An Efficient Certificateless Threshold Decryption Schemes Based On Pairings

Fei Li
Department of Mathematics and Informatics, Ludong University, Yantai, China
Email: miss_lifei@163.com
Wei Gao*
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
Department of Mathematics and Informatics, Ludong University, Yantai, China
*Corresponding Author Email: sdgaowei@gmail.com
Yilei Wang
School of Information and Electrical Engineering, Ludong University, Yantai, China
Email: wangyilei2000@163.com
Xueli Wang
School of Mathematics, South China Normal University, Guangzhou, China
Email: wangxuyuyan@gmail.com

***Abstract*—As the combination of certificateless encryption and threshold cryptography, the certificateless threshold decryption (CLTD) scheme can avoid both the single point of failure in the distributed networks and the inherent key escrow problem in identity-based cryptosystem. This paper presents a CLTD schemes from pairings. We use the Shamir secret sharing method to indirectly split the private keys in the group $\mathbb{G}$ into the shares in the finite field $\mathbb{Z}_q$. Our CLTD scheme succeed in involving much less pairing computation for generating or verifying decryption shares. The decryption share supports short length, fast generation and batch verification. Sharing the private key is completed by its holder himself without needing help from KGC. The security proof is provided in the random oracle model.**

***Index Terms*—certificateless cryptography, certificateless threshold decryption, provable security, random oracle model, bilinear pairing**

I. Introduction

In ID-based cryptography [1], the user's identity, such as the e-mail address, is taken as the public key and so the certificate for certifying the public key is not needed. The secret key, calculated from the identity, is issued by a trusted authority called private key generator (PKG). The first practical ID-based encryption (IBE) scheme was proposed in 2001 by Boneh and Franklin [2], which was proved to be secure in random oracle model. Bilinear pairing [2] is the main tool to construct identity-based cryptographic primitives. An inherent problem of ID-based cryptography is the key escrow problem, i.e., the private key of a user is known to the PKG. Therefore, it seems that ID-based cryptography may only be suitable for small private networks with low security requirements.

To tackle the key escrow problem, Al-Riyami and Paterson [3] successfully proposed the concept of certificateless public key cryptography (CL-PKC). Unlike ID-based cryptography, a user's private key in a CL-PKC scheme is not generated by the Key Generation Center (KGC) alone. Instead, it is a combination of some contribution of the KGC (called partial-private-key) and some user-chosen secret, in such a way that the key escrow problem can be solved. In particular, the KGC cannot obtain the user's private-key. Meanwhile, CL-PKC schemes are not purely ID-based, and there exists an additional public-key for each user. In order to encrypt a message, one has to know both the users identity and this additional public key. This additional public key does not need to be certified by any trusted authority. The structure of the scheme ensures that the key can be verified without a certificate. In [3], two efficient constructions of certificateless encryption schemes based on bilinear pairings were proposed. There also appears some certificateless signature schemes [5]. and certificateless key exchange schemes [6]

Threshold cryptography [4], [7]–[9], including signature and encryption, increases the availability of the cryptographic operations which need involvement of secret keys, and at the same time enhances the protection against the attacker whose target is to compromise secret keys. In a threshold public-key decryption system [4], [8], the decryption key is shared among a set of n users (or servers). In such a system, a ciphertext can be decrypted only if at least t users cooperate, where t is the threshold value. However, the cooperation of less than t users cannot leak any information about the plaintext or signature. This is crucial in all the applications where one cannot fully trust a unique person, but possibly a pool of individuals.

A combination of CL-PKE with threshold cryptosystems will introduce the interesting concept of certificateless threshold decryption (CLTD) schemes. It is able to avoid both the single point of failure in the distributed networks and the inherent key escrow problem in identity-based cryptosystem. In this scheme, the power of decryption is shared among a set of decryption servers, each of

This work is partially supported by National Natural Science Foundation of China (No. 60973135, No.60970111) and Humanities and Social Science Research Project of the Ministry of Education (11YJCZH039).

doi:10.4304/jcp.7.12.2987-2996

which holds a piece of the private key associated with an entity. When given a ciphertext, which is the output of a CL-PKE algorithm, a quorum of servers can act together to decrypt it. Thus, the threshold certificateless cryptosystem remains to be secure unless the majority of servers are compromised by outside attackers. Now we review some results related to CLTD schemes. In 2007, Long and Chen first proposed the notion of CLTD schemes and constructed a CLTD scheme by extending the certificateless encryption scheme due to Al-Riyami and Paterson [3]. In 2010, Long and Chen further proposed the concept of threshold key encapsulation mechanism and construct a concrete scheme. All these results due to Long and Chen are constructed based on bilinear pairings. The main technical tools for extending the certificateless encryption scheme to the CLTD scheme come from the work of Baek and Zheng for constructing ID-based threshold decryption scheme [10]. In [10], [11], Baek and Zheng designed a suit of basic tools for threshold cryptography based on pairings. These building blocks will involve many bilinear pairing computations. However, the computation of bilinear pairing is an expensive operation [12]. So the CLTD schemes or ID-based threshold decryption schemes or ID-based threshold signcryption schemes [13] following this paradigm are not very satisfactory in terms of efficiency.

We first propose a new technique that indirectly shares a private key in the bilinear group through simply sharing an element in the finite field. This key sharing technique involves no pairings. Furthermore, with help of this technique, the algebraic property of pairing can be much better developed while the number of involved pairings is very little. In other words, for the bilinear pairing tool, its advantage of good algebraic structure for cryptography is fully utilized, while its disadvantage of expensive computation is well restricted. Concretely, this paper proposes a new certificateless threshold decryption schemes from bilinear pairings that enjoys the following advantages. In terms of computation efficiency, compared with previous works, it uses much less bilinear pairings among all the algorithms. Especially, the basic tools of secret sharing and non-interactive zero-knowledge proof based on pairings are avoided. Additionally, the ciphertext and many decryption shares can be verified in one equation consisting of two pairings, i.e. they support batch verification [14]. In terms of space efficiency, the ciphertext and decryption shares are all short. In terms of functionality, it is the holder of the private key associated with an identity rather than the Private Key Generator (PKG) to share the private key.

The rest of the paper is organized as follows. We first review some preliminaries mainly including the basic property of pairings, the computational assumption on which our scheme is (indirectly) based, and the security notion of CLTD schemes in Section 2 and Section 3. We then present the CLTD scheme in Section 4, and prove its security in Section 5. We next compare it with other CLTD schemes in Section 6. Finally, Section 8 concludes the paper.

II. Preliminaries

A. Bilinear Pairing and Complexity Assumption

This section briefly reviews the definition of bilinear pairings and the related complexity assumptions. For more details, please refer to [3], [15].

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

(1) e is bilinear, i.e. $e(aP, bP) = e(P, P)^{ab}$ for all $a, b \in \mathbb{Z}_q^*$;
(2) e is non-degenerate, i.e. $e(P, P) \neq 1$, where 1 is the identity of $\mathbb{G}_T$;
(3) e is efficiently computable.

Usually, the group $\mathbb{G}$, $\mathbb{G}_T$ and the pair $(\mathbb{G}, \mathbb{G}_T)$ are called a bilinear group, a target group and a bilinear group pair respectively.

Definition 2 (Bilinear Diffie-Hellman Problem,BDH): Let $\mathbb{G}$ and $\mathbb{G}_T$ be two groups of prime order q, P be a generator of $\mathbb{G}$ and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear pairing. The BDH problem is as follows: Given P, aP, bP, cP with uniformly random choices of $a, b, c \in \mathbb{Z}_q$, output $e(P, P)^{abc}$. An algorithm $\mathcal{A}$ has advantage ϵ in solving the BDH problem, if $Pr[\mathcal{A}(P, aP, bP, cP) = e(P, P)^{abc}] = \epsilon]$. The BDH problem is said to be (t, ϵ)-intractable if there is no algorithm to solve this problem with time less than t and advantage greater than ϵ.

Definition 3 (Quadruple Bilinear Diffie Hellman, 4-BDH): Let $\mathbb{G}$ and $\mathbb{G}_T$ be two groups of prime order q, P be a generator of $\mathbb{G}$ and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear pairing. The 4-BDH problem is as follows: given $(P, aP, bP, cP, abP, bcP, acP, dP)$, output $e(P, P)^{abcd}$. An algorithm $\mathcal{A}$ has an advantage ϵ in solving the 4-BDH problem, if $\Pr[\mathcal{A}(aP, bP, cP, abP, acP, bcP, dP) = e(P, P)^{abcd}] = \epsilon$. The 4-BDH problem is said to be (t, ϵ)-intractable if there is no algorithm to solve this problem with time less than t and advantage greater than ϵ.

III. Syntax and Security of Certificateless Threhshold Decryption Schemes

A. Syntax of CLTD Schemes

Based on the formulation work of [3], [15]–[17], we present the syntax of CLTD schemes with some slight modifications. The certificateless threshold decryption scheme consists of the following probabilistic polynomial-time algorithms:

- **Setup**. This algorithm, run by the Key Generating Center KGC to set up a certificateless system, takes as input security parameter 1^k, and returns the secret master key s for KGC and the common parameter cp for the system.

- **Set-User-Key**. This algorithm, run once by the entity A, takes as input the common parameter cp and A's identity ID_A, and outputs A's secret value x_A and public key P_A. Here note in the Al-Riyami and Paterson formulation [3], this algorithm is divided into two ones.
- **Set-Partial-Private-Key**. This algorithm, run by KGC once for the entity A, takes as input s, cp, ID_A and P_A. It returns a partial private key D_A. Here note that the partial private key must be kept secret or can be public, and the public key P_A may be included in the input or not. For example, in [3], the first certificateless encryption scheme requires D_A be kept secret and need not P_A as input, while the second one need not keep D_A secret and binds P_A and ID_A together into D_A.
- **Set-Private-Key**. This algorithm, run once by the entity A, takes cp, ID_A, P_A, D_A, x_A as input and returns the private-key S_A. Sometimes, this process is trivial, i.e., $S_A = (x_A, D_A)$.
- **Share-Private-Key**. This algorithm, usually run by the entity A, takes as input cp, ID_A, P_A, D_A, x_A, S_A and the parameters t, n for threshold sharing. It returns n private key shares $S_{A,j}$ for $1 \leq j \leq n$. To give robustness, n public verification keys $V_{A,j}$ are generated for checking the valid of decryption shares corresponding to n private key shares $S_{A,j}$ respectively. Each $S_{A,j}$ is securely transmitted to the decryption server Γ_j and all verification keys will be published.
- **Encrypt**. This algorithm takes as input cp, ID_A, P_A and the plaintext M. It returns the ciphertext C for the entity A. Here note that no third party is needed to ensure the authenticity of P_A.
- **Ciphertext-Verify**. This algorithm takes as input cp, ID_A, P_A and the ciphertext C. It returns "Valid Ciphertext" or "Invalid Ciphertext". Here note that the publicly checkable validity test for ciphertexts is necessary for almost all threshold decryption schemes [8]. Hence here we explicitly formalize this procedure.
- **Sub-Decrypt**. This algorithm takes as input cp, ID_A, P_A, the j-th private key share $S_{A,j}$, the ciphertext C. It first checks the validity of C. If C is invalid, it outputs "Invalid Ciphertext". Otherwise, it outputs the j-th decryption share δ_j.
- **Decryption-Share-Verify**. This algorithm takes as input cp, ID_A, P_A, the ciphertext C, the verification key $V_{A,j}$, and a decryption share δ_j. It first checks the validity of the ciphertext C. If C is invalid, it outputs "Invalid Ciphertext". Otherwise, it outputs "Valid Share" or "Invalid Share".
- **Combine**. This algorithm takes as input cp, ID_A, P_A, the set of verification keys $\{V_{A,j}\}_{j=1}^{n}$, the ciphertext C and a set of decryption shares $\{\delta_j\}_{j\in\Phi}$ (without loss of generality, we assume that there are at least t valid decryption shares in $\{\delta_j\}_{j\in\Phi}$). It first checks the validity of the ciphertext C. If C is invalid, it returns "Invalid Ciphertext". Otherwise, it determines t valid decryption shares and then combines them into the plaintext M. At last, it returns the plaintext M.

Remark 1: For standard certificateless encryption shcemes, there are a number of different infrastructures that can be put in place to support the distribution of the receiver's public key: (1) In the original Al-Riyami and Paterson (AP) formulation [3], if the identity ID_A and P_A is bound together, the receiver should generate their public key before generating the partial private key; Otherwise, the receiver can generate their public key at any time; (2) In the Baek, Safavi-Naini and Susilo (BSS) formulation [18], the receiver can only generate their public key after receiving the partial private key; (3) In the Lai and Kou (LK) formulation [19], the receiver can only generate their public key after completing a protocol with the key generation centre. The above formulation of ours follows the Al-Riyami and Paterson's method with or without the binding technique.

B. Adversary Model of CLTD Schemes

Next, we review the adversary model against the certificateless threshold decryption scheme [15]. We first clear up the following oracles that the adversary may have access to:

(1) Public key queries: The attacker supplies the identity ID_A of an entity A and the challenger responds with the current public key P_A. If the identity ID_A has no associated public key, then the challenger generates a public key for ID_A by running **Set-User-Key**.
(2) Replacing public key queries: The attacker supplies an entity A's identity ID_A and a public key value P'_A , and the challenger replaces the current public key P_A with the supplied value P'_A. This oracle models the attacker's ability to convince a legitimate sender to use an invalid public key. This can happen because public keys are no longer verified by a trusted third party, and a user may be given a false public key by an attacker and believe it to be correct.
(3) Partial private key queries: The attacker supplies an entity A's identity ID_A and the public key P_A, and the challenger responds with the corresponding partial private key D_A.
(4) Private key queries: The attacker supplies an entity A's identity ID_A and the public key P_A. The challenger responds with the corresponding private key S_A .
(5) Private key share queries: Let Φ be the set of the indices of corrupted servers chosen by the adversary. The attacker supplies entity A's identity ID_A and the public key P_A, the challenger responds with the private key shares $\{S_{A,j}\}_{j\in\Phi}$ and all verification keys $\{V_{A,j}\}_{j=1}^{n}$.
(6) Decryption share queries: The attacker supplies an entity A's identity ID_A, the public key P_A and a ciphertext C. The challenger responds with all the decryption shares of uncorrupted decryption servers.

As in [3], [15], the adversaries are divided into two classes, i.e. Type I adversaries and Type II adversaries:

- The Type I adversary $\mathcal{A}_I$ is used to capture a normal third party (i.e. anyone except the legitimate receiver or the KGC) who does not have access to the system's master-key. If the public key of the entity A has never been replaced, then $\mathcal{A}_I$ can request all above queries (1-6). If the entity A's public key has been changed from the previous value into the current one P_A, the adversary $\mathcal{A}_I$ can not request the queries (3-6) with respect to ID_A, P_A. Additionally, other restrictions with respect to the challenged target ciphertext C^* will be added in the following definition game.
- The Type II adversary $\mathcal{A}_{II}$ is used to capture the honest-but-curious key generation center who is equipped with the master-key, but is not allowed to replace public keys. Formally speaking, it have access to all the above oracles except the above oracle of replacing public keys. Of course, the oracle for partial private key queries is trivial for $\mathcal{A}_{II}$. Additionally, other restrictions with respect to the challenged targe ciphertext C^* will be added in the following definition game.

C. Security Definition of CLTD Schemes

Now we are ready to review the following game-based definition of the CLTD-IND-CCA security, i.e. the formal model describing confidentiality, for CLTD schemes [15].

- Init. $\mathcal{A}$ (Tyep I $\mathcal{A}_I$ or Type II $\mathcal{A}_{II}$) chooses to corrupt a fixed set of $t-1$ decryption servers. Let Φ be the set of the indices of these corrupted servers.
- Setup. The challenger $\mathcal{S}$ runs **Setup** algorithm and gives the resulting public parameters cp to $\mathcal{A}$.
- Queries 1. $\mathcal{A}$ can make some queries as defined above.
- Challenge. $\mathcal{A}$ determines two plaintexts M_0, M_1 of equal length, the entity A^*'s identity ID_{A^*}, the public key P_{A^*} that it wishes to be challenged on. The challenger randomly picks $\beta' \in \{0,1\}$, sets the challenge ciphertext to be
$$C^* = Encrypt(cp, ID_{A^*}, P_{A^*}, M_{\beta'}),$$
and return C^* to $\mathcal{A}$. To prevent the adversary from trivially gaining advantages, there is a natural restriction: $\mathcal{A}$ can not make the decryption query with respect to (C^*, ID_{A^*}, P_{A^*}) or the private key query on (ID_{A^*}, P_{A^*}).
- Queries 2. $\mathcal{A}$ issues more queries as in the phase Queries 1.
- Guess. Finally, $\mathcal{A}$ outputs a guess $\beta'' \in \{0,1\}$ for β'.

$\mathcal{A}$ wins the game if $\beta'' = \beta'$ and its advantage is defined to be:

$$\text{Adv}(\mathcal{A}) = |Pr[\beta'' = \beta'] - \tfrac{1}{2}|.$$

If the advantages $\text{Adv}(\mathcal{A}_I)$ and $\text{Adv}(\mathcal{A}_{II})$ for two types of adversaries $\mathcal{A}_I$ and $\mathcal{A}_{II}$ against CLTD are negligible (i.e., for all polynomial p, there exists an integer $N(p)$ such that $|f(x) \leq \frac{1}{p(x)}|$ for all $x \geq N$), then the CLTD scheme is said to be secure against threshold chosen-ciphertext attacks (denoted by CLTD-IND-CCA).

IV. Construction: CLTD-LGWW

We now describe the first construction of certificate-less threshold decryption scheme. We call this scheme "CLTD-LGWW", which consists of the following algorithms.

- **Setup**. Run by the KGC (key generating center). Given a security parameter 1^k, this algorithm does as follows.
  (1) It generates two groups $\mathbb{G}$ and $\mathbb{G}_T$ of the same prime order $q \geq 2^k$, chooses a generator P of $\mathbb{G}$ and specifies the bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$.
  (2) It specifies the hash functions $H_1 : \{0,1\}^* \times \mathbb{G} \to \mathbb{G}$, $H_2 : \mathbb{G}_T \to \{0,1\}^l$, $H_3 : \mathbb{G} \times \{0,1\}^l \times \mathbb{G} \to \mathbb{G}$, where l denotes the length of a plaintext.
  (3) It chooses its master key s uniformly at random from $\mathbb{Z}_q$ and computes its public key $P_{KGC} = sP$.
  (4) It returns a common parameters $cp = (\mathbb{G}, \mathbb{G}_T, q, P, e, P_{KGC}, H_1, H_2, H_3)$ and keeps the master key s secret.
- **Set-User-Value**. Run by an entity A. It chooses a random $x_A \in \mathbb{Z}_q$ as its secret value and set A's public key
$$P_A = (X_A, Y_A) = (x_A P, x_A P_{KGC}) = (x_A P, x_A s P).$$
- **Set-Partial-Private-Key**. Run by KGC. It takes as input the master-key s, an entity A's identity ID_A, and A's public key P_A, and output A's partial private key
$$D_A = sH_1(ID_A, P_A) = sQ_A.$$
Here note that D_A can be transmitted through a *PUBLIC* channel.
- **Set-Private-Key**. Run by an entity A. It takes as input cp, ID_A, P_A, x_A, D_A and outputs the (full) private key
$$S_A = x_A D_A = x_A s Q_A \ .$$
- **Share-Private-Key**. Here note that S_A is not directly shared. This algorithm shares the secret value x_A and transmits D_A to each party. Of course, once x_A is recovered, then $S_A = x_A D_A$ can be immediately recovered. In other words, the private key share $S_{A,j} = (x_{A,j}, D_A)$ for each decryption server Γ_j can be simply generated by the famous Shamir (t,n) threshold sharing scheme as follows.
  (1) It randomly chooses $a_1, \ldots, a_{t-1} \in \mathbb{Z}_q$, and constructs the polynomial over $\mathbb{Z}_q$
  $$F_A(x) = x_A + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}.$$
  (2) For each decryption server Γ_j, it then computes $x_{A,j} = F_A(j) \pmod q$ and the verification key $V_{A,j} = x_{A,j} P$.

(3) It secretly sends the private key share $S_{A,j} = (x_{A,j}, D_A)$ to server Γ_j for $1 \leq j \leq n$ and publishes $V_{A,1}, V_{A,2}, \ldots, V_{A,n}$.

Here we simply review the Lagrange Interpolation formula. Let Φ be a subset of t integers in $\{0, 1, 2, \ldots, n\}$ and $x_{A,0} = x_A$, then $\{x_{A,j}\}_{j\in\Phi}$ can be used to reconstruct any other $x_{A,i}$ through Lagrange interpolation:

$$x_{A,i} = F_A(i) = \sum_{j\in\Phi} L_{i,j}^{\Phi} x_{A,j},\ 0 \leq i \leq n, i \notin \Phi,$$

where $L_{i,j}^{\Phi} = \prod_{k\in\Phi, k\neq j} \frac{i-k}{j-k}$ is a lagrange efficient. Especially, we have

$$x_A = x_{A,0} = F_A(0) = \sum_{j\in\Phi} L_{0,j}^{\Phi} x_{A,j} = \sum_{j\in\Phi} x_{A,j} \prod_{k\in\Phi, k\neq j} \frac{-k}{j-k}.$$

- **Encrypt**. Given the entity A's identity ID_A, the public key $P_A = (X_A, Y_A)$, the plaintext $M \in \{0,1\}^l$, this algorithm does as follows.
  (1) It chooses the random integers r from $\mathbb{Z}_q$, and computes $Q_A = H_1(ID_A, P_A)$, $K = e(rQ_A, Y_A)$, $V = H_2(K) \oplus M$, where K plays the role of the temporary encryption key.
  (2) It computes $U = rP$, $\overline{P} = H_3(U, V, X_A)$, $\overline{U} = r\overline{P}$, where X_A is the first part of P_A.
  (3) It returns the ciphertext $C = (U, V, \overline{U})$.

  Remark 2: Here note that there is a little differenc between the encrypting algorithm of ours and that of the corresponding certificateless encryption scheme [3] where the input of the hash function H_3 does not inlclude X_A. This difference will make us able to use this random oralce to simulate the decryption shares (See Game 3 in the proof of Lemma 2).
- **Ciphertext-Verify**. Given the entity A's identity ID_A and public key $P_A = (X_A, Y_A)$, the validity of the ciphertext $C = (U, V, \overline{U})$ can be checked through the equation:

  $$e(P, \overline{U}) = e(U, \overline{P}), \text{ where} \overline{P} = H_3(U, V, X_A).$$

  Remark 3: If the encryptor extends the ciphertext $(U, V, \overline{U})$ by the non-interactive proof (c, d) :

  $$t \overset{R}{\leftarrow} \mathbb{Z}_q,\ W = tP, \overline{W} = t\overline{P}, c = H_4(U, \overline{U}, W, \overline{W}), \quad d = t - rc,$$

  then the validity of the ciphertext $C = (U, V, \overline{U})$ can also be checked through the equation:

  $$c = H_4(U, \overline{U}, dP + cU, d\overline{P} + c\overline{U}),$$

  where $\overline{P} = H_3(U, V)$, and $H_4 : \mathbb{G}^4 \to \mathbb{Z}_q$ is a cryptographic hash function. This method is the non-interactive zero knowledge proof for the equality of logarithms [8].
- **Sub-Decrypt**. Given the entity A's public key $P_A = (X_A, Y_A)$, private key share $S_{A,j} = (x_{A,j}, D_A)$ and a ciphertext $C = (U, V, \overline{U})$, the decryption server Γ_j generates its decryption share δ_j as follows.
  (1) It verifies the validity of the ciphertext C by checking the equation $e(P, \overline{U}) = e(U, \overline{P})$, where $\overline{P} = H_3(U, V, X_A)$. If C can not pass this test, it outputs "Invalid Ciphertext".
  (2) Otherwise, Γ_j computes $\delta_j = x_{A,j}U$.
- **Decryption-Share-Verify**. Given a ciphertext $C = (U, V, \overline{U})$, the public key $P_A = (X_A, Y_A)$, the verification key $V_{A,j}$, and a decryption share δ_j, this algorithm does as follows.
  (1) It checks whether $e(P, \overline{U}) = e(U, \overline{P})$, where $\overline{P} = H_3(U, V, X_A)$. If C does note pass the above test, then this algorithm returns "Invalid Ciphertext".
  (2) Otherwise, it checks the equality

  $$e(P, \delta_j) = e(V_{A,j}, U).$$

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

  Remark 4: As in Remark 3, we can use the non-interactive zero knowledge proof for the equality of logarithms [8] as follws. If the decryption server extends the decryption share by setting $\delta_j = (Z_j, c_j, d_j)$ where

  $$Z_j = x_{A,j}U,\ t_j \overset{R}{\leftarrow} \mathbb{Z}_q, \overline{Z}_j = t_jU, \overline{S}_j = t_jP, c_j = H_4(Z_j, S_j, \overline{Z}_j, \overline{S}_j),\ d_j = t_j - \overline{s}_j c_j,$$

  then the validity of the decryption share can also be checked through the equation:

  $$c_j = H_4(Z_j, S_j, d_jU + c_jZ_j, d_jP + c_jS_j).$$

  where $\overline{P} = H_3(U, V)$, and $H_4 : \mathbb{G}^4 \to \mathbb{Z}_q$ is a cryptographic hash function.
- **Combine**. Given a ciphertext $C = (U, V, \overline{U})$, the public key $P_A = (X_A, Y_A)$, and a set of decryption shares $\{\delta_j\}_{j\in\Phi}$ where $|\Phi| \geq t$, this algorithm does as follows. Without loss of generality, we assume that the party for combining shares is one of the decryption servers who knows D_A. Otherwise, D_A can be easily provided to this combiner in other ways, since D_A is not required to be kept secret and hence can be published when sharing the private key.
  (1) It checks whether $e(P, \overline{U}) = e(U, \overline{P})$, where $\overline{P} = H_3(U, V, X_A)$. If C does note pass the above test, then this algorithm returns "Invalid Ciphertext".
  (2) For $j \in \Phi$, it checks the validity of each decryption share δ_j by the equation $e(P, \delta_j) = e(V_{A,j}, U)$, until it finds the t-th valid one. Without loss of generality, we assume that there exist at least t valid ciphertext shares and $\overline{\Phi}$ denotes the subset of t valid decryption share's indices chosen by the combiner.
  (3) It computes the temporary key $K = e(D_A, \sum_{j\in\overline{\Phi}} L_{0,j}^{\overline{\Phi}} \delta_j)$and returns the plaintext $M = H_2(K) \oplus V$, where $L_{0,j}^{\overline{\Phi}} = \prod_{k\in\overline{\Phi}, k\neq j} \frac{-k}{j-k}$ is the Lagrange coefficient.

  Remark 5: The ciphertexts and decryption shares support batch verification following the method called small exponents test [14]. For example, one ciphertext $C = (U, V, \overline{U})$ and t decryption shares δ_j needs to be verified. If

  $$e(P, c_0\overline{U} + \sum_{i=1}^{t} c_i\delta_j) = e(U, c_0\overline{P} + \sum_{i=1}^{t} c_iV_{A,j}),$$

where $\overline{P} = H_3(U, V, X_A)$, and $c_0, c_1, \ldots, c_t$ is randomly chosen integers with l_b bits, then the ciphertext and the decryption shares taken as valid. If C or one of t decryption shares is invalid, then they are accepted with probability $\frac{1}{2^{l_b}}$. In the best case that the ciphertext and the decryption shares are all valid, with the technique of batch verification, the verification equations for verifying the ciphertexts and decryption shares can be checked through one equation consisting of only two pairings.

V. Security Proof for CLTD-LGWW

In this section, we prove that the above certificateless threshold decryption scheme CLTD-LGWW satisfies the required security property. The proof procedure will be structured as a sequence of games [20]. In the following proof, we will use the below lemma [20].

Lemma 1: (Difference Lemma). Let A, B, F be events defined in some probability distribution, and suppose that $A \wedge \neg F \Leftrightarrow B \wedge \neg F$. Then $|Pr[A] - Pr[B]| \leq Pr[F]$.

Lemma 2: Assume that H_1, H_2, H_3 are random oracles. Let $\mathcal{A}_I$ be an CLTD-IND-CCA adversary of Type I that has an advantage ϵ against the above CLTD scheme. Suppose $\mathcal{A}_I$ makes at most q_{pk} private key extraction queries and q_{H_2} random oracle queries of H_2. Then we can construct an algorithm to solve 4-BDH problem such that

$$\Pr[\mathcal{A}(aP, bP, cP, abP, acP, bcP, dP) = e(P, P)^{abcd}] \geq \epsilon',$$

where $\epsilon' = \frac{1}{e \cdot q_{H_2} \cdot q_{pk}}\epsilon - \frac{q_d}{2^n}$, e is the natural logarithm.

Proof. Let $\mathcal{A}_I$ denote an Type I attacker that defeats the CLTD-IND-CCA security of the CLTD scheme CLThdBm-I. Let $\mathcal{S}_I$ be the attacker against the 4-BDH problem who gets as input

$$(\mathbb{G}, \mathbb{G}_T, q, P, e, aP, bP, cP, bcP, acP, abP, dP),$$

and tries to obtain $e(P, P)^{abcd}$. $\mathcal{S}_I$ simulates the challenger in the following sequence of games against the adversary $\mathcal{A}_I$. In the last game, $\mathcal{S}_I$ makes use of $\mathcal{A}_I$ to compute $e(P, P)^{abcd}$.

Game 0: Fix an efficient adversary $\mathcal{A}_I$ of Type I. Game 0 is simply the CLTD-IND-CCA security game in section III. In Game 0, the simulator provides adversary $\mathcal{A}_I$'s environment. Here we repeat it for cleaning up notations.

- Init. $\mathcal{A}_I$ chooses a fixed set of $t-1$ servers that it wants to corrupt. Without loss of generality, assume that $\{\Gamma_j\}_{j=1}^{t-1}$ are the corrupted servers.
- Setup. The simulator $\mathcal{S}_I$ runs the algorithm **Setup** with the key parameter 1^k as input. It outputs the common parameters $cp = (\mathbb{G}, \mathbb{G}_T, q, P, e, P_{KGC}, H_1, H_2, H_3)$ and keeps the master key s secret. It passes cp to $\mathcal{A}_I$. The hash functions are seen as random oracles. Additionally, $\mathcal{S}_I$ set $a' = s$ and randomly chooses $b', c', d' \in \mathbb{G}_q$.
- Queries 1.
  (1) Public key queries: For these queries, the simulator maintains an initially empty list L_{pk} with the entry being $(ID_{A_i}, P_{A_i}, \overline{x}_{A_i}, \gamma_i)$. For the supplied entity A_i, if there is an entry $(ID_{A_i}, P_{A_i}, \overline{x}_{A_i}, \gamma_i)$ in L_{pk}, $\mathcal{S}_I$ returns P_{A_i} to $\mathcal{A}_I$. Otherwise, $\mathcal{S}_I$ chooses a random $x_{A_i} \in \mathbb{Z}_q$ as the secret value, and returns the public key $P_{A_i} = (x_{A_i}P, x_{A_i}P_{KGC})$ to $\mathcal{A}_I$. At last, it adds $(ID_{A_i}, P_{A_i}, \overline{x}_{A_i}, 0)$ to L_{pk}, where
  $$\overline{x}_{A_i} = x_{A_i} b'^{-1} \pmod q.$$
  (2) Replacing public key queries: For supplied entity A_i's identity ID_{A_i} and the public key value P'_{A_i} , and the simulator $\mathcal{S}_I$ finds the corresponding entry $(ID_{A_i}, P_{A_i}, \overline{x}_{A_i}, \gamma_i)$ in the list L_{pk} and changes it into $(ID_{A_i}, P'_{A_i}, \perp, 1)$. Here note that γ_i is the flag to show whether the public key has been replaced.
  (3) Partial private key queries: For the supplied entity A's identity ID_{A_i}, $\mathcal{S}_I$ first finds the entry $(ID_{A_i}, P_{A_i}, \overline{x}_{A_i}, \gamma_i)$. If $\gamma_i = 1$, it rejects this query. If $\gamma_i = 0$, it sends the partial private key $D_{A_i} = a' H_1(ID_{A_i}, P_{A_i})$ to $\mathcal{A}_I$.
  (4) Private key queries: For the supplied entity A's identity ID_{A_i}, $\mathcal{S}_I$ finds the entry $(ID_{A_i}, P_{A_i}, \overline{x}_{A_i}, \gamma_i)$ in L_{pk}. If $\gamma_i = 1$, $\mathcal{S}_I$ rejects this query. Otherwise, it responds with the corresponding private key $S_{A_i} = \overline{x}_{A_i} b' a' H_1(ID_{A_i}, P_{A_i})$.
  (5) Private key share queries: For these queries, $\mathcal{S}_I$ maintains an initially empty list L_{sh} with the entry being $(ID_{A_i}, P_{A_i}, x_{A_i,1}, x_{A_i,2}, \ldots, x_{A_i,t-1})$. Given the supplied entity A_i and the public key P_{A_i}, $\mathcal{S}_I$ finds the entry $(ID_{A_i}, P_{A_i}, \overline{x}_{A_i}, \gamma_i)$ in L_{pk}. If $\gamma_i = 1$, it rejects this query. If $\gamma_i = 0$ and there is no corresponding entry in the list L_{sh}), $\mathcal{S}_I$ randomly chooses $a_1, \ldots, a_{t-1} \in \mathbb{Z}_q$ to construct the polynomial over $\mathbb{Z}_q$
  $$F_{A_i}(x) = \overline{x}_{A_i} b' + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1},$$
  and then returns to $\mathcal{A}_I$ the verification keys $\{V_{A_i,j}\}_{j=1}^{n}$ and the corrupted private key shares $\{S_{A_i,j}\}_{j=1}^{t-1}$ where
  $$x_{A_i,j} = F_{A_i}(j), D_{A_i} = a' H_1(ID_{A_i}, P_{A_i}), S_{A_i,j} = (x_{A_i,j}, D_{A_i}), V_{A_i,j} = x_{A_i,j} P.$$
  At last, $\mathcal{S}_I$ adds $(ID_{A_i}, P_{A_i}, x_{A_i,1}, x_{A_i,2}, \ldots, x_{A_i,t-1})$ to the list L_{sh}.
  (6) Decryption share queries: The attacker supplies an entity A_i's identity ID_{A_i}, public key P_{A_i} and a ciphertext $C_i = (U_i, V_i, \overline{U}_i)$. If $e(P, \overline{U}_i) \neq e(U_i, \overline{P}_i)$, where $\overline{P}_i = H_3(U_i, V_i, X_{A_i})$, $\mathcal{S}_I$ returns "Invalid Ciphertext". Otherwise, the simulator $\mathcal{S}_I$ finds the entry $(ID_{A_i}, P_{A_i}, \overline{x}_{A_i}, \gamma_i)$ in L_{pk} and then $(ID_{A_i}, P_{A_i}, x_{A_i,1}, x_{A_i,2}, \ldots, x_{A_i,t-1})$ in the list L_{sh}. If $\gamma_i = 1$, it rejects this query. If $\gamma_i = 0$, $\mathcal{S}_I$ computes $x_{A_i,k} = \sum_{j \in \Phi} x_{A_i,j} L^{\Phi}_{k,j}$ for $t \leq k \leq n$, where $x_{A_i,0} = \overline{x}_{A_i} b'$, $\Phi = \{0, 1, \ldots, t-1\}$, and $L^{\Phi}_{k,j}$ is the Lagrange

coefficient. It returns all the decryption shares $\delta_k = x_{A_i,k}U_i$, for $k = t, t+1, \ldots, n$. Here note that by Lagrange Interpolation formula, the decryption shares computed in this way are the same to those computed in the real definition game.

- Challenge. $\mathcal{A}_I$ outputs two plaintexts M_0, M_1 of equal length, the entity A^*'s identity ID_{A^*}, and the public key $P_{A^*} = (X_{A^*}, Y_{A^*})$ that it wishes to be challenged on. $\mathcal{S}_I$ chooses the random $r^* \in \mathbb{Z}_q, \beta' \in \{0,1\}$, and computes
$Q_{A^*} = H_1(ID_{A^*}, P_{A^*})$, $K^* = e(r^*Q_{A^*}, Y_{A^*})$, $V^* = H_2(K^*) \oplus M_{\beta'}$, $U^* = r^*P$, $\overline{U}^* = r^*\overline{P}^*$, where $\overline{P}^* = H_3(U^*, V^*, X_{A^*})$, and responds with the ciphertext $C^* = (U^*, V^*, \overline{U}^*)$. For this challenge phase and the following phases of Queries 2, the restrictions are as follows: (1) $\mathcal{A}_I$ can not make a decryption query on the challenge ciphertext C^* for the combination of identity ID_{A^*} and public key P_{A^*} that was used to encrypt M_b. (2) $\mathcal{A}_I$ cannot extract the challenge private key for the challenge (ID_{A^*}, P_{A^*}) at any place.
- Queries 2. $\mathcal{S}_I$ answers the queries as in the Queries 1 phase with additional restrictions as in the challenge phase.
- Guess. The adversary returns its guess β'' to $\mathcal{S}_I$.

We denote by E_0 the event $\beta' = \beta''$ and use a similar notation E_i for all following Game i. Since Game 0 is the same as the real attack game, with H_1, H_2, H_3 seen as random oracles, we have

$$|\Pr[E_0] - \tfrac{1}{2}| = \epsilon.$$

Game 1: We now transform Game 0 into Game 1, by simulating the random oracles H_1, H_2, H_3 as follows.

- H_1-queries. For these queries, it maintains an initially empty list L_{H_1} of tuples $(coin_i, ID_{A_i}, P_{A_i}, Q_{A_i}, b_{A_i}, \gamma_i)$. If ID_{A_i}, P_{A_i} has already appeared in one entry $(coin_i, ID_{A_i}, P_{A_i}, Q_{A_i}, b_{A_i}, \gamma_i)$ on the list L_{H_1}, $\mathcal{S}_I$ returns $H_1(ID_{A_i}, P_{A_i}) = Q_{A_i}$. Otherwise, the challenger $\mathcal{S}_I$ flips one coin denoted by $coin_i$ where $\Pr[coin_i = 0] = \delta$. It next finds the corresponding entry $(ID_{A_i}, P_{A_i}, \overline{x}_{A_i}, \gamma_i)$ in L_{pk}. Then it does as follows.
  - If $coin_i = 0$, then $\mathcal{S}_I$ picks b_{A_i} at random from $\mathbb{Z}_q^*$. It outputs $H_1(ID_{A_i}, P_{A_i}) = Q_{A_i} = b_{A_i}P$, and adds $(coin_i, ID_{A_i}, P_{A_i}, Q_{A_i}, b_{A_i}, \gamma_i)$ to L_{H_1}.
  - Else if $coin_i = 1$, then $\mathcal{S}_I$ picks b_{A_i} at random from $\mathbb{Z}_q^*$, then
    - if $\gamma_i = 0$, it outputs $H_1(ID_{A_i}, P_{A_i}) = Q_{A_i} = b_{A_i}(c'P)$.
    - if $\gamma_i = 1$, it outputs $H_1(ID_{A_i}, P_{A_i}) = Q_{A_i} = b_{A_i}(b'c'P)$, and adds $(coin_i, ID_{A_i}, P_{A_i}, Q_{A_i}, b_{A_i}, \gamma_i)$ to L_{H_1}.
- H_2-queries. $\mathcal{A}_I$ can issue H_2 queries at any time. $\mathcal{S}_I$ maintains an initially empty list L_{H_2} of tuples (K_i, R_i). For the the query K_i, $\mathcal{A}_I$ does as follows.
  - If the query K_i already appears on the L_{H_2} list in a tuple (K_i, R_i), then $\mathcal{S}_I$ responds with $H_2(K_i) = R_i$.
  - Otherwise, $\mathcal{S}_I$ picks a random string $R_i \in \{0,1\}^l$, adds the tuple (K_i, R_i) to L_{H_2} and responds to $\mathcal{A}_I$ with $H_2(K_i) = R_i$.
- H_3-queries. For these queries, $\mathcal{S}_I$ maintains an initially empty list L_{H_3}. For a query $H_3(U_i, V_i, X_{A_i})$, $\mathcal{S}_I$ does as follows.
  - If the query U_i, V_i, X_{A_i} already appears on the L_{H_3} list in a tuple $(t_i, \overline{P}_i, U_i, V_i, X_{A_i})$, then $\mathcal{S}_I$ responds with $\overline{P}_i$
  - If (U_i, V_i, X_{A_i}) does not appears in L_{H_3}, $\mathcal{S}_I$ randomly selects $t_i \in \mathbb{Z}_q$, sets $\overline{P}_i = t_i X_{A_i}$ and adds $(t_i, \overline{P}_i, U_i, V_i, X_{A_i})$ to the L_{H_3}, and returns the answer $H_3(U_i, V_i, X_{A_i}) = \overline{P}_i$.

Note that the attacker $\mathcal{A}_I$'s view has the same distribution in both Game 0 and Game 1, since we have replaced one set of random variables by another set of random variables which are different, yet has the same distribution. In other words, the random oracles are all perfectly simulated. Thus, we have we have

$$|\Pr[E_1] - \tfrac{1}{2}| = |\Pr[E_0] - \tfrac{1}{2}|.$$

Game 2: We now transform Game 1 into Game 2, by modifying the simulations of the answers for decryption share queries. Assume that $\mathcal{A}_I$ asks for the decryption shares of the ciphertext $C_i = (U_i, V_i, \overline{U}_i)$ under the entity A_i's identity ID_{A_i} and the public key P_{A_i}. The simulator $\mathcal{S}_I$ finds the entry $(ID_{A_i}, P_{A_i}, \overline{x}_{A_i}, \gamma_i)$ in L_{pk}. If $\gamma_i = 1$, $\mathcal{S}_I$ rejects this query. If $\gamma_i = 0$, $\mathcal{S}_I$ responds with the decryption shares simulated as follows.

- $\mathcal{S}_I$ checks the equation $e(P, \overline{U}_i) = e(U_i, \overline{P}_i)$, where $\overline{P}_i = H_3(U_i, V_i, X_{A_i})$. If the equation does not hold, $\mathcal{S}_I$ returns "Invalid Ciphertext".
- $\mathcal{S}_I$ searches the list L_{H_3} for a tuple $(t_i, \overline{P}_i, U_i, V_i, X_{A_i})$ containing U_i, V_i, X_{A_i}. If it is nonexistent, $\mathcal{S}_I$ returns "Invalid Ciphertext".
- Just like simulating the verification keys $V_{A_i,k}$ in Game 2, $k \in [t, n]$, $\mathcal{S}_I$ simulates the k-th decryption share $\delta_k = Z_k$ as follows.
  - $\mathcal{S}_I$ finds $(ID_{A_i}, P_{A_i}, x_{A_i,1}, x_{A_i,2}, \ldots, x_{A_i,t-1})$ in the list L_{sh}, and then computes
$Z_0 = \frac{1}{t_i}\overline{U}_i$, $Z_j = x_{A_i,j}U_i$, $j = 1, 2 \ldots, t-1$.
Here note that $Z_0 = x_{A_i,0}U_i = x_{A_i}U_i$. In fact, let $U_i = r_iP$ (r_i is unknown), then
$\frac{1}{t_i}\overline{U}_i = \frac{1}{t_i}(r_i\overline{P}_i) = \frac{1}{t_i}[r_i(t_iX_{A_i})] = r_iX_{A_i} = x_{A_i}b'U_i = x_{A_i}U_i$
  - $\mathcal{S}_I$ computes Z_k through
$Z_k = L_{k,0}^{\Phi}Z_0 + L_{k,1}^{\Phi}Z_1 + \ldots + L_{k,t-1}^{\Phi}Z_{t-1}$,
where
$$L_{k,j}^{\Phi} = \prod_{s \neq j, 0 \leq s \leq t-1} \frac{k-s}{j-s}, j = 0, 1, 2, \ldots, t-1.$$
$$\Phi = \{0, 1, \ldots, t-1\}.$$

Let E_2' denote there exists at least one decryption share query such that there is no tuple $(t_i, \overline{P}_i, U_i, V_i, X_{A_i})$

in L_{H_3} and $\mathcal{A}_I$'s query $(U_i, V_i, \overline{U}_i)$ satisfies $e(P, \overline{U}_i) = e(U_i, \overline{P}_i)$. In other words, this means that $\mathcal{A}_I$ can provide a valid ciphertext without making the relative query for the random oracle of H_3. First, note that, if E_3' does not happen, (i.e., for every $(U_i, V_i, \overline{U}_i)$ satisfying $e(P, \overline{U}_i) = e(U_i, \overline{P}_i)$, there must be a corresponding tuple $(t_i, \overline{P}_i, U_i, V_i, X_{A_i})$ in L_{H_3}), then the above simulation is just restating the corresponding values. Second, for each decryption query, the event that $(t_i, \overline{P}_i, U_i, V_i, X_{A_i})$ is not in L_{H_3} and $\mathcal{A}_I$'s query $(U_i, V_i, \overline{U}_i)$ satisfies $e(P, \overline{U}_i) = e(U_i, \overline{P}_i)$ happens with probability $\frac{1}{q}$. In fact, if $\mathcal{A}_I$ does not make the random oracle query of $H_3(U, V, X_A)$, then in its view, $H_3(U, V, X_A)$ is a completely random value in $\mathbb{G}$ and hence $\overline{U}_i = r_i H_3(U, V, X_A)$ is also completely random. Thus, by Difference Lemma, we have

$$|\Pr[E_2] - |\Pr[E_1]| \leq \Pr[E_2'] \leq \frac{q_d}{q},$$

where q_d is the maximal number of the decryption share queries.

Game 3: We now transform Game 2 into Game 3, by modifying the partial private key queries, private key queries and private key share queries. Let ID_{A_i} and P_{A_i} are the entity A_i's identity and the public key to be queried on for the partial private key, the private key or private key shares. Here assume that $\mathcal{A}_I$ has asked about P_{A_i} and $H_1(ID_{A_i}, P_{A_i})$ before issuing these queries. Let $(coin_i, ID_{A_i}, P_{A_i}, Q_{A_i}, b_{A_i}, \gamma_i)$ be the corresponding tuple on the L_{H_1} list. Based on the simulation of $H_1(ID_{A_i}, P_{A_i})$ in Game 1, it simulates the answers for these queries as follows.

- Partial key queries. If $\gamma_i = 1$, $\mathcal{A}_I$ rejects this query. Otherwise,it does as follows.
  - if $coin_i = 0$, $\mathcal{S}_I$ returns $D_{A_i} = b_{A_i}(a'P)$.
  - if $coin_i = 1$, $\mathcal{S}_I$ returns $D_{A_i} = b_{A_i}(a'c'P)$.
- Private key queries. If $\gamma_i = 1$, $\mathcal{A}_I$ rejects this query. Otherwise, it does as follows.
  - if $coin_i = 0$, $\mathcal{S}_I$ returns $S_{A_i} = x_{A_i} b_{A_i}(a'b'P)$.
  - if $coin_i = 1$, $\mathcal{S}_I$ terminates this game and outputs "Abort".
- Key share queries. If $\gamma_i = 1$, $\mathcal{A}_I$ rejects this query. Otherwise, it randomly chooses $x_{A_i,1}, x_{A_i,2}, \ldots, x_{A_i,t-1} \in \mathbb{Z}_q$, and sets the private key share $S_{A_i,j} = (x_{A_i,j}, D_{A_i})$ and the verification key $\{V_{A_i,j} = x_{A_i,j}P\}$, for $1 \leq j \leq t-1$, where $D_{A_i} = b_{A_i}(a'P)$ for $coin_i = 0$ or $D_{A_i} = b_{A_i}(a'c'P)$ for $coin_i = 1$. Then, it computes other verification key $V_{A_i,k} = \prod_{j \in \Phi} V_{A_i,j}^{L_{k,j}^{\Phi}}$ for $t \leq k \leq n$, where $L_{k,j}^{\Phi}$ is the Lagrange coefficients, $V_{A_i,0}$ is the first part of the publice key $P_{A_i} = (X_{A_i}, Y_{A_i})$, and $\Phi = \{0, 1, \ldots, t-1\}$. At last, $\mathcal{S}_I$ adds $(ID_{A_i}, P_{A_i}, x_{A_i,1}, x_{A_i,2}, \ldots, x_{A_i,t-1})$ to the list L_{sh}. Here note that a polynomial $F_{A_i}(x) = \overline{x}_{A_i} b' + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}$ can be equivalently determined by the coefficients $\{\overline{x}_{A_i} b', a_1, a_2, \ldots, a_{t-1}\}$ or any t values $F_{A_i}(j)$ on different points.

For the above modification, we can see that the answers for the partial private key queries, the private key queries are simulated in different but equivalent ways, if $\mathcal{S}_I$ does not abort. Furthermore, the values of key shares in Game 2 and Game 1 follows the same random distribution. In fact, according to Lagrange Interpolation, a polynomial $F_{A_i}(x) = \overline{x}_{A_i} b' + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}$ can be equivalently determined by the coefficients $\{\overline{x}_{A_i} b', a_1, a_2, \ldots, a_{t-1}\}$ or a set of t values $F_{A_i}(j)$ on different points. Let F_3 denote $\mathcal{A}_I$ aborts in Game 3. Thus we have

$$\Pr[E_3|\neg F_3] = \Pr[E_2],$$
$$\Pr[\neg F_3] = \delta^{q_{pk}},$$

where q_{pk} is the maximal number of private key queries allowed for $\mathcal{A}_I$.

Game 4: We now transform Game 3 into Game 4, by modifying the simulations for the challenge queries. $\mathcal{A}_I$ outputs two plaintexts M_0, M_1 of equal length, a challenge entity A^*'s identity ID_{A^*}, and public key P_{A^*} that it wishes to be challenged on. First, $\mathcal{S}_I$ finds the corresponding tuple $(coin^*, ID_{A^*}, P_{A^*}, Q_{A^*}, b_{A^*}, \gamma^*)$ on the list L_{H_1}. If $coin^* = 0$, $\mathcal{S}_I$ terminates the game and outputs "Abort". Let F_4 denote $\mathcal{A}_I$ aborts in Game 4. For Game 4, we have

$$\Pr[E_4|\neg F_4] = \Pr[E_2],$$
$$\Pr[\neg F_4] = (1-\delta)\delta^{q_{pk}}.$$

Game 5: In this game, we further modify the simulations for the challenge queries. If $coin^* = 1$, $\mathcal{S}_I$ simply chooses $t^* \in \mathbb{Z}_q^*$ and then does as follows.

- If $\gamma^* = 0$, $\mathcal{A}_I$ computes
  $$U^* = \overline{x}_{A^*}^{-1} b_{A^*}^{-1}(d'P),\ V^* \xleftarrow{R} \{0,1\}^l,\ \overline{U}^* = t^* U^*.$$
  Then it adds the tuple $(t^*, t^*P, U^*, V^*, X_{A^*})$ to the list L_{H_3}.
- If $\gamma^* = 1$, $\mathcal{A}_I$ computes
  $$U^* = b_{A^*}^{-1}(d'P),\ V^* \xleftarrow{R} \{0,1\}^l,\ \overline{U}^* = t^* U^*.$$
  Then it adds the tuple $(t^*, t^*P, U^*, V^*, X_{A^*})$ to the list L_{H_3}.

When the adversary $\mathcal{A}_I$ outputs the guess β'', $\mathcal{S}_I$ randomly chooses an entry (K_i, R_i) and outputs K_i as the guess of $e(P,P)^{a'b'c'd'}$.

Let F' denotes the event that $\mathcal{A}_I$ asks the H_2 query of $e(U^*, S_{A^*}) = e(P,P)^{a'b'c'd'}$. If F does not happen, then $H_2(e(P,P)^{a'b'c'd'})$ is a random value in $\{0,1\}^l$ in the view of $\mathcal{A}_I$ in both Game 4 and Game 5. Thus, if F' does not happen, there is no difference between Game 4 and Game 5. Let F_5 denote $\mathcal{A}_I$ aborts in Game 5. So, by Difference Lemma, we have

$$|\Pr[E_4|\neg F_4] - \Pr[E_5|\neg F_5]| = |\Pr[E_4|\neg F_4] - \tfrac{1}{2}| \leq \Pr[F'|\neg F_5].$$

Let E_5' denotes the event that the challenger $\mathcal{S}_I$ outputs the right value of $e(P,P)^{a'b'c'd'}$, then we have

$$\Pr[E_5'|\neg F_5] \geq \tfrac{1}{q_{H_2}} \Pr[F'|\neg F_5].$$

By combining all the above results, we have

$$\begin{aligned}
\Pr[E_5'] &= \Pr[E_5'|\neg F_5]\Pr[\neg F_5] \\
&= \Pr[E_5'|\neg F_5]\Pr[\neg F_4] \\
&\geq \frac{1}{q_{H_2}}(1-\delta)\delta^{q_{pk}}\Pr[F_5'|\neg F_5] \\
&\geq \frac{1}{q_{H_2}}(1-\delta)\delta^{q_{pk}}|\Pr[E_4|\neg F_4]-\frac{1}{2}| \\
&= \frac{1}{q_{H_2}}(1-\delta)\delta^{q_{pk}}|\Pr[E_2]-\frac{1}{2}| \\
&\geq \frac{1}{q_{H_2}}(1-\delta)\delta^{q_{pk}}|\Pr[E_1]-\frac{1}{2}-\frac{q_d}{q}| \\
&= \frac{1}{q_{H_2}}(1-\delta)\delta^{q_{pk}}|\epsilon-\frac{q_d}{q}| \\
&\geq \frac{1}{q_{H_2}}(1-\delta)\delta^{q_{pk}}\epsilon-\frac{q_d}{q}
\end{aligned}$$

If we set $\delta = 1-\frac{1}{q_{pk}+1}$, then the value of $(1-\delta)\delta^{q_{pk}}$ is the maximal one. Additionally, note that for large q_{ce}, $\frac{1}{q_{pk}+1}(1-\frac{1}{q_{pk}+1})^{q_{pk}} \approx \frac{1}{eq_{pk}}$. Thus we have

$$\Pr[E_5'] \geq \frac{1}{q_{H_2}}\frac{1}{q_{pk}+1}(1-\frac{1}{q_{pk}+1})^{q_{pk}}\epsilon-\frac{q_d}{2^n} \approx \frac{1}{q_{H_2}}\frac{1}{eq_{pk}}\epsilon-\frac{q_d}{2^n}$$

Game 6. In this game, the simulator replaces a', b', c', d' with a, b, c, d respectively in all places. It is easy to see that in Game 5, the simulator with its input $(a'P, b'P, c'P, b'c'P, a'c'P, a'b'P, d'P)$ need not know a', b', c', d' at any place. So we have that the simulator in Game 5, with input $(\mathbb{G}, \mathbb{G}_T, q, P, e, aP, bP, cP, bcP, acP, abP, dP)$, can output $e(P,P)^{abcd}$ with the least probability approximately being $\frac{1}{q_{H_2}}\frac{1}{eq_{pk}}\epsilon-\frac{q_d}{2^n}$. □

Lemma 3: Assume that H_1, H_2, H_3 are random oracles. Let $\mathcal{A}_{II}$ be an CLTD-IND-CCA adversary of Type II that has an advantage ϵ against the above CLTD scheme. Suppose $\mathcal{A}_{II}$ makes at most q_{pk} private key extraction queries and q_{H_2} random oracle queries of H_2. Then we can construct an algorithm $\mathcal{S}_{II}$ for BDH problem such that

$$\Pr[\mathcal{S}_{II}(aP, bP, cP) = e(P,P)^{abc}] \geq \epsilon',$$

where $\epsilon' = \frac{1}{e\cdot q_{H_2}\cdot q_{pk}}\epsilon-\frac{q_d}{2^n}$, e is the natural logarithm.

Proof. Lemma 3 can be similarly proved as Lemma 2 with some trivial modification. Here we omit the details and just show the main idea for proof. First, in the proof of Lemma 2 we provide $\mathcal{A}_I$ with the master key but prohibit it from replacing public keys, then the adversary $\mathcal{A}_I$ of Type I will become the $\mathcal{A}_{II}$ adversary of Type II. Second, since the queries for replacing public key is not allowed, then all corresponding operations ,such as the case $\gamma_i = 1$ in Game 1 and Game 5, can now be left out. At the end, the last output $e(P,P)^{abcd}$ can be transformed into $e(P,P)^{bcd}$ with a known and b, c, d unknown. So the BDH problem (Here this means to output $e(P,P)^{bcd}$ with bP,cP,dP as input) can be solved. □

Following Lemma 2 and Lemma 3, we have:

Theorem 1: Assume that 4-BDH problem is intractable and H_1, H_2 and H_3 are random oralces. Then the certificateless threshold decryption scheme CLTD-LGWW in the above section is CLTD-IND-CCA secure.

VI. Efficiency Comparison

This section compares our scheme CLTD-LGWW and the Long-Chen scheme [15] as follows.

(1) For the algorithm to share the private keys, CLTD-LGWW needs n scalar multiplications. In contrast, the Long-Chen scheme needs $2n$ scalar multiplications and n parings.
(2) For the algorithm to generate decryption shares, CLTD-LGWW just needs one scalar multiplication in the group $\mathbb{G}$. In contrast, in addition to 1 scalar multiplication, the Long-Chen scheme needs 3 pairing computation.
(3) For the algorithm to verify decryption shares, CLTD-LGWW needs only 2 pairing computations. In addition to 2 pairings, the Long-Chen scheme needs 2 exponentiations in the group $\mathbb{G}_T$.
(4) For the algorithm to combine decryption shares, CLTD-LGWW needs 1 pairing computation and t scalar multiplications in the group $\mathbb{G}$. In constrast, the Long-Chen scheme needs t exponentiations in the group $\mathbb{G}_T$.
(5) As shown by Remark 5, the decryption share supports batch verification. By applying the technique of batch verification, the 4 pairings for the algorithm **Decryption-Share-Verify** can be reduced to 2 ones.
(6) Following the famous work of short signatures [21], it is obvious that the decryption share also supports fast generating and short length, since the it is computed by $\delta_i = \overline{x}_{A,j}U$.

VII. Conclusion

W proposed a new CLTD scheme based on pairings. Compared with other CLTD schemes, it uses less bilinear pairings and shorter ciphertexts and decryption shares. This makes it suitable for some applications where the resource is very limited, such as wireless networks and smart cards. All these satisfactory properties are based on our new basic technique that shares a private key in a bilinear group using shares in a finite field.

References

[1] A. Shamir, Identity-based cryptosystems and signature schemes, in: Proceedings of Crypto 84, LNCS, vol.196, Springer-Verlag, 1984, pp. 47-53.
[2] D. Boneh, M. Franklin, Identity-Based encryption from the Weil pairing, in: Proceedings of Crypto 2001, LNCS, vol.2139, Springer-Verlag, 2001, pp.213-229.
[3] S.S. Al-Riyami, K.G. Paterson, Certificateless public key cryptography, in: Proceedings of the Ninth International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, 2003, pp. 452-473.
[4] Y. Desmedt and Y. Frankel, Threshold cryptosystems, in: Proceedings of CRYPTO'89, LNCS, vol.435, Spinger-Verlag, 190, pp.307-315.
[5] Z. Wan, J. Weng, J. Li, Security Mediated Certificateless Signatures Without Pairing. JCP 5(12): 1862-1869 (2010)

[6] M. Chen, K. Wu, J. Xu, J. Du, A Certificateless and Across Administrative Domains Authenticated Key Exchange Scheme for E-payment. JSW 6(10): 1985-1992 (2011)
[7] A. Santis, Y. Desmedt, Y. Frankel, and M. Yung, How to share a function securely, in: Proceedings of 26th ACM STOC, ACM Press, 1994, pp.522-533.
[8] V. Shoup and R. Gennaro, Securing Threshold Cryptosystems against Chosen Ciphertext Attack, Journal of Cryptology 15 (2) (2002) 75 -96.
[9] Z. Tan. Improvement on a Threshold Authenticated Encryption Scheme. Journal of Software, Vol 5, No 7 (2010), 697-704, Jul 2010
[10] J. Baek, Y. Zheng, Identity-based threshold decryption, in: Proceedings of PKC04, LNCS, vol.2947, Springer-Verlag, 2004, pp. 26-76.
[11] J. Baek, Y. Zheng, Identity-Based Threshold Signature Scheme from the Bilinear Pairings, in:Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04), vol.2, IEEE Computer Society, PP.124-128.
[12] M. Scott. Computing the Tate Pairing. in: Proceedinsg of CT-RSA 2005, LNCS 3376, Springer-Verlag,2005, pp. 293C304.
[13] W. Yuan, L. Hu, H. Li, J. Chu, H. Wang, Cryptanalysis and Improvement of an ID-Based Threshold Signcryption Scheme. JCP 7(6): 1345-1352 (2012)
[14] A. L. Ferrara, M. Green, S. Hohenberger and M. Pedersen, Short Signature Batch Verification, in: Proceedings of CT-RSA 2009, LNCS, vol.5473, Springer-Verlag, 2009, pp.309-324.
[15] Y. Long, K. Chen, Certificateless threshold cryptosystem secure against chosen-ciphertext attack, Information Sciences 177 (24) (2007) pp.5620-5637.
[16] Alexander W. Dent. A survey of certificateless encryption schemes and security models. International Journal of Information Security. 2008, Volume 7, Number 5, 349-377.
[17] Y. Long, K. Chen, Efficient chosen-ciphertext secure certificateless threshold key encapsulation mechanism. Inf. Sci. 180(7): 1167-1181 (2010)
[18] J. Baek, R. Safavi-Naini, and W. Susilo. Certificateless public key encryption without pairing. In J. Zhou and J. Lopez, editors, Proceedings of the 8th International Conference on Information Security (ISC 2005), volume 3650 of Lecture Notes in Computer Science, pages 134-148. Springer-Verlag, 2005.
[19] J. Lai and K. Kou. Self-generated-certificate public key encryption without pairing. In T. Okamoto and X. Wang, editors, Public Key Cryptography - PKC 2007, volume 4450 of Lecture Notes in Computer Science, pages 476-489. Springer-Verlag, 2007.
[20] V. Shoup, Sequences of Games: A Tool for Taming Complexity in Security Proofs, Cryptology ePrint report 2004/332, 2004
[21] D. Boneh, H. Shacham, and B. Lynn Short signatures from the Weil pairing. Journal of Cryptology, Vol. 17, No. 4, pp. 297-319, 2004

Fei Li received her MS degree in applied mathematics, Guangzhou University, China, in 2008. She is currently a lecturer in Ludong University, China. Her research interests include cryptography and algebra.

Wei Gao received his Ph.D., MS and BS degrees in applied mathematics from Hunan University in 2006, Guangzhou University 2003, and Ludong University in 2000, respectively. He is a lecturer in Ludong University from 2007. His research interests include security, cryptography and number theory.

Yilei Wang received her MS degree in computer science, Shandong Normal University, China, in 2004. She is currently a lecturer in Ludong University, China. Her research interests include cryptography and information security.

Xueli Wang received his PhD degree in mathematics from the Academy of China in 1991, his MS degree in mathematics from Shannxi Normal University in 1987. He is currently Professor of Computer Science at South China Normal University. His current research interests include cryptography, number theory and elliptic curves.