $\kappa = \hat{e}(S, P)$ and $\tilde{\kappa} = \hat{e}(S, \tilde{P})$.*

Proof. By Claim 1, g and $\tilde{g}$ are generators of $\mathcal{F}$. Now, suppose that $(\kappa, \tilde{\kappa}) \in L_{\mathsf{EDLog}^{\mathcal{F}}_{P,\tilde{P}}}$. Then, by definition of $L_{\mathsf{EDLog}^{\mathcal{F}}_{P,\tilde{P}}}$, there exists $x \in \mathbb{Z}_q^*$ such that $g^x = \tilde{g}^x$. Since $g = \hat{e}(P, P)$ and $\tilde{g} = \hat{e}(P, \tilde{P})$, $g^x = \tilde{g}^x$ implies $\hat{e}(P, P)^x = \hat{e}(P, \tilde{P})^x$. But, since $\hat{e}(P, P)^x = \hat{e}(xP, P)$ and $\hat{e}(P, \tilde{P})^x = \hat{e}(xP, \tilde{P})$ by the bilinear property of $\hat{e}$, we obtain $\kappa = \hat{e}(S, P)$ and $\tilde{\kappa} = \hat{e}(S, \tilde{P})$ by letting $S = xP$. The proof of converse is also easy. □

Now, we show that the protocol is complete. That is, if the Prover and the Verifier follow the protocol without cheating, the Verifier accepts the Prover's claim with overwhelming probability: Assume that $(\kappa, \tilde{\kappa}) \in L_{\mathsf{EDLog}^{\mathcal{F}}_{P,\tilde{P}}}$. By Claim 2, we have $\kappa = \hat{e}(S, P)$ and $\tilde{\kappa} = \hat{e}(S, \tilde{P})$ for some $S \in \mathcal{G}$. Assume that the Prover sends $(\gamma, \tilde{\gamma})$ where $\gamma = \hat{e}(T, P)$ and $\tilde{\gamma} = \hat{e}(T, \tilde{P})$ for random $T \in \mathcal{G}$ to the honest Verifier. Now, observe from the above protocol that $\hat{e}(L, P) = \hat{e}(T + hS, P)$ and that $\gamma\kappa^h = \hat{e}(T, P)\hat{e}(S, P)^h = \hat{e}(T, P)\hat{e}(hS, P)$. By the bilinear property of $\hat{e}$, we have $\hat{e}(T, P)\hat{e}(hS, P) = \hat{e}(T + hS, P)$. Thus, we obtain $\hat{e}(L, P) = \gamma\kappa^h$ and this implies that the above protocol satisfies completeness property.

Second, we show the soundness of the protocol: Assume that $(\kappa, \tilde{\kappa}) \notin L_{\mathsf{EDLog}^{\mathcal{F}}_{P,\tilde{P}}}$. Namely, we have $\kappa = \hat{e}(S, P)$ and $\tilde{\kappa} = \hat{e}(S', \tilde{P})$ for some $S \neq S' \in \mathcal{G}$. Assume that a cheating Prover sends $(\gamma, \tilde{\gamma})$ where $\gamma = \hat{e}(T, P)$ and $\gamma = \hat{e}(T', \tilde{P})$ to the honest Verifier. If the Verifier is to accept this, we should have that $\hat{e}(L, P) = \gamma\kappa^h$ and $\hat{e}(L, \tilde{P}) = \tilde{\gamma}\tilde{\kappa}^h$, which implies $T + hS = T' + hS'$. Now suppose that $T = tP$, $T' = t'P$; $S = xP$ and $S' = x'P$ for $t, t', x, x' \in \mathbb{Z}_q^*$. Then, $T + hS = T' + hS'$ implies $(t - t') + h(x - x') = 0$. However, this happens with probability $1/q$, since we have assumed that $S' \neq S$ which implies $x' \neq x$.

Finally, we can construct a simulator which simulates the communication between the Prover and the Verifier provided that the Verifier behaves *honestly.* More precisely, the simulator chooses $\bar{h}$ and $\bar{L}$ uniformly at random from $\mathbb{Z}_q^*$ and $\mathcal{G}$ respectively. Then, it computes $\bar{\gamma} = \hat{e}(\bar{L}, P)/\kappa^{\bar{h}}$ and $\bar{\tilde{\gamma}} = \hat{e}(\bar{L}, \tilde{P})/\tilde{\kappa}^{\bar{h}}$. The output of the simulator is a tuple $(\bar{\gamma}, \bar{\tilde{\gamma}}, \bar{h}, \bar{L})$. It can be easily verified that the simulated values are identically distributed as those in the real communication if the Verifier behaves honestly. As a result, the above protocol becomes a zero-knowledge proof against a honest Verifier. $\square$

Note that `ZKBm` can easily be converted to a NIZK proof, making the random challenge an output of a random oracle [1]. Note also that the above protocol can be viewed as a proof that $(g, \tilde{g}, \kappa, \tilde{\kappa})$ is a Diffie-Hellman tuple since if $(\kappa, \tilde{\kappa}) \in L_{\mathsf{EDLog}^{\mathcal{F}}_{P,\tilde{P}}}$ then $\kappa = g^x$ and $\tilde{\kappa} = \tilde{g}^x$ for some $x \in \mathbb{Z}_q^*$ and hence $(g, \tilde{g}, \kappa, \tilde{\kappa}) = (g, \tilde{g}, g^x, \tilde{g}^x) = (g, g^y, g^x, g^{xy})$ for some $y \in \mathbb{Z}_q^*$.

5.2 Description of Our Scheme – `IdThdBm`

We now describe our ID-based threshold decryption scheme. We call our scheme "`IdThdBm`", meaning "ID-based threshold decryption scheme from the bilinear map". `IdThdBm` consists of the following algorithms.

- $\mathsf{GK}(k)$: Given a security parameter k, this algorithm generates two groups $\mathcal{G}$ and $\mathcal{F}$ of the same prime order $q \geq 2^k$ and chooses a generator P of $\mathcal{G}$. Then, it specifies the Bilinear map $\hat{e} : \mathcal{G} \times \mathcal{G} \to \mathcal{F}$ and the hash functions H_1, H_2, H_3 and H_4 such that $\mathsf{H}_1 : \{0,1\}^* \to \mathcal{G}^*$; $\mathsf{H}_2 : \mathcal{F} \to \{0,1\}^l$; $\mathsf{H}_3 : \mathcal{G}^* \times \{0,1\}^l \to \mathcal{G}^*$; $\mathsf{H}_4 : \mathcal{F} \times \mathcal{F} \times \mathcal{F} \to \mathbb{Z}_q^*$, where l denotes the length of a plaintext. Next, it chooses the PKG's master key x uniformly at random from $\mathbb{Z}_q^*$ and computes the PKG's public key $Y_{\text{PKG}} = xP$. Finally, it returns a common parameter $cp = (\mathcal{G}, q, P, \hat{e}, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3, \mathsf{H}_4, Y_{\text{PKG}})$ while keeping the master key x secret.
- $\mathsf{EX}(cp, \mathtt{ID})$: Given an identity $\mathtt{ID}$, this algorithm computes $Q_{\mathtt{ID}} = \mathsf{H}_1(\mathtt{ID})$ and $D_{\mathtt{ID}} = xQ_{\mathtt{ID}}$. Then, it returns the private key $D_{\mathtt{ID}}$ associated with $\mathtt{ID}$.
- $\mathsf{DK}(cp, \mathtt{ID}, D_{\mathtt{ID}}, t, n)$ where $1 \leq t \leq n < q$: Given a private key $D_{\mathtt{ID}}$, the number of decryption servers n and a threshold parameter t, this algorithm first picks $R_1, R_2, \ldots, R_{t-1}$ at random from $\mathcal{G}^*$ and constructs $F(u) = D_{\mathtt{ID}} + \sum_{j=1}^{t-1} u^j R_j$ for $u \in \{0\} \cup \mathbb{N}$. It then computes each server Γ_i's private key $S_i = F(i)$ and verification key $y_i = \hat{e}(S_i, P)$ for $1 \leq i \leq n$. Subsequently, it secretly sends the distributed private key S_i and the verification key y_i to server Γ_i for $1 \leq i \leq n$. Γ_i then keeps S_i as secret while making y_i public.
- $\mathsf{E}(cp, \mathtt{ID}, m)$: Given a plaintext $M \in \{0,1\}^l$ and an identity $\mathtt{ID}$, this algorithm chooses r uniformly at random from $\mathbb{Z}_q^*$, and subsequently computes $Q_{\mathtt{ID}} = \mathsf{H}_1(\mathtt{ID})$ and $\kappa =$

$\hat{e}(Q_{\mathtt{ID}}, Y_{\mathrm{PKG}})^r$. It then computes

$$U = rP; V = \mathsf{H}_2(\kappa) \oplus M; W = r\mathsf{H}_3(U, V)$$

and returns a ciphertext $C = (U, V, W)$.

- $\mathsf{D}(cp, S_i, C)$: Given a private key S_i of each decryption server and a ciphertext $C = (U, V, W)$, this algorithm computes $H_3 = \mathsf{H}_3(U, V)$ and checks if $\hat{e}(P, W) = \hat{e}(U, H_3)$.

  If C has passed the above test, this algorithm computes $\kappa_i = \hat{e}(S_i, U)$, $\tilde{\kappa_i} = \hat{e}(T_i, U)$, $\tilde{y_i} = \hat{e}(T_i, P)$, $\lambda_i = \mathsf{H}_4(\kappa_i, \tilde{\kappa_i}, \tilde{y_i})$, and $L_i = T_i + \lambda_i S_i$ for random $T_i \in \mathcal{G}$, and outputs $\delta_{i,C} = (i, \kappa_i, \tilde{\kappa_i}, \tilde{y_i}, \lambda_i, L_i)$. Otherwise, it returns $\delta_{i,C} =(i,$ "*Invalid Ciphertext*").

- $\mathsf{SV}(cp, \{y_i\}_{1 \le i \le n}, C, \delta_{i,C})$: Given a ciphertext $C = (U, V, W)$, a set of verification keys $\{y_1, \ldots, y_n\}$, and a decryption share $\delta_{i,C}$, this algorithm computes $H_3 = \mathsf{H}_3(U, V)$ and checks if $\hat{e}(P, W) = \hat{e}(U, H_3)$.

  If C has passed the above test then this algorithm does the following:

  - If $\delta_{i,C}$ is of the form $(i,$ "*Invalid Ciphertext*") then return "*Invalid Share*".
  - Else parse $\delta_{i,C}$ as $(i, \kappa_i, \tilde{\kappa_i}, \tilde{y_i}, \lambda_i, L_i)$ and compute $\lambda'_i = \mathsf{H}_4(\kappa_i, \tilde{\kappa_i}, \tilde{y_i})$.
    - Check if $\lambda'_i = \lambda_i$, $\hat{e}(L_i, U)/\kappa_i^{\lambda'_i} = \tilde{\kappa_i}$ and $\hat{e}(L_i, P)/y_i^{\lambda'_i} = \tilde{y_i}$.
    - If the test above holds, return "*Valid Share*", else output "*Invalid Share*".

  Otherwise, does the following:

  - If $\delta_{i,C}$ is of the form $(i,$ "*Invalid Ciphertext*"), return "*Valid Share*", else output "*Invalid Share*".

- $\mathsf{SC}(cp, C, \{\delta_{j,C}\}_{j \in \Phi})$: Given a ciphertext C and a set of valid decryption shares $\{\delta_{j,C}\}_{j \in \Phi}$ where $|\Phi| \ge t$, this algorithm computes $H_3 = \mathsf{H}_3(U, V)$ and checks if $\hat{e}(P, W) = \hat{e}(U, H_3)$.

  If C has not passed the above test, this algorithm returns "*Invalid Ciphertext*". (In this case, all the decryption shares are of the form $(i,$ "*Invalid Ciphertext*").) Otherwise, it computes $\kappa = \prod_{j \in \Phi} \kappa_j^{c_{0j}^{\Phi}}$ and $M = \mathsf{H}_2(\kappa) \oplus V$, and returns M.

5.3 Security Analysis – `IdThdBm`

Bilinear Diffie-Hellman Problem. First, we review the Bilinear Diffie-Hellman (BDH) problem, which was introduced by Boneh and Franklin [4].

Definition 3 (BDH) Let $\mathcal{G}$ and $\mathcal{F}$ be two groups of a prime order $q \ge 2^k$, where k is security parameter. Let $P \in \mathcal{G}^*$ be a generator of $\mathcal{G}$. Suppose that there exists a bilinear map $\hat{e} : \mathcal{G} \times \mathcal{G} \to \mathcal{F}$. Let $\mathsf{A}^{\mathsf{BDH}}$ 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.

$\mathsf{A}^{\mathsf{BDH}}$ is to solve the following problem:

- Given $(\mathcal{G}, q, \hat{e}, P, aP, bP, cP)$, compute $\hat{e}(P, P)^{abc}$.

We define $\mathsf{A}^{\mathsf{BDH}}$'s success by

$$\mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G},\mathsf{A}^{\mathsf{BDH}}}(k) = \Pr[\mathsf{A}^{\mathsf{BDH}} \text{ outputs } \hat{e}(P,P)^{abc}].$$

We denote by $\mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G}}(t_{\mathsf{BDH}})$ the maximal success probability $\mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G},\mathsf{A}^{\mathsf{BDH}}}(k)$ over all attackers having running time bounded by t_{BDH} which is polynomial in the security parameter k.

The BDH problem is said to be computationally intractable if $\mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G}}(t_{BDH})$ is negligible in k.

Proof of Security. Regarding the security of the `IdThdBm` scheme, we obtain the following theorem implying that the `IdThdBm` scheme is IND-IDTHD-CCA secure in the random oracle model assuming that the BDH problem is computationally intractable.

Theorem 1 *Suppose that an IND-IDTHD-CCA attacker for the scheme* `IdThdBm` *issues up to* q_E *private key extraction queries,* q_D *decryption share generation queries,* q_{H_1}*,* q_{H_2}*,* q_{H_3}*, and* q_{H_4} *queries to to the random oracles* H_1*,* H_2*,* H_3*, and* H_4 *respectively. Using this attacker as a subroutine, we can construct an attacker for solving the BDH problem in the group* $\mathcal{G}$*, whose running time is bounded by* t_{BDH}*. Concretely, we obtain the following advantage bound:*

$$\frac{1}{q_{\mathsf{H}_1}}\mathbf{Succ}^{\mathrm{IND-IDTHD-CCA}}_{\mathtt{IdThdBm}}(t_{IDCCA}, q_E, q_D, q_{\mathsf{H}_1}, q_{\mathsf{H}_2}, q_{\mathsf{H}_3}, q_{\mathsf{H}_4}) \leq 2\mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G}}(t_{\mathrm{BDH}}) + \frac{q_D + q_D q_{\mathsf{H}_4}}{2^{k-1}},$$

where $t_{\mathrm{BDH}} = t_{IDCCA} + \max(q_E, q_{\mathsf{H}_1})O(k^3) + q_{\mathsf{H}_1} + q_{\mathsf{H}_2}O(k^3) + q_{\mathsf{H}_4}q_D O(k^3)$ *for a security parameter* k*.*

To prove the above theorem, we derive a non-ID-based threshold decryption scheme called "`ThdBm`" from the `IdThdBm` scheme, which will be described shortly. We then show in Lemma 2 that the IND-THD-CCA security of the `ThdBm` scheme, which will be defined after the description of `ThdBm`, implies the IND-IDTHD-CCA security of the `IdThdBm` scheme. Next, we show in Lemma 3 that the intractability of the BDH problem implies the THD-IND-CCA security of the `ThdBm` scheme. Combining Lemmas 2 and 3, we obtain Theorem 1.

As mentioned, we describe the `ThdBm` scheme. Actually, `ThdBm` is very similar to `IdThdBm` except for some differences in the key/common parameter generation and encryption algorithms. We only describe these two algorithms here.

- $\mathsf{GK}(k,t,n)$: Taking a security parameter k as input, this algorithm generates two groups $\mathcal{G}$ and $\mathcal{F}$ of the same prime order $q \geq 2^k$ and chooses a generator P of $\mathcal{G}$. Then, it specifies the Bilinear map $\hat{e} : \mathcal{G} \times \mathcal{G} \to \mathcal{F}$ and the following hash functions H_2, H_3, and H_4 such that $\mathsf{H}_2 : \mathcal{F} \to \{0,1\}^l$; $\mathsf{H}_3 : \mathcal{G}^* \times \{0,1\}^l \to \mathcal{G}^*$; $\mathsf{H}_4 : \mathcal{F} \to \mathbb{Z}_q^*$, where l denotes the length of a plaintext. Next, it chooses x uniformly at random from $\mathbb{Z}_q^*$ and computes $Y = xP$. Then, it chooses Q uniformly at random from $\mathcal{G}^*$ and computes $D = xQ$. Note that (Y, D) will be a public/private key pair. Now, given a private key D, the number of decryption servers n and a threshold parameter t, this algorithm picks $R_1, R_2, \ldots, R_{t-1}$ at random from $\mathcal{G}$ and computes $F(x) = D + \sum_{j=1}^{t-1} x^j R_j$. Then, it computes each server's private key $S_i = F(i)$ for $1 \leq i \leq n$ and verification key $y_i = \hat{e}(S_i, P)$ for $1 \leq i \leq n$. Finally, it outputs a common parameter $cp = (\mathcal{G}, q, P, \hat{e}, \mathsf{H}_2, \mathsf{H}_3, \mathsf{H}_4, Y, Q)$, and sends the verification/private key pair (y_i, S_i) to each decryption server Γ_i for $1 \leq i \leq n$. Upon receiving (y_i, S_i), each decryption server publishes y_i where $1 \leq i \leq n$.

- $\mathsf{E}(cp, m)$: Given a plaintext message $m \in \{0,1\}^l$, this algorithm chooses r uniformly at random from $\mathbb{Z}_q^*$ and computes $d = \hat{e}(Q, Y)$, $\kappa = d^r$ in turn. Then, it computes $U = rP$, $V = \mathsf{H}_2(\kappa) \oplus m$ and $W = r\mathsf{H}_3(U, V)$, and outputs a ciphertext $C = (U, V, W)$.

We now review the chosen-ciphertext security notion of (non-ID-based) threshold decryption. First, we denote a generic (t, n) threshold decryption scheme in the non-ID-based setting by "$\mathcal{THD}$". The $\mathcal{THD}$ scheme consists of a key/common parameter generation algorithm GK, an encryption algorithm E, a decryption share generation algorithm D, a decryption share verification algorithm SV, and a share combining algorithm SC.

By running GC, a trusted dealer generates a public key and its matching private key, and shares the private key among a n decryption servers. The dealer also generates (public) verification keys that will be used for share verification. Given the public key, a sender encrypts a plaintext by running E. A user who wants to decrypt a ciphertext gives the ciphertext to the decryption servers requesting decryption shares. The decryption servers then run D to generate corresponding decryption shares. The user can check the validity of the shares by running SV. When the user collects valid decryption shares from at least t servers, the ciphertext can be decrypted by running SC.

We now review the security notion for the threshold decryption scheme against chosen-ciphertext attack, which we call "IND-THD-CCA", defined in [18].

Definition 4 (IND-THD-CCA) Let $\mathsf{B}^{\mathsf{CCA}}$ be an attacker assumed to be a probabilistic Turing machine. Suppose that a security parameter k is given to $\mathsf{B}^{\mathsf{CCA}}$ as input. Now, consider the following game in which the attacker $\mathsf{B}^{\mathsf{CCA}}$ interacts with the "Challenger".

Phase 1: $\mathsf{B}^{\mathsf{CCA}}$ corrupts a fixed subset of $t-1$ servers.

Phase 2: The Challenger runs the key/common parameter generation algorithm GK taking a security parameter k as input. The Challenger gives $\mathsf{B}^{\mathsf{CCA}}$ the resulting private keys of the corrupted servers, the public key, the verification key and the common parameter. However, the Challenger keeps the private keys of uncorrupted servers secret from $\mathsf{B}^{\mathsf{CCA}}$.

Phase 3: $\mathsf{B}^{\mathsf{CCA}}$ adaptively interacts with the uncorrupted decryption servers, submitting ciphertexts and obtaining decryption shares.

Phase 4: $\mathsf{B}^{\mathsf{CCA}}$ chooses two equal-length plaintexts (M_0, M_1). If these are given to the encryption algorithm then the Challenger chooses $\beta \in \{0,1\}$ at random and returns a target ciphertext $C^* = \mathsf{E}(cp, pk, M_\beta)$ to $\mathsf{B}^{\mathsf{CCA}}$.

Phase 5: $\mathsf{B}^{\mathsf{CCA}}$ 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: $\mathsf{B}^{\mathsf{CCA}}$ outputs a guess $\tilde{\beta} \in \{0,1\}$.

We define the attacker $\mathsf{B}^{\mathsf{CCA}}$'s success by

$$\mathbf{Succ}^{\mathrm{IND-THD-CCA}}_{\mathcal{THD},\mathsf{B}^{\mathsf{CCA}}}(k) = 2 \cdot \Pr[\tilde{\beta} = \beta] - 1.$$

We denote by $\mathbf{Succ}_{\mathcal{THD}}^{\mathrm{IND-THD-CCA}}(t_{CCA}, q_D)$ the maximum of the attacker $\mathsf{B}^{\mathsf{CCA}}$'s success over all attackers $\mathsf{B}^{\mathsf{CCA}}$ having running time t_{CCA} and making at most q_D decryption share generation queries. Note that the running time and the number of queries are all polynomial in the security parameter k.

The scheme $\mathcal{THD}$ is said to be IND-THD-CCA secure if $\mathbf{Succ}_{\mathcal{THD}}^{\mathrm{IND-THD-CCA}}(t_{CCA}, q_D)$ is negligible in k.

We now prove the following lemma.

Lemma 2 *Suppose that an IND-IDTHD-CCA attacker for the* `IdThdBm` *scheme issues up to* q_E *private key extraction queries,* q_D *decryption share generation queries,* q_{H_1}*,* q_{H_2}*,* q_{H_3}*, and* q_{H_4} *queries to the random oracles* H_1*,* H_2*,* H_3*, and* H_4 *respectively. Using this attacker as a subroutine, we can construct an IND-THD-CCA attacker for the* `ThdBm` *scheme, whose running time and the number of decryption share generation queries and the random oracle queries to* H_2*,* H_3*, and* H_4 *are bounded by* t_{CCA}*,* q'_D *and* q'_{H_2}*,* q'_{H_3}*, and* q'_{H_4} *respectively. Concretely, we obtain the following advantage bound:*

$$\begin{aligned}\frac{1}{q_{\mathsf{H}_1}}&\mathbf{Succ}_{\mathtt{IdThdBm}}^{\mathrm{IDTHD-IND-CCA}}(t_{IDCCA}, q_E, q_D, q_{\mathsf{H}_1}, q_{\mathsf{H}_2}, q_{\mathsf{H}_3}, q_{\mathsf{H}_4})\\ &\leq \mathbf{Succ}_{\mathtt{ThdBm}}^{\mathrm{THD-IND-CCA}}(t_{CCA}, q'_D, q'_{\mathsf{H}_2}, q'_{\mathsf{H}_3}, q'_{\mathsf{H}_4}),\end{aligned}$$

where $t_{CCA} = t_{IDCCA} + \max(q_E, q_{\mathsf{H}_1})O(k^3)$*,* $q'_D = q_D$*,* $q'_{\mathsf{H}_2} = q_{\mathsf{H}_2}$*,* $q'_{\mathsf{H}_3} = q_{\mathsf{H}_3}$ *and* $q'_{\mathsf{H}_4} = q_{\mathsf{H}_4}$ *for a security parameter* k*. Here,* t_{IDCCA} *denotes the running time of the IDTHD-IND-CCA attacker.*

Proof. For notational convenience, we assume that the same group parameters $\{\mathcal{G}, q, \hat{e}, P\}$ and security parameter k are given to attackers for `IdThdBm` and `ThdBm`.

Let $\mathsf{A}^{\mathsf{CCA}}$ denote an attacker that defeats the IND-IDTHD-CCA security of the `IdThdBm` scheme. We assume that $\mathsf{A}^{\mathsf{CCA}}$ has access to the common parameter $cp_{\mathtt{IdThdBm}} = (\mathcal{G},\ q,\ P,\ \hat{e},\ \mathsf{H}_1,\ \mathsf{H}_2,\ \mathsf{H}_3,\ \mathsf{H}_4,\ Y_{\mathrm{PKG}})$ of the `IdThdBm` scheme, where $Y_{\mathrm{PKG}} = x'P$ for random $x' \in \mathbb{Z}_q^*$. We also assume that $\mathsf{A}^{\mathsf{CCA}}$ has access to its decryption servers and a set of verification keys.

Let $\mathsf{B}^{\mathsf{CCA}}$ denote an attacker that defeats the THD-IND-CCA security of the `ThdBm` scheme. We assume that $\mathsf{B}^{\mathsf{CCA}}$ has access to the common parameter $cp_{\mathtt{ThdBm}} = (\mathcal{G},\ q,\ P,\ \hat{e},\ \mathsf{H}_2,\ \mathsf{H}_3,\ \mathsf{H}_4,\ Y,\ Q)$ of the `ThdBm` scheme, where $Y = xP$ for random $x \in \mathbb{Z}_q^*$ and Q has been randomly chosen from $\mathcal{G}$. Also, we assume that $\mathsf{B}^{\mathsf{CCA}}$ has access to its decryption servers and a set of verification keys.

Our aim is to simulate the view of $\mathsf{A}^{\mathsf{CCA}}$ in the real attack game denoted by $\mathbf{G_0}$ until we obtain a game denoted by $\mathbf{G_1}$, which is related to the ability of the attacker $\mathsf{B}^{\mathsf{CCA}}$ to defeat the THD-IND-CCA security of the `ThdBm` scheme.

- Game $\mathbf{G_0}$: As mentioned, this game is identical to the real attack game described in Definition 2. We denote by E_0 the event that $\mathsf{A}^{\mathsf{CCA}}$'s output $\tilde{\beta} \in \{0, 1\}$ is equal to $\beta \in \{0, 1\}$ chosen by the Challenger. We use a similar notation E_i for all Games $\mathbf{G}_i$. Since Game $\mathbf{G_1}$ is the same as the real attack game, we have

$$\Pr[E_0] = \frac{1}{2} + \frac{1}{2}\mathbf{Succ}_{\mathtt{IdThdBm},\mathsf{A}^{\mathsf{CCA}}}^{\mathrm{IND-IDTHD-CCA}}(k).$$

- Game $\mathbf{G_1}$: First, we replace Y_{PKG} of $\mathsf{A}^{\mathsf{CCA}}$'s common parameter $cp_{\mathtt{IdThdBm}}$ by Y of $\mathsf{B}^{\mathsf{CCA}}$'s common parameter $cp_{\mathtt{ThdBm}}$ setting $Y_{\mathrm{PKG}} = Y$. We also replace $\mathsf{A}^{\mathsf{CCA}}$'s decryption servers by $\mathsf{B}^{\mathsf{CCA}}$'s decryption servers. We then randomly choose an index μ from the range $[1, 2, \ldots, q_{\mathsf{H}_1}]$ where q_{H_1} denotes the maximum number queries to the random oracle H_1 made by $\mathsf{A}^{\mathsf{CCA}}$. By $\mathtt{ID}_\mu$, we denote the μ-th query to the random oracle H_1. We hope $\mathtt{ID}_\mu$ would be a target identity $\mathtt{ID}^*$ that $\mathsf{A}^{\mathsf{CCA}}$ outputs in Phase 4 of the real attack game of IND-IDTHD-CCA described in Definition 2.

  Now, we simulate $\mathsf{A}^{\mathsf{CCA}}$'s random oracle H_1, which can be queried at any time during the attack. Whenever H_1 is queried at $\mathtt{ID}$, we perform the following:

  - If the query $\mathtt{ID}$ exists in the entry $\langle(\mathtt{ID}, \tau), Q_{\mathtt{ID}}\rangle \in \mathsf{H}_1\mathsf{List}$, return $Q_{\mathtt{ID}}$ to $\mathsf{A}^{\mathsf{CCA}}$. (Note that $\mathsf{H}_1\mathsf{List}$ is the "input-output" list for the simulation of H_1.)
  - Otherwise, do the following.
    * If $\mathtt{ID} = \mathtt{ID}_\mu$, set $Q_{\mathtt{ID}} = Q$ and return $Q_{\mathtt{ID}}$ to $\mathsf{A}^{\mathsf{CCA}}$. (Note that Q is from $\mathsf{B}^{\mathsf{CCA}}$'s common parameter $cp_{\mathtt{ThdBm}}$.)
    * Else ($\mathtt{ID} \neq \mathtt{ID}_\mu$), do the following.
      · Choose τ uniformly at random from $\mathbb{Z}_q^*$.
      · Compute $Q_{\mathtt{ID}} = \tau P$ and return $Q_{\mathtt{ID}}$ to $\mathsf{A}^{\mathsf{CCA}}$.

  If $\mathsf{A}^{\mathsf{CCA}}$ issues $\mathtt{ID}$ as a private key extraction query, we perform the following:

  - If the query $\mathtt{ID}$ exists in the entry $\langle(\mathtt{ID}, Q_{\mathtt{ID}}), \tau)\rangle \in \mathsf{H}_1\mathsf{List}$, extract τ from it, compute $D_{\mathtt{ID}} = \tau Y$, and return $D_{\mathtt{ID}}$ to $\mathsf{A}^{\mathsf{CCA}}$.
  - Otherwise, do the following.
    * If $\mathtt{ID} = \mathtt{ID}_\mu$, terminate the whole game. (Note, however, that if $\mathtt{ID}_\mu = \mathtt{ID}^*$, this query is not allowed.)
    * Else ($\mathtt{ID} \neq \mathtt{ID}_\mu$), do the following.
      · Choose τ uniformly at random from $\mathbb{Z}_q^*$.
      · Compute $Q_{\mathtt{ID}} = \tau P$ and save $\langle(\mathtt{ID}, \tau), Q_{\mathtt{ID}}\rangle$ into $\mathsf{H}_1\mathsf{List}$.
      · Compute $D_{\mathtt{ID}} = \tau Y$ and return $D_{\mathtt{ID}}$ to $\mathsf{A}^{\mathsf{CCA}}$.

  If $\mathsf{A}^{\mathsf{CCA}}$ corrupts $t-1$ decryption servers during the attack, that is, $\mathsf{A}^{\mathsf{CCA}}$ obtains private keys $\{S_i\}_{1 \le i \le t-1}$ of corrupted decryption servers, we give them to $\mathsf{B}^{\mathsf{CCA}}$.

  If $\mathsf{A}^{\mathsf{CCA}}$ submits a pair of two equal-length plaintexts (M_0, M_1), we give it to $\mathsf{B}^{\mathsf{CCA}}$, then $\mathsf{B}^{\mathsf{CCA}}$ uses (M_0, M_1) as its plaintext-pair to be challenged. $\mathsf{B}^{\mathsf{CCA}}$ queries (M_0, M_1) to its Challenger, and obtains a target ciphertext C^* such that

  $$C^* = (U, V, W) = (rP, M_\beta \oplus \mathsf{H}_2(\hat{e}(Q, Y)^r), r\mathsf{H}_3(U, V)),$$

  where r and β are chosen uniformly at random from $\mathbb{Z}_q^*$ and $\{0, 1\}$ respectively. We simply return the C^* to $\mathsf{A}^{\mathsf{CCA}}$ as a target ciphertext.

  If $\mathsf{A}^{\mathsf{CCA}}$ issues decryption share generation queries after it submits the target identity, $\mathsf{B}^{\mathsf{CCA}}$ uses its decryption servers to answer those queries. Note, however, that $\mathsf{A}^{\mathsf{CCA}}$ is not allowed to query C^* to any of the uncorrupted decryption servers.

  Finally, if $\mathsf{A}^{\mathsf{CCA}}$ submits its guess $\tilde{\beta}$, we give it to $\mathsf{B}^{\mathsf{CCA}}$.

Note that due to the randomness of τ in $\mathbb{Z}_q^*$ and Q, the above simulation of the random oracle H_1 is perfect. Note also that $D_{\mathtt{ID}} = \tau Y_{\mathrm{PKG}} = \tau Y = \tau x P = x \tau P = x Q_{\mathtt{ID}}$ and that

$$
\begin{aligned}
C^* &= (rP, M_\beta \oplus \mathsf{H}_2(\hat{e}(Q_{\mathtt{ID}_\mu}, Y_{\mathrm{PKG}})^r), r\mathsf{H}_3(U,V)) \\
&= (rP, M_\beta \oplus \mathsf{H}_2(\hat{e}(Q_{\mathtt{ID}^*}, Y_{\mathrm{PKG}})^r), r\mathsf{H}_3(U,V)) \\
&= (rP, M_\beta \oplus \mathsf{H}_2(\hat{e}(\mathsf{H}_1(\mathtt{ID}^*), Y_{\mathrm{PKG}})^r), r\mathsf{H}_3(U,V)).
\end{aligned}
$$

Hence, as long as $\mathtt{ID}_\mu = \mathtt{ID}^*$, the private keys associated with IDs and the target ciphertext C^* that $\mathsf{A}^{\mathsf{CCA}}$ obtain in the simulation are identically distributed as those $\mathsf{A}^{\mathsf{CCA}}$ obtain in the real attack. (Note that if $\mathtt{ID}_\mu = \mathtt{ID}^*$, we do not terminate the game.)

Since μ has been uniformly chosen from $[1, q_{\mathsf{H}_1}]$ and the bit β is uniformly chosen from $\{0,1\}$, we have

$$
\Pr[E_1] - \frac{1}{2} \geq \frac{1}{q_{\mathsf{H}_1}}\Big(\Pr[E_0] - \frac{1}{2}\Big).
$$

Thus, by definition of $\Pr[E_0]$ and $\Pr[E_1]$, we obtain

$$
\mathbf{Succ}^{\mathrm{IND-THD-CCA}}_{\mathtt{ThdBm},\mathsf{B}^{\mathsf{CCA}}}(k) \geq \frac{1}{q_{\mathsf{H}_1}} \mathbf{Succ}^{\mathrm{IND-IDTHD-CCA}}_{\mathtt{IdThdBm},\mathsf{A}^{\mathsf{CCA}}}(k).
$$

Finally note that the running time t_{CCA} of an arbitrary IND-THD-CCA attacker for the scheme $\mathtt{ThdBm}$ is lower-bounded by $t_{IDCCA} + \max(q_E, q_{\mathsf{H}_1})O(k^3)$. Note also that the number of queries to the random oracles H_2, H_3, H_4 and the decryption servers made by $\mathsf{B}^{\mathsf{CCA}}$ are the same as the number of those $\mathsf{A}^{\mathsf{CCA}}$ has made. Hence, we obtain the bound in the lemma statement. □

Lemma 3 *Suppose that an IND-THD-CCA attacker for the* $\mathtt{ThdBm}$ *scheme issues up to* q_D *decryption share generation queries,* q_{H_2}*,* q_{H_3}*, and* q_{H_4} *queries to the random oracles* H_2*,* H_3*, and* H_4 *respectively. Using this attacker as a subroutine, we can construct a BDH attacker for the group* $\mathcal{G}$*, whose running time is bounded by* t_{BDH}*. Concretely, we obtain the following advantage bound:*

$$
\frac{1}{2}\mathbf{Succ}^{\mathrm{IND-THD-CCA}}_{\mathtt{ThdBm}}(t, q_{\mathsf{H}_2}, q_{\mathsf{H}_3}, q_{\mathsf{H}_4} q_D) \leq \mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G}}(t_{\mathrm{BDH}}) + \frac{q_D + q_D q_{\mathsf{H}_4}}{2^k},
$$

where $t_{\mathrm{BDH}} = t_{\mathrm{CCA}} + q_{\mathsf{H}_2} + q_{\mathsf{H}_3} O(k^3) + q_{\mathsf{H}_4} q_D O(k^3)$ *for a security parameter* k*.*

Proof. For notational convenience, we assume that the same group parameters $\{\mathcal{G}, q, \hat{e}, P\}$ and security parameter k are given to attackers for $\mathtt{ThdBm}$ and the BDH problem.

Let $\mathsf{B}^{\mathsf{CCA}}$ be an attacker that defeats the THD-IND-CCA security of the $\mathtt{ThdBm}$ scheme. Let $\mathsf{A}^{\mathsf{BDH}}$ be an attacker for the BDH problem. Suppose that $(\mathcal{G}, q, \hat{e}, P, aP, bP, cP)$ is given to $\mathsf{A}^{\mathsf{BDH}}$. Also, suppose that the (same) security parameter k is given to $\mathsf{B}^{\mathsf{CCA}}$.

We start with Game $\mathbf{G_0}$ which is the same as the real attack game associated with $\mathsf{B}^{\mathsf{CCA}}$. Then, we modify this game until we completely simulate the view of $\mathsf{B}^{\mathsf{CCA}}$ and obtain a game in which $\mathsf{A}^{\mathsf{BDH}}$ is able to solve the BDH problem.

- Game $\mathbf{G_0}$: This game is actually the same as the real attack game. However, we repeat it for cleaning up notations.

  First, we run the key/common parameter generation algorithm of the ThdBm scheme on input a security parameter k, a threshold parameter t and a number of decryption servers n. We give $\mathsf{B}^{\mathsf{CCA}}$ the resulting common parameter $cp_{\texttt{ThdBm}} = (\mathcal{G}, q, \hat{e}, P, \mathsf{H}_2, \mathsf{H}_3, \mathsf{H}_4, Y, Q)$ where $Y = xP$ for random $x \in \mathbb{Z}_q^*$ and the set of verification keys $\{y_i\}$, where $1 \le i \le n$. But we keep the private key $D = xQ$ as secret.

  If $\mathsf{B}^{\mathsf{CCA}}$ submits a pair of plaintexts (M_0, M_1), we choose a bit β uniformly at random and create a target ciphertext $C^* = (U^*, V^*, W^*)$ as follows.

  $$U^* = r^*P, V^* = H_2^* \oplus M_\beta, \text{ and } W^* = r^* H_3^*,$$

  where $\kappa^* = \hat{e}(Q, Y)^{r^*}$ for random $r^* \in \mathbb{Z}_q^*$, $H_2^* = \mathsf{H}_2(\kappa^*)$ and $H_3^* = \mathsf{H}_3(U^*, V^*)$.

  Once all the decryption servers are set up, $\mathsf{B}^{\mathsf{CCA}}$ can issue decryption share generation queries at its will. We denote those queries by $C = (U, V, W)$. Note that C is different from the target ciphertext C^*.

  On input C^*, $\mathsf{B}^{\mathsf{CCA}}$ outputs $\tilde{\beta}$. We denote by E_0 the event $\tilde{\beta} = \beta$ and use a similar notation E_i for all $\mathbf{G}_i$. Since game $\mathbf{G_0}$ is the same as the real attack game, we have

  $$\Pr[E_0] = \frac{1}{2} + \frac{1}{2}\mathbf{Succ}^{\mathrm{THD-IND-CCA}}_{\texttt{ThdBm},\mathsf{B}^{\mathsf{CCA}}}(k).$$

- Game $\mathbf{G_1}$: First, we replace replace Y and Q in $cp_{\texttt{ThdBm}}$ by bP and cP respectively, all of which are given to $\mathsf{A}^{\mathsf{BDH}}$. We denote bP and cP by Y_{BDH} and Q_{BDH} respectively. Now, we assume that a subset of $t-1$ decryption servers have been corrupted without loss of generality. Let $\Phi' = \{0, 1, \ldots, t-1\}$. Then, we choose $S_1, S_2, \ldots, S_{t-1}$ uniformly at random from $\mathcal{G}$ and compute

  $$y_i = \hat{e}(Q_{\mathrm{BDH}}, Y_{\mathrm{BDH}})^{c_{i0}^{\Phi'}} \prod_{j=1}^{t-1} \hat{e}(S_j, P)^{c_{ij}^{\Phi'}},$$

  where $t \le i \le n$ and $c_{ij}^{\Phi'}$ denotes a Lagrange coefficient with respect to the set Φ'. We send S_i where $1 \le i \le t-1$ to each of the corrupted servers and send y_i where $t \le i \le n$ to each of the uncorrupted decryption servers. Then, $\mathsf{B}^{\mathsf{CCA}}$ obtains access to $\{S_i\}$ and $\{y_i\}$.

  Now, we modify the target ciphertext $C^* = (U^*, V^*, W^*)$ as follows. First, we choose κ^+ uniformly at random from $\mathcal{F}$ and replace κ^* by κ^+. We also choose H_2^+ uniformly at random from $\{0,1\}^l$, replace H_2^* by H_2^+ and V^* by $V^+ = H_2^+ \oplus M_\beta$. Accordingly, whenever the random oracle H_2 is queried at κ^+, we respond with H_2^+.

  Summing up, we obtain a new challenge ciphertext denoted by C_+^* such that $C_+^* = (U^*, V^+, W^*)$, where $V^+ = H_2^+ \oplus M_\beta$ and $H_2^+ = \mathsf{H}_2(\kappa^+)$ for random $\kappa^+ \in \mathcal{F}$.

  Note that the attacker $\mathsf{B}^{\mathsf{CCA}}$'s view has the same distribution in both Game $\mathbf{G_0}$ and Game $\mathbf{G_1}$, since we have replaced one set of random variables by another set of random variables which is different, yet has the same distribution.

  Thus, we have

  $$\Pr[E_1] = \Pr[E_0].$$

- Game $\mathbf{G_2}$: In this game, we restore the queries to the random oracle H_2. That is, if H_2 is queried at κ^+, we do not respond with H_2^+ any more but respond with an answer from the random oracle H_2 instead. We assume that this rule applies to all the forthcoming games.

  By the above rule, κ^+ and H_2^+ are used only in the target ciphertext C_+^*. Accordingly, the distribution of input to $\mathsf{B}^{\mathsf{CCA}}$ does not depend on β. Hence, we get $\Pr[E_2] = 1/2$.

  Note that Game $\mathbf{G_2}$ and Game $\mathbf{G_1}$ may differ if the random oracle H_1 is queried at κ^*. Let AskH_{2_2} denotes the event that, in game $\mathbf{G_2}$, H_2 is queried at κ^*. We will use the same notation AskH_{2_i} to denote such events in all other games.

  Now, we have

  $$|\Pr[E_2] - \Pr[E_1]| \leq \Pr[\mathsf{AskH}_{2_2}].$$

- Game $\mathbf{G_3}$: In this game, we further modify the target ciphertext $C_+^* = (U^*, V^+, W^*)$. First, we replace U^* by aP. We keep $V^+(= H_2^+ \oplus M_\beta = \mathsf{H}_2(\kappa^+) \oplus M_\beta)$ as it is, but define κ^+ as the BDH key $\hat{e}(P,P)^{abc}$. Then, we choose s^+ uniformly at random from $\mathbb{Z}_q^*$, compute s^+aP and replace W^* by s^+aP. Finally, we modify the computation of the random oracle H_3 as follows. Whenever H_3 is queried at (aP, V^+), we compute $H_3^+ = s^+P$ and respond with H_3^+. Namely, we set $H_3^+ = \mathsf{H}_2(aP, V^+)$.

  Summing up, we have obtained a new target ciphertext denoted by $C_{\mathrm{BDH}}^* = (U_{\mathrm{BDH}}, V_{\mathrm{BDH}}, W_{\mathrm{BDH}})$ such that

  $$U_{\mathrm{BDH}} = aP;\ V_{\mathrm{BDH}} = V^+;\ W_{\mathrm{BDH}} = s^+aP,$$

  where $V^+ = \mathsf{H}_2(\hat{e}(P,P)^{abc}) \oplus M_\beta$. Moreover, we have $\mathsf{H}_3(U_{\mathrm{BDH}}, V_{\mathrm{BDH}}) = H_3^+ = s^+P$.

  Note that we have replaced one set of random variables $\{U^*, W^*\}$ by another set of random variables $\{aP, s^+aP\}$ which is different, yet has the same distribution. Note also that C_{BDH}^* is a valid ciphertext since $\hat{e}(P, W_{\mathrm{BDH}}) = \hat{e}(U_{\mathrm{BDH}}, H_3^+)$ by the construction of H_3^+ and W_{BDH}. Hence, the attacker $\mathsf{B}^{\mathsf{CCA}}$'s view has the same distribution in both Game $\mathbf{G_2}$ and Game $\mathbf{G_3}$, and we have

  $$\Pr[\mathsf{AskH}_{2_3}] = \Pr[\mathsf{AskH}_{2_2}].$$

- Game $\mathbf{G_4}$: In this game, we modify the random oracle H_3. Note that we have already dealt with the simulation of the random oracles H_3 appeared in the target ciphertext C_{BDH}^*, namely, the case when H_3 is queried at $(U_{\mathrm{BDH}}, V_{\mathrm{BDH}})$. In the following, we deal with the rest of simulation.

  Whenever H_3 is queried at $(U, V) \neq (U_{\mathrm{BDH}}, V_{\mathrm{BDH}})$, we choose s uniformly at random from $\mathbb{Z}_q^*$, computes $H_3 = sY$ and respond with H_3. Let $\mathsf{H}_3\mathsf{List}$ be a list of all "input-output" pairs of the random oracle H_3. Specifically, $\mathsf{H}_3\mathsf{List}$ consists of the pairs $\langle (U, V), H_3 \rangle$ where $H_3 = \mathsf{H}_3(U, V) = sY$. Notice that this list grows as $\mathsf{B}^{\mathsf{CCA}}$'s attack proceeds.

  Because H_3 is assumed to be a random oracle, the above generation of the outputs of H_3 perfectly simulates the real oracle. Hence, $\mathsf{B}^{\mathsf{CCA}}$'s view in this game remains the same as that in the previous game. Hence, we have

  $$\Pr[\mathsf{AskH}_{2_4}] = \Pr[\mathsf{AskH}_{2_3}].$$

Note that the decryption oracle has been regarded as perfect up to this game. The rest of games will deal with simulation of the decryption oracle.

- Game $\mathbf{G_5}$: In this game, we make the decryption oracle reject all ciphertexts $C = (U, V, W)$ such that $H_3 = \mathsf{H}_3(U, V)$ has *not* been queried. If C is a valid ciphertext while $\mathsf{H}_3(U, V)$ has *not* been queried, $\mathsf{B}^{\mathsf{CCA}}$'s view in Game $\mathbf{G_5}$ and Game $\mathbf{G_4}$ may differ.

  Note that if a ciphertext C is valid then it should be the case that $\hat{e}(P, W) = \hat{e}(U, H_3)$. However, since we have assumed that H_3 has not been queried in this game, the above equality holds with probability at most $1/2^k$ since output of the simulated random oracle H_3 is uniformly distributed in $\mathcal{G}$. Adding up all the decryption queries up to q_D, we have

$$|\Pr[\mathsf{AskH_2}_5] - \Pr[\mathsf{AskH_2}_4]| \leq \frac{q_D}{2^k}.$$

- Game $\mathbf{G_6}$: In this game, we modify the decryption oracle in the previous game to yield a decryption oracle simulator which decrypts a submitted decryption query $C = (U, V, W)$ without the private key. Note that the case when $\mathsf{H}_3(U, V)$ has not been queried are excluded in this game since it was already dealt with in the previous game. Hence, we assume that $\mathsf{H}_3(U, V)$ has been queried at some point.

  Now we describe the complete specification of the decryption oracle simulator. On input a ciphertext $C = (U, V, W)$, the decryption oracle simulator works as follows.

  - Extract $\langle (U, V), H_3 \rangle$ from $\mathsf{H_3List}$.
  - If $\hat{e}(P, W) = \hat{e}(U, H_3)$
    * Compute $K = (1/s)W$. (Note here that $(1/s)W = (1/s)rsY = rY = rxP$.)
    * Compute $\kappa = \hat{e}(Q, K)$
    * For $t \leq i \leq n$, compute $\kappa_i = \kappa^{c_{i0}^{\Phi'}} \prod_{j=1}^{t-1} \hat{e}(S_j, U)^{c_{ij}^{\Phi'}}$.
    * Return κ_i.
  - Else reject C.

  Note in the above construction that

$$\begin{aligned} \kappa_i &= \kappa^{c_{i0}^{\Phi'}} \prod_{j=1}^{t-1} \hat{e}(S_j, U)^{c_{ij}^{\Phi'}} = \hat{e}(Q, K)^{c_{i0}^{\Phi'}} \prod_{j=1}^{t-1} \hat{e}(S_j, U)^{c_{ij}^{\Phi'}} \\ &= \hat{e}(Q, rxP)^{c_{i0}^{\Phi'}} \prod_{j=1}^{t-1} \hat{e}(S_j, rP)^{c_{ij}^{\Phi'}} = \hat{e}(Q, xP)^{rc_{i0}^{\Phi'}} \prod_{j=1}^{t-1} \hat{e}(S_j, P)^{rc_{ij}^{\Phi'}} \\ &= \left(\hat{e}(Q, Y)^{c_{i0}^{\Phi'}} \prod_{j=1}^{t-1} \hat{e}(S_j, P)^{c_{ij}^{\Phi'}}\right)^r = y_i^r. \end{aligned}$$

  Hence, κ_i is a correct i-th share of the BDH key $\kappa = \hat{e}(Q, Y)^r$. However, we need more efforts to simulate a decryption share δ_i containing κ_i completely. This can be done as follows.

  First, we simulate the random oracle H_4 in a classical way. That is, if H_4 is queried, we choose H_4 uniformly at random from $\mathbb{Z}_q^*$ and respond with it. As usual, we maintain an "input-output" list $\mathsf{H_4List}$ for H_4 whose entry is of the form $\langle(\kappa, \tilde{\kappa}, \tilde{y}), H_4\rangle$. Next, we choose L_i and λ_i uniformly at random from $\mathcal{G}$ and $\mathbb{Z}_q^*$ respectively, and compute $\tilde{\kappa_i} = \hat{e}(L_i, U)/\kappa_i^{\lambda_i}$ and $\tilde{y_i} = \hat{e}(L_i, P)/y_i^{\lambda_i}$. Then, we set $\lambda_i = \mathsf{H}_4(\kappa_i, \tilde{\kappa_i}, \tilde{y_i})$. Finally, we check whether there exists an entry $\langle(\kappa, \tilde{\kappa}, \tilde{y}), H_4\rangle$ in $\mathsf{H_4List}$ satisfying $H_4 = \lambda_i$ but $(\kappa, \tilde{\kappa}, \tilde{y}) \neq (\kappa_i, \tilde{\kappa_i}, \tilde{y_i})$. If such entry exists then we return "*Abort*" message to $\mathsf{B}^{\mathsf{CCA}}$. Otherwise, we return the simulated value $\delta_i = (i, \kappa_i, \tilde{\kappa_i}, \tilde{y_i}, L_i)$ to $\mathsf{B}^{\mathsf{CCA}}$ as a decryption share corresponding to C and save $\langle(\kappa_i, \tilde{\kappa_i}, \tilde{y_i},), \lambda_i\rangle$ to $\mathsf{H_4List}$.

Since H_3 are assumed to have already been queried in this game (i.e., these case were already dealt with in the previous game), the above simulated decryption share generation server perfectly simulates the real one except the error (collision) in the simulation of H_4 occurs. Note that this happens with probability $q_{\mathsf{H}_4}/2^k$, considering up to q_{H_4} queries to H_4. Adding up all the decryption queries up to q_D, we have

$$|\Pr[\mathsf{AskH}_{2_6}] - \Pr[\mathsf{AskH}_{2_5}]| \leq \frac{q_D q_{\mathsf{H}_4}}{2^k}.$$

Now, recall that the target ciphertext used so far is C^*_{BDH} that constructed in Game $\mathbf{G_3}$. Accordingly, AskH_{2_6} denotes an event that the BDH key $\hat{e}(P, P)^{abc}$ has been queried to the random oracle H_2. Note also that we have used the easiness of the DDH problem in the group $\mathcal{G}$ to simulate the decryption oracle.

Therefore, at this stage, $\mathsf{A}^{\mathsf{BDH}}$ can solve the BDH problem by outputting the queries to the random oracle H_2. That is, we have

$$\Pr[\mathsf{AskH}_{2_6}] \leq \mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G},\mathsf{A}^{\mathsf{BDH}}}(k).$$

Thus, putting all the bounds we have obtained in each game together, we have

$$\begin{aligned}\frac{1}{2}\mathbf{Succ}^{\mathrm{IND-THD-CCA}}_{\mathtt{ThdBm},\mathsf{B}^{\mathsf{CCA}}}(k) &= |\Pr[E_0] - \Pr[E_2]| \leq \Pr[\mathsf{AskH}_{2_2}] \leq \Pr[\mathsf{AskH}_{2_5}] + \frac{q_D}{2^k} \\ &\leq \frac{q_D}{2^k} + \Pr[\mathsf{AskH}_{2_6}] + \frac{q_D q_{\mathsf{H}_4}}{2^k} \leq \frac{q_D + q_D q_{\mathsf{H}_4}}{2^k} + \mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G},\mathsf{A}^{\mathsf{BDH}}}(k).\end{aligned}$$

Considering the running time t_{BDH} and queries of an arbitrary BDH-attacker for the group $\mathcal{G}$, we obtain the bound in the lemma statement. □

6 Application to Mediated ID-Based Encryption

6.1 Security Issues in Mediated ID-Based Encryption

The main motivation of mediated cryptography [3] is to revoke a user's privilege to perform cryptographic operations such as decrypting ciphertexts or signing messages *instantaneously*. In [3], Boneh et al. constructed the first mediated encryption and signature schemes using the RSA primitive. Their idea is to split a user's private key into two parts and give one piece to the on-line Security Mediator (SEM) and the other to the user. To decrypt or sign, the user must acquire a message-specific token which is associated with the SEM part of private key from the SEM. As a result, revocation is achieved by instructing the SEM not to issue tokens for the user.

Recently, the problem of realizing mediated encryption in the ID-based setting was considered by Ding and Tsudik [7]. They proposed an ID-based mediated encryption scheme based on RSA-OAEP [2]. Although their scheme offers good performance and practicality, it has a drawback which stems from the fact that a common RSA modulus is used for all the users within the system and hence, to guarantee the security of Ding and Tsudik's scheme, one should assume that the SEM's private key must be protected throughout the life of the system.

As an alternative to Ding and Tsudik's solution, Libert and Quisquater [13] proposed a new mediated ID-based encryption scheme based on Boneh and Franklin's ID-based encryption scheme. In term of security, it has an advantage over Ding and Tsudik's scheme in a sense that a compromise of the SEM's private key does not lead to a break of the whole system. In contrast to this positive result, Libert and Quisquater observed that *even though the SEM's private key is protected*, their scheme as well as Ding and Tsudik's scheme are not secure against "inside attack" in which the attacker who possesses the user part of private key conducts chosen-ciphertext attack. As a result, it should be strictly assumed in those schemes that users' private keys must be protected to ensure chosen-ciphertext security. In practice, *this assumption is fairly strong* in that there may be more chance for users to compromise their private keys than the SEM does since the SEM is usually assumed to be a trusted entity configured by a system administrator.

However, in the following section, we present a new mediated ID-based encryption scheme based on our `IdThdBm` scheme, which is secure against ciphertext attack in a *strong* sense, that is, secure against chosen-ciphertext attack conducted by the attacker that obtains the user part of private key.

6.2 Description of Our Scheme – `mIdeBm`

We describe our mediated ID-based encryption scheme "`mIdeBm`" based on the `IdThdBm` scheme with $(t, n) = (2, 2)$ as follows.

- **Setup**: Given a security parameter k, the PKG runs the key generation algorithm of `IdThdBm`. The output of this algorithm $cp = (\mathcal{G}, q, P, \hat{e}, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3, \mathsf{H}_4, Y_{\text{PKG}})$ is as defined in the description of `IdThdBm`. Note that cp is given to all interested parties while the master key x is kept secret within the PKG.

- **Keygen**: Given a user's identity $\mathtt{ID}$, the PKG computes $Q_{\mathtt{ID}} = \mathsf{H}_1(\mathtt{ID})$ and $D_{\mathtt{ID}} = xQ_{\mathtt{ID}}$. It then splits $D_{\mathtt{ID}}$ using the $(2, 2)$-secret-sharing technique as follows[1].
  - Pick R at random from $\mathcal{G}^*$ and construct $F(u) = D_{\mathtt{ID}} + uR$ for $u \in \{0\} \cup \mathbb{N}$.
  - Compute $D_{\mathtt{ID},sem} = F(1)$ and $D_{\mathtt{ID},user} = F(2)$.

  The PKG gives $D_{\mathtt{ID},sem}$ to the SEM and $D_{\mathtt{ID},user}$ to the user.

- **Encrypt**: Given a plaintext $M \in \{0, 1\}^l$ and a user's identity $\mathtt{ID}$, a sender creates a ciphertext $C = (U, V, W)$ such that

  $$U = rP; V = \mathsf{H}_2(\kappa) \oplus M; W = r\mathsf{H}_3(U, V),$$

  where $\kappa = \hat{e}(\mathsf{H}_1(\mathtt{ID}), Y_{\text{PKG}})^r$ for random $r \in \mathbb{Z}_q^*$.

[1] In this particular case of $(2, 2)$-secret-sharing, one may share $D_{\mathtt{ID}}$ by taking a random $D_{\mathtt{ID},sem}$ and computing $D_{\mathtt{ID},user} = D_{\mathtt{ID}} - D_{\mathtt{ID},sem}$ for efficiency.

- **Decrypt**: When receiving $C = (U, V, W)$, a user forwards it to the SEM. The SEM and the user perform the following in parallel.
  - SEM (We call this procedure "SEM oracle"):
    1. Check if the user's identity ID is revoked. If it is, return "ID *Revoked*".
    2. Otherwise, do the following:
       * Compute $H_3 = \mathsf{H}_3(U, V)$ and check if $\hat{e}(P, W) = \hat{e}(U, H_3)$. If C has passed this test, compute $\kappa_{sem} = \hat{e}(D_{\mathtt{ID},sem}, U)$ and send $\delta_{\mathtt{ID},sem,C} = (sem, \kappa_{sem})$ to the user. Otherwise, send $\delta_{\mathtt{ID},sem,C} = (sem,$ "*Invalid Ciphertext*"$)$ to the user.
  - User (We call this procedure "User oracle"):
    1. Compute $H_3 = \mathsf{H}_3(U, V)$ and check if $\hat{e}(P, W) = \hat{e}(U, H_3)$. If C has passed this test, compute $\kappa_{user} = \hat{e}(D_{\mathtt{ID},user}, U)$. Otherwise, return "*Reject*" and terminate.
    2. Get $\delta_{\mathtt{ID},sem,C}$ from the SEM and do the following:
       * If $\delta_{\mathtt{ID},sem,C}$ is of the form $(sem,$ "*Invalid Ciphertext*"$)$, return "*Reject*" and terminate. Otherwise, compute $\kappa = \kappa_{sem}^{c_{01}^{\Phi}} \kappa_{user}^{c_{02}^{\Phi}}$ where c_{01}^{Φ} and c_{02}^{Φ} denote the Lagrange coefficients for the set $\Phi = \{1, 2\}$ and $M = \mathsf{H}_2(\kappa) \oplus V$, and return M.

Notice that in the SEM oracle of the above scheme, the validity of a ciphertext is checked before generating a token in the same way as the decryption share generation algorithm of `IdThdBm` does.

6.3 Security Analysis – `mIdeBm`

In this section, we show that the chosen-ciphertext security of the above scheme against the strong attacker that obtains the user part of private key is relative to the IND-IDTHD-CCA (Definition 2) security of the $(2, 2)$-`IdThdBm` scheme.

To begin with, we define IND-mID-sCCA (indistinguishability of mediated ID-based encryption against strong chosen-ciphertext attack), which is similar to IND-mID-wCCA ("w" stands for "weak") defined in [13] but assumes the stronger attacker that can corrupt users to get their private keys.

Definition 5 (IND-mID-sCCA) Let $\mathsf{A}^{\mathsf{CCA}'}$ be an attacker that defeats the IND-mID-sCCA security of an mediated ID-based encryption scheme $\mathcal{MIDE}$ which consists of Setup, Keygen, Encrypt and Decrypt algorithms. (For details of these algorithms, readers are referred to `mIdeBm` given in Section 6.2.) We assume that $\mathsf{A}^{\mathsf{CCA}'}$ is a probabilistic Turing machine taking a security parameter k as input. Consider the following game in which the attacker $\mathsf{A}^{\mathsf{CCA}'}$ interacts with the "Challenger".

Phase 1: The Challenger runs the Setup algorithm taking a security parameter k. The Challenger then gives the common parameter to $\mathsf{A}^{\mathsf{CCA}'}$.

Phase 2: Having obtained the common parameter, $\mathsf{A}^{\mathsf{CCA}'}$ issues the following queries.

- "User key extraction" query ID: On receiving this query, the Challenger runs the Keygen algorithm to obtain the user part of private key and sends it to $\mathsf{A}^{\mathsf{CCA}'}$.

– "SEM key extraction" query $\mathtt{ID}$: On receiving this query, the Challenger runs the Keygen algorithm to obtain the SEM part of private key and sends it to $\mathsf{A}^{\mathsf{CCA}'}$.

– "SEM oracle" query $(\mathtt{ID}, C)$: On receiving this query, the Challenger runs the Keygen algorithm to obtain a SEM part of private key. Taking the resulting private key as input, the Challenger runs the SEM oracle in the Decrypt algorithm to obtain a decryption token for C and sends it to $\mathsf{A}^{\mathsf{CCA}'}$.

– "User oracle" query $(\mathtt{ID}, C)$: On receiving this query, the Challenger runs the Keygen algorithm to obtain a User part of private key. Taking the resulting private key as input, the Challenger runs the User oracle in the Decrypt algorithm to obtain a decryption token for C and sends it to $\mathsf{A}^{\mathsf{CCA}'}$.

Phase 3: $\mathsf{A}^{\mathsf{CCA}'}$ selects two equal-length plaintexts (M_0, M_1) and a target identity $\mathtt{ID}^*$ which was not queried before. On receiving (M_0, M_1) and $\mathtt{ID}^*$, the Challenger runs the Keygen algorithm to obtain User and SEM parts of the private key associated with $\mathtt{ID}^*$. The Challenger then chooses $\beta \in \{0, 1\}$ at random and creates a target ciphertext C^* by encrypting M_β under the target identity $\mathtt{ID}^*$. The Challenger gives the target ciphertext and *the User part of the private key* to $\mathsf{A}^{\mathsf{CCA}'}$.

Phase 4: $\mathsf{A}^{\mathsf{CCA}'}$ continues to issue "User key extraction" query $\mathtt{ID} \neq \mathtt{ID}^*$, "SEM key extraction" query $\mathtt{ID} \neq \mathtt{ID}^*$, "SEM oracle" query $(\mathtt{ID}, C) \neq (\mathtt{ID}^*, C^*)$, and "User oracle" query $(\mathtt{ID}, C) \neq (\mathtt{ID}^*, C^*)$. The details of these queries are as described in Phase 2.

Phase 5: $\mathsf{A}^{\mathsf{CCA}'}$ outputs a guess $\tilde{\beta} \in \{0, 1\}$.

We define the attacker $\mathsf{A}^{\mathsf{CCA}'}$'s success by

$$\mathbf{Succ}^{\mathrm{IND-mID-sCCA}}_{\mathcal{MIDE},\mathsf{A}^{\mathsf{CCA}'}}(k) = 2 \cdot \Pr[\tilde{\beta} = \beta] - 1.$$

We denote by $\mathbf{Succ}^{\mathrm{IND-mID-sCCA}}_{\mathcal{MIDE}}(t_{CCA}, q_{E,user}, q_{E,sem}, q_{D,sem}, q_{D,user})$ the maximum of the attacker $\mathsf{A}^{\mathsf{CCA}'}$'s success over all attackers $\mathsf{A}^{\mathsf{CCA}'}$ having running time t_{CCA} and making at most $q_{E,user}$ "User key extraction" queries, $q_{E,sem}$ "SEM key extraction" queries, $q_{D,sem}$ "SEM oracle" queries, and $q_{D,user}$ "User oracle" queries. Note that the running time and the number of queries are all polynomial in the security parameter k.

The mediated ID-based encryption scheme $\mathcal{MIDE}$ is said to be IND-mID-sCCA secure if $\mathbf{Succ}^{\mathrm{IND-mID-sCCA}}_{\mathcal{MIDE}}(t_{CCA}, q_{E,user}, q_{E,sem}, q_{D,sem}, q_{D,user})$ is negligible in k.

We now state and prove the following theorem.

Theorem 2 *Suppose that an IND-mID-sCCA attacker for the* `mIdeBm` *scheme issues up to $q_{E,user}$ "User key extraction" queries, $q_{E,sem}$ "SEM key extraction" queries, $q_{D,sem}$ "SEM oracle", and $q_{D,user}$ "User oracle" queries. Using this attacker as a subroutine, we can construct an IND-IDTHD-CCA attacker for the* `IdThdBm` *scheme with $(t, n) = (2, 2)$, whose running time and the number of private key extraction and decryption share generation queries are bounded by t_{IDCCA}, q_E, and q_D respectively. Concretely, we obtain the following advantage bound:*

$$\mathbf{Succ}^{\mathrm{IND-mID-sCCA}}_{\mathtt{mIdeBm}}(t_{CCA,E,user}, q_{E,sem}, q_{D,sem}, q_{D,user}) \\ \leq \mathbf{Succ}^{\mathrm{IND-IDTHD-CCA}}_{\mathtt{IdThdBm}}(t_{IDCCA}, q_E, q_D),$$

where $t_{IDCCA} = t_{CCA} + \max(q_{E,user}, q_{E,sem}, q_{D,sem}, q_{D,user})O(k^3)$, $q_E = O(1)(q_{E,user} + q_{E,sem} + q_{D,sem} + q_{D,user})$, $q_D = O(1)(q_{D,sem} + q_{D,user})$ *for a security parameter* k. *Here,* t_{CCA} *denotes the running time of the ID-mID-sCCA attacker.*

Proof. For notational convenience, we assume that the same group parameter $cp = \{\mathcal{G}, q, \hat{e}, P, Y_{\text{PKG}}\}$ where $Y_{\text{PKG}} = xP$ and security parameter k are given to attackers for `mIdeBm` and `IdThdBm`.

Let $\mathsf{A}^{\mathsf{CCA}'}$ denote an attacker that defeats the IND-mID-sCCA security of the `mIdeBm` scheme. Let $\mathsf{A}^{\mathsf{CCA}}$ denote an attacker that defeats the IND-IDTHD-CCA security of the `IdThdBm` scheme with $(t, n) = (2, 2)$.

Our aim is to simulate the view of $\mathsf{A}^{\mathsf{CCA}'}$ in the real attack game denoted by $\mathbf{G_0}$ until we obtain a game denoted by $\mathbf{G_1}$, which is related to the ability of the attacker $\mathsf{A}^{\mathsf{CCA}}$ to defeat the IND-IDTHD-CCA security of the `IdThdBm` scheme.

- Game $\mathbf{G_0}$: As mentioned, this game is identical to the real attack game described in Definition 5. We denote by E_0 the event that $\mathsf{A}^{\mathsf{CCA}'}$'s output $\bar{\beta} \in \{0, 1\}$ is equal to $\beta \in \{0, 1\}$ chosen by the Challenger. We use a similar notation E_i for all Games $\mathbf{G}_i$. Since Game $\mathbf{G_1}$ is the same as the real attack game, we have

$$\Pr[E_0] = \frac{1}{2} + \frac{1}{2}\mathbf{Succ}^{\mathrm{IND-mID-sCCA}}_{\mathtt{mIdeBm},\mathsf{A}^{\mathsf{CCA}}}(k).$$

- Game $\mathbf{G_1}$: First, we give $\mathsf{A}^{\mathsf{CCA}}$'s common parameter to $\mathsf{A}^{\mathsf{CCA}'}$. We then deal with the simulation of $\mathsf{A}^{\mathsf{CCA}'}$'s view in Phase 2 of the real attack game as follows.

  On receiving $\mathsf{A}^{\mathsf{CCA}'}$'s "User key extraction" queries, each of which consists of `ID`, we perform the following:

  - Search UserKeyList which consists of ⟨identity, corresponding user part of private key⟩ pairs for an entry that is matched against `ID`.
    - If there exists such an entry, extract a corresponding user part of private key and return it to $\mathsf{A}^{\mathsf{CCA}'}$ as an answer.
    - Otherwise, do the following:
      - · Forward `ID` as a "private key extraction" query to $\mathsf{A}^{\mathsf{CCA}}$'s Challenger to obtain a private key $D_{\mathtt{ID}}$ associated with `ID`. (On receiving `ID`, the Challenger runs the private key extraction algorithm of `IdThdBm` taking `ID` as input and returns a private key $D_{\mathtt{ID}}$ associated with `ID` to $\mathsf{A}^{\mathsf{CCA}}$.)
      - · Get $D_{\mathtt{ID}}$ from the communication between the Challenger and $\mathsf{A}^{\mathsf{CCA}}$ and split it into $D_{\mathtt{ID},sem}$ and $D_{\mathtt{ID},user}$ using the (2, 2) secret-sharing technique presented in Section 5.1.
      - · Return $D_{\mathtt{ID},user}$ to $\mathsf{A}^{\mathsf{CCA}'}$ as an answer.
      - · Add $\langle \mathtt{ID}, D_{\mathtt{ID},user}\rangle$ to UserKeyList. Also, add $\langle \mathtt{ID}, D_{\mathtt{ID},sem}\rangle$ to SEMKeyList which consists of ⟨identity, corresponding SEM part of private key⟩ pairs.

  We answer $\mathsf{A}^{\mathsf{CCA}'}$'s "SEM key extraction" queries in a similar way as we do for the "User key extraction" queries. Note that in this case, SEMKeyList and UserKeyList are also updated *concurrently*.

  On receiving $\mathsf{A}^{\mathsf{CCA}'}$'s "SEM oracle" queries, each of which consists of $(\mathtt{ID}, C)$ where $C = (U, V, W)$, we perform the following:

– Search $\mathsf{SEMKeyList}$ for an entry $\langle \mathtt{ID}, D_{\mathtt{ID},sem} \rangle$.

  * If there exists such an entry, extract $D_{\mathtt{ID},sem}$ from it. Then, check if $\hat{e}(P, W) = \hat{e}(U, H_3)$ for $H_3 = \mathsf{H}_3(U, V)$.
    · If C has passed this test, compute $\kappa_{sem} = \hat{e}(D_{\mathtt{ID},sem}, U)$ and return $\delta_{\mathtt{ID},sem,C} = (sem, \kappa_{sem})$ to $\mathsf{A}^{\mathsf{CCA}'}$.
    · Otherwise, return $\delta_{\mathtt{ID},sem,C} = (sem,$ "*Invalid Ciphertext*") to $\mathsf{A}^{\mathsf{CCA}'}$.
  * If $\langle \mathtt{ID}, D_{\mathtt{ID},sem} \rangle$ does not exist in $\mathsf{SEMKeyList}$, do the following:
    · Forward $\mathtt{ID}$ as a "private key extraction" query to $\mathsf{A}^{\mathsf{CCA}}$'s Challenger to obtain a private key $D_{\mathtt{ID}}$ associated with $\mathtt{ID}$.
    · Get $D_{\mathtt{ID}}$ from the communication between the Challenger and $\mathsf{A}^{\mathsf{CCA}}$ and split it into $D_{\mathtt{ID},sem}$ and $D_{\mathtt{ID},user}$ using the (2, 2) secret-sharing technique.
    · Compute $\kappa_{sem} = \hat{e}(D_{\mathtt{ID},sem}, U)$ and return $\delta_{\mathtt{ID},sem,C} = (sem, \kappa_{sem})$ to $\mathsf{A}^{\mathsf{CCA}'}$.
    · Add $\langle \mathtt{ID}, D_{\mathtt{ID},sem} \rangle$ and $\langle \mathtt{ID}, D_{\mathtt{ID},user} \rangle$ to $\mathsf{SEMKeyList}$ and $\mathsf{UserKeyList}$ respectively.

On receiving $\mathsf{A}^{\mathsf{CCA}'}$'s "User oracle" queries, each of which consists of $(\mathtt{ID}, C)$, we perform the following:

– Search $\mathsf{SEMKeyList}$ for an entry $\langle \mathtt{ID}, D_{\mathtt{ID},sem} \rangle$.

  * If there exists such an entry, search $\mathsf{UserKeyList}$ for the corresponding entry $\langle \mathtt{ID}, D_{\mathtt{ID},user} \rangle$. (Recall that $\mathsf{SEMKeyList}$ and $\mathsf{UserKeyList}$ are updated concurrently. Hence if there exists an entry in $\mathsf{SEMKeyList}$, we can always find the corresponding entry in $\mathsf{UserKeyList}$.) Then, check if $\hat{e}(P, W) = \hat{e}(U, H_3)$ for $H_3 = \mathsf{H}_3(U, V)$.
    · If C has passed this test, compute $\kappa_{sem} = \hat{e}(D_{\mathtt{ID},sem}, U)$ and $\kappa_{user} = \hat{e}(D_{\mathtt{ID},user}, U)$, and combine them using the Lagrange interpolation technique and return the resulting value to $\mathsf{A}^{\mathsf{CCA}'}$.
    · Otherwise, return "*Reject*" to $\mathsf{A}^{\mathsf{CCA}'}$.
  * If $\langle \mathtt{ID}, D_{\mathtt{ID},sem} \rangle$ does not exist in $\mathsf{SEMKeyList}$, do the following:
    · Forward $\mathtt{ID}$ as a "private key extraction" query to $\mathsf{A}^{\mathsf{CCA}}$'s Challenger to obtain a private key $D_{\mathtt{ID}}$ associated with $\mathtt{ID}$.
    · Get $D_{\mathtt{ID}}$ from the communication between the Challenger and $\mathsf{A}^{\mathsf{CCA}}$ and split it into $D_{\mathtt{ID},sem}$ and $D_{\mathtt{ID},user}$ using the (2, 2) secret-sharing technique.
    · Perform the same routine as we do for the case when $\langle \mathtt{ID}, D_{\mathtt{ID},sem} \rangle$ exists in $\mathsf{SEMKeyList}$ and return a special symbol "*Reject*" or a certain value to $\mathsf{A}^{\mathsf{CCA}'}$
    · Add $\langle \mathtt{ID}, D_{\mathtt{ID},sem} \rangle$ and $\langle \mathtt{ID}, D_{\mathtt{ID},user} \rangle$ to $\mathsf{SEMKeyList}$ and $\mathsf{UserKeyList}$ respectively.

In Phase 3, if $\mathsf{A}^{\mathsf{CCA}'}$ issues two equal-length plaintexts (M_0, M_1) and a target identity $\mathtt{ID}^*$, we forward $(M_0, M_1, \mathtt{ID}^*)$ to $\mathsf{A}^{\mathsf{CCA}}$'s Challenger. On receiving $(M_0, M_1, \mathtt{ID}^*)$, the Challenger runs the private key extraction algorithm of `IdThdBm` to get a private key $D_{\mathtt{ID}^*}$ associated with $\mathtt{ID}^*$ and runs the private key distribution algorithm of `IdThdBm` to split $D_{\mathtt{ID}^*}$ into S_1^* and S_2^*. The Challenger returns S_2^* to $\mathsf{A}^{\mathsf{CCA}}$ as a corrupted party's private key. We then rename S_1^* and S_2^* as $D_{\mathtt{ID}^*,sem}$ and $D_{\mathtt{ID}^*,user}$ respectively and send $D_{\mathtt{ID}^*,user}$ to $\mathsf{A}^{\mathsf{CCA}'}$. (That is, the *strong* attacker $\mathsf{A}^{\mathsf{CCA}'}$ possesses the user part of private key.) Now, the Challenger chooses $\beta \in \{0,1\}$ at random and runs the encryption algorithm E of `IdThdBm` taking $(M_\beta, \mathtt{ID}^*)$ as input and gets a target ciphertext C^*. If the Challenger returns C^*, we send that to $\mathsf{A}^{\mathsf{CCA}'}$.

On receiving $\mathsf{A}^{\mathsf{CCA}'}$'s "User key extraction" and "SEM key extraction" queries in Phase 4, we answer them in the same way we did in Phase 2.

If $\mathsf{A}^{\mathsf{CCA}'}$ issues "SEM oracle" queries, each of which consists of $(\mathtt{ID}, C) \neq (\mathtt{ID}^*, C^*)$, in Phase 4, we perform the following:

- If $\mathtt{ID} \neq \mathtt{ID}^*$, answer the query in the same way we did for the SEM oracle query in Phase 2.
- If $\mathtt{ID} = \mathtt{ID}^*$ (in this case, $C \neq C^*$), do the following:
  - Forward C to $\mathsf{A}^{\mathsf{CCA}}$'s Challenger as a "decryption share generation" query. (On receiving C, the Challenger runs the decryption share generation algorithm of `IdThdBm` taking $(S_1^*(= D_{\mathtt{ID}^*,sem}), C)$ as input, gets a corresponding decryption share $\delta_{1,C}$ and returns it to $\mathsf{A}^{\mathsf{CCA}}$.)
  - Get $\delta_{1,C}$ from the communication between the Challenger and $\mathsf{A}^{\mathsf{CCA}}$. Then, do the following:

    If $\delta_{1,C} \neq (1,$ "*Invalid Ciphertext*"),
    - · Take κ_1 out from $\delta_{1,C}$
    - · Rename κ_1 as κ_{sem} and send $\delta_{\mathtt{ID}^*,sem,C} = (sem, \kappa_{sem})$ to $\mathsf{A}^{\mathsf{CCA}'}$.

    If $\delta_{1,C} = (1,$ "*Invalid Ciphertext*"),
    - · Send $\delta_{\mathtt{ID}^*,sem,C} = (sem,$ "*Invalid Ciphertext*") to $\mathsf{A}^{\mathsf{CCA}'}$.

If $\mathsf{A}^{\mathsf{CCA}'}$ issues "User oracle" queries, each of which consists of $(\mathtt{ID}, C) \neq (\mathtt{ID}^*, C^*)$, in Phase 4, we perform the following:

- If $\mathtt{ID} \neq \mathtt{ID}^*$, answer the query in the exactly same way we did for the "User oracle" query in Phase 2.
- If $\mathtt{ID} = \mathtt{ID}^*$ (in this case, $C \neq C^*$), do the following:
  - Forward C to $\mathsf{A}^{\mathsf{CCA}}$'s Challenger as a "decryption share generation" query. (On receiving C, the Challenger runs the decryption share generation algorithm of `IdThdBm` taking $(S_1^*(= D_{\mathtt{ID}^*,sem}), C)$ as input, gets a corresponding decryption share $\delta_{1,C}$ and returns it to $\mathsf{A}^{\mathsf{CCA}}$.)
  - Get $\delta_{1,C}$ from the communication between the Challenger and $\mathsf{A}^{\mathsf{CCA}}$. Then, do the following:

    If $\delta_{1,C} \neq (1,$ "*Invalid Ciphertext*"),
    - · Take out κ_1 from $\delta_{1,C}$, rename it as κ_{sem}, and form $\delta_{\mathtt{ID}^*,sem,C} = (sem, \kappa_{sem})$.
    - · Compute $\kappa_{user} = \hat{e}(D_{\mathtt{ID}^*,user}, U)$. (Recall that the $\mathsf{A}^{\mathsf{CCA}}$'s Challenger returned $D_{\mathtt{ID}^*,user}$ as a corrupted party's private key.)
    - · Check the validity of C. If C is invalid, that is, $\hat{e}(P, W) \neq \hat{e}(U, H_3)$ for $H_3 = \mathsf{H}_3(U, V)$, return "*Reject*". Otherwise, combine the shares κ_{user} and κ_{sem} using the Lagrange interpolation technique and return the resulting value to $\mathsf{A}^{\mathsf{CCA}'}$.

If $\delta_{1,C} = (1, \text{"}\textit{Invalid Ciphertext}\text{"})$,
· Send "*Reject*" to $\mathsf{A}^{\mathsf{CCA}'}$.

Finally, once $\mathsf{A}^{\mathsf{CCA}'}$ outputs a guess $\beta' \in \{0,1\}$, we return it as $\mathsf{A}^{\mathsf{CCA}}$'s guess.

Note from the simulation that $\mathsf{A}^{\mathsf{CCA}'}$'s view in the real attack game is identical to it's view in Game $\mathbf{G_1}$. Note also that the bit β is uniformly chosen. Hence we have

$$\Pr[E_1] - \frac{1}{2} \geq \Pr[E_0] - \frac{1}{2}.$$

By definition of $\Pr[E_0]$ and $\Pr[E_1]$, we obtain

$$\mathbf{Succ}^{\mathrm{IND-IDTHD-CCA}}_{\mathtt{IdThdBm},\mathsf{A}^{\mathsf{CCA}}}(k) \geq \mathbf{Succ}^{\mathrm{IND-mID-sCCA}}_{\mathtt{mIdeBm},\mathsf{A}^{\mathsf{CCA}'}}(k).$$

Considering the running time and the number of queries, we obtain the bound in the theorem statement. □

7 Concluding Remarks

In this paper, we discussed the issues related to the realization of ID-based threshold decryption and proposed the first threshold ID-based decryption scheme provably secure against chosen-ciphertext attack. We also showed how our ID-based threshold decryption scheme can result in a mediated ID-based encryption scheme secure against "inside attack", whereby an attacker who possesses a user part of private key conducts chosen-ciphertext attack.

Interesting future research would be finding more security applications where "ID-based threshold decryption" is particularly useful.

Acknowledgement

The authors are grateful to anonymous referees for their helpful comments. The first author also thanks Ron Steinfeld and John Malone-Lee for their valuable comments on the earlier version of this paper.

References

[1] M. Bellare and P. Rogaway, *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*, Proceedings of the First ACM Conference on Computer and Communications Security 1993, pages 62–73.

[2] M. Bellare and P. Rogaway, *Optimal Asymmetric Encryption*, Advances in Cryptology -Proceedings of Eurocrypt '94, LNCS 950, Springer-Verlag 1994, pages 92–111.

[3] D. Boneh, X. Ding, G. Tsudik and C. Wong, *A Method for Fast Revocation of Public Key Certificates and Security Capabilities*, Proceedings of the 10th USENIX Security Symposium, USENIX, 2001.

[4] D. Boneh and M. Franklin, *Identity-Based Encryption from the Weil Pairing*, Proceedings of CRYPTO 2001, LNCS 2139, Springer-Verlag 2001, pages 213–229.

[5] D. Boneh, B. Lynn and H. Shacham, "Short Signatures from the Weil Pairing", dvances in Cryptology - Proceedings of ASIACRYPT 2001, LNCS 2248 LNCS, pages 566–582, Springer-Verlag, 2001.

[6] D. Chaum and T. Perderson, *Wallet Databases with Observers*, Proceedings of CRYPTO '92, LNCS 740, Springer-Verlag 1992, pages 89–105.

[7] X. Ding and G. Tsudik, *Simple Identity-Based Cryptography with Mediated RSA*, Proceedings CT-RSA 2003, LNCS 2612, Springer-Verlag 2003, pages 192–209.

[8] Y. Dodis and M Yung, *Exposure-Resilience for Free: The Hierarchical ID-based Encryption Case*, Proceedings of IEEE Security in Storage Workshop 2002, pages 45–52.

[9] T. ElGamal, *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE Trans. Info. Theory, 31, 1985, pages 469–472.

[10] P. Fouque and D. Pointcheval, *Threshold Cryptosystems Secure Chosen-Ciphertext Attacks*, Proceedings of ASIACRYPT 2001, LNCS 2248, Springer-Verlag 2001, pages 351–368.

[11] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, *Secure Distributed Key Generation for Discrete-Log Based Cryptosystem*, Proceedings of EUROCRYPT '99, LNCS 1592, Springer-Verlag 1999, pages 295–310.

[12] C. Gentry and A. Silverberg, *Hierarchical ID-Based Cryptography*, Proceedings of ASIACRYPT 2002, LNCS 2501, Springer-Verlag 2002, pages 548–566.

[13] B. Libert and J. Quisquater, *Efficient Revocation and Threshold Pairing Based Cryptosystems*, Principles of Distributed Computing (PODC) 2003.

[14] C. Lim and P. Lee, *Another Method for Attaining Security Against Adaptively Chosen Ciphertext Attack*, Proceedings of CRYPTO '93, LNCS 773, Springer-Verlag 1993, pages 410–434.

[15] A. J. Menezes, T. Okamoto, and S. A. Vanstone: *Reducing Elliptic Curve Logarithms to a Finite Field*, IEEE Tran. on Info. Theory, Vol. 31, pages 1639–1646, IEEE, 1993.

[16] A. Shamir, *How to Share a Secret*, Communications of the ACM, Vol. 22, 1979, pages 612–613.

[17] A. Shamir, *Identity-based Cryptosystems and Signature Schemes*, Proceedings of CRYPTO '84, LNCS 196, Springer-Verlag 1984, pages 47–53.

[18] V. Shoup and R. Gennaro, *Securing Threshold Cryptosystems against Chosen Ciphertext Attack*, Journal of Cryptology, Vol. 15, Springer-Verlag 2002, pages 75–96.