Identity-Based Threshold Decryption

Joonsang Baek[1] 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 consider the problem of constructing an identity-based threshold decryption scheme, that is, an identity-based threshold decryption scheme. In contrast to the previous approaches that had focused on the distribution of the master key of a Private Key Generator (PKG) in an identity-based public key encryption scheme, we point out that distribution of a user's private key after the user acquires it from the PKG is essential in realizing practical identity-based threshold decryption, and propose a possible solution to achieve this. As a result we construct the first identity-based threshold decryption scheme in which the role of the PKG is minimized to issue private keys for users. We also formulate a precise definition for the security of an identity-based threshold decryption scheme against adaptive chosen ciphertext attacks and prove that our scheme is secure in the random oracle model, under the assumption that the Bilinear Diffie-Hellman problem is computationally hard.

1 Introduction

Threshold decryption is particularly useful where the centralization of the power to decrypt is a concern. On the other hand, the motivation of identity (ID)-based public key encryption 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 successful combination of these two concepts will allow one to build an "ID-based threshold decryption" scheme. As an example, consider a situation where Alice wishes to send a confidential message to a committee in an organization. Alice can first encrypt the message using the identity 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.

One possible approach to ID-based threshold decryption was suggested by Boneh and Franklin [4]. The essence of the approach is to split the master secret key of the PKG for an ID-based encryption among a number of PKGs and use these distributed PKGs as decryption servers. While this approach provides a partial solution to the problem of ID-based threshold decryption, it has a drawback that limits its practical usefulness. The drawback stems from the fact that the PKGs are required to be engaged in the decryption of a ciphertext *on line*, which will be discussed in more detail in a later section.

In this paper, we take a fresh approach to ID-based threshold decryption. We focus on ID-based public key encryption schemes in which the role of the PKG is minimized to issue private keys of users and a user who acquired a private key from the PKG can distribute the private key into a number of decryption servers at will.

2 Discussions on Related Work

2.1 ID-based Encryption as Envisioned by Shamir

The concept of ID-based encryption was originally proposed by Shamir [14]. In ID-based encryption, Alice, for example, who wants to send a confidential message to Bob, can use Bob's identity such as email or IP address to encrypt her message. Upon receiving the ciphertext from Alice, Bob checks whether he is already in possession of the private key that matches his public identity. If not, he contacts the trusted entity, which Boneh and Franklin called the "Private Key Generator" (PKG) [4], to obtain the private key. Bob is now able to decrypt the ciphertext with the private key. It is important to note that services of the PKG are needed once only when Bob acquires the private key and hence the PKG *can be closed after the key generation.* This was emphasized by Shamir in his original proposal [14].

2.2 Boneh and Franklin's ID-Based Encryption Scheme

In contrast to the simplicity of the concept, successful realization of the ID-based encryption was only made recently by Boneh and Franklin [4], and Cocks [7]. Since our work is more related to Boneh and Franklin's work, we discuss it in more detail.

2.2.1 Bilinear Map

The admissible bilinear map $\hat{e}$ 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 $\mathcal{G}$ will be implemented using a group of points on certain supersingular elliptic curves 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 [4].

- 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.

2.2.2 The ID-based Encryption Scheme "BasicIdent"

Using the bilinear map, Boneh and Franklin constructed an ID-based encryption scheme called "BasicIdent".

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 is the length a plaintext message. Then, the PKG picks a master key x uniformly at random from $\mathbb{Z}_q^*$ and computes a public key $Y_{\text{PKG}} = xP$. 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 $D_{\texttt{ID}} = xQ_{\texttt{ID}}$ where $Q_{\texttt{ID}} = \mathsf{H}_1(\texttt{ID})$, which matches to his identity $\texttt{ID}$.

Alice, the sender, can now encrypt her message m using the Bob's identity $\texttt{ID}$ by computing $U = rP$ and $V = \mathsf{H}_2(\hat{e}(Q_{\texttt{ID}}, Y_{\text{PKG}})^r) \oplus m$, where r is chosen at random from $\mathbb{Z}_q^*$ and $Q_{\texttt{ID}} = \mathsf{H}_1(\texttt{ID})$. The resulting ciphertext $C = (U, V)$ is sent to Bob.

Bob decrypts C by computing $m = V \oplus \mathsf{H}_2(\hat{e}(D_{\texttt{ID}}, U))$.

2.2.3 Boneh and Franklin's "Distributed PKG"

As an extension of their ID-based encryption scheme, Boneh and Franklin [4] suggested that the BasicIdent scheme should be associated with the techniques of threshold cryptography. Their idea focuses on distributing the PKG's master key x in such a way that each of a number of PKGs is given one share $x_i \in \mathbb{Z}_q^*$ of a Shamir's (t, n) secret sharing [13] of $x \in \mathbb{Z}_q^*$ and can respond to a private key extraction request with $D_{\texttt{ID}}^i = x_i Q_{\texttt{ID}}$, where $Q_{\texttt{ID}} = \mathsf{H}_1(\texttt{ID})$. If the technique of [9] 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.

Now, we investigate the above method of distributing the PKG's master key, called the "Distributed PKG", in more detail. We assume below that Bob, who is not a PKG, has published his identity and has obtained the matching private key from the PKG.

First, we emphasize that as its name suggests, the purpose of the Distributed PKG method is to prevent a single PKG from possessing the whole master key, rather than to distribute Bob's private key. In other words, the method only provides a distribution of the PKG's master key *before* Bob obtains his private key, which we call a "first-level distribution". A distribution of Bob's private key obtained from the PKG (It does not matter whether the PKG is distributed or not.) into other users (decryption servers), which we call a "second-level distribution", was not treated in [4].

Although, in theory, each of the distributed PKGs may also function as a decryption server for the second-level distribution, such a method is not quite practical in practice as will be shown below.

Let us assume that n PKGs have jointly generated a master key x and a global public key $Y_{\text{PKG}} = xP$. Let t be a threshold. Assume further that each of the n distributed PKGs now functions as a decryption server. Suppose that Bob has published an identity $\texttt{ID}$ which Alice can use to encrypt her message. Alice can encrypt her message m by creating a ciphertext $C = (U, V) = (rP, m \oplus \hat{e}(Q_{\texttt{ID}}, Y_{\text{PKG}})^r)$ for random $r \in \mathbb{Z}_q^*$, where $Q_{\texttt{ID}} = \mathsf{H}_1(\texttt{ID})$. Now, a user who wants to decrypt C should get at least t decryption shares to recover the Alice's plaintext m.

For generation of decryption shares of C, Boneh and Franklin [4] suggest that one should use "per-message" basis threshold decryption method. In this method, each of the distributed PKGs responds to the decryption request $C = (U, V)$ with $\hat{e}(x_i Q_{\texttt{ID}}, U)$ where x_i is one share of the master key x. However, this requires each PKG be involved *at all times* in the generation of decryption shares because the value "U" changes whenever a new ciphertext is created, which creates a bottleneck on the PKGs. It also violates one of the basic requirements of an ID-based encryption scheme as envisioned by Shamir, that is "the PKG can be closed after key generation".

Indeed, we do not expect the PKG to be involved in particular applications such as threshold decryption especially where a service of the PKG is provided by a third-party corporation such as a commercial PKI provider. Hence, we still need a new type of ID-based threshold decryption in which the role of the PKG is as minimal as possible.

2.3 Other Related Work

Since Boneh and Franklin's work, there have been interesting proposals on applications of the ID-based encryption scheme. Recently, Chen, Harrison, Soldera and Smart [6] illustrated how the use of multiple PKGs/identities in Boneh and Franklin's ID-based encryption scheme can be applied to the real world situations. Furthermore, they dealt with general cases of disjunction and conjunction of the multiple PKGs/identities exploiting, the algebra of the bilinear maps. Subsequently, more complicated cases of the disjunction and conjunction of the multiple PKGs and their applications to access controls were discussed by Smart [16].

Khalili, Katz and Arbaugh [10] also discussed the use of the distributed PKGs in Boneh and Franklin's scheme, especially focusing on its application to ad-hoc networks.

However, the issue of how to distribute a user's private key after the user acquires it from the PKG was not discussed in any of [6], [16] and [10].

Very recently and independently of our work, Libert and Quisquater [11] proposed an ID-based threshold decryption scheme based on the bilinear map, which is similar to Boneh and Franklin's "per-message" basis threshold decryption discussed in the previous section. The difference is that in Libert and Quisquater's scheme, one PKG is fixed as a dealer to create shared-secret/public key pairs $(x_i Q_{\mathtt{ID}}, x_i P)$ $(Q_{\mathtt{ID}} = \mathsf{H}_1(\mathtt{ID}))$ on behalf of Bob who has created the identity $\mathtt{ID}$, where x_i is a (t, n) Shamir secret sharing [13] of the master key $x \in \mathbb{Z}_q^*$, and send them to decryption servers. The PKG also publishes its global public key $Y_{\mathrm{PKG}} = xP$. Upon receiving a ciphertext $C = (U, V) = (rP, \hat{e}(Q_{\mathtt{ID}}, Y_{\mathrm{PKG}})^r)$ that encrypts Alice's message m, each decryption server simply outputs $\hat{e}(x_i Q_{\mathtt{ID}}, U)$ as a decryption share.

However, unlike our approach to ID-based threshold decryption scheme, the above scheme still does not provide a feature that a user who obtained a private key from the PKG can share the key among decryption servers at will. Moreover, their scheme only provides security against chosen plaintext attack and chosen ciphertext security was not treated in [11].

3 Security Notion for ID-based Threshold Decryption

In this section, we formulate a security notion for ID-based threshold decryption scheme against adaptive chosen ciphertext attacks.

3.1 Generic ID-based Threshold Decryption

We describe a generic (t, n) ID-based threshold decryption scheme "$\mathcal{IDTHD}$", which consists of sub-algorithms GK, EX, DK, E, D, SV and SC.

Like other ID-based cryptographic schemes, there exists a PKG in the ID-based threshold decryption scheme. The PKG runs the key generation algorithm GK to generate its master/public key pair and all other necessary common parameters. The PKG's public key and the common parameters are given to every entity involved.

One of the entities, which is assumed to behave honestly, then creates an identity and contacts the PKG to obtain a private key that associated with the identity. Upon receiving the request, the PKG runs the private key extraction algorithm EX to extract the private key and returns it to the entity. Having obtained the private key, the entity runs the private key distribution algorithm DK to share the private key into n decryption servers. DK makes use of an appropriate threshold 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 each of the decryption servers. Each decryption server then keeps its private key share as secret but publishes the verification key.

Given the entity's identity, any other entity can now encrypt a plaintext by running the encryption algorithm E.

A legitimate entity 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 on input the ciphertext and send corresponding decryption shares to the entity. Note that the validity of the shares can be checked by running the decryption share verification algorithm SV. When the entity collects valid decryption shares from at least t (a threshold) servers, the ciphertext can be decrypted by running SC.

Below, we formally describe the above algorithms.

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

- A randomized key generation algorithm $\mathsf{GK}(k)$: Given a security parameter $k \in \mathbb{N}$, this algorithm computes a 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 entities involved and is provided as input to the following algorithms 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 $\{sk_i\}_{1\leq i\leq n}$ and $\{vk_i\}_{1\leq i\leq n}$, respectively. Note that each (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 denoted by δ_i.

- A decryption share verification algorithm $\mathsf{SV}(cp, vk_i, C, \delta_i)$: Given a ciphertext C, a verification key vk_i of each decryption server Γ_i and a decryption share δ_i, this algorithm checks the validity of δ_i. The output of this algorithm is either "*valid*" or "*invalid*".

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

3.2 Chosen Ciphertext Security for ID-Based Threshold Decryption

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

Definition 2 (IDTHD-IND-CCA) Let $\mathsf{A}^{\mathsf{CCA}}$ be an attacker that defeats the security of an ID-based threshold decryption scheme $\mathcal{IDTHD}$ in the sense of IDTHD-IND-CCA. 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 an 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. The Challenger gives $\mathsf{A}^{\mathsf{CCA}}$ the resulting common parameter cp which includes a 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 $\{(sk_i, vk_i)\}$, 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 on input C 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

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

We denote by $\mathbf{Succ}^{\mathrm{IDTHD-IND-CCA}}_{\mathcal{IDTHD}}(t_{IDCCA}, q_E, q_D)$ 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. We say that the ID-based threshold decryption scheme $\mathcal{IDTHD}$ is secure in the sense of THD-IND-CCA if $\mathbf{Succ}^{\mathrm{IDTHD-IND-CCA}}_{\mathcal{IDTHD}}$ (t_{IDCCA}, q_E, q_D) is negligible in k.

In the next section, we present three important building blocks for constructing our ID-based threshold decryption scheme.

4 Building Blocks for Proposed Scheme

4.1 Publicly Checkable Encryption

Publicly checkable encryption is particularly useful for building threshold decryption schemes as discussed by Lim and Lee [12]. The main reason is that in the threshold cryptosystem the attacker has decryption shares as additional information, as well as decryption of chosen ciphertext. (Readers are referred to [12] and [15] for more detailed explanations.)

Note that public checkability in threshold decryption schemes is usually given by non-interactive zero-knowledge (NIZK) proofs, e.g., [15, 8]. However, we emphasize that in our scheme, this can be done *without* a NIZK proof, exploiting the special property of the underlying group $\mathcal{G}$, as mentioned in [4] and further exploited in [3]. We illustrate this as follows:

Suppose that our scheme encrypts a message m by creating a ciphertext $C = (U, V, W) = (rP, \mathsf{H}_1(\kappa) \oplus m, r\mathsf{H}_2(U, V))$, where $\mathsf{H}_1 : \mathcal{F} \to \{0,1\}^l$ and $\mathsf{H}_2 : \mathcal{G}^* \times \{0,1\}^l \to \mathcal{G}^*$ are appropriate hash functions, and κ is the Bilinear Diffie-Hellman key which will be described in greater detail in Section 5. Without employing the NIZK proof, the validity of C can be checked by verifying if $\hat{e}(P, W) = \hat{e}(U, H_2)$, where $H_2 = \mathsf{H}_2(U, V) \in \mathcal{G}^*$. Note that the existence of this validity check implies the Decisional Diffie-Hellman (DDH) problem can be easily solved in the group $\mathcal{G}$ since, if the above test holds then $(P, U, H_2, W) = (P, rP, sP, rsP)$ assuming that $H_2 = sP \in_R \mathcal{G}^*$ for some $s \in \mathbb{Z}_q^*$, namely, (P, U, H_2, W) is a Diffie-Hellman tuple.

4.2 A New Technique of Sharing Points on Elliptic Curves

Recall that the multiple PKGs in Boneh and Franklin's ID-based encryption scheme can be achieved by sharing the PKG's master key x [4]. Indeed, this can easily be done using the Shamir's secret sharing technique [13] as the master key x is a single element in $\mathbb{Z}_q^*$ and hence Shamir's technique [13] can be used directly to distribute an element in $\mathbb{Z}_q^*$.

However, in order to distribute a private key $D_{\mathtt{ID}}$ obtained from the PKG, we need to devise a new sharing technique since $D_{\mathtt{ID}}$ is a point on an elliptic curve and moreover, its discrete logarithm is unknown, so Shamir's technique cannot be applied.

In what follows we describe our method for sharing a secret point on an elliptic curve. The method represents a simple adaptation of Shamir's classic (t, n) secret sharing technique.

Distribution Phase: Let q be a prime order of a group $\mathcal{G}$ of points on elliptic curve. Let $S \in \mathcal{G}^*$ be a secret (point) to share. Suppose that we have chosen integers t 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

$$F(u) = S + \sum_{l=1}^{t-1} u^l R_l.$$

Now, we 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. Note that when $i = 0$, we obtain the secret itself, that is, $S = F(0)$. (We assume that the "multiplication-by-m map for a positive integer m" denoted by mP is extended to all integer $m \in \mathbb{Z}$ by defining $0P = O$ where O is the identity element of $\mathcal{G}$, and $(-m)P = -(mP)$ [2].)

Reconstruction Phase: Let $\Phi \subset \{1, \ldots, n\}$ be a set such that $|\Phi| \geq t$, where $|\cdot|$ denotes the cardinality of the given set. The function $F(u)$ can be reconstructed by computing

$$F(u) = \sum_{j \in \Phi} c_{uj}^{\Phi} S_j \text{ where } c_{uj}^{\Phi} = \prod_{\iota \in \Phi, \iota \neq j} \frac{u - \iota}{j - \iota} \in \mathbb{Z}_q.$$

Actually, the value $c_{uj}^{\Phi} \in \mathbb{Z}_q$ is the Lagrange interpolation coefficient used in Shamir's secret sharing scheme: If we write $S = sP$ and $R_l = r_l P$ for for some $s, r_l \in \mathbb{Z}_q^*$ and $1 \leq l \leq t-1$ (but, we do not know s and r_l), we have $F(u) = sP + ur_1P + \cdots + u^{t-1}r_{t-1}P = (s + r_1u + \cdots + r_{t-1}u^{t-1})P$. Hence, the Lagrange coefficients c_{uj}^{Φ}'s reconstruct the original function $F(u)$. In practice, we recover the secret S directly (without reconstructing $F(u)$) by computing $\sum_{j \in \Phi} c_{0j}^{\Phi} S_j$ where $c_{0j}^{\Phi} = \prod_{\iota \in \Phi, \iota \neq j} \frac{\iota}{\iota - j}$.

4.3 A New Zero Knowledge Proof System for the Equality of Two Discrete Logarithms Based on the Bilinear Map

For decryption share verification, that is, ensuring that all decryption shares are consistent, we need a certain checking procedure. In contrast to the validity checking method of ciphertexts discussed in Section 4.1, we need a zero-knowledge proof system since the share of BDH key κ appeared in the decryption share in our scheme is resulted from the bilinear map $\hat{e}$ and hence is the element in the group $\mathcal{F}$, where the DDH problem is believed to be hard.

Motivated by [5] and [15], we construct the new zero-knowledge proof of membership system for the following language

$$L_{\mathsf{EDLog}^{\mathcal{F}}_{P,\tilde{P}}} \overset{\text{def}}{=} \{(\mu, \tilde{\mu}) \in \mathcal{F} \times \mathcal{F} \quad | \quad \log_g \mu = \log_{\tilde{g}} \tilde{\mu} \text{ where } g = \hat{e}(P, P) \text{ and } \tilde{g} = \hat{e}(P, \tilde{P}) \text{ for generators } P \text{ and } \tilde{P} \text{ of } \mathcal{G}\},$$

where two groups $\mathcal{G}$ and $\mathcal{F}$, and the bilinear map $\hat{e}$ are as defined in Section 2.2.

It is easy to see that g and $\tilde{g}$ are generators of $\mathcal{F}$, and that $(\kappa, \tilde{\kappa}) \in L_{\mathsf{EDLog}_{\mathcal{F}}}$ if and only if there exists a non-identity element $S \in \mathcal{G}$ such that $\kappa = \hat{e}(S, P)$ and $\tilde{\kappa} = \hat{e}(S, \tilde{P})$. (These have formally been proven as Lemmas 1 and 2 in Appendix A.)

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

- The Prover chooses a non-identity element Q uniformly at random from $\mathcal{G}$ and computes $\gamma = \hat{e}(Q, P)$ and $\tilde{\gamma} = \hat{e}(Q, \tilde{P})$. 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 $L = Q + hS \in \mathcal{G}$ 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 system satisfies completeness, soundness and zero-knowledge property against *honest* verifier. (This has proven as Theorem 2 in Appendix A.) Also, `ZKBm` can easily be converted to a NIZK proof, making the random challenge an output of a random oracle [1]. Note 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 Our ID-Based Threshold Decryption Scheme

5.1 Description of the Scheme

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 $P \in \mathcal{G}$. Then, it specifies the bilinear map $\hat{e} : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{F}$ and the following hash functions H_1, H_2, H_3 and H_4 such that $\mathsf{H}_1 : \mathcal{F} \rightarrow \{0, 1\}^l$, $\mathsf{H}_2 : \mathcal{G}^* \times \{0, 1\}^l \rightarrow \mathcal{G}^*$, $\mathsf{H}_3 : \{0, 1\}^* \rightarrow \mathcal{G}^*$; $\mathsf{H}_4 : \mathcal{F} \rightarrow \mathbb{Z}_q^*$, where l is the length a plaintext message. Next, it chooses a PKG's master key x uniformly at random from $\mathbb{Z}_q^*$ and computes a PKG's public key $Y_{\text{PKG}} = xP$. Finally, it returns a common parameter[1] $cp = (\mathcal{G}, q, P, \hat{e}, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3, \mathsf{H}_4, Y_{\text{PKG}})$. Note that the master key x is kept secret.
- $\mathsf{EX}(cp, \mathtt{ID})$: Given an identity $\mathtt{ID}$, this algorithm computes $Q_{\mathtt{ID}} = \mathsf{H}_3(\mathtt{ID})$ and $D_{\mathtt{ID}} = xQ_{\mathtt{ID}}$. Then, it returns the private key $D_{\mathtt{ID}}$.
- $\mathsf{DK}(cp, \mathtt{ID}, D_{\mathtt{ID}}, t, n)$ 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 it publishes y_i.
- $\mathsf{E}(cp, \mathtt{ID}, m)$: Given a plaintext message $m \in \{0, 1\}^l$ and an identity $\mathtt{ID}$, this algorithm chooses r uniformly at random from $\mathbb{Z}_q^*$, and computes $Q_{\mathtt{ID}} = \mathsf{H}_3(\mathtt{ID})$, $d = \hat{e}(Q_{\mathtt{ID}}, Y_{\text{PKG}})$ and $\kappa = d^r$ in turn. It then computes

$$U = rP; V = \mathsf{H}_1(\kappa) \oplus m; W = r\mathsf{H}_2(U, V)$$

[1] As mentioned, cp is provided as input to all of the following algorithms.

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, this algorithm computes $H_2 = \mathsf{H}_2(U, V)$ and checks if $\hat{e}(P, W) = \hat{e}(U, H_2)$. If the test holds then this algorithm computes $\kappa_i = \hat{e}(S_i, U)$, $\tilde{\kappa_i} = \hat{e}(Q_i, U)$, $\tilde{y_i} = \hat{e}(Q_i, P)$, $\lambda_i = \mathsf{H}_4(\kappa_i, \tilde{\kappa_i}, \tilde{y_i})$ and $L_i = Q_i + \lambda_i S_i$ in turn for random $Q_i \in \mathcal{G}$, and outputs $\delta_i = (i, \kappa_i, \tilde{\kappa_i}, \tilde{y_i}, L_i)$. Otherwise, it returns (i, "*void*").

- $\mathsf{SV}(cp, vk, C, \delta_i)$: Given a ciphertext C and a decryption share δ_i, this algorithm computes $H_2 = \mathsf{H}_2(U, V)$ and checks if $\hat{e}(P, W) = \hat{e}(U, H_2)$. If the tests holds then it does the following [2] :

  - If δ_i is of the form (i, "*void*") then output "*invalid*".
  - Else parse δ_i as $(i, \kappa_i, \tilde{\kappa_i}, \tilde{y_i}, L_i)$ and compute $\lambda_i = \mathsf{H}_4(\kappa_i, \tilde{\kappa_i}, \tilde{y_i})$.
    - Check if $\hat{e}(L_i, U)/\kappa_i^{\lambda_i} = \tilde{\kappa_i}$ and $\hat{e}(L_i, P)/y_i^{\lambda_i} = \tilde{y_i}$.
    - If the test above holds, output "*valid*", else output "*invalid*".

  Otherwise, do the following:

  - If δ_i is of the form (i, "*void*"), output "*valid*", else output "*invalid*".

- $\mathsf{SC}(cp, C, \{\delta_j\}_{j \in \Phi})$: Given a ciphertext C and a set of decryption shares $\{\delta_j\}_{j \in \Phi}$ where $|\Phi| \geq t$, this algorithm computes $H_2 = \mathsf{H}_2(U, V)$ and checks if $\hat{e}(P, W) = \hat{e}(U, H_2)$. If the ciphertext passes the test then this algorithm computes $\kappa = \prod_{j \in \Phi} \kappa_j^{c_{0j}^{\Phi}}$ and $m = \mathsf{H}_1(\kappa) \oplus V$ in turn, and outputs m. Otherwise, it outputs "*void*"[3].

It is easy to see that if C is a valid ciphertext and $|\Phi| \geq t$ then $\mathsf{SC}(C, \{\delta_j\}_{j \in \Phi}) = m$: Indeed, if $C = (U, V, W)$ has passed all the validity checks above,

$$\begin{aligned}
\prod_{j \in \Phi} \kappa_j^{c_{0j}^{\Phi}} &= \prod_{i \in \Phi} \hat{e}(S_j, U)^{c_{0j}^{\Phi}} = \prod_{j \in \Phi} \hat{e}(S_j, rP)^{c_{0j}^{\Phi}} = \hat{e}(\sum_{j \in \Phi} c_{0j}^{\Phi} S_j, rP) \\
&= \hat{e}(D_{\mathtt{ID}}, P)^r = \hat{e}(xQ_{\mathtt{ID}}, P)^r = \hat{e}(Q_{\mathtt{ID}}, xP)^r \\
&= \hat{e}(Q_{\mathtt{ID}}, Y_{\mathrm{PKG}})^r = d^r = \kappa,
\end{aligned}$$

where c_{0j}^{Φ} is the Lagrange coefficient defined in Section 4.2. Hence, $\mathsf{H}_1(\kappa) \oplus V = \mathsf{H}_1(\kappa) \oplus (\mathsf{H}_1(d^r) \oplus m) = m$.

5.2 Extension to the First-Level Distribution

As emphasized through previous sections, our approach to ID-based threshold decryption focuses on providing the *second-level distribution* in which the user's private key obtained from the PKG is distributed into a number of decryption servers. However, our scheme provides the first-level distribution as well. More precisely, if the PKG's master key is of particular importance and is required to be distributed into multiple PKGs, one can use, e.g., the technique of [9] for the master key x to be jointly generated by the multiple PKGs as suggested by Boneh and Franklin (Distributed PKG method). Holding a share x_i of x, each of the multiple PKGs responds to Bob's private key extraction request with $D_{\mathtt{ID}}^i = x_i Q_{\mathtt{ID}}$ then he can collect these shares and recover the private key $D_{\mathtt{ID}}$. After Bob acquires $D_{\mathtt{ID}}$, he can distribute it into decryption servers in the group at will, using our method proposed in this paper.

[2] If we let $g = \hat{e}(P, P)$ and $\tilde{g} = \hat{e}(P, U)$ then passing the above test means $(g, \tilde{g}, y_i, \kappa_i)$ is a Diffie-Hellman tuple.

[3] Note that this algorithm is allowed to output a symbol "*void*", which is distinct from all possible plaintexts.

6 Security Analysis

6.1 Bilinear Diffie-Hellman Problem

Before analyzing our scheme, we review the Bilinear Diffie-Hellman (BDH) problem which is a new class of computational problem introduced by Boneh and Franklin [4].

Definition 3 (BDH) Let $\mathcal{G}$ and $\mathcal{F}$ be two groups of order q where q is prime, as defined in Section 2.2. Let $P \in \mathcal{G}^*$ be a generator of $\mathcal{G}$. Suppose that there exists a bilinear map $\hat{e} : \mathcal{G} \times \mathcal{G} \to \mathcal{F}$. Let $\mathsf{A}^{\mathsf{BDH}}$ be an attacker modelled as a probabilistic Turing machine.

The BDH problem refers to the computational problem in which $\mathsf{A}^{\mathsf{BDH}}$ tries 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 $\mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G},\mathsf{A}^{\mathsf{BDH}}}(k)$ by the probability $\mathsf{A}^{\mathsf{BDH}}$ 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. We say that the BDH problem is intractable if $\mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G}}(t_{\mathsf{BDH}})$ is negligible in k.

6.2 Proof of Security

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

Theorem 1 *Suppose that an IDTHD-IND-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 each of the random oracles. Using this attacker as a subroutine, we can construct a BDH attacker for the group* $\mathcal{G}$*, whose advantage including running time* t_{BDH} *is bounded as follows.*

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

where $t_{\mathrm{BDH}} = t_{IDCCA} + \max(q_E, q_{\mathsf{H}_3})O(k^3) + q_{\mathsf{H}_1} + q_{\mathsf{H}_2}O(k^3) + q_{\mathsf{H}_4}q_D O(k^3)$ *for a security parameter* k*.*

To prove the above theorem, we define an ordinary (non-ID-based) threshold decryption scheme called "`ThdBm`" by modifying the `IdThdBm` scheme, which will be described shortly. Then, we show in Lemma 3 that the security of the `ThdBm` scheme in the sense of THD-IND-CCA defined in Appendix B implies the IDTHD-IND-CCA security of the `IdThdBm` scheme. Next, we show in Lemma 4 that the intractability of the BDH problem implies the THD-IND-CCA security of the `ThdBm` scheme. Combining Lemmas 3 and 4, we obtain Theorem 1 above. Precise statements and proofs of Lemmas 3 and 4 are given in Appendices C and D respectively.

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

- $\mathsf{GK}(k, t, n)$: On input 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 $P \in \mathcal{G}$. Then, it specifies the bilinear map $\hat{e} : \mathcal{G} \times \mathcal{G} \to \mathcal{F}$ and the following hash functions H_1, H_2 and H_4 such that

$\mathsf{H}_1 : \mathcal{F} \to \{0,1\}^l$, $\mathsf{H}_2 : \mathcal{G}^* \times \{0,1\}^l \to \mathcal{G}^*$ and $\mathsf{H}_4 : \mathcal{F} \to \mathbb{Z}_q^*$, where l is the length of a plaintext to be encrypted. Next, it chooses x uniformly at random from $\mathbb{Z}_q^*$ and computes $Y = xP$. It then chooses Q uniformly at random from $\mathcal{G}^*$ and computes $D = xQ$. Note that (Q, Y) and D will serve as public and private key respectively. 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 constructs $F(u) = D + \sum_{j=1}^{t-1} u^j R_j$ for $u \in \{0\} \cup \mathbb{N}$. Then, it computes each server's private key $S_i = F(i)$ for $1 \le i \le n$ and verification key $y_i = \hat{e}(S_i, P)$ for $1 \le i \le n$. Finally, it outputs a common parameter $cp = (\mathcal{G}, q, P, \hat{e}, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_4, Y, Q)$, and sends the verification/private key pair (y_i, S_i) to each decryption server Γ_i for $1 \le i \le n$. Upon receiving (y_i, S_i), each decryption server publishes y_i where $1 \le i \le n$.

- $\mathsf{E}(cp, m)$: 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 $U = rP$, $V = \mathsf{H}_1(\kappa) \oplus m$ and $W = r\mathsf{H}_2(U, V)$, and outputs a ciphertext $C = (U, V, W)$.

7 Concluding Remarks

In this paper, we discussed the issues on realization of ID-based threshold decryption and gave a precise definition for secure ID-based decryption schemes. By taking advantage of a new technique of sharing a point on elliptic curves, we constructed an ID-based threshold decryption scheme in which a user who obtained a private key from the PKG can distribute the private key into a number of decryption servers at will, and proved its chosen ciphertext security in the random oracle model, assuming the Bilinear Diffie-Hellman (BDH) problem is intractable.

There are some interesting problems remained. From a theoretical point of view, how to build up a new ID-based threshold decryption scheme that gives chosen ciphertext security but depends on other computational primitives such as the integer factorization problem is one of them. From a practical point of view, how to apply the ID-based threshold decryption scheme to various security applications such as access controls is another interesting problem.

References

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

[2] I. F. Blake, G. Seroussi and N. P. Smart: *Elliptic Curves in Cryptography*, London Mathematical Society Lecture Note Series 265, 1999, Cambridge University Press.

[3] A. Boldyreva: *Efficient Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-group Signature Scheme*, Proceedings of Public Key Cryptography 2003 (PKC 2003), Vol. 2567 of LNCS, Springer-Verlag 2003, pages 31–46.

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

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

[6] L. Chen, K. Harrison, D. Soldera and N. P. Smart: *Applications of Multiple Trust Authorities in Pairing Based Cryptosysems*, Proceedings of InfraSec 2002, Vol. 2437 of LNCS, Springer-Verlag 2002, pages 260–275.

[7] C. Cocks: *An Identity Based Encryption Scheme Based on Quadratic Residues*, Cryptography and Coding - Proceedings of the 8th IMA International Conference, Vol. 2260 of LNCS, Springer-Verlag 2001, pages 360–363.

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

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

[10] A. Khalili, J. Katz and W. Arbaugh:*Toward Secure Key Distribution in Truly Ad-Hoc Networks*, Proceedings of IEEE Workshop on Security and Assurance in Ad-Hoc Networks, 2003.

[11] B. Libert and J. Quisquater: *Efficient Revocation and threshold Pairing Based Cryptosystems*, Principles of Distributed Computing (PODC) 2003, to appear.

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

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

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

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

[16] N. P. Smart: *Access Control Using Pairing Based Cryptography*, Proceedings of Topics in Cryptology-CT-RSA 2003, Vol. 2612 of LNCS, Springer-Verlag 2003, pages 111–121.

A Proofs of Lemmas for `ZKBm`

Lemma 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. □

Lemma 2 *Let P and $\tilde{P}$ be generators of $\mathcal{G}$. Then, $(\kappa, \tilde{\kappa}) \in L_{\mathsf{EDLog}_{\mathcal{F}}}$ 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 Lemma 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, by the bilinear property of $\hat{e}$, we have $\hat{e}(P, P)^x = \hat{e}(P, xP)$ and $\hat{e}(P, \tilde{P})^x = \hat{e}(P, x\tilde{P})$. Hence letting $S = xP$, we obtain $\tilde{\kappa} = \hat{e}(S, \tilde{P})$ and $\tilde{\kappa} = \hat{e}(S, \tilde{P})$. The proof of converse is also easy. □

Theorem 2 *The* ZKBm *protocol satisfies completeness, soundness and zero-knowledge against the honest Verifier.*

Proof. First, 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 Lemma 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 $\gamma = \hat{e}(Q, P)$ and $\tilde{\gamma} = \hat{e}(Q, \tilde{P})$ for random $Q \in \mathcal{G}$ to the honest Verifier. Now, observe from the above protocol that $\hat{e}(L, P) = \hat{e}(Q + hS, P)$ and that $\gamma\kappa^h = \hat{e}(Q, P)\hat{e}(S, P)^h = \hat{e}(Q, P)\hat{e}(hS, P)$. By the bilinear property of $\hat{e}$, we have $\hat{e}(Q, P)\hat{e}(hS, P) = \hat{e}(Q + hS, P)$. 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 $\gamma = \hat{e}(Q, P)$ and $\gamma = \hat{e}(Q', \tilde{P})$ 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 $Q + hS = Q' + hS'$. Now suppose that $Q = tP$, $Q' = t'P$; $S = xP$ and $S' = x'P$ for $t, t', x, x' \in \mathbb{Z}_q^*$. Then, $Q + hS = Q' + hS'$ 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. □

B Non-ID-Based Threshold Decryption Schemes

B.1 Security Notion for Threshold Cryptosystem

A (t, n)-threshold decryption scheme $\mathcal{THD}$ in the normal (non-ID-based) public key setting consists of a key 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 GK, 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.

The security notion for the threshold decryption scheme $\mathcal{THD}$ used in this paper is the indistinguishability under adaptive chosen ciphertext attack, which we call "THD-IND-CCA" and can be found in [15].

Definition 4 (THD-IND-CCA) Let $\mathsf{A}^{\mathsf{CCA}}$ be an attacker that defeats the security of the scheme $\mathcal{THD}$ in the sense of THD-IND-CCA. 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 an attacker $\mathsf{A}^{\mathsf{CCA}}$ interacts with the "Challenger".

> **Phase 1**: $\mathsf{A}^{\mathsf{CCA}}$ corrupts a fixed subset of $t-1$ servers.
>
> **Phase 2**: The Challenger runs the key generation algorithm taking a security parameter k. The Challenger gives $\mathsf{A}^{\mathsf{CCA}}$ 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 $\mathsf{A}^{\mathsf{CCA}}$.
>
> **Phase 3**: $\mathsf{A}^{\mathsf{CCA}}$ adaptively interacts with the uncorrupted decryption servers, submitting ciphertexts and obtaining decryption shares.
>
> **Phase 4**: $\mathsf{A}^{\mathsf{CCA}}$ 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 $\mathsf{A}^{\mathsf{CCA}}$.
>
> **Phase 5**: $\mathsf{A}^{\mathsf{CCA}}$ 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**: $\mathsf{A}^{\mathsf{CCA}}$ outputs a guess $\tilde{\beta} \in \{0,1\}$.

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

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

We denote by $\mathbf{Succ}^{\mathrm{THD-IND-CCA}}_{\mathcal{THD}}(t_{CCA}, q_D)$ 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_D decryption share generation queries. Note that the running time and the number of queries are all polynomial in the security parameter k. We say that the threshold decryption scheme $\mathcal{THD}$ is secure in the sense of THD-IND-CCA if $\mathbf{Succ}^{\mathrm{THD-IND-CCA}}_{\mathcal{THD}}(t_{CCA}, q_D)$ is negligible in k.

C Proof of Lemma 3

Lemma 3 *Suppose that an IDTHD-IND-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 each of the random oracles. Using this attacker as a subroutine, we can construct an THD-IND-CCA attacker for the* `ThdBm` *scheme, whose advantage including running time t_{CCA}, decryption share generation queries q'_D and random oracle queries q'_{H_1}, q'_{H_2} and q'_{H_4} are as follows.*

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

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

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 IDTHD-IND-CCA security of the `IdThdBm` scheme. We assume that $\mathsf{A}^{\mathsf{CCA}}$ has access to the common parameter $cp_{\texttt{IdThdBm}} = (\mathcal{G}, q, P, \hat{e}, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3, \mathsf{H}_4, Y_{\mathrm{PKG}})$ 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 $cp_{\texttt{ThdBm}} = (\mathcal{G}, q, P, \hat{e}, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_4, Y, Q)$ 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 security of the `ThdBm` scheme in the sense of THD-IND-CCA.

- 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 $\bar{\beta} \in \{0,1\}$ 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

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

- Game $\mathbf{G_1}$: First, we deal with the simulation of $\mathsf{A}^{\mathsf{CCA}}$'s view in Phase 1 of the real attack game as follows.

  We replace Y_{PKG} in $\mathsf{A}^{\mathsf{CCA}}$'s common parameter $cp_{\texttt{IdThdBm}}$ by Y in $\mathsf{B}^{\mathsf{CCA}}$'s common parameter $cp_{\texttt{ThdBm}}$, that is, we set $Y_{\mathrm{PKG}} = Y$. We also replace $\mathsf{A}^{\mathsf{CCA}}$'s decryption servers by $\mathsf{B}^{\mathsf{CCA}}$'s decryption servers.

  Then, we randomly choose an index μ from the range $[1, q_{\mathsf{H}_3}]$ where q_{H_3} is the maximum number of queries made by $\mathsf{A}^{\mathsf{CCA}}$ to the random oracle H_3. We denote the μ-th private key extraction query by $\texttt{ID}_\mu$.

  Now, we simulate $\mathsf{A}^{\mathsf{CCA}}$'s random oracle H_3, which can be queried at any time during the attack. To respond to queries to H_3, we maintain an "input-output" list $\mathsf{H}_3\mathsf{List}$ whose entry is of the form $\langle(\texttt{ID}, Q_{\texttt{ID}}), \tau, c)\rangle$ as explained below. Whenever H_3 is queried at `ID`, we perform the following.

  - If the query `ID` exists in the entry $\langle(\texttt{ID}, Q_{\texttt{ID}}), \tau, c)\rangle$, we respond with $Q_{\texttt{ID}}$.
  - Else we do the following.
    - If $\texttt{ID} = \texttt{ID}_\mu$ then we set $Q_{\texttt{ID}} = Q$, where Q is from $\mathsf{B}^{\mathsf{CCA}}$'s common parameter $cp_{\texttt{ThdBm}}$.
      - Set $c = 1$; Return $Q_{\texttt{ID}}$.
    - Else ($\texttt{ID} \neq \texttt{ID}_\mu$) do the following.
      - Choose τ uniformly at random from $\mathbb{Z}_q^*$.
      - Compute $Q_{\texttt{ID}} = \tau P$; Set $c = 0$; Return $Q_{\texttt{ID}}$.

  We continue to deal with the simulation of $\mathsf{A}^{\mathsf{CCA}}$'s view in other phases of the real attack game.

  If $\mathsf{A}^{\mathsf{CCA}}$ issues a private key extraction query `ID` in Phase 2 then we first run the simulator for the random oracle H_3 described above to obtain the corresponding entry $\langle(\texttt{ID}, Q_{\texttt{ID}}), \tau, c)\rangle$ in $\mathsf{H}_3\mathsf{List}$. If $c = 1$ then we output "*abort*" and make $\mathsf{B}^{\mathsf{CCA}}$ terminate the game, otherwise we compute $D_{\mathtt{ID}} = \tau Y_{\mathrm{PKG}}$ and respond with it. Note here that, $D_{\mathtt{ID}} = \tau Y_{\mathrm{PKG}} = \tau Y = \tau x P = x \tau P = x Q_{\mathtt{ID}}$. Hence $\mathsf{A}^{\mathsf{CCA}}$'s view in this step of the game is identical to that in the real attack as long as $\mathsf{B}^{\mathsf{CCA}}$ doesn't abort.

If $\mathsf{A}^{\mathsf{CCA}}$ corrupts $t-1$ decryption servers in Phase 3, that is, it obtains private keys $\{S_i\}_{1 \le i \le t-1}$ of corrupted decryption servers, we give them to $\mathsf{B}^{\mathsf{CCA}}$.

Next, if $\mathsf{A}^{\mathsf{CCA}}$ submits a target identity $\mathtt{ID}^*$ in Phase 4 then we run the simulator for the random oracle H_3 to obtain the corresponding entry $\langle (\mathtt{ID}^*, Q_{\mathtt{ID}^*}), \tau, c) \rangle$ in $\mathsf{H}_3\mathsf{List}$. This time, if $c = 0$ then we output "*abort*" and make $\mathsf{B}^{\mathsf{CCA}}$ terminate the game, otherwise we continue to deal with the next game. Note that if $\mathsf{B}^{\mathsf{CCA}}$ dose not terminate, we have $Q_{\mathtt{ID}^*} = Q_{\mathtt{ID}_\mu} = Q$. Since Q was randomly chosen from $\mathcal{G}$, $\mathsf{A}^{\mathsf{CCA}}$'s view in this step of the game is identical to that in the real attack as long as $\mathsf{B}^{\mathsf{CCA}}$ doesn't abort.

In Phase 5, if $\mathsf{A}^{\mathsf{CCA}}$ issues private key extraction queries $\mathtt{ID} \neq \mathtt{ID}^*$, we respond to these queries as we did in the simulation for Phase 2 above. Note in this phase that $\mathsf{A}^{\mathsf{CCA}}$ submits a ciphertext (i, C) to the i-th uncorrupted decryption server and obtains a corresponding decryption share. (Remember that $\mathsf{B}^{\mathsf{CCA}}$'s decryption servers were provided as its decryption servers to $\mathsf{A}^{\mathsf{CCA}}$.)

If $\mathsf{A}^{\mathsf{CCA}}$ submits a pair of two equal-length plaintexts (m_0, m_1) in Phase 6, we give it to $\mathsf{B}^{\mathsf{CCA}}$, then $\mathsf{B}^{\mathsf{CCA}}$ uses (m_0, m_1) as its plaintexts-pair to be challenged. After $\mathsf{B}^{\mathsf{CCA}}$ queries (m_0, m_1) to its encryption oracle, it obtains a target ciphertext C^* such that $C^* = (U, V, W) = (rP, m_\beta \oplus \mathsf{H}_1(\hat{e}(Q, Y)^r), r\mathsf{H}_2(U, V))$, where r and β are chosen uniformly at random from $\mathbb{Z}_q^*$ and $\{0, 1\}$ respectively.

Now, we return C^* to $\mathsf{A}^{\mathsf{CCA}}$ as its target ciphertext. Note here that

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

Hence, C^* is exactly the same as the target ciphertext that $\mathsf{A}^{\mathsf{CCA}}$ acquires in Phase 6 of the real attack.

In Phase 7, we answer $\mathsf{A}^{\mathsf{CCA}}$'s queries in the same way we did in the simulation for Phase 5. 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 $\tilde{\beta}$ in Phase 8, we give it to $\mathsf{B}^{\mathsf{CCA}}$.

Now we quantitatively analyze the simulations above. Note that if $\mathsf{B}^{\mathsf{CCA}}$ does not abort in the simulation of Phases 2, 3, 5 and 7, $\mathsf{A}^{\mathsf{CCA}}$'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

$$\Pr[E_1] - \frac{1}{2} \ge \epsilon \Big(\Pr[E_0] - \frac{1}{2} \Big),$$

where ϵ denotes the probability that $\mathsf{B}^{\mathsf{CCA}}$ does not abort in Phases 2, 3, 5 and 7, that is, the chance that $\mathtt{ID}_\mu = \mathtt{ID}^*$. Since μ has been uniformly chosen from $[1, q_{\mathsf{H}_3}]$, we have $\epsilon = \frac{1}{q_{\mathsf{H}_3}}$.

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

$$\mathbf{Succ}_{\mathtt{ThdBm}, \mathsf{B}^{\mathsf{CCA}}}^{\mathrm{THD-IND-CCA}}(k) \ge \frac{1}{q_{\mathsf{H}_3}} \mathbf{Succ}_{\mathtt{IdThdBm}, \mathsf{A}^{\mathsf{CCA}}}^{\mathrm{IDTHD-IND-CCA}}(k).$$

Finally note that the running time t_{CCA} of an arbitrary THD-IND-CCA attacker for the scheme `ThdBm` is lower-bounded by $t_{IDCCA} + \max(q_E, q_{\mathsf{H}_3})O(k^3)$. Note also that the number of queries to the random oracles H_1, H_2, 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. □

D Proof of Lemma 4

Lemma 4 *Suppose that an THD-IND-CCA attacker for the* `ThdBm` *scheme issues up to* q_D *decryption share generation queries,* q_{H_1}*,* q_{H_2} *and* q_{H_4} *queries to each of the random oracles. Using this attacker as a subroutine, we can construct a BDH attacker for the group* $\mathcal{G}$*, whose advantage including running time* t_{BDH} *is bounded as follows.*

$$\frac{1}{2}\mathbf{Succ}_{\mathtt{ThdBm}}^{\mathrm{THD-IND-CCA}}(t, q_{\mathsf{H}_1}, q_{\mathsf{H}_2}, q_{\mathsf{H}_4} q_D) \le \mathbf{Succ}_{\mathcal{G}}^{\mathrm{BDH}}(t_{\mathrm{BDH}}) + \frac{q_D + q_D q_{\mathsf{H}_4}}{2^k},$$

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

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 `ThdBm` and the BDH problem.

Let $\mathsf{B}^{\mathsf{CCA}}$ be an attacker that defeats the THD-IND-CCA security of the `ThdBm` scheme. Let $\mathsf{A}^{\mathsf{BDH}}$ be an attacker for the BDH problem given $(\mathcal{G}, q, \hat{e}, P, aP, bP, cP)$, as defined in Section 6.1. We provide a same security parameter k as input to both $\mathsf{B}^{\mathsf{CCA}}$ and $\mathsf{A}^{\mathsf{BDH}}$.

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 $cp_{\mathtt{ThdBm}} = (\mathcal{G}, q, \hat{e}, P, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_4, Y, Q)$ 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.

  $$U^* = r^*P, V^* = H_1^* \oplus m_\beta, \text{ and } W^* = r^* H_2^*,$$

  where $\kappa^* = \hat{e}(Q, Y)^{r^*}$ for random $r^* \in \mathbb{Z}_q^*$, $H_1^* = \mathsf{H}_1(\kappa^*)$ and $H_2^* = \mathsf{H}_2(U^*, V^*)$.

  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

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

- 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 $y_i = \hat{e}(Q_{\text{BDH}}, Y_{\text{BDH}})^{c_{i0}^{\Phi'}} \prod_{j=1}^{t-1} \hat{e}(S_j, P)^{c_{ij}^{\Phi'}}$, 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_1^+ uniformly at random from $\{0,1\}^l$, replace H_1^* by H_1^+ and V^* by $V^+ = H_1^+ \oplus m_\beta$. Accordingly, whenever the random oracle H_1 is queried at κ^+, we respond with H_1^+.

  Summing up, we obtain a new challenge ciphertext denoted by C_+^* such that $C_+^* = (U^*, V^+, W^*)$, where $V^+ = H_1^+ \oplus m_\beta$ and $H_1^+ = \mathsf{H}_1(\kappa^+)$ 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_1. That is, if H_1 is queried at κ^+, we do not respond with H_1^+ any more but respond with an answer from the random oracle H_1 instead. We assume that this rule applies to all the forthcoming games.

  By the above rule, κ^+ and H_1^+ 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_{1_2} denotes the event that, in game $\mathbf{G_2}$, H_1 is queried at κ^*. We will use the same notation AskH_{1_i} to denote such events in all other games.

  Now, we have

  $$|\Pr[E_2] - \Pr[E_1]| \le \Pr[\mathsf{AskH}_{1_2}].$$

- 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 $V^+ (= H_1^+ \oplus m_\beta = \mathsf{H}_1(\kappa^+) \oplus m_\beta)$ 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_2 as follows. Whenever H_2 is queried at (aP, V^+), we compute $H_2^+ = s^+P$ and respond with H_2^+. Namely, we set $H_2^+ = \mathsf{H}_2(aP, V^+)$.

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

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

  where $V^+ = \mathsf{H}_1(\hat{e}(P, P)^{abc}) \oplus m_\beta$. Moreover, we have $\mathsf{H}_2(U_{\text{BDH}}, V_{\text{BDH}}) = H_2^+ = s^+P$.

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 $\hat{e}(P, W_{\mathrm{BDH}}) = \hat{e}(U_{\mathrm{BDH}}, H_2^+)$ by the construction of H_2^+ 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

$$\Pr[\mathsf{AskH1}_3] = \Pr[\mathsf{AskH1}_2].$$

- Game $\mathbf{G_4}$: In this game, we modify the random oracle H_2. Note that we have already dealt with the simulation of the random oracles H_2 appeared in the target ciphertext C^*_{BDH}, namely, the case when H_2 is queried at $(U_{\mathrm{BDH}}, V_{\mathrm{BDH}})$. In the following, we deal with the rest of simulation.

  Whenever H_2 is queried at $(U, V) \neq (U_{\mathrm{BDH}}, V_{\mathrm{BDH}})$, we choose s uniformly at random from $\mathbb{Z}_q^*$, computes $H_2 = sY$ and respond with H_2. Let $\mathsf{H_2List}$ be a list of all "input-output" pairs of the random oracle H_2. Specifically, $\mathsf{H_2List}$ consists of the pairs $\langle (U, V), H_2 \rangle$ where $H_2 = \mathsf{H}_2(U, V) = sY$. Notice that this list grows as $\mathsf{B}^{\mathsf{CCA}}$'s attack proceeds.

  Because H_2 is assumed to be a random oracle, the above generation of the outputs of H_2 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

$$\Pr[\mathsf{AskH1}_4] = \Pr[\mathsf{AskH1}_3].$$

  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 $H_2 = \mathsf{H}_2(U, V)$ has *not* been queried. If C is a valid ciphertext while $\mathsf{H}_2(U, V)$ 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 $\hat{e}(P, W) = \hat{e}(U, H_2)$. However, since we have assumed that H_2 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_2 is uniformly distributed in $\mathcal{G}$. Adding up all the decryption queries up to q_D, we have

$$|\Pr[\mathsf{AskH1}_5] - \Pr[\mathsf{AskH1}_4]| \leq \frac{q_D}{2^k}.$$

- 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 $\mathsf{H}_2(U, V)$ has not been queried are excluded in this game since it was already dealt with in the previous game. Hence, we assume that $\mathsf{H}_2(U, V)$ 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 $\langle (U, V), H_2 \rangle$ from $\mathsf{H_2List}$.
  - If $\hat{e}(P, W) = \hat{e}(U, H_2)$
    - 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 $\kappa_i = \kappa^{c^{\Phi'}_{i0}} \prod_{j=1}^{t-1} \hat{e}(S_j, U)^{c^{\Phi'}_{ij}}$.
    ∗ Return κ_i.
- Else reject C.

Note in the above construction that

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

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_1 and H_2 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

$$|\Pr[\mathsf{AskH1}_6] - \Pr[\mathsf{AskH1}_5]| \leq \frac{q_D q_{\mathsf{H}_4}}{2^k}.$$

Now, recall that the target ciphertext used so far is C^*_{BDH} that constructed in Game $\mathbf{G_3}$. Accordingly, $\mathsf{AskH1}_6$ denotes an event that the BDH key $\hat{e}(P, P)^{abc}$ has been queried to the random oracle H_1. 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_1. That is, we have

$$\Pr[\mathsf{AskH1}_6] \leq \mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G}, \mathsf{A}^{\mathsf{BDH}}}(k).$$

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

$$\begin{aligned}\frac{1}{2}\mathbf{Succ}^{\mathrm{THD-IND-CCA}}_{\mathtt{ThdBm}, \mathsf{B}^{\mathsf{CCA}}}(k) &= |\Pr[E_0] - \Pr[E_2]| \leq \Pr[\mathsf{AskH1}_2] \leq \Pr[\mathsf{AskH1}_5] + \frac{q_D}{2^k} \\ &\leq \frac{q_D}{2^k} + \Pr[\mathsf{AskH1}_6] + \frac{q_D q_{\mathsf{H}_4}}{2^k} \leq \frac{q_D + q_D q_{\mathsf{H}_4}}{2^k} + \mathbf{Succ}^{\mathrm{BDH}}_{\mathcal{G}, \mathsf{A}^{\mathsf{BDH}}}(k).\end{aligned}$$

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. □