Identity-Based Threshold Decryption

Joonsang Baek[1] and Yuliang Zheng[2]

[1] School of Network Computing, Monash University, McMahons Road, Frankston, VIC 3199, Australia
joonsang.baek@infotech.monash.edu.au
[2] Dept. Software and Info. Systems, UNC 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 of 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 [5].

Bilinear Map. The admissible bilinear map $\hat{e}$ [5] 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 $\hat{e} : \mathcal{G} \times \mathcal{G} \to \mathcal{F}$, 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 "BasicIdent" Scheme. We now describe Boneh and Franklin's basic version of ID-based encryption scheme called "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 $\hat{e} : \mathcal{G} \times \mathcal{G} \to \mathcal{F}$. It also specifies two hash functions $\mathsf{H}_1 : \{0,1\}^* \to \mathcal{G}^*$ and $\mathsf{H}_2 : \mathcal{F} \to \{0,1\}^l$, 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 [5] 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.

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 1 (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 $\mathsf{A}^{\mathsf{CCA}}$'s success as a function $\mathbf{Succ}^{\mathrm{IND-IDTHD-CCA}}_{\mathcal{IDTHD},\mathsf{A}^{\mathsf{CCA}}}(k) = 2 \cdot \Pr[\tilde{\beta} = \beta] - 1$. The ID-based threshold decryption scheme $\mathcal{IDTHD}$ is said to be IND-IDTHD-CCA secure if, for any attacker $\mathsf{A}^{\mathsf{CCA}}$ whose running time is polynomially bounded, $\mathbf{Succ}^{\mathrm{IND-IDTHD-CCA}}_{\mathcal{IDTHD},\mathsf{A}^{\mathsf{CCA}}}(k)$ 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 as follows.

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 $F : \mathbb{N} \cup \{0\} \rightarrow \mathcal{G}$ 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 "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 (The proof is given in the full version of this paper [1].) Note that ZKBm can easily be converted to a NIZK proof, making the random challenge an output of a random oracle [2]. 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 $(\kappa, \tilde{\kappa}) \in L_{\mathsf{EDLog}_{P,\tilde{P}}^{\mathcal{F}}}$ 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 $\hat{e} : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{F}$ and the hash functions H_1, H_2, H_3 and H_4 such that $\mathsf{H}_1 : \{0,1\}^* \rightarrow \mathcal{G}^*$; $\mathsf{H}_2 : \mathcal{F} \rightarrow \{0,1\}^l$; $\mathsf{H}_3 : \mathcal{G}^* \times \{0,1\}^l \rightarrow \mathcal{G}^*$; $\mathsf{H}_4 : \mathcal{F} \times \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{Z}_q^*$, 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 $Y_{\mathrm{PKG}} = xP$. Finally, it returns a common parameter $cp = (\mathcal{G},\ q,\ P,\ \hat{e},\ \mathsf{H}_1,\ \mathsf{H}_2,\ \mathsf{H}_3,\ \mathsf{H}_4,\ Y_{\mathrm{PKG}})$ 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*").
- $\mathsf{SV}(cp, \{y_i\}_{1 \leq i \leq n}, C, \delta_{i,C})$: 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 $|\Phi| \geq t$, 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 [5].

Definition 2 (BDH). Let $\mathcal{G}$ and $\mathcal{F}$ be two groups of order q where q is prime, as defined in Section 2. Let $P \in \mathcal{G}^*$ be a generator of $\mathcal{G}$. Suppose that there exists a Bilinear map $\hat{e} : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{F}$. Let $\mathsf{A}^{\mathsf{BDH}}$ be an attacker modelled as a probabilistic Turing machine.

The BDH problem refers to the computational problem in which $\mathsf{A}^{\mathsf{BDH}}$ is to compute the BDH key $\hat{e}(P, P)^{abc}$ given $(\mathcal{G}, q, P, aP, bP, cP)$ and a security parameter k. We define $\mathsf{A}^{\mathsf{BDH}}$'s success as a function $\mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G},\mathsf{A}^{\mathsf{BDH}}}(k)$=Pr[$\mathsf{A}^{\mathsf{BDH}}$ outputs $\hat{e}(P, P)^{abc}$]. The BDH problem is said to be computationally intractable if, for any attacker $\mathsf{A}^{\mathsf{BDH}}$ whose running time is polynomially bounded, $\mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G},\mathsf{A}^{\mathsf{BDH}}}(k)$ is negligible in k.

Proof of Security. Regarding the security of the `IdThdBm` scheme, we obtain the following theorem. (For a more detailed proof, we refer readers to the full version of this paper [1].)

Theorem 1. *In the random oracle model, the* `IdThdBm` *scheme is IND-IDTHD-CCA secure if the BDH problem is computationally intractable.*

Proof. (Sketch) To prove the above theorem, we derive a non-ID-based threshold decryption scheme, which we call "`ThdBm`", from the `IdThdBm` scheme. Actually, `ThdBm` is the same as `IdThdBm` except that it does not have a private key extraction algorithm and hence the hash function $\mathsf{H}_1 : \{0, 1\}^* \rightarrow \mathcal{G}^*$ is not used. The private key D of this scheme is generated by choosing Q and x uniformly at random from $\mathcal{G}^*$ and $\mathbb{Z}_q^*$ respectively, and computing $D = xQ$. The public key of this scheme consists of (Q, Y), where $Y = xP$. Note that the private key D is shared among n decryption servers. The encryption of a plaintext message $m \in \{0, 1\}^l$ can be done by choosing r uniformly at random from $\mathbb{Z}_q^*$ and computing $U = rP$, $V = \mathsf{H}_2(\kappa) \oplus m$, and $W = r\mathsf{H}_3(U, V)$, where $d = \hat{e}(Q, Y)$ and $\kappa = d^r$.

As a first step, we show how to use the IND-IDTHD-CCA attacker $\mathsf{A}^{\mathsf{CCA}}$ for `IdThdBm` to construct an IND-THD-CCA attacker $\mathsf{B}^{\mathsf{CCA}}$ for `ThdBm`. (IND-THD-CCA denotes the chosen-ciphertext security notion for non-ID-based threshold decryption defined in [18].) First, $\mathsf{B}^{\mathsf{CCA}}$ gives Y as the PKG's public key $\mathsf{A}^{\mathsf{CCA}}$. $\mathsf{B}^{\mathsf{CCA}}$ then randomly chooses an index μ from the range $[1, q_{\mathsf{H}_1}]$ where q_{H_1} denotes the maximum number of queries made by $\mathsf{A}^{\mathsf{CCA}}$ to the random oracle H_1. By

$\mathtt{ID}_\mu$, we denote the μ-th query to the random oracle H_1. $\mathsf{B}^{\mathsf{CCA}}$ hopes $\mathtt{ID}_\mu$ to be a target identity $\mathtt{ID}^*$ that $\mathsf{A}^{\mathsf{CCA}}$ outputs at some stage. Now, if $\mathsf{A}^{\mathsf{CCA}}$ queries H_1 at $\mathtt{ID} \neq \mathtt{ID}_\mu$, $\mathsf{B}^{\mathsf{CCA}}$ responds with τP, where τ is randomly chosen from $\mathbb{Z}_q^*$. Otherwise, $\mathsf{B}^{\mathsf{CCA}}$ responds with Q. Similarly, if $\mathsf{A}^{\mathsf{CCA}}$ issues $\mathtt{ID} \neq \mathtt{ID}_\mu$ as a private key extraction query, $\mathsf{B}^{\mathsf{CCA}}$ responds to the query with τY, where τ is randomly chosen from $\mathbb{Z}_q^*$, and stops the simulation otherwise. (However, if $\mathtt{ID}_\mu = \mathtt{ID}^*$, this query is not allowed.) 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. Notice that if $\mathtt{ID}_\mu = \mathtt{ID}^*$, which happens with probability $1/q_{\mathsf{H}_1}$, the simulation is perfect.

The next step is to show how to use the IND-THD-CCA attacker $\mathsf{B}^{\mathsf{CCA}}$ for `ThdBm` to construct an attacker $\mathsf{A}^{\mathsf{BDH}}$ for solving the BDH problem. Suppose that $(\mathcal{G}, q, \hat{e}, P, aP, bP, cP)$ for random $a, b, c \in \mathbb{Z}_q^*$ are given to $\mathsf{A}^{\mathsf{BDH}}$. Assume that $\mathsf{B}^{\mathsf{CCA}}$ has access to the common parameter $(\mathcal{G}, q, P, \hat{e}, \mathsf{H}_2, \mathsf{H}_3, \mathsf{H}_4, Y, Q)$. First, $\mathsf{A}^{\mathsf{BDH}}$ replaces Y by bP and Q by cP. If $\mathsf{B}^{\mathsf{CCA}}$ corrupts a subset of $t-1$ servers, where t is a threshold parameter, $\mathsf{A}^{\mathsf{BDH}}$ assumes that the servers $\Gamma_1, \Gamma_2, \ldots, \Gamma_{t-1}$ have been corrupted without loss of generality. $\mathsf{A}^{\mathsf{BDH}}$ then chooses $S_1, S_2, \ldots, S_{t-1}$ uniformly at random from $\mathcal{G}$ and computes $y_i = \hat{e}(Q,Y)^{c_{i0}^{\Phi}} \prod_{j=1}^{t-1} \hat{e}(S_j, P)^{c_{ij}^{\Phi}}$, where $t \le i \le n$ and c_{ij}^{Φ} denotes the Lagrange coefficient for a set $\Phi = \{0, 1, \ldots, t-1\}$. $\mathsf{A}^{\mathsf{BDH}}$ sends y_i to each of the uncorrupted decryption servers, that is, $\mathsf{A}^{\mathsf{BDH}}$ replaces the verification keys with the new y_i computed above.

Whenever the random oracle H_3 is queried at some point by $\mathsf{B}^{\mathsf{CCA}}$, $\mathsf{A}^{\mathsf{BDH}}$ picks s uniformly at random from $\mathbb{Z}_q^*$, computes $H_3 = sY$, and responds $\mathsf{B}^{\mathsf{CCA}}$ with it. On receiving queries to other random oracles, $\mathsf{A}^{\mathsf{BDH}}$ picks values at random from the ranges of the random oracles, and responds with them.

When $\mathsf{B}^{\mathsf{CCA}}$ submits two plaintexts (M_0, M_1) to the encryption oracle, $\mathsf{A}^{\mathsf{BDH}}$ creates a target ciphertext $C^* = (U^*, V^*, W^*)$ as follows. First, $\mathsf{A}^{\mathsf{BDH}}$ sets $U^* = aP$. $\mathsf{A}^{\mathsf{BDH}}$ then picks a string V^* at random from $\{0,1\}^l$, computes $H_3^* = s^*P$ for random $s^* \in \mathbb{Z}_q^*$, and sets $H_3^* = \mathsf{H}_3(U^*, V^*)$. $\mathsf{A}^{\mathsf{BDH}}$ also computes $W^* = s^*U^*$. Having created C^*, $\mathsf{A}^{\mathsf{BDH}}$ returns it to $\mathsf{B}^{\mathsf{CCA}}$ as a target ciphertext. Note here that $\hat{e}(P, W^*) = \hat{e}(U^*, H_3^*)$ since $(P, U^*, H_3^*, W^*) = (P, aP, s^*P, s^*aP)$ and hence is a legitimate Diffie-Hellman tuple. Therefore, as long as $\mathsf{B}^{\mathsf{CCA}}$ does not query the random oracle H_2 at the point $\hat{e}(P,P)^{abc}$, the simulation is perfect. However, happening such an event means that $\mathsf{A}^{\mathsf{BDH}}$ is able to solve the BDH problem. So $\mathsf{A}^{\mathsf{BDH}}$ simulates $\mathsf{B}^{\mathsf{CCA}}$'s view up to this event.

Now, suppose that $\mathsf{A}^{\mathsf{CCA}}$ has already made a query (U, V) to the random oracle H_3. By the construction of the simulator for H_3, we have $H_3 = \mathsf{H}(U, V) = sY$ for random $s \in \mathbb{Z}_q^*$. Since $\mathsf{A}^{\mathsf{BDH}}$ knows the value s, $\mathsf{A}^{\mathsf{BDH}}$ can compute $K = (1/s)W$ and hence $\kappa = \hat{e}(Q, K)$. Note here that $(1/s)W = (1/s)rsY = rY = rxP$ and $Q = cP$. Then, $\mathsf{A}^{\mathsf{BDH}}$ computes $\kappa_i = \kappa^{c_{i0}^{\Phi}} \prod_{j=1}^{t-1} \hat{e}(S_j, U)^{c_{ij}^{\Phi}}$ for $t \le i \le n$. It is easy to check κ_i is a correct i-th share of the BDH key $\kappa = \hat{e}(Q, Y)^r$.

The rest is a simulation of a full decryption share $\delta_{i,C} = (i, \kappa_i, \tilde{\kappa_i}, \tilde{y_i}, L_i)$. This can easily be done by the zero-knowledge simulation technique, responding to queries to the random oracle H_4 with an element randomly chosen from $\mathbb{Z}_q^*$. $\square$

6 Application to Mediated ID-Based Encryption

6.1 Security Issues in Mediated ID-Based Encryption

The main motivation of mediated cryptography [4] is to revoke a user's privilege to perform cryptographic operations such as decrypting ciphertexts or signing messages *instantaneously*. In [4], 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 [3]. 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 $cp = (\mathcal{G}, q, P, \hat{e}, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3, \mathsf{H}_4, Y_{\mathrm{PKG}})$ 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 `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[3].
  • 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 `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 $\kappa = \hat{e}(\mathsf{H}_1(\mathtt{ID}), Y_{\mathrm{PKG}})^r$ for random $r \in \mathbb{Z}_q^*$.
– **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 1) 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.

[3] 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.

Definition 3 (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 `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 $\mathsf{A}^{\mathsf{CCA}'}$'s success as a function $\mathbf{Succ}^{\mathrm{IND-mID-sCCA}}_{\mathcal{MIDE},\mathsf{A}^{\mathsf{CCA}'}}(k) = 2 \cdot \Pr[\tilde{\beta} = \beta] - 1$. The mediated ID-based encryption scheme $\mathcal{MIDE}$ is said to be IND-mID-sCCA secure if, for any attacker $\mathsf{A}^{\mathsf{CCA}}$ whose running time is polynomially bounded, $\mathbf{Succ}^{\mathrm{IND-mID-sCCA}}_{\mathcal{MIDE},\mathsf{A}^{\mathsf{CCA}'}}(k)$ is negligible in k.

We now state and prove the following theorem. (Readers are referred to [1] for a more detailed proof.)

Theorem 2. *If the* $(2, 2)$-`IdThdBm` *scheme is IND-IDTHD-CCA secure then the* `mIdeBm` *scheme is IND-mID-sCCA secure.*

Proof. (Sketch) We show how to use the IND-mID-sCCA attacker $\mathsf{A}^{\mathsf{CCA}'}$ for $\mathtt{mIdeBm}$ to construct an IND-IDTHD-CCA attacker $\mathsf{A}^{\mathsf{CCA}}$ for $\mathtt{IdThdBm}$.

When $\mathsf{A}^{\mathsf{CCA}'}$ issues a new "User key extraction" or "SEM key extraction" query, which is an $\mathtt{ID}$, $\mathsf{A}^{\mathsf{CCA}}$ forwards $\mathtt{ID}$ to its Challenger as a private key extraction query, obtains a private key $D_{\mathtt{ID}}$ associated with $\mathtt{ID}$, and gives $D_{\mathtt{ID}}$ to $\mathsf{A}^{\mathsf{CCA}'}$. Having done this, $\mathsf{A}^{\mathsf{CCA}}$ splits $D_{\mathtt{ID}}$ into $D_{\mathtt{ID},sem}$ and $D_{\mathtt{ID},user}$ using the (2, 2)-secret-sharing technique. $\mathsf{A}^{\mathsf{CCA}}$ then adds $\langle \mathtt{ID}, D_{\mathtt{ID},user}\rangle$ and $\langle \mathtt{ID}, D_{\mathtt{ID},sem}\rangle$ to $\mathsf{UserKeyList}$ and $\mathsf{SEMKeyList}$ respectively. Using these lists, $\mathsf{A}^{\mathsf{CCA}}$ answers $\mathsf{A}^{\mathsf{CCA}'}$'s "SEM oracle" and "User oracle" queries, each of which consists of $(\mathtt{ID}, C)$. If necessary, $\mathsf{A}^{\mathsf{CCA}}$ forwards the $\mathtt{ID}$ in those queries to its Challenger to get a private key associated with it. It should be emphasized here that $\mathsf{A}^{\mathsf{CCA}}$ always checks the validity of the ciphertext $C = (U, V, W)$ by testing whether $\hat{e}(P, W)$ equals to $\hat{e}(U, \mathsf{H}_3(U, V))$. If C does not pass this test, $\mathsf{A}^{\mathsf{CCA}}$ rejects it.

Once $\mathsf{A}^{\mathsf{CCA}'}$ issues two equal-length plaintexts (M_0, M_1) and a target identity $\mathtt{ID}^*$, $\mathsf{A}^{\mathsf{CCA}}$ forwards $(M_0, M_1, \mathtt{ID}^*)$ to its Challenger. On receiving $(M_0, M_1, \mathtt{ID}^*)$, the Challenger runs the private key extraction algorithm of $\mathtt{IdThdBm}$ to get a private key $D_{\mathtt{ID}^*}$ associated with $\mathtt{ID}^*$ and runs the private key distribution algorithm of $\mathtt{IdThdBm}$ to split $D_{\mathtt{ID}^*}$ into $D_{\mathtt{ID}^*,sem}$ and $D_{\mathtt{ID}^*,user}$. The Challenger gives $D_{\mathtt{ID}^*,user}$ to $\mathsf{A}^{\mathsf{CCA}}$ as a corrupted party's private key. $\mathsf{A}^{\mathsf{CCA}}$ then sends this back to $\mathsf{A}^{\mathsf{CCA}'}$. In doing so, 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 $\mathtt{IdThdBm}$ taking $(M_\beta, \mathtt{ID}^*)$ as input and gets a target ciphertext C^*. The Challenger gives it to $\mathsf{A}^{\mathsf{CCA}}$. Then, $\mathsf{A}^{\mathsf{CCA}}$ sends C^* back to $\mathsf{A}^{\mathsf{CCA}'}$.

$\mathsf{A}^{\mathsf{CCA}}$ answers "User key extraction", "SEM key extraction", "SEM oracle", and "User oracle" queries in the same way it did before. Note, however, that the cases when $(\mathtt{ID}, C^*)$ and $(\mathtt{ID}^*, C)$ are asked as "SEM oracle" and "User oracle" queries should be handled at this stage. Especially, $\mathsf{A}^{\mathsf{CCA}}$ uses its decryption servers to handle the query $(\mathtt{ID}^*, C)$.

Finally, if $\mathsf{A}^{\mathsf{CCA}'}$ outputs a guess $\beta' \in \{0, 1\}$, $\mathsf{A}^{\mathsf{CCA}}$ returns it as its guess. $\square$

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. J. Baek and, Y. Zheng, *Identity-Based Threshold Decryption*, IACR ePrint Archive Report 2003/164.
2. 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.
3. M. Bellare and P. Rogaway, *Optimal Asymmetric Encryption*, Advances in Cryptology -Proceedings of Eurocrypt '94, LNCS 950, Springer-Verlag 1994, pages 92–111.
4. 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.
5. D. Boneh and M. Franklin, *Identity-Based Encryption from the Weil Pairing*, Proceedings of CRYPTO 2001, LNCS 2139, Springer-Verlag 2001, pages 213–229.
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.