Efficient CCA-secure Threshold Public-Key Encryption Scheme

Xi-Jun Lin *and Lin Sun †

November 13, 2013

Abstract: In threshold public-key encryption, the decryption key is divided into n shares, each one of which is given to a different decryption user in order to avoid single points of failure. In this study, we propose a simple and efficient non-interactive threshold public-key encryption scheme by using the hashed Diffie-Hellman assumption in bilinear groups. Compared with the other related constructions, the proposed scheme is more efficient.

Key words: Threshold public-key encryption; Chosen-ciphertext security; Hashed Diffie-Hellman assumption; CCA-secure

1 Introduction

In a threshold public-key encryption scheme, the private key corresponding to a public key is shared among a set of n decryption users. In such a scheme, a message is encrypted and sent to a group of decryption users, in such a way that the cooperation of at least t of them (where t is the threshold) is necessary in order to recover the original message. Moreover, no information about the message is leaked, even if the number of the corrupted users is up to $t-1$. Such schemes have many applications in situations where one cannot fully trust a unique person, but possibly a pool of individuals, such as electronic voting, electronic auctions, key-escrow, etc.

In a non-interactive threshold public-key encryption scheme, no communication is needed amongst the decryption users performing the partial decryptions. Furthermore, such schemes are often required to be robust in that if threshold decryption of a valid ciphertext fails, the combiner can identify the decryption users who supply invalid partial decryption shares. Recently, we have seen many studies of such schemes in the crypto/security community [1, 3, 5, 7, 8].

In this study, we propose a more efficient non-interactive threshold public-key encryption scheme than the other related constructions, and the proposed scheme is proved to be CCA-secure under the hashed Diffie-Hellman (HDH) assumption in bilinear groups [2, 5].

In the proposed scheme, the decryption user needs to verify the ciphertext C before attempting to generate its partial decryption share. This validity check which is performed using two exponentiations in group $\mathcal{G}$ is more efficient than that in the other related construc-

*X.J.Lin is with the Department of Computer Sciences and Technology, Ocean University of China. Qingdao 266100, P.R.China. email: linxj77@163.com

†L. Sun is with the College of Liberal Arts, Qingdao University. Qingdao 266071, P.R.China. email: sunlin9@126.com

tions in which the pairing computation is employed. Moreover, each of partial decryption share will be verified before running *Combine* algorithm.

The rest of this paper is organized as follows: After recalling the relevant technical definitions in the next section, the definitions and the security models of the threshold public-key encryption scheme are given in Section 3. In Section 4 and Section 5, we propose a more efficient non-interactive threshold public-key encryption scheme based on HDH assumption in bilinear groups, and then prove its security. Furthermore, the comparisons with the related constructions are given in Section 6 which is followed by the last section to conclude our works.

2 Preliminaries

2.1 Bilinear Pairing

Let $\mathcal{G}$ be an additive group of prime order p, $\mathcal{F}$ be a multiplicative group of the same order. Bilinear pairing is a map $\hat{e} : \mathcal{G} \times \mathcal{G} \to \mathcal{F}$ which satisfies the following properties:

- Bilinearity: given any $g, h \in \mathcal{G}$ and $a, b \in \mathbb{Z}_p^*$, we have $\hat{e}(g^a, g^b) = \hat{e}(g, g)^{ab} = \hat{e}(g^{ab}, g), etc.$
- Non-Degeneracy: There exists a $g \in \mathcal{G}$ such that $\hat{e}(g, g) \neq 1$.
- Computability: $\hat{e}(g, h)$ can be computed in polynomial time.

2.2 HDH Assumption

Let $\mathcal{G}$ be a group of prime order p and g be a generator of $\mathcal{G}$. Let H be a one-way hash function $H : \mathcal{G} \to \{0, 1\}^l$. Let $\mathcal{A}$ be an adversary. We define HDH advantage of $\mathcal{A}$ against $\mathcal{G}$ at a security parameter λ as

$$Adv_{\mathcal{A},\mathcal{G}}^{HDH}(\lambda) = |Pr[\mathcal{A}(g, g^a, g^b, H(g^{ab})) = 1] - Pr[\mathcal{A}(g, g^a, g^b, T \in_R \{0, 1\}^l) = 1]|.$$

The HDH assumption is that for every polynomial-time adversary $\mathcal{A}$, the function $Adv_{\mathcal{A},\mathcal{G}}^{HDH}(\lambda)$ is negligible.

2.3 Lagrange Interpolation

Let $f(x) = \sum_{j=0}^{t-1} a_j x^j$ be a polynomial over $\mathbb{Z}_p$ with degree $t-1$ where p is a prime, and let $(x_1, f(x_1)), (x_2, f(x_2)), \cdots, (x_t, f(x_t))$ be t distinct points over $f(x)$.

Then, given $(x_1, f(x_1)), (x_2, f(x_2)), \cdots, (x_t, f(x_t))$, $f(x)$ can be reconstructed as follows

$$f(x) = f(x_1)\lambda_{x_1}^x + f(x_2)\lambda_{x_2}^x + \cdots + f(x_t)\lambda_{x_t}^x,$$

where

$$\lambda_{x_j}^x = \frac{(x - x_1) \cdots (x - x_{j-1})(x - x_{j+1}) \cdots (x - x_t)}{(x_j - x_1) \cdots (x_j - x_{j-1})(x_j - x_{j+1}) \cdots (x_j - x_t)},$$

for any $1 \leq j \leq t$.

3 Definitions

We follow the notation of CCA-secure threshold public-key encryption scheme from [5]. A threshold public-key encryption scheme consists of six algorithms.

1. **Setup**(n, t, λ): Takes as input the number of decryption users n, a threshold t , where $1 \leq t \leq n$, a security parameter $\lambda \in \mathbb{Z}$. It outputs a triple (PK, SK, VK), where PK is the public key, $SK = (SK_1, \cdots, SK_n)$ is a vector of n secret keys and $VK = (VK_1, \cdots, VK_n)$ is the corresponding vector of verification keys. The verification key VK_i is used to check the validity of partial decryption shares generated by using SK_i. The secret key SK_i is secretly given to the ith user, for $i = 1, \cdots, n$.

2. **Encrypt**(PK, M): Takes as input the public key PK and a message M to be encrypted. It outputs ciphertext C.

3. **ValidateCT**(PK, C): Takes as input the public key PK, and ciphertext C. It checks whether C is a valid ciphertext with respect to PK.

4. **ShareDecrypt**(PK, i, SK_i, C): Takes as input the public key PK, ciphertext C, a decryption user i and its secret key SK_i. It outputs a partial decryption share $\sigma_i = (i, \vartheta_i)$, or a special symbol $(i, \perp)$ if C is invalid.

5. **ShareVerify**(PK, VK_i, C, σ_i): Takes as input the public key PK, the verification key VK_i, as well as ciphertext C and partial decryption share σ_i. It checks whether σ_i is a valid partial decryption share with respect to VK_i and C.

6. **Combine**(PK, VK, C, Ω): Takes as input the public key PK, the verification key VK, as well as ciphertext C, and $\Omega = (\sigma_1, \cdots, \sigma_t)$ a list of t partial decryption shares. It outputs plaintext M or $\perp$.

We require, for all ciphertext C, **ShareVerify**$(PK, VK_i, C,$ **ShareDecrypt**$(PK, i, SK_i, C)) = valid$. In addition, let $\Omega = (\sigma_1, \cdots, \sigma_t)$ be t distinct valid decryption shares of C, where $C =$ **Encrypt**(PK, M), then we require **Combine**$(PK, VK, C, \Omega) = M$.

Security against chosen ciphertext attack is defined using the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{R}$, and both of them are given as input the system parameters $n, t, \lambda \in \mathbb{N}$ with $t \leq n$.

- **Init**: The adversary $\mathcal{A}$ outputs a set $S \subset \{1, \cdots, n\}$ of $t - 1$ decryption users to corrupt.

- **Setup**: The challenger $\mathcal{R}$ runs **Setup**(n, t, λ) algorithm to obtain a triple (PK, SK, VK), where $SK = (SK_1, \cdots, SK_n)$ and $VK = (VK_1, \cdots, VK_n)$. It gives PK, VK and all (j, SK_j) (where $j \in S$) to adversary $\mathcal{A}$.

- **Phase 1**: Adversary $\mathcal{A}$ adaptively issues **ShareDecrypt** queries with (i, C), where $i \in \{1, \cdots, n\}$ and $C \in \{0, 1\}^*$.

  Challenger $\mathcal{R}$ runs the **ShareDecrypt** algorithm using C, SK_i to get σ_i , and gives σ_i to adversary $\mathcal{A}$.

- **Challenge**: Adversary $\mathcal{A}$ outputs two equal length messages M_0 and M_1. Challenger $\mathcal{R}$ picks a random bit $\delta \in \{0,1\}$, and sends $C^* = \textbf{Encrypt}(PK, M_\delta)$ to adversary $\mathcal{A}$.
- **Phase 2**: Adversary $\mathcal{A}$ makes further queries as in **Phase 1** but is not allowed to make **ShareDecrypt** queries on C^*.
- **Guess**: Finally, adversary $\mathcal{A}$ outputs a guess $\delta' \in \{0,1\}$ and wins the game if $\delta = \delta'$.

4 The Proposed Scheme

4.1 Construction

- **Setup**(n, t, λ): The trust center generates the system parameters $(p, \mathcal{G}, \mathcal{F}, \hat{e})$ by running the group generator algorithm. It then does the following:
  1. Pick two generators $g, X \in_R \mathcal{G}$.
  2. Pick two hash functions H_1 and H_2, where $H_1 : \{0,1\}^l \times \mathcal{G} \times \mathcal{G} \to \mathbb{Z}_p^*$ is a secure hash function and $H_2 : \mathcal{G} \to \{0,1\}^l$ is a random instance of a hash function such that the HDH assumption holds in bilinear groups.
  3. Pick a random polynomial $f(x) = a + \sum_{j=1}^{t-1} a_j x^j$ with degree $t-1$ (where $a, a_1, \cdots, a_{t-1} \in_R \mathbb{Z}_p^*$, t is the value of threshold).
  4. Compute $SK_i = f(i)$ and $VK_i = X^{f(i)}$, for $i = 1, \cdots, n$.
  5. Let $g_1 = g^a$.
  6. Publish system parameters $PK = (p, \mathcal{G}, \mathcal{F}, \hat{e}, g, g_1, X, H_1, H_2)$ and verification key $VK = (VK_1, \cdots, VK_n)$. Secret key SK_i is given to user i privately, for $i = 1, \cdots, n$.
- **Encrypt**(PK, M): To encrypt $M \in \{0,1\}^l$, this algorithm picks $k, r \in_R \mathbb{Z}_p^*$, and computes

$$C_0 = M \oplus H_2(g_1^k),$$

$$C_1 = g^k, C_2 = g^r,$$

$$\beta = kH_1(C_0, C_1, C_2) + r \pmod{p-1}.$$

  The output is $C = (C_0, C_1, C_2, \beta)$.
- **ValidateCT**(PK, C): To validate ciphertext $C = (C_0, C_1, C_2, \beta)$, this algorithm checks whether

$$g^\beta = C_2 \cdot C_1^{H_1(C_0, C_1, C_2)}.$$

- **ShareDecrypt**(PK, i, SK_i, C): Decryption user i uses its secret key $SK_i = f(i)$ to partially decrypt ciphertext $C = (C_0, C_1, C_2, \beta)$ as follows:
  1. Run **ValidateCT**(PK, C) algorithm to check whether or not C is a valid ciphertext. If the verification fails, it outputs $\sigma_i = (i, \perp)$;
  2. Otherwise, compute $\vartheta_i = C_1^{f(i)}$ and output partial decryption share $\sigma_i = (i, \vartheta_i)$.

- **ShareVerify**(PK, VK_i, C, σ_i): To verify a partial decryption share σ_i with respect to ciphertext $C = (C_0, C_1, C_2, \beta)$ under verification key VK_i, this algorithm firstly runs **ValidateCT**$(PK,\ C)$ to check whether C is a valid ciphertext. If C and σ_i are well formed, it checks whether the following equation holds:

$$\hat{e}(\vartheta_i, X) = \hat{e}(C_1, VK_i).$$

- **Combine**$(PK, VK, C, \{\sigma_1, \cdots, \sigma_t\})$: To decrypt ciphertext $C = (C_0, C_1, C_2, \beta)$ using the partial decryption shares $\{\sigma_1, \cdots, \sigma_t\}$, this algorithm firstly checks whether $\sigma_i = (i, \vartheta_i)$ is valid by running **ShareVerify**(PK, VK_i, C, σ_i), for $i = 1, \cdots, t$. Then, it performs as follows:

  1. Determine the Lagrange coefficients $(\lambda_1^0, \lambda_2^0, \cdots, \lambda_t^0) \in \mathbb{Z}_q^t$, and then compute
  $$\mu = \prod_{i=1}^{t} (\vartheta_i)^{\lambda_i^0}.$$
  2. Compute and output $M = C_0 \oplus H_2(\mu)$ to decrypt $C = (C_0, C_1, C_2, \beta)$ with μ.

4.2 Correctness

If the ciphertext $C = (C_0, C_1, C_2, \beta)$ and partial decryptions $\{\sigma_1, \cdots, \sigma_t\}$ are valid, the **Combine** algorithm will output the correct plaintext.

$$\begin{array}{rcl}
\mu & = & \prod_{i=1}^{t} (\vartheta_i)^{\lambda_i^0} \\
& = & \prod_{i=1}^{t} (C_1^{f(i)})^{\lambda_i^0} \\
& = & C_1^{\sum_{i=1}^{t} f(i)\lambda_i^0} \\
& = & C_1^{f(0)} \\
& = & C_1^{a}, \\
& & \\
C_0 \oplus H_2(\mu) & = & C_0 \oplus H_2(C_1^a) \\
& = & C_0 \oplus H_2((g^k)^a) \\
& = & C_0 \oplus H_2(g_1^k) \\
& = & (M \oplus H_2(g_1^k)) \oplus H_2(g_1^k) \\
& = & M.
\end{array}$$

5 Security

Theorem 1 *Assume that H_1 is a random oracle and H_2 is a random instance of a hash function such that the HDH assumption holds in bilinear groups. Suppose that there exists a polynomial time adversary $\mathcal{A}$ that breaks chosen-ciphertext security of the proposed scheme with non-negligible advantage. We show that there exists an algorithm $\mathcal{B}$ that runs in polynomial time and runs adversary $\mathcal{A}$ as a subroutine to break the HDH assumption in bilinear groups.*

Proof: The algorithm $\mathcal{B}$ is given group parameters $(p, g, \mathcal{G}, \mathcal{F}, \hat{e})$ and a random HDH instance tuple (g, g^a, g^b, T, H_2), where T is equal to $H_2(g^{ab})$ or a random element in $\{0, 1\}^l$.

If T is equal to $H_2(g^{ab})$, $\mathcal{B}$ outputs 1; otherwise, it outputs 0. Set $g_1 = g^a$. $\mathcal{B}$ performs by interacting with the adversary $\mathcal{A}$ in the following game:

- **Init**: The adversary $\mathcal{A}$ chooses a set S of $t-1$ decryption users that it wants to corrupt. Without loss of generality, we let $S = \{1, \cdots, t-1\} \subset \{1, \cdots, n\}$.
- **Setup**: $\mathcal{B}$ does as follows:
  1. Pick $x \in_R \mathbb{Z}_p^*$ and compute $X = g^x$, and then give $(p, \mathcal{G}, \mathcal{F}, \hat{e}, g, g_1, X)$ to $\mathcal{A}$ as the system parameters. Two lists H_1-list and H_2-list are maintained by $\mathcal{B}$ to answer H_1 oracle queries and H_2 oracle queries, respectively.
  2. Pick integers $a_i \in_R \mathbb{Z}_p^*$ where $i = 1, \cdots, t-1$. Note that there exists an interpolation polynomial $f(x)$ with degree $t-1$, such that $f(0) = a$ and $f(i) = a_i$. However, $\mathcal{B}$ does not know $f(x)$ since it does not know a. $\mathcal{B}$ gives the $t-1$ secret keys $SK_i = f(i) = a_i$ to $\mathcal{A}$.
  3. Construct the verification key $VK = (VK_1, \cdots, VK_n)$ as follows:
     (a) For $i \in S$, $VK_i = X^{a_i}$ since $f(i) = a_i$ which is known to $\mathcal{B}$.
     (b) For $i \notin S$, $\mathcal{B}$ computes the Lagrange coefficients $\lambda_0^i, \lambda_1^i, \cdots, \lambda_{t-1}^i \in \mathbb{Z}_p$, and sets $VK_i = g_1^{x\lambda_0^i} X^{a_1\lambda_1^i} \cdots X^{a_{t-1}\lambda_{t-1}^i}$. We claim that VK_i is a valid verification key of the decryption user i. To verify the correctness, we have that

$$\begin{aligned} VK_i &= g_1^{x\lambda_0^i} X^{a_1\lambda_1^i} \cdots X^{a_{t-1}\lambda_{t-1}^i} \\ &= X^{a\lambda_0^i} X^{a_1\lambda_1^i} \cdots X^{a_{t-1}\lambda_{t-1}^i} \\ &= X^{a\lambda_0^i + a_1\lambda_1^i + \cdots + a_{t-1}\lambda_{t-1}^i} \\ &= X^{f(0)\lambda_0^i + f(1)\lambda_1^i + \cdots + f(t-1)\lambda_{t-1}^i} \\ &= X^{f(i)} \end{aligned}$$

  $\mathcal{B}$ gives the verification key VK to the adversary $\mathcal{A}$.
- **Phase** 1: $\mathcal{A}$ can adaptively issue the following queries:
  - H_1-query: After receiving (C_0, C_1, C_2) from $\mathcal{A}$, $\mathcal{B}$ performs as follows:
    If there exists an item $[C_0, C_1, C_2, h_1]$ in the H_1-list with respect to (C_0, C_1, C_2), $\mathcal{B}$ responds with h_1; otherwise, $\mathcal{B}$ picks $h_1 \in_R \mathbb{Z}_p^*$, stores $[C_0, C_1, C_2, h_1]$ into the H_1-list and responds with h_1.
  - H_2-query: After receiving γ from $\mathcal{A}$, $\mathcal{B}$ performs as follows:
    If there exists an item $[\gamma, h_2]$ in the H_2-list with respect to γ, $\mathcal{B}$ responds with h_2; otherwise, $\mathcal{B}$ computes $h_2 = H_2(\gamma)$, stores $[\gamma, h_2]$ into the H_2-list and responds with h_2.
  - *ShareDecrypt*-query: $\mathcal{A}$ issues decryption queries of the form (i, C), where $C = (C_0, C_1, C_2, \beta)$ and $i \in \{1, \cdots, n\}$. For each such decryption query, $\mathcal{B}$ performs as follows:
    1. Check whether $g^\beta = C_2 \cdot C_1^{H_1(C_0, C_1, C_2)}$. If not, respond with $\sigma_i = (i, \perp)$;
    2. Otherwise, perform as follows:
       If $i \in S$, compute $\vartheta_i = C_1^{a_i}$. Then, we have that $\vartheta_i = C_1^{f(i)}$ since $f(i) = a_i$ which is known to $\mathcal{B}$.

If $i \notin S$, compute the Lagrange coefficients $\lambda_0^i, \lambda_1^i, \cdots, \lambda_{t-1}^i \in \mathbb{Z}_p$. Suppose $C_1 = g^k$. Since H_2 is a random oracle, the probability of computing $H_2(g_1^k)$ without issuing H_2-query is negligible. Then, we claim that there exists an item $[\gamma = g_1^k = g^{ak}, H_2(\gamma)]$ in the H_2-list if the equation $\hat{e}(\vartheta_i, X) = \hat{e}(C_1, VK_i)$ holds, where $\vartheta_i = \gamma^{\lambda_0^i} C_1^{\sum_{j=1}^{t-1} a_j \lambda_j^i}$. The correctness is given as follows:
For $\vartheta_i = \gamma^{\lambda_0^i} C_1^{\sum_{j=1}^{t-1} a_j \lambda_j^i}$, we have the following two equations:

$$\hat{e}(\vartheta_i, X) = \hat{e}(\gamma^{\lambda_0^i} C_1^{\sum_{j=1}^{t-1} a_j \lambda_j^i}, X)$$

$$\begin{aligned} \hat{e}(C_1, VK_i) &= \hat{e}(C_1, X^{f(i)}) \\ &= \hat{e}(C_1^{f(i)}, X) \\ &= \hat{e}(C_1^{a\lambda_0^i + \sum_{j=1}^{t-1} a_j \lambda_j^i}, X) \\ &= \hat{e}(C_1^{a\lambda_0^i} C_1^{\sum_{j=1}^{t-1} a_j \lambda_j^i}, X) \\ &= \hat{e}((g_1^k)^{\lambda_0^i} C_1^{\sum_{j=1}^{t-1} a_j \lambda_j^i}, X) \end{aligned}$$

Then, the equation $\gamma = g_1^k = g^{ak}$ holds.
$\mathcal{B}$ sends $\sigma_i = (i, \vartheta_i)$ to $\mathcal{A}$. We claim that σ_i is a valid partial decryption about $C = (C_0, C_1, C_2, \beta)$. To verify the correctness, we have that

$$\begin{aligned} \vartheta_i &= \gamma^{\lambda_0^i} C_1^{\sum_{j=1}^{t-1} a_j \lambda_j^i} \\ &= (g_1^k)^{\lambda_0^i} C_1^{\sum_{j=1}^{t-1} a_j \lambda_j^i} \\ &= (g^{ak})^{\lambda_0^i} C_1^{\sum_{j=1}^{t-1} a_j \lambda_j^i} \\ &= (g^k)^{a\lambda_0^i} C_1^{\sum_{j=1}^{t-1} a_j \lambda_j^i} \\ &= C_1^{a\lambda_0^i + \sum_{j=1}^{t-1} a_j \lambda_j^i} \\ &= C_1^{f(i)} \end{aligned}$$

- **Challenge**: $\mathcal{A}$ outputs two equal length messages M_0 and M_1 on which it wishes to be challenged. $\mathcal{B}$ picks $\delta \in_R \{0, 1\}$ and $\beta^* \in_R \mathbb{Z}_p^*$, and computes as follows:

$$C_0^* = M_\delta \oplus T, C_1^* = g^b, C_2^* = g^{\beta^*}/(g^b)^{H_1(C_0^*, C_1^*, C_2^*)}.$$

If $T = H_2(g^{ab}) = H_2((g^a)^b)$, the challenge ciphertext $C^* = (C_0^*, C_1^*, C_2^*, \beta^*)$ given to the adversary $\mathcal{A}$ is a valid ciphertext on M_δ. To verify the correctness, we have that

$$\begin{aligned} C_0^* &= M_\delta \oplus T \\ &= M_\delta \oplus H_2((g^a)^b) \\ &= M_\delta \oplus H_2(g_1^b) \end{aligned}$$

$$C_1^* = g^b$$

$$\begin{aligned} C_2^* &= g^{\beta^*}/(g^b)^{H_1(C_0^*, C_1^*, C_2^*)} \\ &= g^{\beta^* - bH_1(C_0^*, C_1^*, C_2^*)} \\ &= g^r \end{aligned}$$

where $r = \beta^* - bH_1(C_0^*, C_1^*, C_2^*)$, i.e. $\beta^* = bH_1(C_0^*, C_1^*, C_2^*) + r$.

- **Phase** 2: $\mathcal{A}$ continues to issue further decryption queries (i, C) under the constraint that $C \neq C^*$.
- **Guess**: Eventually, $\mathcal{A}$ outputs a guess bit $\delta' \in \{0, 1\}$ for δ. $\mathcal{B}$ concludes its own game by outputting a guess as follows.

  If $\delta' = \delta$, $\mathcal{B}$ outputs 1 meaning that $T = H_2(g^{ab})$; otherwise, it outputs 0 meaning $T \neq H_2(g^{ab})$.

  We can see that $\mathcal{B}$ can break the HDH assumption in bilinear groups with non-negligible advantage in polynomial time if $\mathcal{A}$ wins the game. □

6 Comparisons

The comparison with other related constructions [1, 3, 5, 7, 8] is given in Table 1, where AT09_1 and AT09_2 denote two constructions, TPKE1 and TPKE2 in [1], respectively. Let e_N denote the pairing with composite order $N = p_1p_2p_3$, which is about 50 times of that for computing e_p which denotes the pairing with prime order p [4]. Let E denote the exponentiation over finite fields which is more efficient than the pairing computation. The time of executing the *ValidateCT* algorithm in the *ShareVerify* algorithm is not counted, since the time of checking the validity of ciphertext is included in the *ShareDecrypt* algorithm.

With the comparison, we claim that the proposed scheme is more efficient than the other related constructions, especially in *Encrypt* and *ShareDecrypt*.

Table 1: Efficiency comparisons

Scheme	Setup	Encrypt	ShareDecrypt	ShareVerify	Combine
AT09_1 [1]	$(n+3)E$	$1e_p + 4E$	$2e_p + 2E$	$2e_p$	$1e_p + tE$
AT09_2 [1]	$(2n+4)E$	$7E$	$4e_p + 3E$	$4e_p$	tE
BBH06 [3]	$2nE$	$1e_p + 4E$	$2e_p + 4E$	$3e_p + 1E$	$2e_p + 2tE$
LDLK10 [7]	$1e_p + (2n+1)E$	$5E$	$2e_p + 6E$	$3e_p + 2E$	$2e_p + 2tE$
LY11 [8]	$(n+1)e_N + nE$	$4E$	$4e_N + 4E$	$2e_N + 1E$	$2e_N + 2tE$
GWWPY13 [5]	nE	$5E$	$2e_p + 3E$	$2e_p$	tE
Ours	nE	$3E$	$3E$	$2e_p$	tE

7 Conclusions

In this study, we proposed a simple and efficient non-interactive threshold public-key encryption scheme based on the HDH assumption in bilinear groups, and proved its security. Compared with the other related constructions, the proposed scheme is more efficient.

References

[1] S.Arita and K.Tsurudome. Construction of Threshold Public-Key Encryptions Through Tag-Based Encryptions. Applied Cryptography and Network Security.2009. pp: 186-200.

[2] M.Abdalla, M.Bellare and P. Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. CT-RSA'01. 2001. pp: 143-158.

[3] D.Boneh, X.Boyen and S.Halevi. Chosen Ciphertext Secure Public Key Threshold Encryption Without Random Oracles. CT-RSA'06. 2006. pp: 226-243.

[4] D.Freeman. Converting Pairing-Based Cryptosystems from Composite-order Groups to Prime-order Groups. EUROCRYPT'10. 2010. pp:44-61.

[5] Y.Gan, L.Wang, L.Wang, P.Pan and Y.Yang. Efficient Construction of CCA-Secure Threshold PKE Based on Hashed Diffie-Hellman Assumption. The Computer Journal. 2013.56(10).pp:1249-1257.

[6] E.Kiltz. Chosen-Ciphertext Secure Key-Encapsulation Based on Gap Hashed Diffie-Hellman. PKC'07. pp: 282-297.

[7] J.Lai, R.Deng, S.Liu and W.Kou. Efficient CCA-Secure PKE from Identity-Based Techniques. CT-RSA'10.2010. pp: 132-147.

[8] B.Libert and M.Yung. Adaptively Secure Noninteractive Threshold Cryptosystems. I-CALP'11.2011. pp: 588-600.

[9] C.P.Schnorr. Efficient Identification and Signatures for Smart Cards. CRYPTO'89. pp:688-689.