This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

An Efficient Certificateless Encryption for Secure Data Sharing in Public Clouds

Seung-Hyun Seo, *Member, IEEE,* Mohamed Nabeel, *Member, IEEE,* Xiaoyu Ding, *Student Member, IEEE,* and Elisa Bertino, *Fellow, IEEE*

Abstract—We propose a mediated certificateless encryption scheme without pairing operations for securely sharing sensitive information in public clouds. Mediated certificateless public key encryption (mCL-PKE) solves the key escrow problem in identity based encryption and certificate revocation problem in public key cryptography. However, existing mCL-PKE schemes are either inefficient because of the use of expensive pairing operations or vulnerable against partial decryption attacks. In order to address the performance and security issues, in this paper, we first propose a mCL-PKE scheme without using pairing operations. We apply our mCL-PKE scheme to construct a practical solution to the problem of sharing sensitive information in public clouds. The cloud is employed as a secure storage as well as a key generation center. In our system, the data owner encrypts the sensitive data using the cloud generated users' public keys based on its access control policies and uploads the encrypted data to the cloud. Upon successful authorization, the cloud partially decrypts the encrypted data for the users. The users subsequently fully decrypt the partially decrypted data using their private keys. The confidentiality of the content and the keys is preserved with respect to the cloud, because the cloud cannot fully decrypt the information. We also propose an extension to the above approach to improve the efficiency of encryption at the data owner. We implement our mCL-PKE scheme and the overall cloud based system, and evaluate its security and performance. Our results show that our schemes are efficient and practical.

Index Terms—Cloud Computing, Certificateless cryptography, confidentiality, access control.

1 Introduction

Due to the benefits of public cloud storage, organizations have been adopting public cloud services such as Microsoft Skydrive [18] and Dropbox [11] to manage their data. However, for the widespread adoption of cloud storage services, the public cloud storage model should solve the critical issue of data confidentiality. That is, shared sensitive data must be strongly secured from unauthorized accesses. In order to assure confidentiality of sensitive data stored in public clouds, a commonly adopted approach is to encrypt the data before uploading it to the cloud. Since the cloud does not know the keys used to encrypt the data, the confidentiality of the data from the cloud is assured. However, as many organizations are required to enforce fine-grained access control to the data, the encryption mechanism should also be able to support fine-grained encryption based access control. As shown in Figure 1, a typical approach used to support fine-grained encryption based access control is to encrypt different sets of data items to which the same access control policy applies with different symmetric keys and give users either the relevant keys [4], [19] or the ability to derive the keys [20], [23]. Even though the key derivation-based approaches reduce the number of keys to be managed, symmetric key based mechanisms in general have the problem of high costs for key management. In order to reduce the overhead of key management, an alternative is to use a public key cryptosystem. However, a traditional public key cryptosystem requires a trusted Certificate Authority (CA) to issue digital certificates that bind users to their public keys. Because the CA has to generate its own signature on each user's public key and manage each user's certificate, the overall certificate management is very expensive and complex. To address such shortcoming, Identity-Based Public Key Cryptosystem (IB-PKC) was introduced, but it suffers from the key escrow problem as the key generation server learns the private keys of all users. Recently, Attribute Based Encryption (ABE) has been proposed that allows one to encrypt each data item

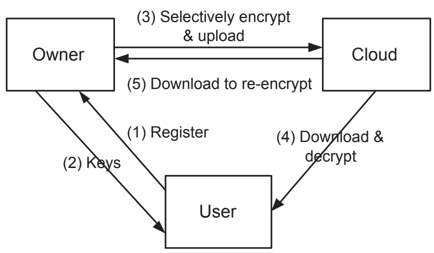

Fig. 1. Symmetric Key Based Fine-Grained Encryption

- *Seung-Hyun Seo, Mohamed Nabeel, Xiaoyu Ding and Elisa Bertino are with the Department of Computer Science, Purdue University, West Lafayette, IN, 47907.*
E-mail: seo29@purdue.edu, nabeel@purdue.edu, ding55@purdue.edu, bertino@purdue.edu

Digital Object Indentifier 10.1109/TKDE.2013.138

based on the access control policy applicable to the data. However, in addition to the key escrow problem, ABE has the revocation problem as the private keys given to existing users should be updated whenever a user is revoked. In order to address the key escrow problem in IB-PKC, Al-Riyami and Paterson introduced a new cryptosystem called Certificateless Public Key Cryptography (CL-PKC) [2].

Lei et al.[16] then proposed the CL-PRE (Certificateless Proxy Re-Encryption) scheme for secure data sharing in public cloud environments. Although their scheme is based on CL-PKC to solve the key escrow problem and certificate management, it relies on pairing operations. Despite recent advances in implementation techniques, the computational costs required for pairing are still considerably high compared to the costs of standard operations such as modular exponentiation in finite fields. Moreover, their scheme only achieves Chosen Plaintext Attack (CPA) security. As pointed out in [3], CPA security is often not sufficient to guarantee security in general protocol settings. For example, CPA is not sufficient for many applications such as encrypted email forwarding and secure data sharing that require security against Chosen Ciphertext Attack (CCA).

In this paper, we address the shortcomings of such previous approaches and propose a novel mediated Certificateless Public Key Encryption (mCL-PKE) scheme that does not utilize pairing operations. Since most CL-PKC schemes are based on bilinear pairings, they are computationally expensive. Our scheme reduces the computational overhead by using a pairing-free approach. Further, the computation costs for decryption at the users are reduced as a semi-trusted security mediator partially decrypts the encrypted data before the users decrypt. The security mediator acts as a policy enforcement point as well and supports instantaneous revocation of compromised or malicious users. In section 5, we show that our scheme is much more efficient than the pairing based scheme proposed by Lei et al. [16]. Moreover, compared to symmetric key based mechanisms, our approach can efficiently manage keys and user revocations. In symmetric key systems, users are required to manage a number of keys equal to at least the logarithm of the number of users, whereas in our approach, each user only needs to maintain its public/private key pair. Further, revocation of users in a typical symmetric key system requires updating the private keys given to all the users in the group, whereas in our approach private keys of the users are not required to be changed.

Based on our mCL-PKE scheme, we propose a novel approach to assure the confidentiality of data stored in public clouds while enforcing access control requirements. There are five entities in our system: the data owner, users, the Security Mediator (SEM), the Key Generation Center (KGC), and the storage service (see Figure 2 for a high-level architecture of our approach). The SEM, KGC, and the storage service are semi-trusted and reside in a public cloud. Although they are not trusted for the confidentiality of the data and the keys, they are trusted for executing the protocols correctly. According to the access control policy, the data owner encrypts a symmetric data encryption key using mCL-PKE scheme and encrypts the data items using symmetric encryption algorithm. Then, data owner uploads encrypted data items and the encrypted data encryption key to the cloud. Notice that a major advantage of our approach compared to conventional approaches is that the KGC, which is the entity in charge of generating the keys, resides in a public cloud. Thus, it simplifies a task of key management for organizations.

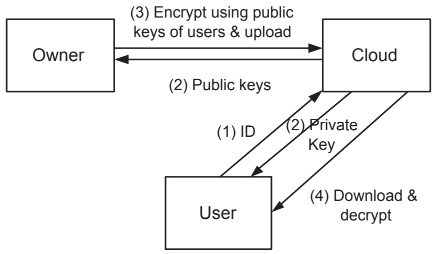

Fig. 2. CL-PKE Based Fine-Grained Encryption

In a conventional CL-PKE scheme, user's complete private key consists of a secret value chosen by the user and a partial private key generated by the KGC. Unlike the CL-PKE scheme, the partial private key is securely given to the SEM, and the user keeps only the secret value as its own private key in the mCL-PKE scheme. So, each user's access request goes through the SEM which checks whether the user is revoked before it partially decrypts the encrypted data using the partial private key. It does not suffer from the key escrow problem, because the user's own private key is not revealed to any party. It should be noted that neither the KGC nor the SEM can decrypt the encrypted data for specific users. Moreover, since each access request is mediated through the SEM, our approach supports immediate revocation of compromised users.

It is important to notice that if one directly applies our basic mCL-PKE scheme to cloud computing and if many users are authorized to access the same data, the encryption costs at the data owner can become quite high. In such case, the data owner has to encrypt the same data encryption key multiple times, once for each user, using the users' public keys. To address this shortcoming, we introduce an extension of the basic mCL-PKE scheme. Our extended mCL-PKE scheme requires the data owner to encrypt the data encryption key only once and to provide some additional information to the cloud so that authorized

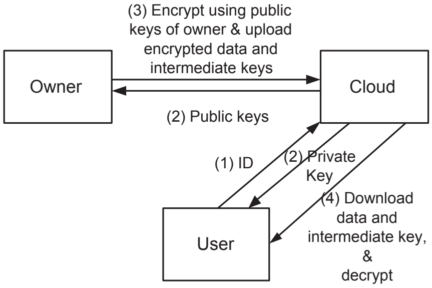

Fig. 3. CL-PKE with Intermediate Keys Based Fine-Grained Encryption

users can decrypt the content using their private keys. Figure 3 gives a high-level view of the extension. The idea is similar to Proxy Re-Encryption (PRE) by which the data encryption key is encrypted using the data owner's public key and later can be decrypted by different private keys after some transformation by the cloud which acts as the proxy. However, in our extension, the cloud simply acts as storage and does not perform any transformation. Instead, the user is able to decrypt using its own private key and an intermediate key issued by the data owner.

Our main contributions are summarized as follows:

- We propose a new mCL-PKE scheme. We present the formal security model and provide the security proof. Since our mCL-PKE scheme does not depend on the pairing-based operation, it reduces the computational overhead. Moreover, we introduce an extension of mCL-PKE scheme to efficiently encrypt data for multiple users.
- We propose a novel approach to securely share data in a public cloud. Unlike conventional approaches, the KGC only needs to be semi-trusted and can reside in the public cloud, because our mCL-PKE scheme does not suffer from the key escrow problem.
- We have implemented our mCL-PKE scheme and the extension to evaluate the performance. The experimental result shows that our mCL-PKE scheme can be realistically applied in a public cloud for secure data sharing.

The remainder of this paper is organized as follows: Section 2 introduces our mCL-PKE scheme without pairing, and presents a security model and security proof. Section 3 proposes an approach for secure sharing data in public clouds. Section 4 proposes the extended scheme for secure cloud storage. Section 5 shows the performance evaluation. Section 6 discusses related works and Section 7 concludes the paper.

2 mCL-PKE Scheme without Pairings

In this section, we present the mediated Certificateless Public Key Encryption (mCL-PKE) scheme and its security model. Then, we prove the formal security of mCL-PKE scheme.

2.1 Definitions

Definition 1. The mediated certificateless public key encryption scheme is a 7-tuple mCL-PKE=(SetUp, SetPrivateKey, SetPublicKey, SEM-KeyExtract, Encrypt, SEM-Decrypt, USER-Decrypt). The description of each algorithm is as follows.

- SetUp: It takes a security parameter k as input and returns system parameters params and a secret master key mk. We assume that params are publicly available to all users.
- SetPrivateKey: It takes params and ID as input and outputs the user's (the owner of ID) secret value SK_{ID}. Each user runs this algorithm.
- SetPublicKey: It takes params and a user's secret value SK_{ID} as input and returns the user's public key PK_{ID}.
- SEM-KeyExtract: Each user registers its own identity and public key to the KGC. After the KGC verifies the user's knowledge of the private key corresponding to its public key, the KGC takes params, mk and user identity ID as input and generates a SEM-key corresponding to ID required during decryption time by the SEM. The KGC runs this algorithm for each user, and we assume that the SEM-key is distributed securely to the SEM.
- Encrypt: It takes params, a user's identity ID, a user's public key PK_{ID}, and a message M as inputs and returns either a ciphertext C_{ID} or a special symbol $\perp$ meaning an encryption failure. Any entity can run this algorithm.
- SEM-Decrypt: It takes params, a SEM-key, and a ciphertext C_{ID} as input, and then returns either a partial decrypted message C'_{ID} for the user or a special symbol $\perp$ meaning an decryption failure. Only the SEM runs this algorithm using SEM-key.
- USER-Decrypt: It takes params, a user's private key SK_{ID}, the partial decrypted message C'_{ID} by the SEM as input and returns either a fully decrypted message M or a special symbol $\perp$ meaning an decryption failure. Only the user can run this algorithm using its own private key and the partial decrypted message by the SEM.

Definition 2. The Computational Diffie-Hellman (CDH) problem is defined as follows: Let p and q be primes such that $q|(p-1)$. Let g be a generator of $\mathbb{Z}_p^*$. Let $\mathcal{A}$ be an adversary. $\mathcal{A}$ tries to solve the following problem: Given (g, g^a, g^b) for uniformly chosen $a, b, c \in \mathbb{Z}_q^*$, compute $k = g^{ab}$. We define $\mathcal{A}$'s advantage in solving the CDH problem by **Adv($\mathcal{A}$)**=Pr[$\mathcal{A}(g, g^a, g^b) = g^{ab}$].

2.2 Security Model of Mediated CL-PKE

In general, in order to construct the security model of a mediated CL-PKE scheme [9], we must consider two types of adversaries: Type I adversary $\mathcal{A}_I$ and Type II adversary $\mathcal{A}_{II}$. A type I adversary $\mathcal{A}_I$ means a normal third party attacker which does not know the master key, but can replace public keys of users. That is, $\mathcal{A}_I$ does not have access to the master key, but is able to choose any public key to be used for the challenge ciphertext. A type II adversary $\mathcal{A}_{II}$ is a malicious KGC which has the master key, but is unable to replace public keys of users. That is, $\mathcal{A}_{II}$ can have access to the master key, but can use only a registered public key for the challenge ciphertext. We do not need to consider a malicious SEM explicitly, because it is weaker than $\mathcal{A}_{II}$.

In order to describe the security of the mediated CL-PKE scheme, we consider a formal game where the adversary $\mathcal{A}$ interacting with their Challenger as follows. The adversary $\mathcal{A}$ can be either $\mathcal{A}_I$ or $\mathcal{A}_{II}$. The Challenger should keep a history of query-answer while interacting with the adversaries.

A Formal Game for an adversary $\mathcal{A}$

- SetUp: The Challenger runs SetUp by taking a security parameter k as input in order to return system parameters `params` and a master key `mk`. The Challenger gives `params` to the adversary $\mathcal{A}$ and keeps `mk` secret.
- Phase 1: The adversary $\mathcal{A}$ can adaptively make various queries and the Challenger can respond to the queries as follows:
  - SEM-key for ID Extraction: The Challenger runs SEM-KeyExtract to generate the SEM-key d_0 using an identity ID and `params` as the input.
  - Public Key Request for ID: The Challenger runs SetPrivateKey to generate SK_{ID}, and then runs SetPublicKey to generate the public key PK_{ID} using ID, SK_{ID} and `params` as the input. It returns PK_{ID} to $\mathcal{A}$.
  - Public Key Replacement: The adversary $\mathcal{A}$ can repeatedly replace the public key for any identity with any value of its choice. The SEM-key is also updated if the Challenger bundles the public key with the identity for SEM-key creation. The replaced public key will be used in the rest of the game unless replaced again.
  - Private Key Extraction for ID: The Challenger runs SetPrivateKey to generate SK_{ID} using ID as the input. It returns SK_{ID} to $\mathcal{A}$. However, if the public key of ID has been already replaced by the adversary $\mathcal{A}$, this query is disallowed.
  - SEM-Decryption: The adversary provides an identity ID and a ciphertext C_{ID}. The Challenger responds with the partial decryption C'_{ID} under the SEM-key d_0 that is associated with the identity ID.
  - USER-Decryption: The adversary provides an identity ID and a ciphertext C'_{ID}. The Challenger responds with the decryption of C'_{ID} under the private key SK_{ID} that is associated with the identity ID.
- Challenge Phase: Once $\mathcal{A}$ determines that Phase 1 is over, it outputs a challenge identity ID^* and a pair of plaintext (M_0, M_1) with an equal length. In case that $\mathcal{A}$ is a $\mathcal{A}_I$, it chooses a public key of identity ID^*, PK_{ID^*} by using the Public Key Replacement query. For the identity ID^*, $\mathcal{A}_I$ cannot ask both the SEM-key Extraction query and Private Key Extraction query. If $\mathcal{A}$ is a $\mathcal{A}_{II}$, the public key of identity ID^* cannot be replaced. For the identity ID^*, $\mathcal{A}_{II}$ cannot ask Private Key Extraction query. The Challenger picks $\beta \in_R \{0,1\}$ and creates a target ciphertext C_{ID^*} which is the encryption of M_β under the public key of ID^*. In case of $\mathcal{A}_I$, the public key of ID^* is PK_{ID^*}. Otherwise, the public key of ID^* is the original one. The Challenger returns C_{ID^*} to $\mathcal{A}$.
- Phase 2: $\mathcal{A}$ continues to issue more queries, but it cannot issue both the SEM-key Extraction query and Private Key Extraction query for the ID^*. If $\mathcal{A}_{\mathcal{I}}$ has requested the private key corresponding to the public key PK_{ID^*}, then it cannot make a SEM-Decrypt query for C_{ID^*}. On the other hand, if $\mathcal{A}_{\mathcal{II}}$ has requested the SEM-key corresponding to ID^*, it cannot make a USER-Decrypt query for C'_{ID^*} where C'_{ID^*} is the result of SEM-Decrypt query for C_{ID^*}.
- Guess: $\mathcal{A}$ outputs its guess bit $\beta' \in_R \{0,1\}$.

In case of $\beta' = \beta$, $\mathcal{A}$ wins. We define $\mathcal{A}_i$'s advantage in the above game by $2 \times |Pr[\beta' = \beta] - \frac{1}{2}|, i \in \{I, II\}$. **A mediated CL-PKE scheme** is **IND-CCA secure** if there is no probabilistic polynomial-time adversary in the above games with non-negligible advantage in the security parameter k. The security of our mediated certificateless public key encryption scheme is based on the assumed intractability of the CDH problem.

2.3 Basic Algorithm

- SetUp:
  KGC takes as input a security parameter k to generate two primes p and q such that $q|p-1$. It then performs the following steps:
  1) Pick a generator g of $\mathbb{Z}_p^*$ with order q.
  2) Select $x \in \mathbb{Z}_q^*$ uniformly at random and compute $y = g^x$.
  3) Choose cryptographic hash functions $H_1 : \{0,1\}^* \times \mathbb{Z}_p^* \rightarrow \mathbb{Z}_q^*$, $H_2 : \{0,1\}^* \times \mathbb{Z}_p^* \times \mathbb{Z}_p^* \rightarrow \mathbb{Z}_q^*$, $H_3 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$, $H_4 : \mathbb{Z}_p^* \rightarrow \{0,1\}^{n+k_0}$, $H_5 : \mathbb{Z}_p^* \rightarrow \{0,1\}^{n+k_0}$, and $H_6 : \mathbb{Z}_p^* \times \{0,1\}^{n+k_0} \times$

$\mathbb{Z}_p^* \times \{0,1\}^{n+k_0} \to \mathbb{Z}_q^*$, where n, k_0 are the bit-length of a plaintext and a random bit string, respectively.

The system parameters $params$ are $(p, q, n, k_0, g, y, H_1, H_2, H_3, H_4, H_5, H_6)$. The master key of KGC is x. The plaintext space is $\mathtt{M} = \{0,1\}^n$ and the ciphertext space is $\mathtt{C} = \mathbb{Z}_p^* \times \{0,1\}^{n+k_0} \times \mathbb{Z}_q^*$

- SetPrivateKey:
  The entity **A** chooses $z_A \in \mathbb{Z}_q^*$ uniformly at random as the private key of **A**.
- SetPublicKey:
  The entity **A** computes $U_A = g^{z_A}$.
- SEM-KeyExtract:
  KGC selects $s_0, s_1 \in \mathbb{Z}_q^*$ and computes $w_0 = g^{s_0}$, $w_1 = g^{s_1}$, $d_0 = s_0 + xH_1(ID_A, w_0)$, $d_1 = s_1 + xH_2(ID_A, w_0, w_1)$. KGC sets d_0 as the SEM-key for **A**. After **A** proves the knowledge of the secret value z_A such that $U_A = g^{z_A}$, KGC sets (U_A, w_0, w_1, d_1) as the **A**'s public keys.
- Encrypt:
  To encrypt a plaintext $M \in \{0,1\}^n$ for the entity **A** with identity ID_A and public keys (U_A, w_0, w_1, d_1), it performs the following steps:
  1) Check whether $g^{d_1} = w_1 \cdot y^{H_2(ID_A, w_0, w_1)}$.
     If the checking result is not valid, encryption algorithm must be aborted.
  2) Choose $\sigma \in \{0,1\}^{k_0}$ and
     compute $r = H_3(M, \sigma, ID_A, U_A)$.
  3) Compute $C_1 = g^r$.
  4) Compute $C_2 = (M||\sigma) \oplus H_4({U_A}^r) \oplus H_5(w_0^r \cdot y^{H_1(ID_A, w_0) \cdot r})$.
  5) Compute C_3
     $C_3 = H_6(U_A, (M||\sigma) \oplus H_4(U_A^r), C_1, C_2)$.

  Output the ciphertext $C = (C_1, C_2, C_3)$.
  In Step 1, an entity who wants to encrypt a message can verify the validity of receiver's public key. From Step 2 to Step 5 are the process of encryption.
- SEM-Decrypt:
  Given the ciphertext $C = (C_1, C_2, C_3)$, a ID_A, **A**'s public keys (U_A, w_0, w_1, d_1), SEM performs the following steps using the SEM-key d_0:
  1) Check that ID_A is a legitimate user whose key has not been revoked.
  2) Compute ${C_1}^{d_0}$.
     ${C_1}^{d_0} = g^{r \cdot d_0} = g^{r \cdot (s_0 + xH_1(ID_A, w_0))}$
     $= g^{r \cdot s_0} \cdot g^{r \cdot xH_1(ID_A, w_0)} = {w_0}^r \cdot y^{r \cdot H_1(ID_A, w_0)}$
  3) Compute $C_2 \oplus H_5(C_1^{d_0})$.
     $C_2 \oplus H_5(C_1^{d_0})$
     $= (M||\sigma) \oplus H_4({U_A}^r) \oplus H_5(w_0^r \cdot y^{H_1(ID_A, w_0) \cdot r}) \oplus H_5(C_1^{d_0}) = (M||\sigma) \oplus H_4({U_A}^r) \oplus H_5(w_0^r \cdot y^{H_1(ID_A, w_0) \cdot r}) \oplus H_5(w_0^r \cdot y^{H_1(ID_A, w_0) \cdot r}) = (M||\sigma) \oplus H_4({U_A}^r)$
  4) Check whether $C_3 = H_6(U_A, C_2 \oplus H_5(C_1^{d_0}), C_1, C_2)$.

  If it is valid, SEM sends C_1 and $C_2' = (M||\sigma) \oplus H_4({U_A}^r)$ to **A**. Otherwise, abort SEM-Decrypt.
  In Step 1, SEM ascertains whether the user's identification information is valid. In Step 2 SEM performs the partial decryption of the ciphertext C using SEM-key. In Step 3, SEM computes token information that is needed for complete decryption in USER-Decrypt algorithm. After SEM finishes executing the partial decryption and the token generation, it performs the validity checking for the ciphertext C in Step 4. In order to prevent from the partial decryption attack, Step 4 must be required.
- USER-Decrypt:
  Given C_1 and C_2' from the SEM, **A** performs the following steps using his private key z_A:
  1) Compute ${C_1}^{z_A}$
     ${C_1}^{z_A} = g^{r \cdot z_A} = g^{z_A \cdot r} = {U_A}^r$
  2) Parse M' and σ' from $M'||\sigma' = H_4({C_1}^{z_A}) \oplus C_2'$
  3) Compute $r' = H_3(M', \sigma', ID_A, U_A)$ and $g^{r'}$
  4) Check whether $g^{r'} = C_1$

  If the verification succeeds then return the fully decrypted message $M' = M$. Otherwise, the USER-Decrypt must be aborted. In Step 1 and Step 2, user **A** fully decrypts C_2' using own private key z_A. After **A** computes the value for a validity checking in Step 3, **A** ascertains whether the decryption is successful in Step 4.

2.4 Security Analysis

The security of our mCL-PKE scheme is based on the assumed intractability of the CDH problem. The following theorem summaries the security of our scheme.

Theorem 1. Our mediated certificateless public key encryption scheme **mCL-PKE** is **IND-CCA secure** against Type I and Type II adversaries in the random oracle model, under the assumption that the **CDH** problem is intractable.

In order to prove the theorem 1, we have to consider both kinds of adversaries(Type I and Type II) to establish the chosen ciphertext security of the above mCL-PKE scheme. Thus, the theorem 1 is proved based on Lemma 1 and 2. We adopt the security proof techniques from [25].

Lemma 1. Suppose that the hash functions $H_i (i = 1, 2, 3, 4, 5, 6)$ are random oracles and there exists a Type I IND-CCA adversary $\mathcal{A_I}$ against the mCL-PKE scheme with advantage ε when running in time t, making q_{pub} public key requests queries, q_{sem} SEM-key extraction queries, q_{pri} private key extraction queries, q_{pubR} public key replacement queries, q_{D_S} SEM decryption queries, q_{D_U} USER decryption queries and q_i random oracle queries to H_i $(1 \leq i \leq 6)$. Then, for any $(0 \leq \delta \leq \varepsilon)$, there exists either an algorithm $\mathcal{B}$ to solve the CDH problem with advantage $\epsilon' \geq \frac{1}{q_5} Pr[\mathsf{Ask}H_5^*] \geq \frac{1}{q_5} (\frac{\epsilon (1-\delta)^{q_{pubR}}}{e(q_{sem}+q_{pri}+1)} -$

$\frac{q_6}{2^{k_0}} - \frac{q_4}{2^{k_0}} - \frac{q_3}{2^{k_0}} - \frac{q_{D_S}+q_{D_U}}{q}$) and running in time $T = max\{t + (q_1 + q_2 + q_3 + q_4 + q_5 + q_6)O(1) + (q_{pub}+q_{pubR}+q_{D_S}+q_{D_U})(5t_{exp}+O(1)), cq_2t/\epsilon\}$, where t_{exp} denotes the time for computing exponentiation in $\mathbb{Z}_p^*$, and c is constant greater than 120686 assuming that $\epsilon \geq 10(q_{sem}+1)(q_{sem}+q_2)/q$, or an attacker who breaks the EUF-CMA(existential unforgeability under adaptive chosen message attack) security of the Schnorr signature with advantage δ within time T.

Proof. In order to prove Lemma 1, we assume that the Schnorr signature scheme is EUF-CMA secure with advantage δ $(0 \leq \delta \leq \varepsilon)$ within time T. Let $\mathcal{A_I}$ be a Type I IND-CCA adversary against the mCL-PKE scheme. By using $\mathcal{A_I}$, we show how to construct an algorithm $\mathcal{B}$ to solve the CDH problem. Suppose that $\mathcal{B}$ is given a random instance (g, g^a, g^b) of the CDH problem. $\mathcal{B}$ sets $y = g^a$ and simulates the SetUp algorithm of the mCL-PKE scheme by supplying $\mathcal{A_I}$ with $(p, q, n, k_0, g, y, H_1, H_2, H_3, H_4, H_5, H_6)$ as public parameters, where $H_1, H_2, H_3, H_4, H_5, H_6$ are random oracles controlled by $\mathcal{B}$. $\mathcal{B}$ can simulate the Challenger's execution of each phase of the formal Game. $\mathcal{A_I}$ may make queries to random oracles $H_i (1 \leq i \leq 6)$ at any time and $\mathcal{B}$ responds as follows:

H_1 **queries**: $\mathcal{B}$ maintains a H_1 list of tuples $\langle (ID_i, w_{0i}), e_{0i}\rangle$. On receiving such a query on (ID_i, w_{0i}), $\mathcal{B}$ first check if there is a tuple $\langle (ID_i, w_{0i}), e_{0i}\rangle$ on the H_1 list. If there is, then $\mathcal{B}$ returns e_{0i}. Otherwise, $\mathcal{B}$ chooses $e_{0i} \in_R \mathbb{Z}_q^*$, adds $\langle (ID_i, w_{0i}), e_{0i}\rangle$ to the H_1 list and returns e_{0i}.

H_2 **queries**: $\mathcal{B}$ maintains a H_2 list of tuples $\langle (ID_i, w_{0i}, w_{1i}), e_{1i}\rangle$. On receiving such a query on (ID_i, w_{0i}, w_{1i}), $\mathcal{B}$ first checks if there is a tuple $\langle (ID_i, w_{0i}, w_{1i}), e_{1i}\rangle$ on the H_2 list. If there is, return e_{1i}. Otherwise, $\mathcal{B}$ picks $e_{1i} \in_R \mathbb{Z}_q^*$, adds $\langle (ID_i, w_{0i}, w_{1i}), e_{1i}\rangle$ to the H_2 list and returns e_{1i}.

H_3 **queries**: $\mathcal{B}$ maintains a H_3 list of tuples $\langle (M_i, \sigma_i, ID_i, U_i), r_i\rangle$. On receiving such a query on $(M_i, \sigma_i, ID_i, U_i)$, $\mathcal{B}$ first checks if there is a tuple $\langle (M_i, \sigma_i, ID_i, U_i), r_i\rangle$ on the H_3 list. If there is, return r_i. Otherwise, $\mathcal{B}$ picks $r_i \in_R \mathbb{Z}_q^*$ and returns r_i.

H_4 **queries**: $\mathcal{B}$ maintains a H_4 list of tuples $\langle A, h_1\rangle$. On receiving such a query on A, $\mathcal{B}$ first checks if there is a tuple $\langle A, h_1\rangle$ on the H_4 list. If there is, return h_1. Otherwise, $\mathcal{B}$ picks $h_1 \in_R \{0,1\}^{n+k_0}$, adds $\langle A, h_1\rangle$ to the H_4 list and returns h_1.

H_5 **queries**: $\mathcal{B}$ maintains a H_5 list of tuples $\langle B, h_2\rangle$. On receiving such a query on A, $\mathcal{B}$ first checks if there is a tuple $\langle B, h_2\rangle$ on the H_5 list. If there is, return h_2. Otherwise, $\mathcal{B}$ picks $h_2 \in_R \{0,1\}^{n+k_0}$, adds $\langle B, h_2\rangle$ to the H_5 list and returns h_2.

H_6 **queries**: $\mathcal{B}$ maintains a H_6 list of tuples $\langle (U_i, C, D, E), h_3\rangle$. On receiving such a query on (U_i, C, D, E), $\mathcal{B}$ first checks if there is a tuple $\langle (U_i, C, D, E), h_3\rangle$ on the H_6 list. If there is, return h_3. Otherwise, $\mathcal{B}$ picks $h_3 \in_R \{0,1\}^{n+k_0}$ and returns h_3.

Phase 1: $\mathcal{A_I}$ launches Phase 1 of its attack by making a series of requests, each of which is either a Public Key Request, a SEM-key Extraction, a Private Key Extraction, a Public Key Replacement, a SEM-Decryption or a USER-Decryption query.

Public Key Request: $\mathcal{B}$ maintains a public key list $\langle ID_i, (U_i, w_{0i}, w_{1i}, d_{1i}), coin\rangle$. On receiving such a query on ID_i, $\mathcal{B}$ responds as follows:

1) If there is a tuple $\langle ID_i, (U_i, w_{0i}, w_{1i}, d_{1i}), coin\rangle$ on the list, $\mathcal{B}$ returns $(U_i, w_{0i}, w_{1i}, d_{1i})$.
2) Otherwise, $\mathcal{B}$ picks $coin \in \{0,1\}$ with $Pr[coin = 0] = \gamma$ (γ will be determined in the **Guess**).
   - In case of $coin = 0$, choose $d_{0i}, d_{1i}, e_{0i}, e_{1i}, z_i \in_R \mathbb{Z}_q^*$, compute $w_{0i} = g^{d_{0i}} y^{-e_{0i}}, w_{1i} = g^{d_{1i}} y^{-e_{1i}}, U_i = g^{z_i}$. (Check the H_1 list and if there is a tuple $\langle (ID_i, w_{0i}), e_0\rangle$, select again $d_{0i}, e_{0i} \in_R \mathbb{Z}_q^*$; Check the H_2 list and if there is a tuple $\langle (ID_i, w_{0i}, w_{1i}), e_1\rangle$, select again $d_{1i}, e_{1i} \in_R \mathbb{Z}_q^*$.) Add $\langle (ID_i, w_{0i}), e_{0i}\rangle$ to the H_1 list, $\langle (ID_i, w_{0i}, w_{1i}), e_{1i}\rangle$ to the H_2 list, $\langle ID_i, d_{0i}, (w_{0i}, w_{1i}, d_{1i})\rangle$ to the partial key list, $\langle ID_i, z_i\rangle$ to the private key list and $\langle ID_i, (U_i, w_{0i}, w_{1i}, d_{1i})\rangle$ to the public key list. Return $(U_i, w_{0i}, w_{1i}, d_{1i})$ as answer.
   - In case of $coin = 1$, choose $s_{0i}, d_{1i}, e_{1i}, z_i \in_R \mathbb{Z}_q^*$, compute $w_{0i} = g^{s_{0i}}, w_{1i} = g^{d_{1i}} y^{-e_{1i}}, U_i = g^{z_i}$. (Check the H_2 list and if there is a tuple $\langle (ID_i, w_{0i}, w_{1i}), e_1\rangle$, select again $d_{1i}, e_{1i} \in_R \mathbb{Z}_q^*$.) Add $\langle (ID_i, w_{0i}, w_{1i}), e_{1i}\rangle$ to the H_2 list and $\langle ID_i, (U_i, w_{0i}, w_{1i}, d_{1i}), s_{0i}, coin\rangle$ to the public key list. Return $(U_i, w_{0i}, w_{1i}, d_{1i})$ as answer.

SEM-Key Extraction: $\mathcal{B}$ maintains a partial key list of tuples $\langle ID_i, d_{0i}, (w_{0i}, w_{1i}, d_{1i})\rangle$. On receiving such a query on ID_i, $\mathcal{B}$ responds as follows:

1) If a tuple $\langle ID_i, d_{0i}, (w_{0i}, w_{1i}, d_{1i})\rangle$ exists on the list, $\mathcal{B}$ returns d_{0i} as the SEM-key and (w_{0i}, w_{1i}, d_{1i}) as the partial public key.
2) Otherwise, $\mathcal{B}$ runs the simulation algorithm for public key request by using ID_i as input.
   - In case of $coin = 0$, $\mathcal{B}$ searches the partial key list of the form $\langle ID_i, d_{0i}, (w_{0i}, w_{1i}, d_{1i})\rangle$ and returns d_{0i} as the SEM-key.
   - In case of $coin = 1$, $\mathcal{B}$ aborts.

Private Key Extraction: $\mathcal{B}$ maintains a private key list of tuples $\langle ID_i, z_i\rangle$. On receiving such a query on ID_i, $\mathcal{B}$ responds as follows:

1) If a tuple $\langle ID_i, z_i\rangle$ exists on the list, $\mathcal{B}$ returns z_i as the private key.
2) Otherwise, $\mathcal{B}$ runs the simulation algorithm for public key request by using ID_i as input.
   - In case of $coin = 0$, $\mathcal{B}$ searches the partial key list $\langle ID_i, z_i\rangle$ and returns z_i as the private key.
   - In case of $coin = 1$, $\mathcal{B}$ aborts.

Public Key Replacement: $\mathcal{A_I}$ can replace the public key of any user ID_i, $(U_i, w_{0i}, w_{1i}, d_{1i})$ with any value $(U_i', w_{0i}', w_{1i}', d_{1i}')$ of its choice. If $(w_{0i}', w_{1i}', d_{1i}') \neq$

(w_{0i}, w_{1i}, d_{1i}) but it satisfies $g^{d'_{1i}} = w'_{1i} \cdot y^{H_2(ID_i, w'_{0i}, w'_{1i})}$, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ records the change.

SEM-Decryption queries: $\mathcal{A_I}$ can request $\mathcal{B}$ to decrypt partially ciphertext C for ID_i. $\mathcal{B}$ searches the public key list for a tuple $\langle ID_i, (U_i, w_{0i}, w_{1i}, d_{1i}), coin\rangle$. Then $\mathcal{B}$ responds as follows:

1) If the public key has not been replaced and $coin = 0$,
   - $\mathcal{B}$ searches the partial key list for a tuple $\langle ID_i, d_{0i}, (w_{0i}, w_{1i}, d_{1i})\rangle$.
   - $\mathcal{B}$ computes ${C_1}^{d_{0i}}$ and $C_2 \oplus H_5(C_1^{d_{0i}})$
   - $\mathcal{B}$ checks whether $C_3 = H_6(U_i, C_2 \oplus H_5(C_1^{d_{0i}}), C_1, C_2)$. If it is valid, $\mathcal{B}$ returns $C'_2 = C_2 \oplus H_5(C_1^{d_{0i}})$. Otherwise, $\mathcal{B}$ outputs $\perp$.
2) Otherwise, $\mathcal{B}$ searches the H_3 list for a tuple $\langle (M_i, \sigma_i, ID_i, U_i), r_i\rangle$, where $C_1 = g^{r_i}$, $C_2 = (M_i||\sigma_i) \oplus H_4({U_i}^{r_i}) \oplus H_5(w_{0i}^{r_i} \cdot y^{H_1(ID_i, w_{0i}) \cdot r_i})$ and $C_3 = H_6(U_i, (M_i||\sigma_i) \oplus H_4(U_i^{r_i}), C_1, C_2)$. $\mathcal{B}$ returns the corresponding $C'_2 = (M_i||\sigma_i) \oplus H_4({U_i}^{r_i})$ if such a tuple exists. Otherwise, $\mathcal{B}$ outputs $\perp$.

USER-Decryption queries: This query should be performed after SEM-Decryption query performing. $\mathcal{A_I}$ can request $\mathcal{B}$ to perform User-decryption for C_1, C'_2. $\mathcal{B}$ searches the public key list for a tuple $\langle ID_i, (U_i, w_{0i}, w_{1i}, d_{1i}), coin\rangle$. Then $\mathcal{B}$ responds as follows:

1) If the public key has not been replaced and $coin = 0$,
   - $\mathcal{B}$ searches private key list of tuples $\langle ID_i, z_i\rangle$.
   - $\mathcal{B}$ computes ${C_1}^{z_i}$ and $H_4({C_1}^{z_i})$.
   - $\mathcal{B}$ parses M' and σ' from $M'||\sigma' = H_4({C_1}^{z_i}) \oplus C'_2$.
   - $\mathcal{B}$ computes $r' = H_3(M'||\sigma'||ID_i||U_i)$ and $g^{r'}$.
   - $\mathcal{B}$ checks whether $g^{r'} = C_1$. If $g^{r'} = C_1$, it returns $M' = M$. Otherwise, $\mathcal{B}$ outputs $\perp$.
2) Otherwise, $\mathcal{B}$ searches the H_3 list for a tuple $\langle (M_i, \sigma_i, ID_i, U_i), r_i\rangle$, where $C_1 = g^{r_i}$, $C_2 = (M_i||\sigma_i) \oplus H_4({U_i}^{r_i}) \oplus H_5(w_{0i}^{r_i} \cdot y^{H_1(ID_i, w_{0i}) \cdot r_i})$ and $C_3 = H_6(U_i, (M_i||\sigma_i) \oplus H_4(U_i^{r_i}), C_1, C_2)$. $\mathcal{B}$ returns the corresponding M_i if such a tuple exists. Otherwise, $\mathcal{B}$ outputs $\perp$.

Challenge Phase: $\mathcal{A_I}$ outputs ID^* and two messages M_0, M_1 on which it wishes to be challenged. On receiving ID^*, $\mathcal{B}$ searches the public key list for the tuple $\langle ID^*, (U^*, w_0^*, w_1^*, d_1^*), coin\rangle$. Then $\mathcal{B}$ responds as follows:

1) If $coin = 0$, $\mathcal{B}$ aborts the game.
2) Otherwise, $\mathcal{B}$ performs the following actions:
   - $\mathcal{B}$ picks $\sigma^* \in_R \{0,1\}^{k_0}$, $\beta \in_R \{0,1\}$ and $C_2^*, C_3^* \in_R \{0,1\}^{n+k_0}$.
   - $\mathcal{B}$ sets $C_1^* = g^b$, $e_0^* = H_1(ID^*, w_0^*)$, $b = H_3(M_\beta, \sigma^*, ID^*, U^*), c = H_4(U^{*b})$, $H_5(w_0^{*b} \cdot y^{e_0^* b}) = C_2^* \oplus (M_\beta||\sigma^*) \oplus c$ and $H_6(U^*||(M_\beta||\sigma^*) \oplus c||C_1^*||C_2^*) = C_3^*$.
   - $\mathcal{B}$ outputs $\langle C_1^*, C_2^*, C_3^*\rangle$ as the challenge ciphertext. According to the above construction, $C_2^* = (M_\beta||\sigma^*) \oplus H_4(U^{*b}) \oplus H_5(w_0^{*b} \cdot y^{e_0^* b}) = (M_\beta||\sigma^*) \oplus H_4(U^{*b}) \oplus H_5(g^{bs_0^*} \cdot g^{abe_0^*})$

Phase 2: $\mathcal{B}$ continues to respond to $\mathcal{A_I}$'s requests in the same way as it did in Phase 1. $\mathcal{A_I}$ cannot make a SEM-Key Extraction query or a Private Key Extraction query on ID^*. For the combination of ID^* and $(U^*, w_0^*, w_1^*, d_1^*)$ used to encrypt M_β, $\mathcal{A_I}$ should not make decryption query on $\langle C_1^*, C_2^*, C_3^*\rangle$.

Guess: Eventually, $\mathcal{A_I}$ outputs its guess. $\mathcal{B}$ chooses a random pair $\langle B, h\rangle$ from the H_5 list and outputs $(\frac{B}{g^{bs_0^*}})^{\frac{1}{e_0^*}} (= g^{ab})$ as the solution to the CDH problem.

Analysis. First of all, we evaluate the simulation of the random oracles given above. It is evident that the simulations of H_1 and H_2 are perfect through the constructions of H_1 and H_2. Moreover, as long as $\mathcal{A_I}$ does not query $(M_\beta, \sigma^*, ID^*, U^*)$ to H_3 nor U^{*b} to H_4 nor $w_0^{*b} \cdot y^{e_0^* b}$ to H_5 nor $U^*||(M_\beta||\sigma^*) \oplus c||C_1^*||C_2^*$ to H_6, the simulations of H_3, H_4, H_5 and H_6 are perfect. Let $\mathsf{Ask}H_3^*$ denote the event that $(M_\beta, \sigma^*, ID^*, U^*)$ has been queried to H_3, $\mathsf{Ask}H_4^*$ denote the event that U^{*b} has been queried to H_4, $\mathsf{Ask}H_5^*$ denote the event that $w_0^{*b} \cdot y^{e_0^* b}$ has been queried to H_5 and $\mathsf{Ask}H_6^*$ denote the event that $U^*||(M_\beta||\sigma^*) \oplus c||C_1^*||C_2^*$ has been queried to H_6.

The simulated challenge ciphertext is identically distributed as the real one, because H_3, H_4, H_5 and H_6 are random oracles. Now we evaluate the simulation of the decryption oracle. As to the simulation of decryption oracle, $\mathcal{B}$ will wrongly reject a valid ciphertext during the simulation with probability smaller than $\frac{q_{D_S} + q_{D_U}}{q}$. That is, $Pr[DecErr] \leq \frac{q_{D_S} + q_{D_U}}{q}$, where $DecErr$ denotes the event that $\mathcal{B}$ rejects a valid ciphertext during the simulation.

Let $E = (\mathsf{Ask}H_6^* \vee \mathsf{Ask}H_5^* \vee \mathsf{Ask}H_4^* \vee \mathsf{Ask}H_3^* \vee DecErr)|\neg Abort$. If E does not happen during the simulation, $\mathcal{B}$ will not gain any advantage greater than 1/2 to guess β, because of the randomness of the output of H_5.

In other words, $Pr[\beta' = \beta|\neg E] \leq 1/2$. We obtain $Pr[\beta' = \beta] = Pr[\beta' = \beta|\neg E]Pr[\neg E] + Pr[\beta' = \beta|E]Pr[E] \leq \frac{1}{2}Pr[\neg E] + Pr[E] = \frac{1}{2} + \frac{1}{2}Pr[E]$

By definition of ϵ, we have $\epsilon \leq 2(Pr[\beta' = \beta] - \frac{1}{2}) \leq Pr[E] \leq \frac{Pr[\mathsf{Ask}H_6^*] + Pr[\mathsf{Ask}H_5^*] + Pr[\mathsf{Ask}H_4^*] + Pr[\mathsf{Ask}H_3^*] + Pr[DecErr]}{Pr[\neg Abort]}$

The probability that $\mathcal{B}$ does not abort during the simulation is given by $\gamma^{q_{sem}+q_{pri}}(1-\gamma)(1-\delta)^{q_{pubR}}$. This probability is maximized at $\gamma = 1 - \frac{1}{q_{sem}+q_{pri}+1}$. Therefore, we have $Pr[\neg Abort] \geq \frac{(1-\delta)^{q_{pubR}}}{e(q_{sem}+q_{pri}+1)}$, where e denotes the base of the natural logarithm.

Hence, we obtain the following $Pr[\mathsf{Ask}H_5^*] \geq \epsilon Pr[\neg Abort] - Pr[\mathsf{Ask}H_6^*] - Pr[\mathsf{Ask}H_4^*] - Pr[\mathsf{Ask}H_3^*] - Pr[DecErr]$

$$\geq \frac{\epsilon(1-\delta)^{q_{pubR}}}{e(q_{sem}+q_{pri}+1)} - \frac{q_6}{2^{k_0}} - \frac{q_4}{2^{k_0}} - \frac{q_3}{2^{k_0}} - \frac{q_{D_S}+q_{D_U}}{q}$$

If $\mathsf{Ask}H_5^*$ happens, then $\mathcal{A}_\mathcal{I}$ will be able to distinguish the simulation from the real one. $\mathcal{A}_\mathcal{I}$ can tell that the challenge ciphertext C^* by the simulation is invalid. $H_5(w_0^{*b} \cdot y^{e_0^* b})$ has been recorded on the H_5 list. Then, $\mathcal{B}$ wins if it chooses the correct element from the H_5 list. Therefore, we obtain the advantage for $\mathcal{B}$ to solve the CDH problem.

$\epsilon' \geq \frac{1}{q_5} Pr[\mathsf{Ask}H_5^*] \geq \frac{1}{q_5}(\frac{\epsilon(1-\delta)^{q_{pubR}}}{e(q_{sem}+q_{pri}+1)} - \frac{q_6}{2^{k_0}} - \frac{q_4}{2^{k_0}} - \frac{q_3}{2^{k_0}} - \frac{q_{D_S}+q_{D_U}}{q})$

The running time of the $\mathcal{B}$ who is the CDH attacker is bounded by $T = max\{t + (q_1 + q_2 + q_3 + q_4 + q_5 + q_6)O(1) + (q_{pub} + q_{pubR} + q_{D_S} + q_{D_U})(5t_{exp} + O(1)), cq_2 t/\epsilon\}$, where t_{exp} denotes the time for computing exponentiation in $\mathbb{Z}_p^*$, and c is constant greater than 120686 assuming that $\epsilon \geq 10(q_{sem}+1)(q_{sem}+q_2)/q$. This estimation is from the result of [21].

Lemma 2. Suppose that the hash functions $H_i (i = 1, 2, 3, 4, 5, 6)$ are random oracles and there exists a Type II IND-CCA adversary $\mathcal{A}_\mathcal{II}$ against the mCL-PKE scheme with advantage ε when running in time t, making q_{pub} public key requests queries, q_{sem} SEM-key extraction queries, q_{pri} private key extraction queries, q_{pubR} public key replacement queries, q_{D_S} SEM decryption queries, q_{D_U} USER decryption queries and q_i random oracle queries to H_i $(1 \leq i \leq 6)$. Then, there exists an algorithm $\mathcal{B}$ to solve the CDH problem with advantage $\epsilon' \geq \frac{1}{q_4} Pr[\mathsf{Ask}H_4^*] \geq \frac{1}{q_4}(\frac{\epsilon}{e(q_{pri}+1)} - \frac{q_6}{2^{k_0}} - \frac{q_5}{2^{k_0}} - \frac{q_3}{2^{k_0}} - \frac{q_{D_S}+q_{D_U}}{q})$ running in time $T < t + (q_1 + q_2 + q_3 + q_4 + q_5 + q_6)O(1) + (q_{pub} + q_{D_S} + q_{D_U})(4t_{exp} + O(1))$, where t_{exp} denotes the time for computing exponentiation in $\mathbb{Z}_p^*$.

Proof. Let $\mathcal{A}_\mathcal{II}$ be a Type II IND-CCA adversary against the mCL-PKE scheme. By using $\mathcal{A}_\mathcal{II}$, we show how to construct an algorithm $\mathcal{B}$ to solve the CDH problem. Suppose that $\mathcal{B}$ is given a random instance (g, g^a, g^b) of the CDH problem. $\mathcal{B}$ chooses $x \in_R \mathbb{Z}_q^*$, computes $y = g^x$ and simulates the SetUp algorithm of the mCL-PKE scheme by supplying $\mathcal{A}_\mathcal{II}$ with $(p, q, n, k_0, g, y, H_1, H_2, H_3, H_4, H_5, H_6)$, where $H_1, H_2, H_3, H_4, H_5, H_6$ are random oracles controlled by $\mathcal{B}$. $\mathcal{B}$ can simulate the Challenger's execution of each phase of Game. $\mathcal{A}_\mathcal{II}$ may make queries to $H_i (1 \leq i \leq 6)$ at any time during its attack and $\mathcal{B}$ responds as follows:

H_1 **queries**: $\mathcal{B}$ maintains a H_1 list of tuples $\langle (ID_i, w_{0i}), e_{0i} \rangle$. On receiving a query on (ID_i, w_{0i}), $\mathcal{B}$ does the following:

1) If $\langle (ID_i, w_{0i}), e_{0i} \rangle$ is on the H_1 list, $\mathcal{B}$ returns e_{0i}.
2) Otherwise, $\mathcal{B}$ chooses $e_{0i} \in_R \mathbb{Z}_q^*$, adds $\langle (ID_i, w_{0i}), e_{0i} \rangle$ to the H_1 list and returns e_{0i}.

H_2 **queries**: $\mathcal{B}$ maintains a H_2 list of tuples $\langle (ID_i, w_{0i}, w_{1i}), e_{1i} \rangle$. On receiving a query on (ID_i, w_{0i}, w_{1i}), $\mathcal{B}$ first checks if $\langle (ID_i, w_{0i}, w_{1i}), e_{1i} \rangle$ is on the H_2 list, return e_{1i}. Otherwise, $\mathcal{B}$ picks $e_{1i} \in_R \mathbb{Z}_q^*$, adds $\langle (ID_i, w_{0i}, w_{1i}), e_{1i} \rangle$ to the H_2 and returns e_{1i}.

H_3 **queries**: $\mathcal{B}$ maintains a H_3 list of tuples $\langle (M_i, \sigma_i, ID_i, U_i), r_i \rangle$. On receiving a query on $(M_i, \sigma_i, ID_i, U_i)$, $\mathcal{B}$ first checks if $\langle (M_i, \sigma_i, ID_i, U_i), r_i \rangle$ is on the H_3 list, return r_i. Otherwise, $\mathcal{B}$ picks $r_i \in_R \mathbb{Z}_q^*$ and returns r_i.

H_4 **queries**: $\mathcal{B}$ maintains a H_4 list of tuples $\langle A, h_1 \rangle$. On receiving a query on A, $\mathcal{B}$ first checks if $\langle A, h_1 \rangle$ is on the H_4 list, return h_1. Otherwise, $\mathcal{B}$ picks $h_1 \in_R \{0,1\}^{n+k_0}$, adds $\langle A, h_1 \rangle$ to the H_4 and returns h_1.

H_5 **queries**: $\mathcal{B}$ maintains a H_5 list of tuples $\langle B, h_2 \rangle$. On receiving a query on A, $\mathcal{B}$ first checks if $\langle B, h_2 \rangle$ is on the H_5 list, return h_2. Otherwise, $\mathcal{B}$ picks $h_2 \in_R \{0,1\}^{n+k_0}$, adds $\langle B, h_2 \rangle$ to the H_5 list and returns h_2.

H_6 **queries**: $\mathcal{B}$ maintains a H_6 list of tuples $\langle (U_i, C, D, E), h_3 \rangle$. On receiving a query on (U_i, C, D, E), $\mathcal{B}$ first checks if $\langle (U_i, C, D, E), h_3 \rangle$ is on the H_6 list, return h_3. Otherwise, $\mathcal{B}$ picks $h_3 \in_R \{0,1\}^{n+k_0}$ and returns h_3.

Phase 1: $\mathcal{A}_\mathcal{II}$ launches Phase 1 of its attack by making a series of requests, each of which is either a Public Key Request, a Private Key Extraction, a SEM-Decryption or a USER-Decryption query.

Compute SEM-Key: $\mathcal{A}_\mathcal{II}$ computes the SEM-key d_{0i} and the partial public key (w_{0i}, w_{1i}, d_{1i}) for ID_i, $\mathcal{B}$ keeps $\langle ID_i, d_{0i}, (w_{0i}, w_{1i}, d_{1i}) \rangle$ to the partial key list.

Public Key Request: $\mathcal{B}$ maintains a public key list of tuples $\langle ID_i, (U_i, w_{0i}, w_{1i}, d_{1i}), coin \rangle$. On receiving a query on ID_i, $\mathcal{B}$ responds as follows:

1) If $\langle ID_i, (U_i, w_{0i}, w_{1i}, d_{1i}), coin \rangle$ is on the public key list, $\mathcal{B}$ returns $(U_i, w_{0i}, w_{1i}, d_{1i})$.
2) Otherwise, $\mathcal{B}$ picks $coin \in \{0, 1\}$ with $Pr[coin = 0] = \gamma$ (γ will be determined later).
   - In case of $coin = 0$, $\mathcal{B}$ chooses $z_i \in_R \mathbb{Z}_q^*$, compute $U_i = g^{z_i}$. Then, it searches the partial key list to get the partial public key (w_{0i}, w_{1i}, d_{1i}), adds $\langle ID_i, (U_i, w_{0i}, w_{1i}, d_{1i}), z_i, coin \rangle$ to the public key list and returns $(U_i, w_{0i}, w_{1i}, d_{1i})$.
   - In case of $coin = 1$, $\mathcal{B}$ sets $U_i = g^a$. Then, it searches the partial key list to get the partial public key (w_{0i}, w_{1i}, d_{1i}), adds $\langle ID_i, (U_i, w_{0i}, w_{1i}, d_{1i}), ?, coin \rangle$ to the public key list and returns $(U_i, w_{0i}, w_{1i}, d_{1i})$.

Private Key Extraction: $\mathcal{B}$ maintains a private key list of tuples $\langle ID_i, z_i \rangle$. On receiving a query on ID_i, $\mathcal{B}$ responds as follows:

1) If $\langle ID_i, z_i \rangle$ exists on the private key list, $\mathcal{B}$ returns z_i.
2) Otherwise, $\mathcal{B}$ runs the simulation algorithm for public key request by using ID_i as input in order to get a tuple $\langle ID_i, (U_i, w_{0i}, w_{1i}, d_{1i}), z_i, coin \rangle$.
   - In case of $coin = 0$, $\mathcal{B}$ returns z_i.
   - In case of $coin = 1$, $\mathcal{B}$ aborts.

SEM-Decryption queries: $\mathcal{A}_\mathcal{II}$ can request $\mathcal{B}$ to decrypt partially $C = (C_1, C_2, C_3)$ for ID_i. $\mathcal{B}$ runs the simulation algorithm for public key request taking

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

ID_i as input to get $\langle ID_i, (U_i, w_{0i}, w_{1i}, d_{1i}), z_i, coin\rangle$. Then, $\mathcal{B}$ performs the following:

1) If $coin = 0$, $\mathcal{B}$ searches the partial key list and the private key list for a tuple $\langle ID_i, (d_{0i}, z_i)\rangle$. Then, it computes ${C_1}^{d_{0i}}$ and $C_2 \oplus H_4(C_1^{d_{0i}})$. $\mathcal{B}$ checks whether $C_3 = H_6(U_i, C_2 \oplus H_4(C_1^{d_{0i}}), C_1, C_2)$. If it is valid, $\mathcal{B}$ returns $C_2' = C_2 \oplus H_4(C_1^{d_{0i}})$. Otherwise, $\mathcal{B}$ outputs $\perp$.
2) Otherwise, $\mathcal{B}$ searches the H_3 list for a tuple $\langle (M_i, \sigma_i, ID_i, U_i), r_i\rangle$ satisfying $C_1 = g^{r_i}$, $C_2 = (M_i||\sigma_i) \oplus H_4({U_i}^{r_i}) \oplus H_5(w_{0i}^{r_i} \cdot y^{H_1(ID_i, w_{0i})\cdot r_i})$ and $C_3 = H_6(U_i, (M_i||\sigma_i) \oplus H_4(U_i^{r_i}), C_1, C_2)$. $\mathcal{B}$ returns the corresponding $C_2' = (M_i||\sigma_i) \oplus H_4({U_i}^{r_i})$ if such a tuple exists. Otherwise, $\mathcal{B}$ outputs $\perp$.

USER-Decryption queries: This query should be performed after SEM-Decryption query performing. $\mathcal{A_{II}}$ can request $\mathcal{B}$ to perform User-decryption for C_1, C_2' for ID_i. $\mathcal{B}$ searches the public key list taking ID_i as input to get $\langle ID_i, (U_i, w_{0i}, w_{1i}, d_{1i}), z_i, coin\rangle$. Then $\mathcal{B}$ responds as follows:

1) If $coin = 0$,
   - $\mathcal{B}$ searches private key list of tuples $\langle ID_i, z_i\rangle$.
   - $\mathcal{B}$ computes ${C_1}^{z_i}$ and $H_4({C_1}^{z_i})$.
   - $\mathcal{B}$ parses M' and σ' from $M'||\sigma' = H_4({C_1}^{z_i}) \oplus C_2'$.
   - $\mathcal{B}$ computes $r' = H_3(M'||\sigma'||ID_i||U_i)$, $g^{r'}$.
   - $\mathcal{B}$ checks if $g^{r'} = C_1$, it returns $M' = M$. Otherwise, $\mathcal{B}$ outputs $\perp$.
2) Otherwise, $\mathcal{B}$ searches the H_3 list for $\langle (M_i, \sigma_i, ID_i, U_i), r_i\rangle$ satisfying $C_1 = g^{r_i}$, $C_2 = (M_i||\sigma_i) \oplus H_4({U_i}^{r_i}) \oplus H_5(w_{0i}^{r_i} \cdot y^{H_1(ID_i, w_{0i})\cdot r_i})$ and $C_3 = H_6(U_i, (M_i||\sigma_i) \oplus H_4(U_i^{r_i}), C_1, C_2)$. $\mathcal{B}$ returns the corresponding M_i if such a tuple exists. Otherwise, outputs $\perp$.

Challenge Phase: $\mathcal{A_{II}}$ outputs ID^* and M_0, M_1 on which it wishes to be challenged. $\mathcal{B}$ runs the simulation algorithm for public key request taking ID^* as input to get $\langle ID^*, (U^*, w_0^*, w_1^*, d_1^*), z_i, coin\rangle$. Then $\mathcal{B}$ performs as follows:

1) If $coin = 0$, $\mathcal{B}$ aborts the game.
2) Otherwise, $\mathcal{B}$ performs the following actions:
   - $\mathcal{B}$ picks $\sigma^* \in_R \{0,1\}^{k_0}$, $\beta \in_R \{0,1\}$ and $C_2^*, C_3^* \in_R \{0,1\}^{n+k_0}$.
   - $\mathcal{B}$ sets $C_1^* = g^b$, $e_0^* = H_1(ID^*, w_0^*)$, $b = H_3(M_\beta, \sigma^*, ID^*, U^*), c = H_4(U^{*b})$, $H_5(w_0^{*b} \cdot y^{e_0^* b}) = C_2^* \oplus (M_\beta||\sigma^*) \oplus c$ and $H_6(U^*||(M_\beta||\sigma^*) \oplus c||C_1^*||C_2^*) = C_3^*$.
   - $\mathcal{B}$ outputs $\langle C_1^*, C_2^*, C_3^*\rangle$ as the challenge ciphertext. According to the above construction, $C_2^* = (M_\beta||\sigma^*) \oplus H_4(U^{*b}) \oplus H_5(w_0^{*b} \cdot y^{e_0^* b}) = (M_\beta||\sigma^*) \oplus H_4(g^{ab}) \oplus H_5(w_0^{*b} \cdot y^{e_0^* b})$

Phase 2: $\mathcal{B}$ continues to respond to $\mathcal{A_{II}}$'s requests in the same way as it did in Phase 1. $\mathcal{A_{II}}$ cannot make a Private Key Extraction queries on ID^*. For ID^*, if any decryption query is equal to the challenge ciphertext $\langle C_1^*, C_2^*, C_3^*\rangle$, then $\mathcal{B}$ aborts.

Guess: Eventually, $\mathcal{A_{II}}$ outputs its guess. $\mathcal{B}$ chooses a random pair $\langle A, h\rangle$ from the H_4 list and outputs $U^{*b} (= g^{ab})$ as the solution to the CDH problem.

Analysis. First of all, we evaluate the simulation of the random oracles given above. It is evident that the simulations of H_1 and H_2 are perfect through the constructions of H_1 and H_2. Moreover, as long as $\mathcal{A_{II}}$ does not query $(M_\beta, \sigma^*, ID^*, U^*)$ to H_3 nor U^{*b} to H_4 nor $w_0^{*b} \cdot y^{e_0^* b}$ to H_5 nor $U^*||(M_\beta||\sigma^*) \oplus c||C_1^*||C_2^*$ to H_6, the simulations of H_3, H_4, H_5 and H_6 are perfect.

Let $\mathsf{Ask}H_3^*$ denote the event that $(M_\beta, \sigma^*, ID^*, U^*)$ has been queried to H_3, $\mathsf{Ask}H_4^*$ denote the event that U^{*b} has been queried to H_4, $\mathsf{Ask}H_5^*$ denote the event that $w_0^{*b} \cdot y^{e_0^* b}$ has been queried to H_5 and $\mathsf{Ask}H_6^*$ denote the event that $U^*||(M_\beta||\sigma^*) \oplus c||C_1^*||C_2^*$ has been queried to H_6.

The simulated challenge ciphertext is identically distributed as the real one, because H_3, H_4, H_5 and H_6 are random oracles. Now we evaluate the simulation of the decryption oracle. As to the simulation of decryption oracle, $\mathcal{B}$ will wrongly reject a valid ciphertext during the simulation with probability smaller than $\frac{q_{D_S}+q_{D_U}}{q}$. That is, $Pr[DecErr] \le \frac{q_{D_S}+q_{D_U}}{q}$, where $DecErr$ denotes the event that $\mathcal{B}$ rejects a valid ciphertext during the simulation. Let $E = (\mathsf{Ask}H_6^* \vee \mathsf{Ask}H_5^* \vee \mathsf{Ask}H_4^* \vee \mathsf{Ask}H_3^* \vee DecErr)|\neg Abort$. If E does not happen during the simulation, $\mathcal{B}$ will not gain any advantage greater than $1/2$ to guess β, because of the randomness of the output of H_4.
In other words, $Pr[\beta' = \beta|\neg E] \le 1/2$. We obtain $Pr[\beta' = \beta] = Pr[\beta' = \beta|\neg E]Pr[\neg E] + Pr[\beta' = \beta|E]Pr[E] \le \frac{1}{2}Pr[\neg E] + Pr[E] = \frac{1}{2} + \frac{1}{2}Pr[E]$ By definition of ϵ, we have $\epsilon \le 2(Pr[\beta' = \beta] - \frac{1}{2}) \le Pr[E] \le \frac{Pr[\mathsf{Ask}H_6^*]+Pr[\mathsf{Ask}H_5^*]+Pr[\mathsf{Ask}H_4^*]+Pr[\mathsf{Ask}H_3^*]+Pr[DecErr]}{Pr[\neg Abort]}$ The probability that $\mathcal{B}$ does not abort during the simulation is given by $\gamma^{q_{pri}}(1-\gamma)$. This probability is maximized at $\gamma = 1 - \frac{1}{q_{pri}+1}$. Therefore, we have $Pr[\neg Abort] \ge \frac{1}{e(q_{pri}+1)}$, where e denotes the base of the natural logarithm. Hence, we obtain the following $Pr[\mathsf{Ask}H_4^*] \ge \epsilon Pr[\neg Abort] - Pr[\mathsf{Ask}H_6^*] - Pr[\mathsf{Ask}H_5^*] - Pr[\mathsf{Ask}H_3^*] - Pr[DecErr]$

$$\ge \frac{\epsilon}{e(q_{pri}+1)} - \frac{q_6}{2^{k_0}} - \frac{q_5}{2^{k_0}} - \frac{q_3}{2^{k_0}} - \frac{q_{D_S}+q_{D_U}}{q}$$

If $\mathsf{Ask}H_4^*$ happens, then $\mathcal{A_{II}}$ will be able to distinguish the simulation from the real one. $\mathcal{A_{II}}$ can tell that the challenge ciphertext C^* is invalid. $H_4(U^{*b})$ has been recorded on the H_4 list. Then, $\mathcal{B}$ wins if it chooses the correct element from the H_4 list. Therefore, we obtain the advantage for $\mathcal{B}$ to solve the CDH problem. $\epsilon' \ge \frac{1}{q_4}Pr[\mathsf{Ask}H_4^*] \ge \frac{1}{q_4}(\frac{\epsilon}{e(q_{pri}+1)} - \frac{q_6}{2^{k_0}} - \frac{q_5}{2^{k_0}} - \frac{q_3}{2^{k_0}} - \frac{q_{D_S}+q_{D_U}}{q})$

The running time of the $\mathcal{B}$ who is the CDH attacker is bounded by $T < t + (q_1+q_2+q_3+q_4+q_5+q_6)O(1) + (q_{pub} + q_{D_S} + q_{D_U})(4t_{exp} + O(1))$, where t_{exp} denotes the time for computing exponentiation in $\mathbb{Z}_p^*$.$\square$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

3 SECURE CLOUD STORAGE

In this section we provide a detailed description of our system for privacy preserving cloud storage using our mCL-PKE scheme.

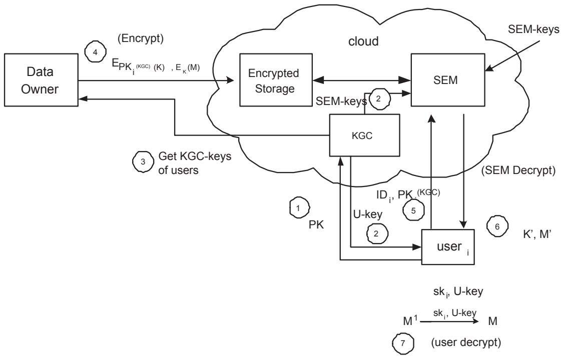

Fig. 4. The overall system

As shown in Figure 4, our scheme consists of three entities: *data owner*, *cloud*, and *users*. The data owner possesses sensitive content that it wants to share with authorized users by storing it in the public cloud and requesting the cloud to partially decrypt the encrypted content when users request the data. The cloud consists of three main services: an encrypted content storage; a key generation center (KGC), which generates public/private key pairs for each user as explained in Section 2; and a security mediation server (SEM), which acts as a security mediator for each data request and partially decrypts encrypted data for authorized users. The cloud is trusted to perform the security mediation service and key generation correctly, but it is not trusted for the confidentiality of the content and key escrowing. Our approach allows one to have most of the key generation and management functionality deployed in the untrusted cloud as our mCL-PKE scheme does not have the problem of key escrowing and thus the KGC is unable to learn the full private keys of users.

Our scheme consists of four phases: (1) Cloud set up; (2) User registration; (3) Data encryption and uploading and (4) Data decryption. Now we describe each of these phases in detail.

3.1 Cloud set up

The KGC in the cloud runs the SetUp operation of the mCL-PKE scheme and generates the master key *MK* and the system parameters *params*. It should be noted that this setup operation is a one-time task.

3.2 User registration

Each user first generates its own private and public key pair, called SK and PK, using the SetPrivateKey and SetPublicKey operations respectively using our mCL-PKE scheme. The user then sends its public keys and its identity (ID) to the KGC in the cloud. The KGC in turn generates two partial keys and a public key for the user. One partial key, referred to as *SEM-key*, is stored at the SEM in the cloud. The other partial key, referred to as *U-key*, is given to the user. The public key, referred to as *KGC-key*, consists of the user generated public key as well as the KGC generated public key. The KGC-key is used to encrypt data. The SEM-key, U-key, and SK are used together to decrypt encrypted data. We denote the partial private key and the public key for user$_i$ as SEM-key$_i$, U-key$_i$, KGC-key$_i$ respectively.

3.3 Data encryption and uploading

The data owner obtains the KGC-keys of users from the KGC in the cloud. The data owner then symmetrically encrypts each data item for which the same access control policy applies using a random session key K and then the data owner encrypts K using the KGC-keys of users. The encrypted data along with the access control list is uploaded to the cloud. The encrypted content is stored in the storage service in the cloud and the access control list, signed by the data owner, is stored in the SEM in the cloud.

3.4 Data retrieval and decryption

When a user wants to read some data, it sends a request to the SEM to obtain the partially decrypted data. The SEM first checks if the user is in the access control list and if the user's KGC-key encrypted content is available in the cloud storage. If the verification is successful, the SEM retrieves the encrypted content from the cloud and partially decrypts the content using the SEM-key for the user. The partial decryption at the SEM reduces the load on users. The user uses its SK and U-key to fully decrypt the data.

In order to improve the efficiency of the system, once the initial partial decryption for each user is performed, the SEM stores back the partially decrypted data in the cloud storage.

If a user is revoked, the data owner updates the access control list at the SEM so that future access requests by the user are denied. If a new user is added to the system, the data owner encrypts the data using the public key of the user and uploads the encrypted data along with the updated access control list to the cloud. Note that existing users are not affected by revoking or adding users to the system.

4 IMPROVED SECURE CLOUD STORAGE

In our basic scheme, the data owner has to encrypt the same data encryption key multiple times for each authorized user. This can be a huge bottleneck at the data owner if many users are authorized to access the same data as the number of mCL-PKE encryptions is proportional to the number of authorized users. We provide an extension to our basic mCL-PKE scheme so that the data owner encrypts the data encryption key once for a data item and provides some additional information to the cloud so that authorized users can decrypt the content using their private keys. The idea is similar to Proxy Re-Encryption (PRE) where the content encrypted using the data owner's public key is allowed to be decrypted by different private keys after some transformation by the cloud which acts as the proxy. However, in our improved scheme, the cloud simply acts as a storage for the proxy keys, referred to as intermediate keys, and gives these keys to users at the time of data requests.

Now we give the details of the extension. Let the data owner's private and public key pair be z_O and $U_O = g^{z_O}$ respectively, where g is a generator of $\mathbb{Z}_p^*$ with order q and z_O is a random number in $\mathbb{Z}_q^*$. The following modifications to the basic mCL-PKE scheme are performed to support single encryption at the data owner per data item.

- Encrypt: Along with $C_1 = g^r$, where r is computed as in the second step of Encrypt operation of the basic mCL-PKE scheme, the data owner computes the intermediate key INT-Key$_i$ for each authorized user$_i$, $\{g^{rz_oz_i} | i = 1, 2, \cdots, m\}$ and gives the keys to the cloud. Unlike the typical PRE schemes, the transformation at the cloud does not utilize the intermediate keys. The intermediate keys are given to authorized users when they request for data.
- USER-Decrypt: A user$_i$ having INT-Key$_i$ (= $g^{rz_oz_i}$) can compute ${U_O}^r$ using its private key, z_i, as follows and perform the decryption using this value and the public key of the data owner.

$$(g^{rz_oz_i})^{1/z_i} = {U_O}^r$$

Notice that the knowledge of ${U_O}^r$ allows user$_i$ to decrypt the message encrypted using the the data owner's public key following the steps in the User-Decrypt operation in the basic mCL-PKE scheme.

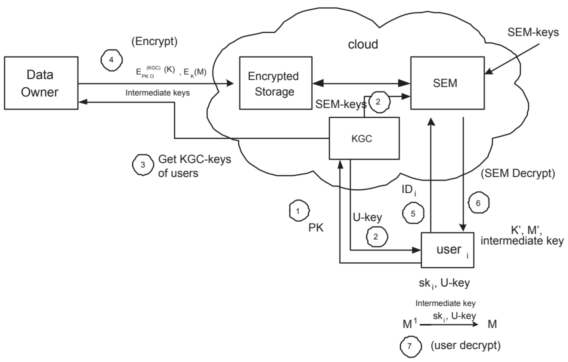

Fig. 5. The Overall System with Intermediate Keys

Figure 5 shows the overall system with the utilization of intermediate keys. The phases in this approach are very similar to those of the basic approach presented in Section 3 except for the following differences.

- During the data encryption and download phases, the data owner downloads the public keys of users to generate the intermediate keys as shown above. Unlike the basic approach, the data owner encrypts each data item only once using a random symmetric key K and then mCL-PKE encrypts K using its public key. The data owner uploads the encrypted data along with the intermediate keys to the cloud. The encrypted data is stored in the storage service in the cloud and the intermediate keys are stored at the SEM in the cloud.
- During the data retrieval and decryption phases, upon successful authorization, the SEM partially decrypts the data encrypted using the data owner's public key as input to the SEM-decryption operation of the basic mCL-PKE scheme, and gives the partially decrypted data along with the intermediate keys. The intermediate keys along with private keys allow users to fully decrypt the partially decrypted data using User-Decrypt operation of the basic mCL-PKE scheme.

5 EXPERIMENTAL RESULTS

In this section, we first present the experimental results for our mCL-PKE scheme. We then compare the basic approach with the improved approach. Finally we compare our approach with Lei et al.'s scheme[16]. The experiments were performed on a machine running 32 bits GNU Linux kernel version 3.2.0-30 with an Intel®Core ™i5-2430 CPU @ 2.40GHZ and 8 GBytes memory. Our prototype system is implemented in C/C++. We use V. Shoup's NTL library [24] version 5.5.2 for big number calculation and field arithmetic. The NTL library is compiled along with the GMP library [12] in order to improve the performance of computations involving large numbers. We construct the hash function required for the mCL-PKE scheme based on SHA1.

For the hash functions, we use SHA1 as the elementary operation. However, SHA1 can easily be replaced with other cryptographic hash functions such as SHA2. The basic idea of the hash function construction is as follows. Based on the field of the hash function output, we break the input into multiple blocks, and the block number is dynamically adjusted. For each block, we execute SHA1, convert the output into decimal numbers, say $a_1, a_2, \cdots, a_n$. If the output field is $\mathbb{Z}_q^*$, then we compute $(a_1||a_2||a_3|| \cdots ||a_n) \mod q$, where $||$ denotes the concatenation operation, to get the final result of the hash function. This operation is very efficient even if other hash functions are used. According to our experimental results, it takes less than 1 ms to do this operation for a message of size 16KB.

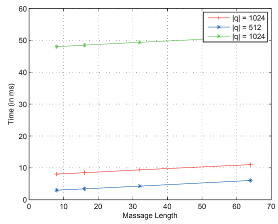

Fig. 6. Basic Encryption

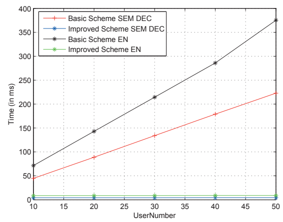

Fig. 7. Improved scheme

Figure 6 shows the time required to perform the encryption operation in the mCL-PKE scheme for different message sizes. Since our scheme does not use pairing operations, it performs encryption efficiently. As can be seen from the graph, the encryption time increases linearly as the message size increases. As the bit length of q increases, the cost increases non-linearly since the encryption algorithm performs exponentiation operations. A similar observation applies to the the SEM decryption and user decryption.

We also implemented the improved scheme where the data owner performs only one encryption per data item and creates a set of intermediate keys that allows authorized users to decrypt the data, as described in Section 4. In Figure 7, we compare the time to perform encryption and decryption in the basic scheme and the improved scheme as the number of users who can access the same data increases from 10 to 50. We fixed the length of q to 1024 bits and the message size to 16KB. It is evident from the graph that as more users are allowed to access the same data item, the better the improved scheme performs compared to the basic scheme. The cost of the basic scheme is high since the encryption algorithm is executed for each user.

Finally we implemented Lei et al.[16]'s CL-PRE scheme based on pairing. According to the results reported in their paper, proxy-encryption takes 7-8ms to encrypt a message with length 3K bits. We reimplemented their scheme using the PBC-library [17]. Our implementation of their scheme is actually more efficient and the time for encrypting a message of 8K Bytes is about 3ms. We then compared our scheme with their scheme for encryption. Even with the improved implementation, as shown in Figure 8, our encryption algorithm is more efficient than their algorithm for message sizes above 16K bytes. A similar observation is made for the decryption algorithm. This observation is consistent with the fact that our scheme uses an efficient hash function and XOR operations to perform encryption and decryption whereas their scheme uses more expensive constructs.

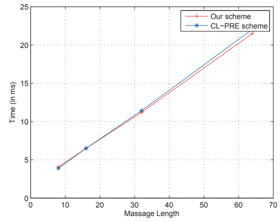

Fig. 8. Comparison of Encryption

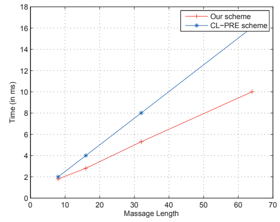

Fig. 9. Comparison of Decryption

6 Related Work

6.1 Security Mediated CL-PKE

In 2003, Al-Riyami and Paterson [2] introduced a Certificateless Public Key Cryptography (CL-PKC). Since each user holds a combination of KGC produced partial private key and an additional user-chosen secret, the key escrow problem can be resolved. As the structure of CL-PKC guarantees the validity of the user's public key without the certificate, it removes the certificate management problem. Since the advent of CL-PKC [2], many CL-PKE schemes have been proposed based on bilinear pairings. The computational cost required for pairing is still considerably high compared to standard operations such as modular exponentiation in finite fields. To improve efficiency, Sun et al. [25] presented a strongly secure CL-PKE without pairing operations. However, previous CL-PKE schemes could not solve the key revocation problem. In public key cryptography, we should consider scenarios where some private keys are compromised. If the private keys are compromised, then it is no longer secure to use the corresponding public keys. To address this problem, Boneh et al. [6] proposed the concept of mediated cryptography to support immediate revocation. The basic concept of the mediated cryptography is to utilize a security mediator (SEM) which can control security capabilities for every transaction. Once the SEM is notified that a user's public key should be revoked, it can immediately stop the user's participation in a transaction. Chow et al. [9] introduced the notion of security-mediated certificateless cryptography and presented a mediated CL-PKE relying on pairing operations. Yang et al. [26] first proposed a mediated CL-PKE without pairings. Unfortunately, Yang et al.'s scheme was found to be insecure against partial decryption attack, since their security model did not consider the capabilities of the adversary in requesting partial decryptions. Thus, a secure mediated CL-PKE without pairings is needed. Our proposed pairing-free mediated CL-PKE scheme is secure against the partial decryption attack.

6.2 Functional Encryption

Functional encryption allows one to encode an arbitrary complex access control policy with the encrypted message. The message can then be decrypted only by the users satisfying the encoded policy. In predicate encryption with public index, the policy under which the encryption is performed is public. Unlike public key cryptosystems, the public key is not a random string but some publicly known values such as ID that bind to users. Attribute based encryption (ABE) introduced by Sahai and Waters [22] is a more expressive predicate encryption with a public index. It can be considered as a generalization of IBE. In ABE, the public keys of a user are described by a set of identity attributes the user has. Key Policy ABE (KP-ABE) [13] and Ciphertext Policy ABE (CP-ABE) [5] are two popular extensions of ABE. An ABE based approach supports expressive Access Control Policies (ACPS). However, such approach suffers from some major drawbacks. Whenever the group dynamic changes, the rekeying operation requires to update the private keys given to existing members in order to provide backward/forward secrecy. Further, the ABE scheme suffers from the key escrow problem. Predicate encryption schemes without public index such as Anonymous IBE [1], [14], Hidden Vector Encryption [7], and Inner product predicate [15] preserve the privacy of the access control policies. Even though they preserve the privacy of the policy, they have limited expressibility compared to the former schemes and also suffer from the same limitations as the former schemes.

6.3 Symmetric Key Based Systems

In push-based approaches [4], [19] data items are encrypted with different keys, which are provided to users at the beginning. The encrypted data is then broadcast to all users. However, such approaches require that all [4] or some [19] keys be distributed in advance during user registration phase. This requirement makes it difficult to assure forward and backward key secrecy when user groups are dynamic or the ACPS change. Further, the rekey process is not transparent, thus shifting the burden of acquiring new keys to users. Shang et al. [23] proposed an approach to solve such problem. It lays the foundation to make rekey transparent to users and protect the privacy of the users who access the content. However, it does not support expressive access control policies. In order to address such limitations, Nabeel et. al. [20] recently proposed a more expressive attribute based group key management scheme that can be utilized to support fine-grained encryption based access control to data uploaded to public clouds. While such approaches solve the key management problem and provide expressive access control, they still suffer from the key escrow problem.

6.4 Secure Cloud Storage

Some recent research efforts [8], [10] have been proposed to build privacy preserving access control systems by combining oblivious transfer and anonymous credentials. The goal of such work is similar to ours but we identify the following limitations. Each transfer protocol allows one to access only one record from the database, whereas our approach does not have any limitation on the number of records that can be accessed at once since we separate the access control from the authorization. Yu et al. [27] proposed an approach based on ABE utilizing PRE (Proxy Re-Encryption) to handle the revocation problem of ABE. The approach still does not solve the key escrow and revocation problems. Further, it is based on pairing based cryptography whereas we avoid pairing operations.

Recently, Lei et al. [16] proposed the CL-PRE (Certificateless Proxy Re-Encryption) scheme for public cloud computing environments. While Lei et al.'s CL-PRE scheme solves the key escrow problem and certificate management, it utilizes expensive pairing operations. Further, their scheme only achieves CPA (Chosen Plaintext Attack) security which is not sufficient to protect real-world applications. They do not establish a strong security model with two types of adversaries. In CPA, the ability of the adversary is limited to obtaining ciphertexts of plaintexts of their choice. Therefore CPA is too weak to be considered viable for real-world applications. In contrast with Lei et al.'s scheme, our proposed scheme achieves CCA (chosen ciphertext attack) security. Under CCA, the ability of an adversary is more powerful than the ability of the adversary under CPA. In addition to the public key, the adversary under CCA is given access to a "decryption oracle" which decrypts arbitrary ciphertexts at the adversary's request, returning the plaintext. Moreover, our scheme does not utilize bilinear pairings to improve efficiency.

7 Conclusions

In this paper we have proposed the first mCL-PKE scheme without pairing operations and provided its formal security. Our mCL-PKE solves the key escrow problem and revocation problem. Using the mCL-PKE scheme as a key building block, we proposed an improved approach to securely share sensitive data in public clouds. Our approach supports immediate revocation and assures the confidentiality of the data stored in an untrusted public cloud while enforcing the access control policies of the data owner. Our experimental results show the efficiency of basic mCL-PKE scheme and improved approach for the public cloud. Further, for multiple users satisfying the same access control policies, our improved approach performs only a single encryption of each data item and reduces the overall overhead at the data owner.

References

[1] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi.

Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. *J. Cryptol.*, 21(3):350–391, Mar. 2008.
[2] S. Al-Riyami and K. Paterson. Certificateless public key cryptography. In C.-S. Laih, editor, *Advances in Cryptology - ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 452–473. Springer Berlin / Heidelberg, 2003.
[3] M. Bellare, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. pages 26–45. Springer-Verlag, 1998.
[4] E. Bertino and E. Ferrari. Secure and selective dissemination of XML documents. *ACM Transactions on Information and System Security*, 5(3):290–331, 2002.
[5] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 321–334, Washington, DC, USA, 2007. IEEE Computer Society.
[6] D. Boneh, X. Ding, and G. Tsudik. Fine-grained control of security capabilities. *ACM Transactions on Internet Technology*, 4(1):60–82, Feb. 2004.
[7] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *Proceedings of the 4th conference on Theory of cryptography*, TCC'07, pages 535–554, Berlin, Heidelberg, 2007. Springer-Verlag.
[8] J. Camenisch, M. Dubovitskaya, and G. Neven. Oblivious transfer with access control. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, pages 131–140, New York, NY, USA, 2009. ACM.
[9] S. S. M. Chow, C. Boyd, and J. M. G. Nieto. Security-mediated certificateless cryptography. In *Proceedings of the 9th international conference on Theory and Practice of Public-Key Cryptography*, PKC'06, pages 508–524, 2006.
[10] S. Coull, M. Green, and S. Hohenberger. Controlling access to an oblivious database using stateful anonymous credentials. In *Irvine: Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography*, pages 501–520, Berlin, Heidelberg, 2009. Springer-Verlag.
[11] I. Dropbox. Dropbox. https://www.dropbox.com/.
[12] The gnu multiple precision arithmetic library. http://gmplib.org/.
[13] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98, New York, NY, USA, 2006. ACM.
[14] C. Gu, Y. Zhu, and H. Pan. Information security and cryptology. In D. Pei, M. Yung, D. Lin, and C. Wu, editors, *Inscrypt*, chapter Efficient Public Key Encryption with Keyword Search Schemes from Pairings, pages 372–383. Springer-Verlag, Berlin, Heidelberg, 2008.
[15] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology*, EUROCRYPT'08, pages 146–162, Berlin, Heidelberg, 2008. Springer-Verlag.
[16] X. W. Lei Xu and X. Zhang. CL-PKE: A certificateless proxy re-encryption scheme for secure data sharing with public cloud. In *ACM Symposium on Information, Computer and Communications Security*, 2012.
[17] B. Lynn. Pairing-based cryptography. http://crypto.stanford.edu/pbc.
[18] microsoft co. ltd. Microsoft skydrive. https://skydrive.live.com/.
[19] G. Miklau and D. Suciu. Controlling access to published data using cryptography. In *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, pages 898–909. VLDB Endowment, 2003.
[20] M. Nabeel, N. Shang, and E. Bertino. Privacy preserving policy based content sharing in public clouds. *IEEE Transactions on Knowledge and Data Engineering*, 2012.
[21] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–396, 2000.
[22] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *Eurocrypt 2005, LNCS 3494*, pages 457–473. Springer-Verlag, 2005.
[23] N. Shang, M. Nabeel, F. Paci, and E. Bertino. A privacy-preserving approach to policy-based content dissemination. In *ICDE '10: Proceedings of the 2010 IEEE 26th International Conference on Data Engineering*, 2010.
[24] V. Shoup. NTL library for doing number theory. http://www.shoup.net/ntl/.
[25] Y. Sun, F. Zhang, and J. Baek. Strongly secure certificateless public key encryption without pairing. In *Proceedings of the 6th international conference on Cryptology and network security*, CANS'07, pages 194–208, 2007.
[26] C. Yang, F. Wang, and X. Wang. Efficient mediated certificates public key encryption scheme without pairings. In *Advanced Information Networking and Applications Workshops, 2007*, volume 1, pages 109 –112, may 2007.
[27] S. Yu, C. Wang, K. Ren, and W. Lou. Attribute based data sharing with attribute revocation. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 261–270, New York, NY, USA, 2010. ACM.

Seung-Hyun Seo She is a post-doctoral researcher of Computer Science at Purdue University since 2012 and a senior researcher of KISA (Korea Internet and Security Agency) since 2010. Before joining KISA, she was a researcher for 3 years in FSA (Financial Security Agency), Korea. She received her B.S.(2000), M.S.(2002), and Ph.D.(2006) from Ewha Womans University in Korea. Her main research interests include cryptography, mobile security, secure cloud computing, and malicious code analysis.

Mohamed Nabeel He is a post-doctoral researcher of Computer Science at Purdue University and a Fulbright Alumnus from Sri Lanka. He received Ph.D.(2012) from Purdue University. His main research interests include distributed systems security and applied cryptogrphy.

Xiaoyu Ding He is a PhD candidate at the department of computer science, Purdue university. He received the B.E. degrees from Wuhan University, China in 2011. He joined the department of computer science in Purdue University since 2011. His main research interests include cryptography and access control.

Elisa Bertino She is professor of Computer Science at Purdue University and serves as director of Purdue Cyber Center (Discovery Park) and as research director of CERIAS. Previously, she was a faculty member in the Department of Computer Science and Communication of the University of Milan. Her main research interests include security, privacy, digital identity management systems, database systems, distributed systems, and multimedia systems. She is a fellow of the IEEE and a fellow of the ACM. She received the 2002 IEEE Computer Society Technical Achievement Award for outstanding contributions to database systems and database security and advanced data management systems and the 2005 IEEE Computer Society Tsutomu Kanai Award for pioneering and innovative research contributions to secure distributed systems