Provably-Secure Certificateless Key Encapsulation Mechanism for e-Healthcare System

Hui-Xian Shi[1] and Rui Guo[2]
(Corresponding author: Rui Guo)
Department of Mathematics and Information Science, Shaanxi Normal University, Xi'an 710119, China[1]
China Mobile Group Shaanxi Company Limited Xi'an Brand, Xi'an 710077, China[2]
(Email: grbupt@gmail.com)
(Received Feb. 10, 2015; revised and accepted Mar. 28 & Apr. 26, 2015)

Abstract

Modern information and communications technology have facilitated the traditional medical services in the healthcare system, which exchanges the physiological condition and diagnosis timely and remotely between the patient and the physician. However, there exist several privacy concerns as personal health information could be exposed to unauthorized parties. To ensure the confidentiality of this sensitive data, it is a promising method to encrypt it before delivering. Moreover, to generate and distribute a secure session key is a significant issue in the encryption algorithm. In this paper, we put forward a novel certificateless key encapsulation mechanism for the e-healthcare system, which is proven secure under the computational Diffie-Hellman assumption in the random oracle model. Furthermore, we compare our proposal with others in performance. Under the same simulation environment, the results show that the proposed scheme needs less computation and communication cost and appropriate to encapsulate the session key in the e-healthcare system.

Keywords: Certificateless key encapsulation mechanism, e-healthcare system, hybrid encryption, IND-CCA secure

1 Introduction

In recent years, information and communications technology have been employed in the traditional healthcare system. Some lightweight devices, such as wireless medical sensors, PDA and iPhone, increase the efficiency of this system and provide high-quality of care without sacrificing the patient comfort [17]. In the e-healthcare system, the wearable medical sensors are fixed on the patient to collect his/her physiological signals (e.g., blood pressure, pulse oximeter and temperature). Then, via a public wireless channel, these data are transmitted to the physician's handheld terminals, and the patient can be diagnosed timely and remotely.

However, the Health Insurance Portability and Accountability Act (HIPAA) enacted in 1996 [11] demonstrated that the patient's physiological conditions are all the sensitive information, which relate to his/her privacy and should be protected. If the privacy is eavesdropped by an unauthorized party, the safety and economic interests of the patient would be threatened. Thus, during the transmission in the public channel, it needs a secure encryption scheme to ensure the confidentiality of the transmitted data between the patient and the physician in the e-healthcare system.

Considering the encryption scheme for e-healthcare system, for the reason that this system consists of the lightweight devices with limited memory, small bandwidth and low power, it should preserve two outstanding characteristics with efficiency and confidentiality during designing an encryption scheme.

There are two models of encryption in cryptography, asymmetric and symmetric key encryption systems. In the public key encryption (i.e., asymmetric encryption), the most of schemes in the literature have limited message spaces, which means that a message to be encrypted is assumed to have a limited length or belong to a specific group. It is inconvenient and expensive for ensuring the confidentiality of arbitrary messages by using of a purely public key encryption. As enjoy high efficiency, symmetric encryption schemes are usually employed to encrypt large messages, such as DES [4, 16]. Unfortunately, they also suffer from the key distribution problem. To achieve high efficiency while avoiding the key distribution problem in the encryption system, the normal method of performing the public key encryption is to divide the encryption scheme into two parts: one part uses the public key techniques to encrypt a one-time symmetric key; the other part takes use of this symmetric key to encrypt the transmitted message. In such a construction, the public key part of the algorithm is called the key en-

capsulation mechanism (KEM) while the symmetric key part (where the message is actually encrypted) is known as the data encapsulation mechanism (DEM). According to this encryption model, KEM can provide an efficient and secure method to deliver a random key from a sender to a designated receiver, and DEM enables to increase efficiency over public key encryption. Combining KEM and DEM, the resulting scheme is then called a hybrid encryption scheme which has received much attention in recent years [1, 9, 10].

In the traditional public key infrastructure (PKI), encryption is achieved through the certificates issued by a trust certification authority (CA). In [5], Dent presented a lot of generic constructions of KEM from standard public key encryption, however these led to the problem of certificates management (including distribution, storage, revocation and verification of certificates), which placed a large computation cost on the system. To avoid these weaknesses, Shamir [15] proposed the identity based public key cryptography (ID-PKC) by deriving the user's public key directly from some public parameters and the user's identity, such as email and IP address. In 2008, Bentahar et al. [3] extended the concept of key encapsulation to the primitives of identity based encryption that are provably secure in the random oracle model. Nevertheless, there is a trusted third party called the private key generator (PKG) in ID-PKC whose behavior is in possession of a master secret key (which is used to derive the private key of any user in this system). Thus, the private key of all the users in ID-PKC is known to the PKG. This inherent issue in ID-PKC is called the key escrow problem [14]. Therefore, these two types of cryptographic primitive above are not suitable for protecting the entity's privacy with lightweight mobile devices, such as in e-healthcare system.

To overcome the key escrow problem, certificateless public key cryptography (CL-PKC) was introduced by Al-Riyami and Paterson [2]. In the certificateless key encapsulation mechanism (CL-KEM), the user's private key is split into two parts: one is the partial private key obtained from the key generation center (KGC), the other one is a user's selected secret value. Consequently, the trusted third party KGC cannot access the user's private key to reveal his/her privacy any more. Several CL-KEM protocols have been proposed in the last decade [3, 8, 12]. Huang and Wong [8] proposed the first generic construction of CL-KEM in the standard model, which was secure against malicious-but-passive KGC attacks. In [3], Bentahar et al. also took any IBE scheme plus a special form of public key scheme, such as RSA or ElGamal in certain groups, and used them to construct a CL-KEM, which was secure in a strong sense. However, these two schemes combined a public key based encryption scheme and an identity based KEM and thus very inefficient. Lippold and Boyd [12] presented a direct construction for a chosen ciphertext secure (CCA secure) CL-KEM in the standard model that was more efficient than the generic constructions.

In this paper, we put forward a certificateless key encapsulation mechanism for e-healthcare system, and prove that it is secure in the random oracle model against the chosen ciphertext attacks. Furthermore, this scheme achieves the Girault's trust Level 3 which ensures the credibility of the authority [6]. Compared to the related schemes, through the evaluations and experiments, our protocol offers a better performance in the computation and communication cost.

In the next section, we review some computational assumptions, the model and the security definitions of CL-KEM that will be used throughout the paper. In Section 3, we design a new protocol for e-healthcare and analyze the security of it. In Section 4, we compare the efficiency with related schemes and conclude the paper in Section 5.

2 Preliminaries

2.1 Complexity Assumptions

Let G be a cyclic additive group with prime order p, and P be a generator of G.

Definition 1. *Discrete Logarithm (DL) problem: Given* $(P, Q \in G)$*, find an integer* $x \in Z_p^*$ *satisfying* $Q = xP$*.*

The DL assumption is that there is no polynomial time algorithm that can solve the DL problem with non-negligible probability.

Definition 2. *Computational Diffie-Hellman (CDH) problem: Given* $(Q = xP, R = yP) \in G^2$ *for any* $x, y \in Z_p^*$*, compute* xyP*.*

The CDH assumption is that there is no polynomial time algorithm that can solve the CDH problem with non-negligible probability.

2.2 Certificateless Key Encapsulation Mechanism

A CL-KEM for e-healthcare system consists of the following seven probabilistic polynomial time (PPT) algorithms: *Setup*, *User-Key-Generation*, *Partial-Key-Extract*, *Set-Private-Key*, *Set-Public-Key*, *Encap* and *Decap*.

Setup. On input a security parameter 1^k, the medical server (MS) returns the system parameters *params*, the master public/secret key (*mpk*,*msk*). Then, MS publishes *params* and *mpk*, and keeps the *msk* secret.

User-Key-Generation. On input the system parameters *params*, the patient returns a pair of public/secret key (*pk*,*sk*).

Partial-Key-Extract. On input *params*, *msk*, patient's identity ID_P and his/her public key *pk*, MS

executes this algorithm and returns a partial private key D_P to patient via a confidential and authentic channel, and the corresponding partial public key P_P.

Set-Private-Key. On input *params*, patient's partial private key D_P and his/her secret key sk, this algorithm returns private key SK_P to the patient.

Set-Public-Key. On input *params*, patient's partial public key P_P and his/her public key pk, this algorithm returns the patient's public key PK_P.

Encap. Running by a doctor. On input *params*, the patient's identity ID_P, and his/her public key PK_P, this algorithm outputs an encapsulation key pair $(K, c) \in (\mathcal{K}, \mathcal{C})$, where c is called the encapsulation of key K, and K is considered to be distributed uniformly in the key space $\mathcal{K}$. (In the hybrid encryption primitive, the doctor encrypts the privacy data by using of this K in symmetric encryption scheme.)

Decap. Running this deterministic algorithm by a patient. On input *params*, the encapsulation c, and his/her private key SK_P, this algorithm outputs the corresponding key K, or an invalid encapsulation $\perp$. (Similarly, the patient decrypts the ciphertext above with this decapsulation K.)

In this system, to achieve the Girault's trust Level 3, the *User-Key-Generation* algorithm must be run prior to the *Partial-Key-Extract* algorithm. The patient fixes his/her secret key sk and public key pk firstly. Then, MS generates patient's partial key D_P by binding his/her public key to an identity ID_P. According to this way, although MS can replace patient's public key pk, there will exist a pair of working public keys (pk, pk') for only one patient. Moreover, two working different public keys (PK_P, PK'_P) binding one patient's identity can result from two partial private keys, and only the MS has ability to generate these two working partial private keys. Hence, the MS's forgery is easily tracked, which means that the trust level of MS is achieving to the Girault's trust Level 3 as described in [6].

2.3 Security Model

In certificateless cryptography, there are two types of adversaries $\mathcal{A}_I$ and $\mathcal{A}_{II}$. Type-I adversary $\mathcal{A}_I$ acts as a dishonest user who does not have access to MS's master secret key and patient's partial key, but it enables to compromise user's private key or replace the public key of any patient with its own choices value. By contrast, Type-II adversary $\mathcal{A}_{II}$ plays the part of a malicious-but-passive MS who controls the master secret key msk (hence it can compute patient's partial secret key). Besides, Type-II adversary $\mathcal{A}_{II}$ is allowed to receive private keys for arbitrary identities but cannot replace any patient's public key. The following oracles are the interactive game between challenger $\mathcal{C}$ and adversary $\mathcal{A}$.

Setup. The challenger $\mathcal{C}$ runs this algorithm to generate the public parameters *params* and the master public/private key pair (mpk, msk).

Partial-Key-Extract-Oracle. Upon receiving an identity ID, this oracle computes the corresponding partial public/private key pair $(P_{\mathrm{ID}}, D_{\mathrm{ID}})$ and sends this tuple to $\mathcal{A}$.

Private-Key-Request-Oracle. Upon receiving an identity ID, if the ID's public key has not been replaced, $\mathcal{C}$ responds it with the private key SK_{ID}. Otherwise, $\mathcal{C}$ does not provide the corresponding private key to $\mathcal{A}$.

Public-Key-Request-Oracle. Upon receiving an identity ID, $\mathcal{C}$ responds it with the public key PK_{ID}.

Replace-Public-Key-Oracle. $\mathcal{A}$ can repeatedly replace the public key PK_{ID} with any value PK'_{ID} of its own choice. The current value of the user's public key is used by $\mathcal{C}$ in any computations or to response to $\mathcal{A}$'s queries.

Decapsulation-Oracle. Upon receiving an identity ID and an encapsulation c, if there is no query on ID, return $\perp$. Otherwise, return $K \leftarrow Decap(\mathrm{ID}, SK_{\mathrm{ID}}, c)$ as a decapsulation of c.

We now specify two games for Type-I and Type-II security described as follows.

Game-I: Let $\mathcal{C}_I$ be a challenger to Type-I adversary $\mathcal{A}_I$ and 1^k be a security parameter.

1) $\mathcal{C}_I$ computes $(mpk, msk) \leftarrow Setup(1^k)$, and runs $\mathcal{A}_I$ on input 1^k and mpk.
2) $\mathcal{A}_I$ can query *Partial-Key-Extract-Oracle*, *Private-Key-Request-Oracle*, *Public-Key-Request-Oracle*, *Replace-Public-Key-Oracle* and *Decapsulation-Oracle*. Then, $\mathcal{A}_I$ submits a target identity $\mathrm{ID}^* \in \{0,1\}^*$.
3) $\mathcal{C}_I$ runs $(K_1, c^*) \leftarrow Encap(mpk, PK_{\mathrm{ID}^*}, \mathrm{ID}^*)$ and randomly selects $(K_0 \leftarrow \mathcal{K})$. Then, $\mathcal{C}_I$ flips a coin b, and returns (K_b, c^*) to $\mathcal{A}_I$.
4) $\mathcal{A}_I$ continues to issue queries as in Step (2). Finally, it outputs a bit b'.

$\mathcal{A}_I$ wins this game if $b' = b$. Note that $\mathcal{A}_I$ is not allowed to query *Partial-Key-Extract-Oracle* on ID^* and *Decapsulation-Oracle* on (ID^*, c^*). We define the advantage of $\mathcal{A}_I$ in **Game-I** to be $Adv(\mathcal{A}_I) = |\Pr(b' = b) - \frac{1}{2}|$.

Game-II: Let $\mathcal{C}_{II}$ be a challenger to Type-II adversary $\mathcal{A}_{II}$ and 1^k be a security parameter.

1) $\mathcal{C}_{II}$ runs $\mathcal{A}_{II}$ on input 1^k and returns $(mpk, msk) \leftarrow Setup(1^k)$ as an answer.

2) $\mathcal{A}_{II}$ can query *Private-Key-Request-Oracle*, *Public-Key-Request-Oracle* and *Decapsulation-Oracle*. Then $\mathcal{A}_{II}$ submits a target identity $\mathrm{ID}^* \in \{0,1\}^*$. Note that *Partial-Key-Extract-Oracle* is not allowed by $\mathcal{A}_{II}$ because of the knowledge of msk.

3) $\mathcal{C}_{II}$ runs $(K_1, c^*) \leftarrow Encap(mpk, PK_{\mathrm{ID}^*}, \mathrm{ID}^*)$ and randomly selects $(K_0 \leftarrow \mathcal{K})$. Then, $\mathcal{C}_{II}$ flips a coin b, and returns (K_b, c^*) to $\mathcal{A}_{II}$.

4) $\mathcal{A}_{II}$ continues to issue queries as in Step (2). Finally, it outputs a bit b'.

$\mathcal{A}_{II}$ wins this game if $b' = b$. Note that $\mathcal{A}_{II}$ is not allowed to query *Private-Key-Request-Oracle* on ID^* and *Decapsulation-Oracle* on (ID^*, c^*). We define the advantage of $\mathcal{A}_{II}$ in **Game-II** to be $Adv(\mathcal{A}_{II}) = |\Pr(b' = b) - \frac{1}{2}|$.

Definition 3. *A CL-KEM* Π *is secure against chosen ciphertext attack (IND-CCA secure) if neither polynomial bounded adversary* $\mathcal{A}$ *of Type-I nor Type-II has a non-negligible advantage against the challenger in the* ***Game-I*** *and* ***Game-II****.*

$\mathcal{A}$ breaks an IND-CCA secure CL-KEM Π with $(q_H, q_{par}, q_{pri}, q_{pub}, q_D, \varepsilon)$ if and only if the advantage of $\mathcal{A}$ that makes q_H times to the random oracle $H(\cdot)$, q_{par} times *Partial-Key-Extract-Oracle*, q_{pri} times *Private-Key-Request-Oracle*, q_{pub} times *Public-Key-Request-Oracle* and q_D times *Decapsulation-Oracle* queries is greater than ε. The scheme Π is said to be $(q_H, q_{par}, q_{pri}, q_{pub}, q_D, \varepsilon)$-IND-CCA secure if there is no adversary $\mathcal{A}$ that breaks IND-CCA secure scheme Π with $(q_H, q_{par}, q_{pri}, q_{pub}, q_D, \varepsilon)$.

3 Our CL-KEM

In this section, we put forward a novel CL-KEM without bilinear pairing to encapsulate a one-time symmetric key between the patient and doctor. The notations used throughout this protocol are listed in Table 1.

Table 1: Notions of this scheme

ID_P	the identity of Patient
$H_i(\cdot)$	the collision-resistant hash function (i=1,2)
p	the large prime number
G	the cyclic additive group
P	the generator of G
x	the master secret key
X	the master public key
P_P	the Patient's partial public key
D_P	the Patient's partial private key
PK_P	the Patient's public key
SK_P	the Patient's private key
$\|$	the connection operation

3.1 Construction

The proposed CL-KEM as shown in Figure 1 consists of the following seven PPT algorithms.

Setup. Let G be a cyclic group of prime order p with an arbitrary generator $P \in G$. The MS selects $x \in Z_p^*$ randomly and computes $X = xP$ as the master public key. Then, it chooses two collision resistant hash functions $H_1 : \{0,1\}^{l_0} \times G^* \times G^* \to Z_p^*$ and $H_2 : \{0,1\}^{l_0} \times G^{*5} \to \{0,1\}^*$. The system parameters are $params = (p, G, P, X, H_1, H_2)$, and the master secret key is $msk = x$.

User-Key-Generation. Patient picks $y \in Z_p^*$ uniformly at random and computes $Y = yP$, and he/she returns $(sk, pk) = (y, Y)$.

Partial-Key-Extract. MS picks $\alpha \in Z_p^*$ at random and computes $r_P = \alpha P$ and $z_P = \alpha + xH_1(\mathrm{ID}_P \parallel r_P \parallel pk)$, where ID_P is the patient's identity. Then MS returns $(P_P, D_P) = (r_P, z_P)$ as a pair of patient's partial key.

Set-Private-Key. Set $SK_P = (sk, D_P) = (y, z_P)$, it returns SK_P as the patient's private key.

Set-Public-Key. Let $PK_P = (pk, P_P) = (Y, r_P)$, it returns PK_P as the patient's public key.

Encap. Doctor picks $u \in Z_p^*$ randomly and computes the ciphertext:

$$\begin{aligned} c &= uP, \\ c_1 &= u(Y + r_P + XH_1(\mathrm{ID}_P \parallel r_P \parallel pk)), \\ c_2 &= uY, \end{aligned}$$

and the corresponding session key is

$$K = H_2(\mathrm{ID}_P \parallel PK_P \parallel c \parallel c_1 \parallel c_2).$$

Then the doctor delivers the encapsulation $\{c\}$ to patient.

Decap. To decapsulate c, the patient reconstructs the session key as

$$K = H_2(\mathrm{ID}_P \parallel PK_P \parallel c \parallel (y + z_P)c \parallel yc).$$

Then in the hybrid scheme, a symmetric encryption scheme is taken to protect the privacy under this K.

The above *Decap* algorithm is consistent if c is a valid encapsulation, then it is easy to verify that,

$$\begin{aligned} (y + z_P)c &= yuP + (\alpha + xH_1(\mathrm{ID}_P \parallel r_P \parallel pk))uP \\ &= u(yP + r_P + XH_1(\mathrm{ID}_P \parallel r_P \parallel pk)) \\ &= c_1, \end{aligned}$$

and

$$yc = yuP = uY = c_2.$$

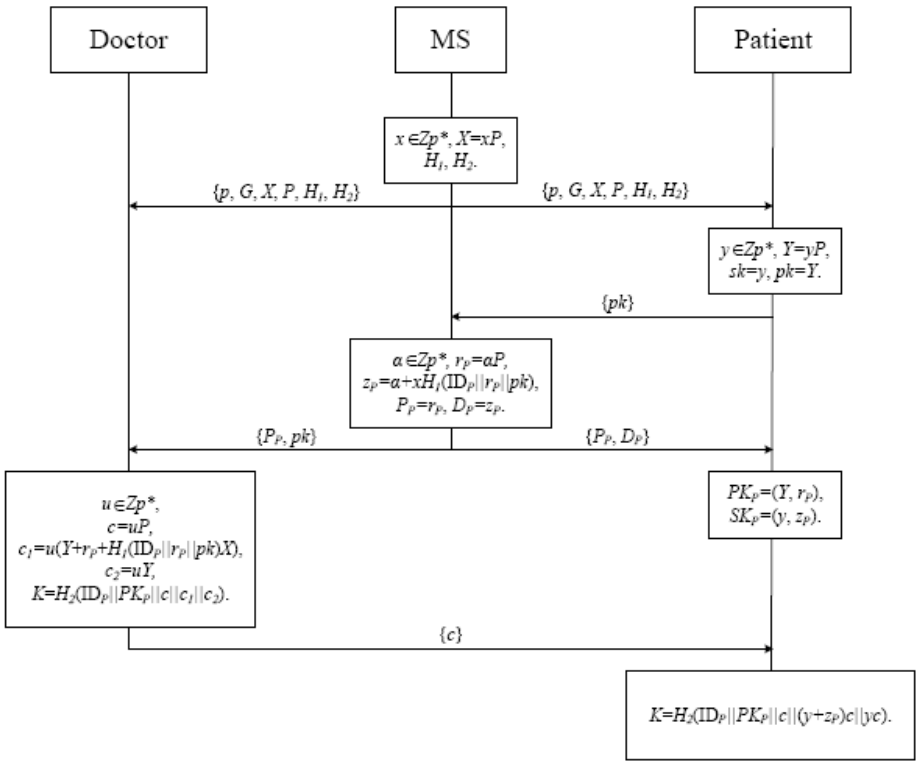

Figure 1: Our CL-KEM for e-healthcare system

3.2 Security Analysis

In this subsection, we prove that the CL-KEM presented in the previous is secure in the random oracle model.

Theorem 1. *Provided that H_1 and H_2 are two collision resistant hash functions. This CL-KEM is IND-CCA secure in the random oracle model assuming that there is no polynomial time algorithm that can solve the CDH problem with non-negligible probability.*

This theorem following from two lemmas will show that our CL-KEM is secure against the Type-I and Type-II adversaries whose behaviors are as described in the **Game-I** and **Game-II**.

Lemma 1. *This CL-KEM is $(q_H, q_{par}, q_{pri}, q_{pub}, q_D, \varepsilon)$-IND-CCA secure against the Type-I adversary $\mathcal{A}$ in the random oracle model, then there exists an algorithm $\mathcal{B}$ that solves the CDH problem with the following advantage*

$$\varepsilon' > \frac{1}{q_{H_2}}\left(\frac{2\varepsilon}{e(q_{prv}+1)} - \frac{q_D q_{H_1}}{2^{l_0}p^2} - \frac{q_D}{2^{l_0}p^5}\right).$$

Proof. Assuming there exists a Type-I adversary $\mathcal{A}_I$ imitating an "outside" adversary, who replaces the public key of arbitrary identities but cannot corrupt the master secret key.

Suppose that there is another PPT algorithm $\mathcal{B}$ can solve the CDH problem in the instance of (p, P, aP, xP) with probability at least ε' by interacting with $\mathcal{A}_I$. To solve this problem, $\mathcal{B}$ needs to simulate a challenger to run each algorithm of **Game-I** for $\mathcal{A}_I$ as follows:

Setup. Algorithm $\mathcal{B}$ sets the master public key $X = xP$, where $x \in Z_p^*$ is the master secret key that is unknown to $\mathcal{B}$. Then $\mathcal{B}$ gives $\mathcal{A}_I$ the $params = \{p, G, P, X, H_1, H_2\}$ as system parameters. $\mathcal{A}_I$ performs a series of polynomially bounded number of queries according to the following oracles:

H_1 **Queries.** $\mathcal{B}$ maintains a list of tuples $\langle(\mathrm{ID}, r_{\mathrm{ID}}, Y), v\rangle$ in H_1-**List** L_1. On receiving a query $(\mathrm{ID}, r_{\mathrm{ID}}, Y)$ to H_1:

1) If $\langle(\mathrm{ID}, r_{\mathrm{ID}}, Y), v\rangle$ already appears on the list L_1, $\mathcal{B}$ responds v as an answer.
2) Otherwise, pick $v \in Z_p^*$ randomly, add $\langle(\mathrm{ID}, r_{\mathrm{ID}}, Y), v\rangle$ to L_1 and return v as an answer.

H_2 **Queries.** $\mathcal{B}$ maintains a list of tuples $\langle(\mathrm{ID}, T), R\rangle$ in H_2-**List** L_2, where $T \in G^{*5}$. On receiving a query (ID, T) to H_2:

1) If $\langle(\mathrm{ID}, T), R\rangle$ exists in the list L_2, $\mathcal{B}$ responds R as an answer.
2) Otherwise, choose $R \in \{0,1\}^*$ uniformly at random, add $\langle(\mathrm{ID}, T), R\rangle$ to L_2 and return R as an answer.

Phase 1. $\mathcal{A}_I$ can issue a number of the following oracle queries.

Partial-Key-Extract-Oracle. $\mathcal{B}$ maintains a **PartialKeyList** of tuples $\langle \mathrm{ID}, (r_{\mathrm{ID}}, z_{\mathrm{ID}})\rangle$. On receiving a query ID, $\mathcal{B}$ responds as follows:

1) If $\langle \mathrm{ID}, (r_{\mathrm{ID}}, z_{\mathrm{ID}})\rangle$ exists in **PartialKeyList**, return $(r_{\mathrm{ID}}, z_{\mathrm{ID}})$ as an answer.
2) Otherwise, pick z_{ID}, $v \in Z_p^*$ at random, and compute $r_{\mathrm{ID}} = z_{\mathrm{ID}}P - vX$. Add $(\mathrm{ID}, r_{\mathrm{ID}}, v)$ to L_1 and $\langle \mathrm{ID}, (r_{\mathrm{ID}}, z_{\mathrm{ID}})\rangle$ to **PartialKeyList**, return $(r_{\mathrm{ID}}, z_{\mathrm{ID}})$ as an answer.

Public-Key-Request-Oracle. $\mathcal{B}$ maintains a **PublicKeyList** of tuples $\langle \mathrm{ID}, (r_{\mathrm{ID}}, Y), coin\rangle$. On receiving a query ID, $\mathcal{B}$ responds as follows:

1) If $\langle \mathrm{ID}, (r_{\mathrm{ID}}, Y), coin\rangle$ exists in **PublicKeyList**, return $PK_{\mathrm{ID}} = (r_{\mathrm{ID}}, Y)$ as an answer.
2) Otherwise, choose $coin \in \{0, 1\}$ at random so that $\Pr[coin = 0] = \delta$ (δ will be defined later).
3) If $coin = 0$, do the following:
   a. If $\langle \mathrm{ID}, (r_{\mathrm{ID}}, z_{\mathrm{ID}})\rangle$ exists in **PartialKeyList**, pick $y \in Z_p^*$ at random and compute $Y = yP$. Then, add $\langle \mathrm{ID}, (y, z_{\mathrm{ID}})\rangle$ to **PrivateKeyList** (which will be defined later) and $\langle \mathrm{ID}, (r_{\mathrm{ID}}, Y), coin\rangle$ to **PublicKeyList** respectively, return $PK_{\mathrm{ID}} = (r_{\mathrm{ID}}, Y)$ as an answer.
   b. Otherwise, run the *Partial-Key-Extract-Oracle* to get partial keys $(r_{\mathrm{ID}}, z_{\mathrm{ID}})$ about ID. Pick $y \in Z_p^*$ at random and compute $Y = yP$. Then, add $\langle \mathrm{ID}, (r_{\mathrm{ID}}, z_{\mathrm{ID}})\rangle$ to **PrivateKeyList** and $\langle \mathrm{ID}, (r_{\mathrm{ID}}, Y), coin\rangle$ to **PublicKeyList** respectively, return $PK_{\mathrm{ID}} = (r_{\mathrm{ID}}, Y)$ as an answer.
4) Otherwise (if $coin = 1$), pick $\alpha, y \in Z_p^*$ at random and compute $r_{\mathrm{ID}} = \alpha P$, $Y = yP$, add $\langle \mathrm{ID}, (y, *), \alpha\rangle$ to **PrivateKeyList** (where * denotes the arbitrary value), and $\langle \mathrm{ID}, (r_{\mathrm{ID}}, Y), coin\rangle$ to **PublicKeyList**, return $PK_{\mathrm{ID}} = (r_{\mathrm{ID}}, Y)$ as an answer.

Private-Key-Request-Oracle. $\mathcal{B}$ maintains a **PrivateKeyList** of tuples $\langle \mathrm{ID}, (y, z_{\mathrm{ID}}), \alpha\rangle$. On receiving a query ID, $\mathcal{B}$ responds as follows:

1) Perform *Public-Key-Request-Oracle* on ID to get a tuple $\langle \mathrm{ID}, (r_{\mathrm{ID}}, Y), coin\rangle$ from **PublicKeyList**.
2) If $coin = 0$, search a tuple $\langle \mathrm{ID}, (y, z_{\mathrm{ID}}), \alpha\rangle$ in **PrivateKeyList** and return $SK_{\mathrm{ID}} = (y, z_{\mathrm{ID}})$ as an answer.
3) Otherwise, return "*Abort*" and terminate this algorithm.

Replace-Public-Key-Oracle. $\mathcal{A}_I$ may replace any public key with a new value of its choice and $\mathcal{B}$ records all the changes.

Decapsulation-Oracle. On receiving a query $\langle \mathrm{ID}, PK_{\mathrm{ID}}, c\rangle$, where $PK_{\mathrm{ID}} = (r_{\mathrm{ID}}, Y)$. $\mathcal{B}$ responds as follows:

1) Search a tuple $\langle \mathrm{ID}, (r_{\mathrm{ID}}, Y), coin\rangle$ in **PublicKeyList**.
2) If such a tuple exists and $coin = 0$.
   a. Search **PrivateKeyList** for a tuple $\langle \mathrm{ID}, (y, z_{\mathrm{ID}})\rangle$.
   b. Compute $K = H_2(\mathrm{ID} \parallel PK_{\mathrm{ID}} \parallel c \parallel (y + z_{\mathrm{ID}})c \parallel yc)$.
3) Else, if such a tuple exists and $coin = 1$.
   a. Perform H_1 **queries** to get a tuple $\langle \mathrm{ID}, (r_{\mathrm{ID}}, Y), v\rangle$.
   b. If there exists $\langle (\mathrm{ID}, T), R\rangle \in L_2$ such that $R = H_2(\mathrm{ID} \parallel T)$, return R as the session key and "*Reject*" otherwise.
4) Else, if such a tuple does not exist (which means that the public key of a target user is replaced by $\mathcal{A}_I$), run the same algorithm in (3).

Challenge Phase. Once $\mathcal{A}_I$ decides that *Phase 1* is over, it outputs a challenge identity ID^*. On receiving a challenge query ID^*, $\mathcal{B}$ responds as follows:

1) Run *Public-Key-Request-Oracle* on ID^* to get a tuple $\langle \mathrm{ID}^*, (r_{\mathrm{ID}^*}, Y^*), coin\rangle$ in **PublicKeyList**.
2) If $coin = 0$, return "*Abort*" and terminate.
3) Otherwise, do the following:
   a. Search a tuple $\langle \mathrm{ID}^*, (y^*, *), \alpha\rangle$ in **PrivateKeyList**. (In this case, we know that $r_{\mathrm{ID}^*} = \alpha^* P$, $Y^* = y^* P$).
   b. Set $c^* = aP$, $c_1^* = a(Y^* + r_{\mathrm{ID}^*} + XH_1(\mathrm{ID}^* \parallel r_{\mathrm{ID}^*} \parallel Y^*))$ and $c_2^* = aY^*$. Note that $\mathcal{B}$ does not know "a".
   c. Compute $\Gamma = ar_{\mathrm{ID}^*}$ and $v^* = H_1(\mathrm{ID}^* \parallel r_{\mathrm{ID}^*} \parallel Y^*)$.
   d. Pick $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space.
   e. Compute $K_1 = H_2(\mathrm{ID}^* \parallel (r_{\mathrm{ID}^*}, Y^*) \parallel c^* \parallel c_1^* \parallel c_2^*)$.
4) Choose a bit $\beta \in_R \{0, 1\}$ and return (c^*, K_β) to $\mathcal{A}_I$.

Phase 2. $\mathcal{A}_I$ repeats the queries in *Phase 1*. However, there is no *Partial-Key-Extract-Oracle* or *Private-Key-Request-Oracle* query on ID^* is allowed. Also, no *Decapsulation-Oracle* query should be made on the encapsulation c^* for ID^*.

Guess. $\mathcal{A}_I$ outputs a guess β' for β, and wins the game if $\beta' = \beta$. Then, $\mathcal{B}$ will be able to solve the CDH problem by computing $(c^* \cdot z_{\mathrm{ID}^*} - \Gamma)/v^*$.

Analysis. We denote the event that ID^* has been queried to H_1 as $\mathbf{Ask}H_1^*$. Also, by $\mathbf{Ask}H_2^*$, we denote the event that $\langle(\mathrm{ID}^*, T^*), R^*\rangle$ has been queried to H_2. Provided that the event $\mathbf{Ask}H_2^*$ happens, $\mathcal{B}$ will solve the CDH problem by picking a tuple $\langle(\mathrm{ID}^*, T^*), R^*\rangle$ in L_2 and computing $(c^* \cdot z_{\mathrm{ID}^*} - \Gamma)/v^*$ with probability at least $1/q_{H_2}$. Hence, we have $\varepsilon' \geq (1/q_{H_2})\Pr[\mathbf{Ask}H_2^*]$.

If $\mathcal{B}$ does not abort in the **Game-I**, the simulations of *Partial-Key-Extract-Oracle*, *Public-Key-Request-Oracle*, *Private-Key-Request-Oracle* and the target encapsulation is identically distributed in our construction. Also, $\mathcal{B}$'s responses to all hash queries are uniformly and independently distributed as in the real attack, and all responses to $\mathcal{A}_I$ can pass the validity test unless $\mathcal{B}$ aborts. Thus, we find that when a public key PK_{ID} has not been replaced or produced under $coin = 1$, the simulation is perfect as $\mathcal{B}$ knowing the corresponding private key SK_{ID}. Otherwise, a simulation error may occur in *Decapsulation-Oracle*, and let **DecErr** denote this event. Suppose that ID, $PK_{\mathrm{ID}} = (r_{\mathrm{ID}}, Y)$ and c have been issued as a valid decapsulation query. Even if K is a valid session key, there is a possibility that K can be produced without querying $\langle(\mathrm{ID}, T), R\rangle$ to H_2. Let **Valid** be an event that K is a valid session key, $\mathbf{Ask}H_1$ and $\mathbf{Ask}H_2$ be events that $(\mathrm{ID}, r_{\mathrm{ID}}, Y)$ has been queried to H_1 and (ID, T) to H_2 respectively. Since **DecErr** is an event that $\mathbf{Valid}|\neg\mathbf{Ask}H_2$ happens during the entire simulation and q_D *Decapsulation-Oracle* queries are operated, we have $\Pr[\mathbf{DecErr}]=q_D\Pr[\mathbf{Valid}|\neg\mathbf{Ask}H_2]$, where $\Pr[\mathbf{Valid}|\neg\mathbf{Ask}H_2] \leq \Pr[\mathbf{Valid}\wedge\mathbf{Ask}H_1|\neg\mathbf{Ask}H_2]$ + $\Pr[\mathbf{Valid}\wedge\neg\mathbf{Ask}H_1|\neg\mathbf{Ask}H_2] \leq \Pr[\mathbf{Ask}H_1|\neg\mathbf{Ask}H_2]$ + $\Pr[\mathbf{Valid}|\neg\mathbf{Ask}H_1 \wedge \neg\mathbf{Ask}H_2] \leq (q_{H_1}/(2^{l_0}p^2)) + (1/(2^{l_0}p^5))$.

Let the event $(\mathbf{Ask}H_2^* \vee \mathbf{DecErr})|\neg\mathbf{Abort}$ be denoted by $\mathbf{E}$, where **Abort** is an event that $\mathcal{B}$ aborts during the simulation. The probability $\neg\mathbf{Abort}$ that happens is given by $\delta^{q_{prv}}(1-\delta)$ which is maximized at $\delta = 1 - 1/(q_{prv}+1)$. Hence, we have $\Pr[\neg\mathbf{Abort}] \leq 1/(e(q_{prv}+1))$, where e denotes the base of the natural logarithm.

If $\mathbf{E}$ does not happen, it is clear that $\mathcal{A}_I$ does not gain any advantage greater than 1/2 to guess β due to the randomness of the output of the random oracle H_2. Namely, we have $\Pr[\beta' = \beta \mid \neg\mathbf{E}] \leq 1/2$.

By definition of ε, we have $\varepsilon < |\Pr[\beta' = \beta] - (1/2)| = |\Pr[\beta' = \beta|\neg\mathbf{E}]\Pr[\neg\mathbf{E}] + \Pr[\beta' = \beta|\mathbf{E}]\Pr[\mathbf{E}] - (1/2)| \leq |(1/2)\Pr[\neg\mathbf{E}] + \Pr[\mathbf{E}] - (1/2)| = |(1/2)(1 - \Pr[\mathbf{E}]) + \Pr[\mathbf{E}] - (1/2)| = (1/2)\Pr[\mathbf{E}] \leq (\Pr[\mathbf{Ask}H_2^*] + \Pr[\mathbf{DecErr}])/(2\Pr[\neg\mathbf{Abort}]) \leq (e(q_{prv}+1)/2)(q_{H_2}\varepsilon' + (q_D q_{H_1}/(2^{l_0}p^2)) + (q_D/(2^{l_0}p^5)))$. Consequently, we obtain

$$\varepsilon' > \frac{1}{q_{H_2}}\left(\frac{2\varepsilon}{e(q_{prv}+1)} - \frac{q_D q_{H_1}}{2^{l_0}p^2} - \frac{q_D}{2^{l_0}p^5}\right).$$

□

The following lemma shows that our CLE scheme is secure against the Type-II adversary.

Lemma 2. *This CL-KEM is $(q_H, q_{par}, q_{pri}, q_{pub}, q_D, \varepsilon)$-IND-CCA secure against the Type-II adversary $\mathcal{A}$ in the random oracle model, then there exists an algorithm $\mathcal{B}$ that solves the CDH problem with the following advantage*

$$\varepsilon' > \frac{1}{q_{H_2}}\left(\frac{2\varepsilon}{e(q_{prv}+1)} - \frac{q_D q_{H_1}}{2^{l_0}p^2} - \frac{q_D}{2^{l_0}p^5}\right).$$

Proof. Assuming there exists an algorithm $\mathcal{A}_{II}$ who impersonates an "insider" adversary. Suppose that there is another PPT algorithm $\mathcal{B}$ can solve the CDH problem in the instance of (p, P, aP, bP) with probability at least ε' by interacting with $\mathcal{A}_{II}$. To solve this problem, $\mathcal{B}$ needs to simulate a challenger to run each algorithm of **Game-II** for $\mathcal{A}_{II}$ as follows:

Setup. Algorithm $\mathcal{B}$ picks the master secret key $x \in Z_p^*$ randomly and computes $X = xP$. Then $\mathcal{B}$ gives the system parameters $params = \{p, G, P, X, H_1, H_2\}$ to $\mathcal{A}_{II}$, where H_1 and H_2 are random oracles. Adversary $\mathcal{A}_{II}$ queries these two random oracles at any time during its attack. $\mathcal{B}$ responds as follows:

H_1 **Queries.** $\mathcal{B}$ maintains a list of tuples $\langle(\mathrm{ID}, r_{\mathrm{ID}}, Y), v\rangle$ in H_1-**List** L_1. On receiving a query $(\mathrm{ID}, r_{\mathrm{ID}}, Y)$ to H_1:

1) If $\langle(\mathrm{ID}, r_{\mathrm{ID}}, Y), v\rangle$ already appears on the list L_1, responds v as an answer.
2) Otherwise, pick $v \in Z_p^*$ randomly, add $\langle(\mathrm{ID}, r_{\mathrm{ID}}, Y), v\rangle$ to L_1 and return v as an answer.

H_2 **Queries.** $\mathcal{B}$ maintains a list of tuples $\langle(\mathrm{ID}, T), R\rangle$ in H_2-**List** L_2, where $T \in G^{*5}$. On receiving a query (ID, T) to H_2:

1) If $\langle(\mathrm{ID}, T), R\rangle$ exists in the list L_2, return R as an answer.
2) Otherwise, choose $R \in \{0,1\}^*$ uniformly at random, add $\langle(\mathrm{ID}, T), R\rangle$ to L_2 and return R as an answer.

Phase 1. $\mathcal{A}_{II}$ issues the following oracle queries.

Public-Key-Request-Oracle. $\mathcal{B}$ maintains a **PublicKeyList** of tuples $\langle\mathrm{ID}, (r_{\mathrm{ID}}, Y), coin\rangle$. On receiving a query ID, $\mathcal{B}$ responds as follows:

1) If $\langle\mathrm{ID}, (r_{\mathrm{ID}}, Y), coin\rangle$ exists in **PublicKeyList**, return $PK_{\mathrm{ID}} = (r_{\mathrm{ID}}, Y)$ as an answer.
2) Otherwise, pick $coin \in \{0,1\}$ at random so that $\Pr[coin = 0] = \delta$ (δ is the same as it in the proof of **Lemma 1**).
3) If $coin = 0$, choose $y, \alpha \in Z_p^*$ at random and compute $Y = yP$, $r_{\mathrm{ID}} = \alpha P$ and $z_{\mathrm{ID}} = \alpha + xH_1(\mathrm{ID}\|r_{\mathrm{ID}}\|Y)$. Then, add $\langle\mathrm{ID}, (y, z_{\mathrm{ID}}), \alpha\rangle$ to PrivateKeyList and $\langle\mathrm{ID}, (r_{\mathrm{ID}}, Y), coin\rangle$ to PublicKeyList respectively, return $PK_{\mathrm{ID}} = (r_{\mathrm{ID}}, Y)$ as an answer.

4) Otherwise (if $coin = 1$), pick $\alpha, y \in Z_p^*$ at random and compute $r_{\mathrm{ID}} = \alpha aP$, $Y = yP$ and $z_{\mathrm{ID}} = \alpha + bxH_1(\mathrm{ID}\|r_{\mathrm{ID}}\|Y)$. Then, add $\langle \mathrm{ID}, (y, *), \alpha \rangle$ to **PrivateKeyList** (where * denotes the arbitrary value), and $\langle \mathrm{ID}, (r_{\mathrm{ID}}, Y), coin \rangle$ to **PublicKeyList**, return $PK_{\mathrm{ID}} = (r_{\mathrm{ID}}, Y)$ as an answer.

Private-Key-Request-Oracle. $\mathcal{B}$ maintains a **PrivateKeyList** of tuples $\langle \mathrm{ID}, (y, z_{\mathrm{ID}}), \alpha \rangle$. On receiving a query ID, $\mathcal{B}$ responds as follows:

1) Perform *Public-Key-Request-Oracle* on ID to get a tuple $\langle \mathrm{ID}, (r_{\mathrm{ID}}, Y), coin \rangle$ from **PublicKeyList**.
2) If $coin = 0$, search **PrivateKeyList** for a tuple $\langle \mathrm{ID}, (y, z_{\mathrm{ID}}), \alpha \rangle$ and return $SK_{\mathrm{ID}} = (y, z_{\mathrm{ID}})$ as an answer.
3) Otherwise, return "*Abort*" and terminate.

Decapsulation-Oracle. On receiving a query $\langle$ID, PK_{ID}, $c\rangle$, where $PK_{\mathrm{ID}} = (r_{\mathrm{ID}}, Y)$. $\mathcal{B}$ responds as follows:

1) Search a tuple $\langle \mathrm{ID}, (r_{\mathrm{ID}}, Y), coin \rangle$ in **PublicKeyList**. If such a tuple exists and $coin = 0$, search a tuple $\langle \mathrm{ID}, (y, z_{\mathrm{ID}}) \rangle$ in **PrivateKeyList** (Note that $\langle \mathrm{ID}, (r_{\mathrm{ID}}, Y), coin \rangle$ must exist in **PublicKeyList**. While $coin = 0$, the tuple $\langle \mathrm{ID}, (y, z_{\mathrm{ID}}), \alpha \rangle$ exists in **PrivateKeyList**). Then, set $SK_{\mathrm{ID}} = (y, z_{\mathrm{ID}})$ and run the algorithm of *Decap*. Finally, return the results of the *Decap*.
2) Otherwise (if $coin = 1$), run H_1 **queries** to access a tuple $\langle (\mathrm{ID}, r_{\mathrm{ID}}, Y), v \rangle$. If there exists $\langle (\mathrm{ID}, T), R \rangle \in L_2$ such that $R = H_2(\mathrm{ID}\|T)$, return R as the session key and "*Reject*" otherwise.

Challenge Phase. Once $\mathcal{A}_{II}$ decides that *Phase 1* is over, it outputs a challenge identity ID*. On receiving a challenge query ID*, $\mathcal{B}$ responds as follows:

1) Taking ID* as input, $\mathcal{B}$ runs *Public-Key-Request-Oracle* and gets a tuple $\langle \mathrm{ID}^*, (r_{\mathrm{ID}^*}, Y^*), coin \rangle$ from **PublicKeyList**.
2) If $coin = 0$, return "*Abort*" and terminate.
3) Otherwise, do the following:
   a. Search for a tuple $\langle \mathrm{ID}^*, (y^*, z_{\mathrm{ID}^*}), \alpha^* \rangle$ from **PrivateKeyList** (In this case, we know that $r_{\mathrm{ID}^*} = \alpha^* aP$, $Y^* = y^* P$).
   b. Set $c^* = aP$, $c_1^* = a(Y^* + r_{\mathrm{ID}^*} + XH_1(\mathrm{ID}^* \| r_{\mathrm{ID}^*} \| Y^*))$ and $c_2^* = aY^*$. Also, note that $\mathcal{B}$ does not know "a". Then compute $v^* = H_1(\mathrm{ID}^* \| r_{\mathrm{ID}^*} \| Y^*)$.
   c. Pick $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space.
   d. Compute $K_1 = H_2(\mathrm{ID}^* \| (r_{\mathrm{ID}^*}, Y^*) \| c^* \| c_1^* \| c_2^*)$.
4) Choose a bit $\beta \in_R \{0, 1\}$ and return (c^*, K_β) to $\mathcal{A}_{II}$.

Phase 2. $\mathcal{A}_{II}$ repeats the same methods as in *Phase 1*. Moreover, no private key extraction on ID* is allowed and no *Decapsulation-Oracle* query should be made on the encapsulation c^* for ID*.

Guess. $\mathcal{A}_{II}$ outputs a guess β' for β, and wins the game if $\beta' = \beta$. Then, $\mathcal{B}$ enables to solve the CDH problem by computing $(c^* \cdot z_{\mathrm{ID}^*} - r_{\mathrm{ID}^*})/(x \cdot v^*)$.

Analysis. Similar to Analysis in the proof of **Lemma 1**.

Consequently, we obtain

$$\varepsilon' > \frac{1}{q_{H_2}}\left(\frac{2\varepsilon}{e(q_{prv}+1)} - \frac{q_D q_{H_1}}{2^{l_0} p^2} - \frac{q_D}{2^{l_0} p^5}\right).$$

□

In conclusion, based on these two lemmas, we complete the proof of **Theorem 1**.

4 Comparisons

In this section, we compare our CL-KEM with previous protocols [7, 10, 12] on the computation complexity of encapsulation (**Enc**) and decapsulation (**Dec**), the bandwidth of the encapsulation (**Bandwidth**) and the running time (**Time**) of one-round *Encap-Decap* of each scheme. Without considering the addition of two points, hash function and exclusive-OR operations, we denote the cost of a bilinear pairing by P, the cost of an exponentiation by E , and the cost of a scalar multiplication in the additive cyclic group by S.

This CL-KEM is tested on a laptop with the Intel Core i5-2400 at a frequency of 3.10 GHz processor, 3GB memory and Ubuntu-12.04 operation system, using the pairing based cryptography (PBC) library (version 0.5.13 [13]). The implementation takes use of a 160-bit elliptic curve group based on the supersingular curve $y^2 = x^3 + x$ over a 512-bit finite field with embedding degree 2. Then, the average running time of each operation is obtained and demonstrated in Table 2.

Table 2: Cryptographic operation time

Pairing	Exponentiation	Scalar multiplication
3.93 ms	3.35 ms	3.28 ms

As to communication cost, we analyze it in terms of bandwidth of transmitting encapsulation. Suppose that the output of one way Hash function is 160-bit, and the elements of multiplicative group is 1024-bit (e.g., parameters in RSA). In our protocol, one encapsulation contains one point, thus the bandwidth of our protocol is $160/8 = 20$ bytes. In [7, 10], each encapsulation contains two exponentiations, thus the bandwidths of [7, 10] are

$(1024 \times 2)/8 = 256$ bytes respectively. At last, in Lippold et al.'s scheme [12], the encapsulation contains two exponentiations and one hash value, the bandwidth of it is $(1024 \times 2 + 160)/8 = 276$ bytes. The detailed results are listed in Table 3, and the bandwidth of our scheme is the smallest one.

Table 3: Comparison of the related schemes

Schemes	Enc	Dec	Bandwidth	Time
[10]	4E	2E	256 bytes	20.10 ms
[12]	5E	3P+6E	276 bytes	48.64 ms
[7]	4E	2E	256 bytes	20.10 ms
Ours	4S	2S	20 bytes	19.68 ms

The computation and communication cost in this scheme is less than others, which shows that our scheme enables to provide an efficient method to protect the confidential of the session key between patient and doctor in e-healthcare system.

5 Conclusions

We have proposed an efficient certificateless key encapsulation mechanism for e-healthcare system to protect the confidentiality of the session key in the hybrid encryption scheme. In terms of security, we prove that this scheme is IND-CCA secure in the random oracle model assuming that CDH problem is intractable. Furthermore, our protocol promotes the trust hierarchy of the medical server to the Girault's trust Level 3. A thorough performance evaluation and experiments on PC indicate that the proposal is advantageous over the related schemes in efficiency. Thus, all these attributes render this scheme a promising approach in session key protection to e-healthcare system.

Acknowledgments

This work was supported by National Natural Science Foundation of China (Grant Nos. 11171200, 11426148) and Fundamental Research Funds for the Central Universities (Grant No. GK201402006).

References

[1] M. Abe, R. Gennaro, and K. Kurosawa, "Tag-kem/dem: A new framework for hybrid encryption," *Journal of Cryptology*, vol. 21, no. 1, pp. 97–130, 2008.

[2] S. S. Al-Riyami and K. G. Paterson, "Certificateless public key cryptography," in *Advances in Cryptology (ASIACRYRT03)*, pp. 452–473, Taipei, Taiwan, Nov. 2003.

[3] K. Bentahar, P. Farshim, J. M. Lee, and N. P. Smart, "Generic constructions of identity-based and certificateless KEMs," *Journal of Cryptology*, vol. 21, no. 1, pp. 178–199, 2008.

[4] J. Daemen and V. Rijmen, *Advanced Encryption Standard (AES)*, Technical Report 197, Dec. 2001.

[5] A. Dent, "A designers guide to KEMs," in *Cryptography and Coding*, pp. 133–151, Cirencester, UK, Dec. 2003.

[6] M. Girault, "Self-certificated public keys," in *Advances in Cryptology (EUROCRYPTO91)*, pp. 34–46, Brighton, UK, Apr. 2010.

[7] D. Hofheinz and E. Kiltz, "Secure hybrid encryption from weakened key encapsulation," in *Advances in Cryptology (CRYPTO07)*, pp. 553–571, California, USA, Aug. 2007.

[8] Q. Huang and D. S. Wong, "Generic certificateless key encapsulation mechanism," in *Information Security and Privacy*, pp. 215–229, Townsville, Australia, July 2007.

[9] E. Kiltz, "Chosen-ciphertext secure key-encapsulation based on gap hashed diffie-hellman," in *Public Key Cryptography (PKC'07)*, pp. 282–297, Beijing, China, Apr. 2007.

[10] K. Kurosawa and Y. Desmedt, "A new paradigm of hybrid encryption scheme," in *Advances in Cryptology (CRYPTO04)*, pp. 426–442, California, USA, Aug. 2004.

[11] Congress Public Law, *Health Insurance Portability and Accountability Act of 1996*, Technical Report 104, June 1996.

[12] G. Lippold, C. Boyd, and J. M. G. Nieto, "Efficient certificateless KEM in the standard model," in *Information, Security and Cryptology*, pp. 34–46, Seoul, Korea, Dec. 2010.

[13] B. Lynn, *The Pairing-based Cryptography Library*, PBC Library, May 2015. (http://crypto.stanford.edu/pbc/)

[14] J. H. Oh, K. K. Lee, and S. J. Moon, "How to solve key escrow and identity revocation in identity based encryption scheme," in *Proceedings of 1st International Conference on Information System Security*, pp. 290–303, Kolkata, India, Dec. 2005.

[15] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Advances in Cryptology-CRYPTO84*, pp. 47–53, California, USA, Dec. 1985.

[16] W. Tuchman and C. Meyer, *Date Encryption Standard (DES)*, Technical Report 46, July 1977.

[17] M. K. Watfa, *E-healthcare Systems and Wireless Communications: Current and Future Challenges*, Technical Report 27, Sep. 2012.

Shi Hui-Xian: received the B.S. and Ph.D degrees in Department of Mathematics and Information Science from Shaanxi Normal University, Xi'an, China, in 2007 and 2013, respectively. Now she is a post-doctoral in Department of Computer Science in Shaanxi normal University. Her present research interests include model

checking, fuzzy logic and uncertainty reasoning.

Guo Rui: received the BS degree in Math and Applied Math from Henan University of Science and Technology, the MS degree in Applied Math from Shaanxi Normal University. He is currently a candidate for Ph.D in the Department of State Key Laboratory of Networking and Switch Technology, Beijing University of Posts and Telecommunications. His present research interests include cryptography, information security and applied mathematics.