Stateful Certificateless Public Key Encryption with Application in Public Cloud

S. Sree Vivek(✉)

Samsung Research Institute, Bangalore, India
sreevivek.s@samsung.com

Abstract. Certificateless cryptography eliminates the key escrow problem inherent in identity based cryptosystem. Certificatateless systems are preferred in public cloud to offer security because it solves two different problems simultaneously, namely, the key escrow problem and the cumbersome certificate management. A stateful public key encryption scheme is a cryptographic primitive, in which the sender maintains state information to perform encryption. The encryption algorithm takes the intended message, receiver's public key and the current state information to produce the ciphertext, and possibly updates the state information. Decryption is straightforward and depends only on the ciphertext and secret key of the receiver. In this paper, we propose the first stateful certificateless public key encryption scheme and prove the security of the scheme in the random oracle model. This scheme finds very interesting application for sharing data in an encrypted cloud storage system.

Keywords: Certificateless encryption · Stateful cryptography · Random oracle model · Provable security · Cloud data security · Sharing cloud data

1 Introduction

Certificateless Cryptography (CLC) introduced by Al-Riyami et al. [2] is a variant of Identity Based Cryptography (IBC), which intends to prevent the key escrow problem. Usually, in IBC the private key of a user is generated by the Private Key Generator (PKG), who has to be trusted by all users of the system. In the case of a PKG compromise, a total-break of the system is possible. This is called the key escrow problem. In order to prevent this, the key generation process is split between the KGC (Key Generation Center - The central authority in CLC) and the user. The KGC first generates the private key for a user, which is called as the partial private key of the user. The remaining part of the private key is a random secret value generated by the user, and is never revealed to anyone, not even to the KGC. This key is called as the user secret value and the user generates the public key corresponding to this key. All cryptographic operations by the user are performed by using the full private key which involves both the partial private key and the user secret value.

I. Bica et al. (Eds.): SECITC 2015, LNCS 9522, pp. 130–149, 2015.
DOI: 10.1007/978-3-319-27179-8_10

Having introduced CLC, we now move on to stateful public key encryption (PKE) schemes. PKE schemes make use of compute intensive exponentiation computations to perform encryption as well as decryption. The order of complexity is roughly considered to be one thousand times that of a block cipher or hash function evaluation. This results in slowdown of the system as well as hinders the use of public key cryptography in systems with limited computing power. Public key cryptography operations are very expensive that they drain the battery of devices easily. This seems to be a very important and severe limitation on cell phones, personal digital assistants, tablets, wearables, RFID chips and sensors. Hence, researchers are very much interested in reducing the cost of exponentiation, which is a very crucial operation for PKE schemes. It was stated by Bellare et al. [5] that "a 10% improvement would be very welcome and a 50% improvement would be dramatic". However, lot of intellectual energy is pumped in to improve the schemes by proposing time-space trade-off mechanisms like pre-computation of exponentiation and faster implementations for exponentiations.

In a stateful encryption scheme, the sender maintains a state information that can be reused across various encryptions during a session. A session may be marked by the communication between a sender and a fixed receiver. Thus if the communication has to occur between two fixed entities, the sender has to use a symmetric key (the key used for encryption and decryption in any symmetric key encryption scheme) which is derived using the public key of the receiver. A stateful encryption algorithm is deterministic with respect to the state and public key. Thus, this key has to be computed only for the first time the sender communicates with the receiver. After which the key can be reused through out the state, which reduces the cost of further public key encryptions to that receiver.

Moreover, it should be noticed that reusing randomness is not straightforward in any cryptographic operation. In the history, we have learned hard lessons due to reuse of randomness. One of the well known examples is the attack on Sony's PlayStation 3 in 2010. A group of attackers recovered the private key of Elliptic Curve Digital Signature Algorithm (ECDSA) used by Sony to sign software for the PlayStation 3 game console. This attack was possible because ECDSA has a randomized signature generation algorithm and Sony reused the randomness used to generate the signature [6]. One more well known example is in the case of RSA. The entropy of the output distribution of standardized RSA key generation is always almost maximal. The outputs are hard to factor if factoring in general is hard where the primes were chosen at random but it was identified in [12] that the random primes did not satisfy this requirement on the distribution of the RSA keys. This exposed a considerable number of RSA private keys used in PGP system. Hence we emphasize that reuse of randomness should be done with utmost care and the resulting scheme should be proven secure, taking the reuse into account.

Motivation: Recent advancements in technology has made huge data storage available for users in the name of Cloud Storage. Cloud platforms such as

Dropbox, Skybox, Oracle, Amazon provide users with huge space for them to store their data. However reputed the cloud storage provider is, other organizations who want to make use of the cloud storage do no trust them to store sensitive data. Hence a need for secure cloud storage arose. In secure cloud storage, each user has a public/private key pair created by the user. The public key is used to encrypt a symmetric key which in turn is used to encrypt the data. The encrypted data (ciphertext) is then stored in the cloud. The corresponding private key is used to decrypt and obtain the symmetric key which is used to decrypt the ciphertext and obtain the actual message. However, this approach requires a certified public key in order to withstand man-in-the-middle attacks and public key replacement attacks, and requires the presence of a Certification Authority (CA).

To make the system more convenient, the identity of the user could be used to generate the public keys and the corresponding private key could be obtained from the trusted authority (PKG). In this case the PKG (the cloud service provider) is a fully trusted entity and knows the private key of all users in the system. Certificateless Encryption (CLE) schemes find great application in this scenario. In a CLE, the identity of a user along with a user defined public key acts as the full public key of the user. Unlike Public Key Infrastructure (PKI), these public keys need not be certified by centralized authorities, because changing the public key in the public repository will be useful only to the trusted KGC in the CLE and hence the KGC will be accountable for any replacement of the public keys in the repository.

Consider a scenario wherein a user has n different files (may be photos, documents etc.). In order to maintain privacy, the user has to encrypt each file with different symmetric keys. This is because, if all the files were encrypted with the same key and if the user shares the key of one file, he is loosing the keys of all other files too. In order to avoid this each file should be encrypted with different symmetric keys. Thus all the n files are encrypted with n different symmetric keys and stored in the cloud storage. In case, if the user (owner of the files) wants to share a subset of k files to another user (the receiver), the naive way is to use a PKE scheme and encrypt the symmetric keys to the receiver. The receiver on receiving the encrypted keys can decrypt them using his private key and use the symmetric keys to decrypt the actual file from the downloaded ciphertext. If k is large and if the owner uses a stateless encryption scheme, he has to perform $\mathcal{O}(k)$ exponentiations. The advantage of using a stateful PKE scheme is that it requires only $\mathcal{O}(1)$ exponentiation when the receiver is fixed.

Figure 1 shows how a user registers with the KGC to avail secure cloud storage and how his data is encrypted and stored in the cloud. User A sends his identity, ID_A and requests for the partial private key to the KGC. The KGC has a master private key msk and a set of public system parameters $params$, which are used to generate the partial private key SK_A of the user A. SK_A is sent to A through a secure channel. After receiving the partial private key user A chooses his own user defined private key sk_A and computes the corresponding public key pk_A. The user defined public key pk_A is sent to the KGC and is stored in the user

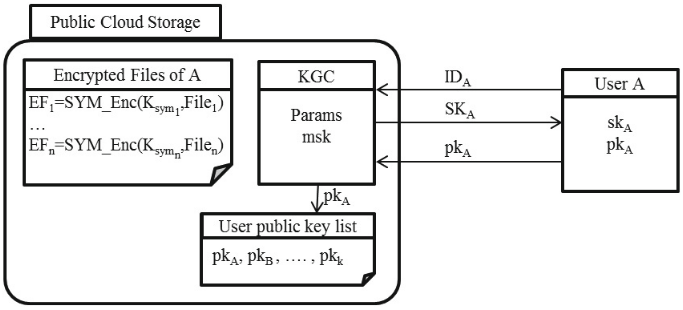

Fig. 1. User registration and encryption on cloud

public key list maintained by the KGC. Hence the private key of A is $\langle sk_A, SK_A\rangle$ and the corresponding public key is $\langle pk_A, ID_A\rangle$. To encrypt a message to A, the sender has to use ID_A and pk_A along with $params$. When user A wants to upload a file $File_i$ to the cloud, he has to encrypt the file using a symmetric key encryption scheme with a symmetric key K_{sym_i} to obtain the encrypted file $EF_i = SYM\_Enc(K_{sym_i}, File_i)$. (Note that we assume that each symmetric key is unique and there is an efficient way for the owner of the file to uniquely obtain the key using a private Pseudo Random Number Generator and other attributes such as file name, modified and created date etc., in a secure way. We do not explain it here since it is out of the scope of the problem addressed here) The encrypted file EF_i is then stored in the cloud.

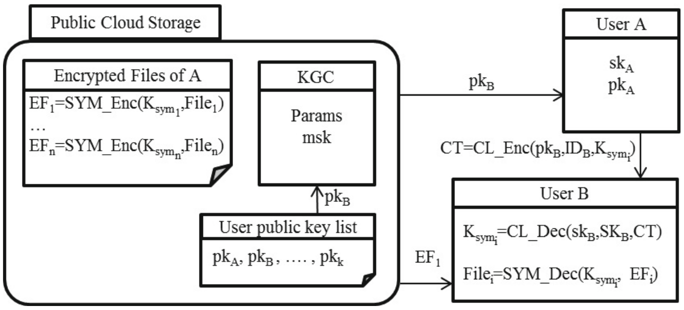

Fig. 2. Sharing encrypted contents using CLE

Figure 2 shows how user A shares an encrypted file to another user. Let us consider that user A wants to share his contents to user B, whose public key is $\langle pk_B, ID_B\rangle$. The owner A shares the encrypted file to user B first. Then he obtains the user public key pk_B of B from the user public key list maintained by the KGC. Using pk_B and ID_B with a CLE scheme he encrypts the symmetric key K_{sym_i} corresponding to the file $File_i$ to user B as $CT =$

$CL\_Enc(pk_B, ID_B, K_{sym_i})$ and sends CT to B. User B upon receiving CT, decrypts it as: $K_{sym_i} = CL\_Dec(sk_B, SK_B, CT)$ and obtains the symmetric key. Now, B uses K_{sym_i} to decrypt EF_i as $File_i = SYM\_Dec(K_{sym_i}, EF_i)$.

Related Work: Al-Riyami and Paterson in [2] have shown realization for CLE, signature (CLS) and key exchange (CLK) schemes in their work. Huang et al. [10] and Castro et al. [7] independently showed that the signature scheme in [2] is not secure against Type-I adversary (explained in later sections). In fact they showed that it is possible for a Type-I adversary to replace the public key of the user and attack the scheme. They also gave a new certificateless signature scheme. A lot of CLE schemes were proposed, whose security were proved both in the random oracle model [4,8,17,18] and standard model [13,15]. Recently, Dent [9] gave a survey on the various security models for CLE schemes, mentioning the subtle difference in the level of security offered by each model. Dent also gave a generic construction and an efficient construction for CLE. The initial constructs for certificateless cryptosystem were all based on bilinear pairing [8,13,15,17]. Baek et al. [4] were the first to propose a CLE scheme without bilinear pairing. Certificateless cryptosystem are prone to key replacement attack because the public keys are not certified and anyone can replace the public key of any legitimate user in the system. The challenging task in the design of certificateless cryptosystem is to come up with schemes which resists key replacement attacks. The CLE in [4] did not withstand key replacement attack, which was pointed out by Sun et al. in [18]. Sun et al. fixed the problem by changing the partial key extract and setting public key procedures. Both the aforementioned schemes, namely [4,18] were based on multiplicative groups. Lai et al. in [11] proposed the first RSA-based CLE scheme. They have proved their scheme secure against chosen plaintext attack (CPA). Later, in [19] Vivek et al. proposed a CCA secure scheme based on the RSA assumption.

There are several PKE schemes, which make use of transformations to achieve CCA security and some of them are customized design. There is no known straightforward ways to make these schemes stateful. Even though some efforts were made in this direction, the ciphertext size will be large due to the usage of CCA secure symmetric key encryption schemes. However, there are PKE schemes that are designed to be stateful, namely [3,5,20] and few stateful IBE schemes were also found in the literature [3,16]. There are no known stateful certificateless PKE schemes present in the literature, which motivated us to look forward in this direction.

Our Contribution: In this paper we propose the first stateful certificateless encryption scheme. Our scheme finds straightforward application in sharing encrypted contents in the cloud efficiently when the cloud data is accessed through resource constrained devices. This efficiency comes in due to the fact that when different files are encrypted with different keys as shown in Fig. 1, sharing huge number of encrypted files will involve transportation of huge number of symmetric keys. Since in a stateful encryption scheme, the randomness used to encrypt the data is reused across a session for communication with a fixed receiver, we are able to reduce the cost of encryption required to share the keys. In the example stated above if there are k files to be shared with a receiver, the owner has to perform only $\mathcal{O}(1)$ exponentiation operations, where as in the naive way, the owner has to perform $\mathcal{O}(k)$ exponentiations. Our scheme offers compact ciphertext with ciphertext verifiability.

2 Preliminaries

In this section, we give the definition of hardness assumptions, framework and the security model used in our paper.

2.1 Review of Computational Assumptions

Definition 1 ***Computational Diffie-Hellman Problem (CDHP).*** *Given $\langle g, g^a, g^b \rangle \in \mathbb{G}^3$ for unknown $a, b \in \mathbb{Z}_q^*$, where $\mathbb{G}$ is a cyclic prime order multiplicative group with g as a generator and q the order of the group, the CDH problem in $\mathbb{G}$ is to compute g^{ab}.*

The advantage of any probabilistic polynomial time algorithm $\mathcal{A}$ in solving the CDH problem in $\mathbb{G}$ is defined as

$$Adv_{\mathcal{A}}^{CDH} = Pr\left[\mathcal{A}(g, g^a, g^b) = g^{ab} \mid a, b \in \mathbb{Z}_q^*\right]$$

The CDH Assumption is that, for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage $Adv_{\mathcal{A}}^{CDH}$ is negligibly small.

Definition 2 ***(Strong Diffie Hellman (SDH) Problem as given in*** [1]***).*** *Let κ be the security parameter and $\mathbb{G}$ be a multiplicative group of order q, where $|q| = \kappa$. Given $\langle g, g^a, g^b \rangle \in_R \mathbb{G}^3$ and access to a Decision Diffie Hellman (DDH) oracle $\mathcal{DDH}_{g,a}(.,.)$ which on input g^b and g^c outputs* `true` *if and only if $g^{ab} = g^c$, the strong Diffie Hellman problem is to compute $g^{ab} \in \mathbb{G}$.*

The advantage of an adversary $\mathcal{A}$ in solving the strong Diffie Hellman problem is defined as the probability with which $\mathcal{A}$ solves the above strong Diffie Hellman problem.

$$Adv_{\mathcal{A}}^{SDH} = Pr[\mathcal{A}(g, g^a, g^b) = g^{ab} | \mathcal{DDH}_{g,a}(.,.)]$$

The strong Diffie Hellman assumption holds in $\mathbb{G}$ if for all polynomial time adversaries $\mathcal{A}$, the advantage $Adv_{\mathcal{A}}^{SDH}$ is negligible.

Note: In pairing groups (also known as gap groups), the DDH oracle can be efficiently instantiated and hence the strong Diffie Hellman problem is equivalent to the Gap Diffie Hellman problem [14].

2.2 Framework for Stateful CLE

In this section, we discuss the general framework for stateful CLE. We adopt the definition of certificateless public key encryption, given by Baek et al. [4]. Their definition of CLE is weaker than the original definition by Al-Riyami and Paterson [2] because the user has to obtain a partial public key from the KGC before he can create his public key (While in Al-Riyami and Paterson's original CLE this is not the case). A stateful certificateless public-key encryption scheme is defined by seven probabilistic, polynomial-time algorithms which are defined below:

Setup: This algorithm takes as input a security parameter κ and returns the master private key msk and the system public parameters $params$. This algorithm is run by the KGC in order to initialize a certificateless system.

Partial Private Key Extract: This algorithm takes as input the public parameters $params$, the master private key msk and an identity $ID_A \in \{0,1\}^*$ of a user A. It outputs the partial private key d_A and a partial public key y_A of user A. This algorithm is run by the KGC once for each user and the corresponding partial private key and partial public key is given to A through a secure and authenticated channel.

Set Full Private Key: This algorithm is run once by each user. It takes the public parameters $params$, the user identity ID_A and A's partial private key d_A as input. The algorithm generates a secret value $sk_A \in \mathcal{S}$, where $\mathcal{S}$ is the secret value space. Now, the full private key D_A of A, is a combination of the secret value sk_A and the partial private key d_A.

Set Full Public Key: This algorithm run by the user, takes as input the public parameters $params$, a user, say A's partial public key y_A and the full private key D_A. It outputs the full public key PK_A for A. This algorithm is run once by each user and the resulting full public key is widely and freely distributed. The full public key of user A consists of PK_A and ID_A.

New State Generation: This algorithm is used to generate a set of state information used for encryption. The sender executes this algorithm and keeps the information confidential. The sender's state information is not known to any entity. The state information st is generated by taking $params$ as input.

Encryption: This algorithm takes as input the public parameters $params$, a receiver identity, say ID_A, the corresponding full public key PK_A, the state information st_i (corresponding to the i^{th} state) and a message $m \in \mathcal{M}$. The output of this algorithm is the ciphertext $CT \in \mathcal{CS}$. Note that $\mathcal{M}$ is the message space and $\mathcal{CS}$ is the ciphertext space.

Decryption: This algorithm takes as input the public parameters $params$, a user, say A's private key D_A and a ciphertext $CT \in \mathcal{CS}$. It returns either a message $m \in \mathcal{M}$ - if the ciphertext is valid, or $'Invalid'$ - otherwise.

2.3 Security Model for CLE

We review the notion of Type-I and Type-II adversaries and provide the security model for stateful CLE. The confidentiality of a stateful CLE scheme is proved by means of an interactive game between a challenger $\mathcal{C}$ and an adversary. In the confidentiality game for stateful certificateless encryption (IND-stCLE-CCA2), the adversary is given access to the following six oracles. These oracles are simulated by $\mathcal{C}$:

Partial Key Extract for $\boldsymbol{ID_A}$: $\mathcal{C}$ responds by returning the partial private key SK_A and the partial public key PPK_A of the user A.

Extract Secret Value for $\boldsymbol{ID_A}$: If A's public key has not been replaced then $\mathcal{C}$ responds with the secret value sk_A for user A. If the adversary has already replaced A's public key, then $\mathcal{C}$ does not provide the corresponding private key to the adversary.

Request Public Key for $\boldsymbol{ID_A}$: $\mathcal{C}$ responds by returning the full public key PK_A for user A. (First by choosing a secret value if necessary).

Replace Public Key for $\boldsymbol{ID_A}$: The adversary can repeatedly replace the public key PK_A for a user A with any valid public key PK'_A of its choice. The adversary generates the new valid public key and sends it to $\mathcal{C}$. The current value of the user's public key is used by $\mathcal{C}$ in any computations or responses.

Encryption($\boldsymbol{ID_i, st_j, m_k}$): Encryption queries for any number of messages ($k = 1$ to $\hat{m}$) for a given state st_j ($j = 1$ to $\hat{n}$), where $\hat{m}$ and $\hat{n}$ are the upper bounds for the number of messages that can be encrypted in a state and total number of states respectively, for whose combination $\mathcal{A}$ can query this oracle.

Decryption($\boldsymbol{CT, ID_A}$): The adversary can issue a decryption query for ciphertext CT and identity ID_A of its choice, $\mathcal{C}$ decrypts CT and returns the corresponding message to the adversary. $\mathcal{C}$ should be able to properly decrypt ciphertexts, even for those users whose public key has been replaced, i.e. this oracle provides the decryption of a ciphertext, which is generated with the current valid public key. The strong decryption oracle returns *Invalid*, if the ciphertext corresponding to any of the previous public keys were queried. This is a strong property of the security model (Note that, $\mathcal{C}$ may not know the private key corresponding to the current public key of the user. This is true if public key is replaced by the adversary). However, this property ensures that the model captures the fact that changing a user's public key to a value of the adversary's choice may give the adversary an advantage in breaking the scheme. This is called as strong decryption in [9]. Our scheme provides strong decryption for Type-I adversary.

There are two types of adversaries (namely Type-I and Type-II) to be considered for stateful certificateless encryption scheme. The Type-I adversary models the attack by a third party attacker, (i.e. anyone except the legitimate receiver or the KGC) who is trying to gain some information about a message from the encryption. The Type-II adversary models the honest-but-curious KGC who tries to break the confidentiality of the scheme. Here, the attacker is allowed to have access to master private key msk. This means that we do not have to give the attacker explicit access to partial key extraction, as the adversary is able to compute these value on its own. The most important point about Type-II security is that the adversary modeling the KGC should not have replaced the public key for the target identity before the challenge is issued.

Constraints for Type-I and Type-II Adversaries: The IND-stCLE-CCA2 security model distinguishes the two types of adversary Type-I and Type-II with the following constraints.

- Type-I adversary $\mathcal{A}_I$ is allowed to change the public keys of users at will but does not have access to the master private key msk.
- Type-II adversary $\mathcal{A}_{II}$ is equipped with the master private key msk but is not allowed to replace public keys corresponding to the target identity.

IND-stCLE-CCA2 Game for Type-I Adversary: The game is named as IND-stCLE-CCA2-I. This game, played between the challenger $\mathcal{C}$ and the Type-I adversary $\mathcal{A}_I$, is defined below:

Setup: Challenger $\mathcal{C}$ runs the setup algorithm to generate master private key msk and public parameters $params$. $\mathcal{C}$ gives $params$ to $\mathcal{A}_I$ while keeping msk secret. After receiving $params$, $\mathcal{A}_I$ interacts with $\mathcal{C}$ in two phases:

Phase I: $\mathcal{A}_I$ is given access to all the six oracles. $\mathcal{A}_I$ adaptively queries the oracles consistent with the constraint that the type-I adversary $\mathcal{A}_I$ is allowed to change the public keys of users at will but does not have access to the master private key msk.

Challenge: At the end of ***Phase I***, $\mathcal{A}_I$ gives two messages m_0 and m_1 of equal length to $\mathcal{C}$ on which it wishes to be challenged. $\mathcal{C}$ randomly chooses a bit $\delta \in_R \{0,1\}$ and encrypts m_δ with the target identity ID^*'s public key for the state st^* to form the challenge ciphertext CT^* and sends it to $\mathcal{A}_I$ as the challenge. (Note that the partial Private Key corresponding to ID^* should not be queried by $\mathcal{A}_I$ but the secret value corresponding to ID^* may be queried. This makes our security model stronger when compared to the security models of [11,18].)

Phase II: $\mathcal{A}_I$ adaptively queries the oracles consistent with the constraints for Type-I adversary described above. Besides this $\mathcal{A}_I$ cannot query *Decryption* on (CT^*, ID^*) and the partial private key of the receiver should not have been queried to the *Extract Partial Private Key* oracle. $\mathcal{A}_I$ gets oracle access to all ciphertexts for any message including m_0 and m_1 for the state information st^* through the encryption oracle $Encryption(params, st^*, m_j)$, where $j \leq \hat{m}$.

Guess: $\mathcal{A}_I$ outputs a bit δ' at the end of the game. $\mathcal{A}_I$ wins the IND-stCLE-CCA2-I game if $\delta' = \delta$. The advantage of $\mathcal{A}_I$ is defined as -

$$Adv_{\mathcal{A}_I}^{IND-stCLE-CCA2-I} = |2Pr\left[\delta = \delta'\right] - 1|$$

IND-stCLE-CCA2 Game for Type-II Adversary: The game is named as IND-stCLE-CCA2-II. This game, played between the challenger $\mathcal{C}$ and the Type-II adversary $\mathcal{A}_{II}$, is defined below:

Setup: Challenger $\mathcal{C}$ runs the setup algorithm to generate master private key msk and public parameters $params$. $\mathcal{C}$ gives $params$ and the master private key msk to $\mathcal{A}_{II}$. After receiving $params$, $\mathcal{A}_{II}$ interacts with $\mathcal{C}$ in two phases:

Phase I: $\mathcal{A}_{II}$ is not given access to the *Extract partial Private Key* oracle because $\mathcal{A}_{II}$ knows msk, it can generate the partial private key of any user in the system. All other oracles are accessible by $\mathcal{A}_{II}$. $\mathcal{A}_{II}$ adaptively queries the oracles consistent with the constraint that the type-II adversary $\mathcal{A}_{II}$ is equipped with the master private key msk but is not allowed to replace public keys corresponding to the target identity.

Challenge: At the end of ***Phase I***, $\mathcal{A}_{II}$ gives two messages m_0 and m_1 of equal length to $\mathcal{C}$ on which it wishes to be challenged. $\mathcal{C}$ randomly chooses a bit $\delta \in_R \{0,1\}$ and encrypts m_δ with the target identity ID^*'s public key using the state information st^* to form the challenge ciphertext CT^* and sends it to $\mathcal{A}_{II}$ as the challenge. (Note that the Secret Value Corresponding to ID^* should not be queried by $\mathcal{A}_{II}$ and the public key corresponding to ID^* should not be replaced during ***Phase I***.)

Phase II: $\mathcal{A}_{II}$ adaptively queries the oracles consistent with the constraints for Type-II adversary described above. Besides this $\mathcal{A}_{II}$ cannot query *Decryption* on (CT^*, ID^*) and the Secret Value corresponding to the receiver should not be queried to the *Extract Secret Value* oracle and the public key corresponding to ID^* should not be replaced during ***Phase I***. $\mathcal{A}_{II}$ gets oracle access to all ciphertexts for any message including m_0 and m_1 for the state information st^* through the encryption oracle $Encryption(params, st^*, m_j)$, where $j \leq \hat{m}$.

Guess: $\mathcal{A}_{II}$ outputs a bit δ' at the end of the game. $\mathcal{A}_{II}$ wins the IND-stCLE-CCA2-II game if $\delta' = \delta$. The advantage of $\mathcal{A}_{II}$ is defined as -

$$Adv_{\mathcal{A}_{II}}^{IND-stCLE-CCA2-II} = |2Pr[\delta = \delta'] - 1|$$

3 Our Scheme - `StCLE`

In this section, we propose our stateful certificateless encryption scheme. The scheme has the following algorithms. Unless stated otherwise, all computations are done *mod p*.

Setup: The KGC does the following to initialize the system and to setup the public parameters. Let κ be the security parameter.

– Choose two large primes p and q such that $q|(p-1)$ and $|q| \geq \kappa$. Choose $g \in_R \mathbb{Z}_p^*$ with order q, $z \in_R \mathbb{Z}_q^*$ and compute $y = g^z$. Choose five cryptographic hash functions $F : \mathbb{Z}_p^* \rightarrow \mathbb{Z}_q^*$, $G : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$, $H : \{0,1\}^* \rightarrow \{0,1\}^{l_m}$, $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ and $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$, where l_m is the size of the message.

- The KGC publicizes the system parameters, $params = \langle p, q, g, y, F, G, H, H_1, H_2\rangle$ and keeps z as the master private key.

Partial Key Extract: This algorithm is executed by the KGC and upon receiving the identity ID_A of a user A the KGC performs the following to generate the corresponding partial private key d_A.

- Choose $s_{i0}, s_{i1} \in_R \mathbb{Z}_q^*$, compute $y_{A0} = g^{s_{i0}}$ and $y_{A1} = g^{s_{i1}}$.
- Compute $d_{A0} = s_{i0} + zH_1(ID_A, y_{A0})\ mod\ q$ and $d_{A1} = s_{i1} + zH_2(ID_A, y_{A0}, y_{A1})\ mod\ q$.
- Output $d_A = \langle d_{A0}, d_{A1}\rangle$ and $y_A = \langle y_{A0}, y, A1\rangle$.

The validity of the partial private key can be verified by user A by performing the following check:

$$g^{d_{A0}} g^{d_{A1}} \stackrel{?}{=} y_{A0} y^{H_1(ID_A, y_{A0})} y_{A1} y^{H_2(ID_A, y_{A0}, y_{A1})} \tag{1}$$

Set Full Private Key: On receiving the partial private key the user with identity ID_A does the following to generate his full private key.

- Choose $x_A \in_R \mathbb{Z}_q^*$ as his secret value.
- Set the private key as $D_A = \langle D_A^{(1)}, D_A^{(2)}\rangle = \langle d_{A0}, x_A\rangle$. (Note that both the KGC and the corresponding user knows $D_A^{(1)}$ and the user with identity ID_A alone knows $D_A^{(2)}$).

Set Full Public Key: The user with identity ID_A computes the public key corresponding to his private key as described below:

- Compute $g_A = g^{D_A^{(2)}}$.
- Make $PK_A = \langle PK_A^{(1)}, PK_A^{(2)}, PK_A^{(3)}, PK_A^{(4)}\rangle = \langle y_{A0}, y_{A1}, d_{A1}, g_A\rangle$ public.

Now, any one can verify the public key by checking:

$$g^{PK_A^{(3)}} \stackrel{?}{=} PK_A^{(2)} y^{H_2(ID_A, PK_A^{(1)}, PK_A^{(2)})} \tag{2}$$

It should be noted that there is no verification for $PK_A^{(4)}$.

New State Generation: Recall that the sender's state information is not known to any entity other than the sender himself. Let i represent the index of the current state and hence the current state will be referred as st_i. The sender generates the state information as follows:

- Choose $r_i \in_R \mathbb{Z}_q^*$. Compute $u_i = F(g^{r_i}) \in \mathbb{Z}_q^*$, $s_i = r_i u_i\ mod\ q$, $v_i = g^{s_i}$, $w_{i1} = (PK_A^{(1)} y^{H_1(ID_A, y_{A0})})^{s_i}$ and $w_{i2} = (PK_A^{(4)})^{s_i}$.

The state information $st_i = \langle u_i, v_i, s_i, w_{i1}, w_{i2}, \texttt{index}\rangle$.

Encryption: To encrypt a message m to a user with identity ID_A, one has to perform the following steps:

- Compute $c_1 = g^{si}$, $c_2 = u_i \oplus G(ID_A, c_1, m, w_{i1}, w_{i2}, \texttt{index})$ and $c_3 = m \oplus H(ID_A, c_1, c_2, w_{i1}, w_{i2}, \texttt{index})$

Now, $CT = \langle c_1, c_2, c_3, \texttt{index}\rangle$ is send as the ciphertext to the user A. To reuse the state information st_i, the sender has to just increment `index` and use st_i, It is not required to send the component c_1 throughout the session and hence from the second encryption onwards the ciphertext size will be $|q|+|m|+|\texttt{index}|$ which is much less than $|q|+|m|+|p|$ in the most efficient CLE [4] with ciphertext verifiability.

It should be noted that the maximum number of encryptions to be performed in a session will be determined by the sender. Thus, `index` is a user determined integer value and to perform one million encryptions in a session, the value of index may be utmost 2^{20}. Hence, `index` may typically be a value from $1 \leq \texttt{index} \leq 2^{20}$ and thus of size less than 20-bits.

Decryption: The receiver with identity ID_A does the following to decrypt a ciphertext $CT = (c_1, c_2, c_3, \texttt{index})$:

- Compute $w'_{i1} = c_1^{D_A^{(1)}}$ and $w'_{i2} = c_1^{D_A^{(2)}}$, $m' = c_3 \oplus H(ID_A, c_1, c_2, w'_{i1}, w'_{i2}, \texttt{index})$ and $u' = c_2 \oplus G(ID_A, c_1, m, w'_{i1}, w'_{i2}, \texttt{index})$
- Check whether $u' \stackrel{?}{=} F(c_1^{(u')^{-1}})$. If the check holds output m', otherwise output $\perp$. This check helps in identifying whether a ciphertext is well formed or not.

Correctness: We have to show that the u' computed by the decryption algorithm passes the verification test $u' \stackrel{?}{=} F(c_1^{(u')^{-1}})$, if $u' = u_i = F(g^{r_i})$.

$$\begin{aligned} RHS &= F(c_1^{(u')^{-1}}) = F(v_i^{(u')^{-1}}) = F(g^{s_i(u')^{-1}}) = F(g^{r_i u_i (u')^{-1}}) \\ &= F(g^{r_i}) \text{ (If } u' = u_i = F(g^{r_i})) \\ &= u' = LHS \end{aligned}$$

Thus, the decryption will hold if $u' = u_i = F(g^{r_i})$.

3.1 Security Proof

To prove the confidentiality of a certificateless encryption scheme, it is required to consider the attacks by Type-I and Type-II adversaries. In the two existing secure schemes [11,18], the Type-I adversary is not allowed to extract the secret value corresponding to the target identity. To capture the ability of the adversary who can access the user secret keys of the target identity, we give access to the user secret value of the target identity to the Type-I adversary. We also state that, allowing the extract secret value query corresponding to the target identity makes the security model for Type-I adversary more stronger. For a stateful certificateless encryption scheme, the adversary may be interested in analyzing the ciphertexts of different messages of his choice, encrypted during a particular session. Since the adversary does not know the state information, the challenger has to provide the encryption oracle to the adversary.

Confidentiality Against Type-I Adversary

Theorem 1. *The stateful certificateless encryption scheme* `stCLE` *is IND-stCLE-CCA2-I secure in the random oracle model, if the SDH problem is intractable.*

Proof. The challenger $\mathcal{C}$ is challenged with an instance of the SDH problem, say $\langle g, g^a, g^b \rangle \in_R \mathbb{G}^3$ and is given access to a Decision Diffie Hellman (DDH) oracle $\mathcal{DDH}_{g,a}(.,.)$ which on input g^b and g^c outputs `True` if and only if $g^{ab} = g^c$. The challenger's aim is to solve the SDH problem, which is to compute $g^{ab} \in \mathbb{G}$. In our scheme $\mathbb{Z}_p^*$ forms a group which can be represented as $\mathbb{G}$. Let us consider that there exists an adversary $\mathcal{A}_I$ who is capable of breaking the IND-stCLE-CCA2-I security of the `stCLE` scheme. $\mathcal{C}$ can make use of $\mathcal{A}_I$ to compute g^{ab} by playing the following interactive game with $\mathcal{A}_I$.

Setup: $\mathcal{C}$ begins the game by setting up the system parameters as in the `stCLE` scheme. $\mathcal{C}$ takes g and g^a from the instance of the SDH problem sets $y = g^a$ and sends $params = \langle p, q, g, y \rangle$ to $\mathcal{A}_I$. This makes an implicit assignment to the master private key as $z = a$, where $\mathcal{C}$ doenot know z. $\mathcal{C}$ also designs the five hash functions F, G, H, H_1 and H_2 as random oracles $\mathcal{O}_F$, $\mathcal{O}_G$, $\mathcal{O}_H$, $\mathcal{O}_{H_1}$ and $\mathcal{O}_{H_2}$. $\mathcal{C}$ maintains five lists L_F, L_G, L_H, L_{H_1} and L_{H_2} in order to consistently respond to the queries to the random oracles $\mathcal{O}_F$, $\mathcal{O}_G$, $\mathcal{O}_H$, $\mathcal{O}_{H_1}$ and $\mathcal{O}_{H_2}$ respectively. To maintain the consistency of the private key request and public key request oracle queries, $\mathcal{C}$ maintains lists L_S and L_P respectively. A typical entity in list L_i for $i = \{F, G, H, H_1, H_2\}$ will have the input parameters of the oracles, followed by the corresponding hash value returned as the response to the hash oracle query. The list L_S consists of the tuples of the form $\langle ID_i, D_i^{(1)}, D_i^{(2)} \rangle$ and that of L_P consists of the tuples of the form $\langle ID_i, PK_i^{(1)}, PK_i^{(2)}, PK_i^{(3)}, PK_i^{(4)} \rangle\rangle$. In order to generate stateful encryptions, $\mathcal{C}$ generates $\hat{n}$ tuples of state informations and stores them in a state list L_{st}. Each tuple in the list corresponds to a state information. This is done as follows.
For each identity ID_i created by $\mathcal{A}_I$ and $j = 1$ to $\hat{n}$, $\mathcal{C}$ performs the following:

- Choose $r_j \in_R \mathbb{Z}_q^*$, compute $k_j = g^{r_j}$, choose $u_j \in_R \mathbb{Z}_q^*$ and add the tuple $\langle k_j, u_j \rangle$ in the list L_F.
- Compute $s_j = r_j u_j$ and $v_j = g^{s_j}$.
- The state information $st_j = \langle ID_i, u_j, v_j, s_j, \mathtt{index}_j \rangle$.
- Store the tuple st_j in list L_{st}.

The game proceeds as described in the security model for Type-I adversary in Sect. 2.3.

Phase I: $\mathcal{A}_I$ performs a series of queries to the oracles provided by $\mathcal{C}$. The descriptions of the oracles and the responses given by $\mathcal{C}$ to the corresponding oracle queries by $\mathcal{A}_I$ are described below:
$\mathcal{O}_F(g^{r_i})$: For answering the $\mathcal{O}_F$ query, $\mathcal{C}$ performs the following:

- If a tuple of the form $\langle g^{r_i}, u_i \rangle$ exists in the list L_F then $\mathcal{C}$ retrieves the corresponding u_i and sends it to $\mathcal{A}_I$.

– Else, $\mathcal{C}$ chooses $u_i \in_R \mathbb{Z}_q^*$, stores the tuple $\langle g^{r_i}, u_i \rangle$ in the list L_F and sends u_i to $\mathcal{A}_I$

$\mathcal{O}_G(ID_j, c_1, m, w_{i1}, w_{i2}, \texttt{index})$: For answering the $\mathcal{O}_G$ query, $\mathcal{C}$ performs the following:

– If a tuple of the form $\langle ID_j, c_1, m, w_{i1}, w_{i2}, \texttt{index}, \mathcal{G} \rangle$ exists in the list L_G then $\mathcal{C}$ retrieves the corresponding $\mathcal{G}$ and sends it to $\mathcal{A}_I$.
– Else, $\mathcal{C}$ chooses $\mathcal{G} \in_R \mathbb{Z}_q^*$, stores the tuple $\langle ID_j, c_1, m, w_{i1}, w_{i2}, \texttt{index}, \mathcal{G} \rangle$ in the list L_G and sends $\mathcal{G}$ to $\mathcal{A}_I$

$\mathcal{O}_H(ID_j, c_1, c_2, w_{i1}, w_{i2}, \texttt{index})$: For answering the $\mathcal{O}_H$ query, $\mathcal{C}$ performs the following:

– If a tuple of the form $\langle ID_j, c_1, c_2, w_{i1}, w_{i2}, \texttt{index}, \mathcal{H} \rangle$ exists in the list L_H then $\mathcal{C}$ retrieves the corresponding $\mathcal{H}$ and sends it to $\mathcal{A}_I$.
– Else, $\mathcal{C}$ chooses $\mathcal{H} \in_R \{0,1\}^{l_m}$, stores the tuple $\langle ID_j, c_1, m, w_{i1}, w_{i2}, \texttt{index}, \mathcal{H} \rangle$ in the list L_H and sends $\mathcal{H}$ to $\mathcal{A}_I$

$\mathcal{O}_{H_1}(ID_i, y_{i0})$: To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle ID_i, y_{i0}, h_{i1} \rangle$ exists in the list L_{H_1}. If a tuple of this form exists, $\mathcal{C}$ returns the corresponding h_{i1}, else chooses $h_{i1} \in_R \mathbb{Z}_q^*$ and adds the tuple $\langle ID_i, y_{i0}, h_{i1} \rangle$ to the list L_{H_1} and returns h_{i1} to $\mathcal{A}_I$.
$\mathcal{O}_{H_2}(ID_i, y_{i0}, y_{i1})$: To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle ID_i, y_{i0}, y_{i1}, h_{i2} \rangle$ exists in the list L_{H_2}. If a tuple of this form exists, $\mathcal{C}$ returns the corresponding h_{i2}, else chooses $h_{i2} \in_R \mathbb{Z}_q^*$, adds the tuple $\langle ID_i, y_{i0}, y_{i1}, h_{i2} \rangle$ to the list L_{H_2} and returns h_{i2} to $\mathcal{A}_I$.
$\mathcal{O}_{RequestPublicKey}(ID_i)$: $\mathcal{C}$ selects a random index γ, where $1 \leq \gamma \leq q_{PK}$ and $\mathcal{C}$ does not reveal γ to $\mathcal{A}_I$. Here q_{PK} is the maximum number of Request Public Key oracle queries. When $\mathcal{A}_I$ makes the γ^{th} query on ID_γ, $\mathcal{C}$ fixes ID_γ as target identity for the challenge phase.
If a tuple of the form $\langle ID_i, PK_i^{(1)}, PK_i^{(2)}, PK_i^{(3)}, PK_i^{(4)} \rangle$ exists in the list L_P, return the items corresponding to the identity ID_i in the list as the public key. If a tuple does not exist, check whether $i \neq \gamma$. In this case, $\mathcal{C}$ queries $\mathcal{O}_{PartialKeyExtract}(ID_i)$ and then retrieves the tuple of the form $\langle ID_i, PK_i^{(1)}, PK_i^{(2)}, PK_i^{(3)}, PK_i^{(4)} \rangle$ from the list L_P and returns it as the public key corresponding to the identity ID_i. If $i = \gamma$, then perform the following:

– Choose $s_{i0}, d_{i1}, h_{i1}, h_{i2}, x_i \in_R \mathbb{Z}_q^*$.
– Compute $y_{i0} = g^{s_{i0}}$, $y_{i1} = g^{d_{i1}}\ (g^a)^{-h_{i2}}$ and $g_i = g^{x_i}$.
– Add the tuple $\langle ID_i, D_i^{(1)}, D_i^{(2)} \rangle = \langle ID_i, -, x_i \rangle$ in the list L_S and add the tuple $\langle ID_i, PK_i^{(1)}, PK_i^{(2)}, PK_i^{(3)}, PK_i^{(4)} \rangle = \langle ID_i, y_{i0}, y_{i1}, d_{i1}, g_i \rangle$ to the list L_P.
– Add the tuple $\langle ID_i, y_{i0}, h_{i1} \rangle$ to list L_{H_1} and the tuple $\langle ID_i, y_{i0}, y_{i1}, h_{i2} \rangle$ to list L_{H_2}
– Return $\langle ID_i, PK_i^{(1)}, PK_i^{(2)}, PK_i^{(3)}, PK_i^{(4)} \rangle$ to $\mathcal{A}_I$.

$\mathcal{O}_{PartialKeyExtract}(ID_i)$: In order to answer a query to the oracle, $\mathcal{C}$ checks whether a tuple of the form $\langle ID_i, D_i^{(1)}, D_i^{(2)}\rangle$ exists in the list L_S and if a tuple of this form exists, $\mathcal{C}$ returns the corresponding $D_i^{(1)}$. If it does not exist, $\mathcal{C}$ checks whether $i \overset{?}{=} \gamma$. If $i = \gamma$, $\mathcal{C}$ *Aborts* the game. If $i \neq \gamma$, $\mathcal{C}$ performs the following:

- Choose $d_{i0}, d_{i1}, h_{i1}, h_{i2}, x_i \in_R \mathbb{Z}_q^*$.
- Compute $y_{i0} = g^{d_{i0}}(g^a)^{-h_{i1}}$, $y_{i1} = g^{d_{i1}}\ (g^a)^{-h_{i2}}$ and $g_i = g^{x_i}$.
- Add the tuple $\langle ID_i, D_i^{(1)}, D_i^{(2)}\rangle = \langle ID_i, d_{i0}, x_i\rangle$ in the list L_S and add the tuple $\langle ID_i, PK_i^{(1)},\ PK_i^{(2)}, PK_i^{(3)},\ PK_i^{(4)}\rangle = \langle ID_i, y_{i0}, y_{i1}, d_{i1}, g_i\rangle$ to the list L_P.
- Add the tuple $\langle ID_i, y_{i0},\ h_{i1}\rangle$ to list L_{H_1} and the tuple $\langle ID_i, y_{i0}, y_{i1},\ h_{i2}\rangle$ to list L_{H_2}
- Return $D_i^{(1)}$ to $\mathcal{A}_I$.

$\mathcal{O}_{ExtractSecretValue}(ID_i)$: $\mathcal{C}$ retrieves the tuple of the form $\langle ID_i, d_{i0}, x_i\rangle$ from the list L_S and returns the corresponding x_i as the secret value corresponding to the identity ID_i. If the entry corresponding to x_i in the tuple is "$-$" then it indicates the fact that $\mathcal{A}_I$ has replaced the public key corresponding to ID_i. By the definition of the model, such queries by $\mathcal{A}_I$ are not allowed and hence $\mathcal{C}$ can ignore such queries.
$\mathcal{O}_{ReplacePublicKey}(ID_i, PK_i^{'})$: To replace the public key of ID_i with a new public key $PK_i^{'} = \langle ID_i, PK_i^{(1)'},\ PK_i^{(2)'}, PK_i^{(3)'}, PK_i^{(4)'}\rangle$, sent by $\mathcal{A}_I$, $\mathcal{C}$ updates the corresponding tuples in the list L_P, only if $PK_i^{'}$ satisfies equation (2). If the equation is not satisfied return *Invalid*.
$\mathcal{O}_{Encryption}(ID_i, st_j, m_k)$: $\mathcal{A}_I$ may perform encryption with respect to any state information st_j, chosen by $\mathcal{C}$. $\mathcal{C}$ performs the following to encrypt the message m_k with respect to the state information st_j, where $j = 1$ to $\hat{n}$, where $\hat{n}$ is the upper bound for the total number of states and $k = 1$ to $\hat{m}$ is bound by the maximum number of messages that can be encrypted in one session:

- Retrieves the tuple st_j of the form $\langle ID_i, u_j, v_j,\ s_j, \texttt{index}_j\rangle$ from L_{st}, sets $c_1 = v_j$, compute $w_{j1} = (PK_i^{(1)} y^{h_{i1}})^{s_j}$ and $w_{j2} = (PK_i^{(4)})^{s_j}$
- Choose $\mathcal{G} \in_R \mathbb{Z}_q^*$, store the tuple $\langle ID_i, c_1, m_k, w_{j1}, w_{j2},\ \texttt{index}_j, \mathcal{G}\rangle$ in the list L_G and computes $c_2 = u_j \oplus \mathcal{G}$.
- Choose $\mathcal{H} \in_R \{0,1\}^{l_m}$, store the tuple $\langle ID_i, c_1, c_2,\ w_{j1}, w_{j2}, \texttt{index}_j, \mathcal{H}\rangle$ in the list L_H and computes $c_3 = m_k \oplus \mathcal{H}$.
- Returns $c = \langle c_1, c_2, c_3\rangle$ as the ciphertext, increments $\texttt{index}_j$ and updates the state information st_j.

$\mathcal{O}_{Decryption}(CT = (c_1, c_2, c_3, \texttt{index}), ID_i, PK_i)$: If $i \neq \gamma$, $\mathcal{C}$ performs decryption in the normal way since $\mathcal{C}$ knows the private key corresponding to ID_i. If $i = \gamma$, $\mathcal{C}$ performs the following to decrypt the ciphertext $CT = \langle c_1, c_2, c_3, \texttt{index}\rangle$:

- Check the validity of PK_i and reject the ciphertext CT if this check fails; else, proceed.

- Retrieve the tuples of the form $\langle ID_i, c_1, m, w_{i1}, w_{i2}, \texttt{index}, \mathcal{G}\rangle$ and $\langle ID_i, c_1, c_2, w_{i1}, w_{i2}, \texttt{index}, \mathcal{H}\rangle$ from the lists L_G and L_H respectively. Retrieve the tuple $\langle ID_i, y_{i0}, h_{i1}\rangle$ from the list L_{H_1}.
- Compute $\alpha = (w_{i1}c_1^{-s_{i0}})^{h_{i1}^{-1}}$ by taking the corresponding values from the tuples retrieved in the above step.
- Check whether $\langle g, c_1, g^a, \alpha\rangle$ is a valid DDH tuple using the $\mathcal{DDH}_{g,a}(.,.)$ oracle. If the oracle outputs $\texttt{true}$, proceed else reject the ciphertext CT.
- Compute $m' = c_3 \oplus \mathcal{H}$ and check whether $m' = m$, where $\mathcal{H}$ and m are retrieved from the lists L_H and L_G respectively. If $m' \neq m$ reject the ciphertext CT.
- If the check holds, compute $u' = c_2 \oplus \mathcal{G}$. Retrieve the tuple of the form $\langle g^{r_i}, u'\rangle$ from the list L_F and check whether $c_1^{u'^{-1}} \overset{?}{=} g^{r_i}$. If the check does not hold reject the ciphertext CT.

If any of the fetched tuple is not available in any of the lists or any of the tests fails, returns *Invalid* else return m as the message.

Challenge: At the end of ***Phase I***, $\mathcal{A}_I$ produces two messages m_0 and m_1 of equal length and an identity ID^*. $\mathcal{C}$ *Aborts* the game if $ID^* \neq ID_\gamma$, else randomly chooses a bit $\delta \in_R \{0, 1\}$ and computes a ciphertext CT^* with ID_γ as the receiver by performing the following steps:

- Choose $u \in_R \mathbb{Z}_q^*$ and add the tuple $\langle g^b, u\rangle$ to the list L_F.
- Set $\texttt{index}^* = 1$ and compute $c_1^* = g^{bu}$
- Retrieve the tuple of the form $\langle ID^*, d_0^*, x^*\rangle$ from the list L_S and compute $w_2^* = (c_1^*)^{x^*}$.
- Choose $\mathcal{G} \in_R \mathbb{Z}_q^*$, store the tuple $\langle ID^*, c_1^*, m_\delta, -, w_2^*, \texttt{index}^*, \mathcal{G}\rangle$ in list L_G and compute $c_2^* = u \oplus \mathcal{G}$.
- Choose $\mathcal{H} \in_R \{0, 1\}^{l_m}$, store the tuple $\langle ID^*, c_1^*, m_\delta, -, w_2^*, \texttt{index}^*, \mathcal{H}\rangle$ in list L_H and compute $c_3^* = m_\delta \oplus \mathcal{H}$.
- Here the state information $st^* = \langle ID^*, u^*, v^*, s^*, \texttt{index}^*\rangle = \langle ID^*, u, g^{bu}, -, \texttt{index}^*\rangle$

Now, $CT^* = \langle c_1^*, c_2^*, c_3^*, \texttt{index}^*\rangle$ is sent to $\mathcal{A}_I$ as the challenge ciphertext.

Phase II: $\mathcal{A}_I$ performs the second phase of interaction, where it makes polynomial number of queries to the oracles provided by $\mathcal{C}$ with the following conditions:

- $\mathcal{A}_I$ should not have queried the *Strong Decryption* oracle with $(CT^*, PK_\gamma, ID_\gamma)$ as input. (It is to be noted that PK_γ is the public key corresponding to ID_γ during the challenge phase. $\mathcal{A}_I$ can query the decryption oracle with (CT^*, PK^*, ID_γ) as input, $\forall PK^* \neq PK_\gamma$)
- $\mathcal{A}_I$ should not query the partial private key of ID_γ.
- $\mathcal{A}_I$ can query the secret value and PK_γ of ID_γ.

Encryption oracle has to be provided to $\mathcal{A}_I$ with respect to the state information st^*. This is because, $\mathcal{A}_I$ should have access to any number of ciphertexts generated during this state. Moreover, the decryption oracle has to respond to decryption queries corresponding to any ciphertext generated during the state st^*. These two oracles are described here and all other oracles are same as in Phase I.

$\mathcal{O}_{Encryption}(ID_i, st_j, m_k)$: For any state $st_j \neq st^*$, $\mathcal{C}$ performs the encryption as in Phase I. If $st_j = st^*$, $\mathcal{C}$ performs the following to encrypt the message m_k with respect to the state information st^*.

– Retrieve the tuple st^* of the form $\langle ID^*, u, g^{bu}, -, \texttt{index}^* \rangle$ from the list L_{st}, set $c_1 = g^{bu}$.
– Choose $\mathcal{G} \in_R \mathbb{Z}_q^*$, store the tuple $\langle ID^*, c_1, m_k, -, -, \texttt{index}^*, \mathcal{G} \rangle$ in the list L_G and computes $c_2 = u \oplus \mathcal{G}$.
– Choose $\mathcal{H} \in_R \{0,1\}^{l_m}$, store the tuple $\langle ID^*, c_1, c_2, -, -, \texttt{index}^*, \mathcal{H} \rangle$ in the list L_H and computes $c_3 = m_k \oplus \mathcal{H}$.
– Returns $c = \langle c_1, c_2, c_3 \rangle$ as the ciphertext, increments $\texttt{index}_j$ and updates the state information st_j.

$\mathcal{O}_{Decryption}(CT = (c_1, c_2, c_3, \texttt{index}), ID_i, PK_i)$: If $i \neq \gamma$ and $c_1 \neq c_1^*$, $\mathcal{C}$ performs decryption in the normal way since $\mathcal{C}$ knows the private key corresponding to ID_i. If $i = \gamma$ and $c_1 \neq c_1^*$, $\mathcal{C}$ performs the decryption as in Phase I. If $i = \gamma$ and $c_1 = c_1^*$ then $\mathcal{C}$ performs the following. It should be noted that the state information st^* is with respect to the identity ID^* and hence for all other identities, decryption oracle proceeds as in Phase I.

– Check the validity of PK_i and reject the ciphertext CT if this check fails; else, proceed.
– Retrieve the tuples of the form $\langle ID^*, c_1^*, m, -, -, \texttt{index}^*, \mathcal{G} \rangle$ and $\langle ID^*, c_1^*, c_2, -, -, \texttt{index}, \mathcal{H} \rangle$ from the lists L_G and L_H respectively. Retrieve the tuple $\langle ID_i, y_{i0}, h_{i1} \rangle$ from the list L_{H_1}.
– Compute $m' = c_3 \oplus \mathcal{H}$ and check whether $m' = m$, where $\mathcal{H}$ and m are retrieved from the lists L_H and L_G respectively. If $m' \neq m$ reject the ciphertext CT. Note that $\mathcal{C}$ can even work consistently with the tuples of this form. In this case, $\mathcal{C}$ takes the values $\mathcal{G}$ and $\mathcal{H}$ without consulting the DDH oracle because these tuples were generated by $\mathcal{C}$ without knowing the values of w_1^* and w_2^*.
– Compute $u' = c_2 \oplus \mathcal{G}$. Retrieve the tuple of the form $\langle g^{r_i}, u' \rangle$ from the list L_F and check whether ${c_1^*}^{u'^{-1}} = g^{r_i}$. If the check does not hold reject the ciphertext CT.
– If in the process of finding out the tuples of the form $\langle ID^*, c_1^*, m, w_1^*, w_2^*, \texttt{index}^*, \mathcal{G} \rangle$ and $\langle ID^*, c_1^*, c_2^*, w_1^*, w_2^*, \texttt{index}^*, \mathcal{H} \rangle$ appeared in the lists L_G and L_H respectively then retrieve the tuple $\langle ID^*, y_0^*, h_1^* \rangle$ from the list L_{H_1}, compute $\alpha = (w_1^* {c_1^*}^{-s_0^*})^{h_1^{*-1}}$ by taking the corresponding values from the tuples retrieved in the above step and check whether $\langle g, c_1^*, g^a, \alpha \rangle$ is a valid DDH tuple using the $\mathcal{DDH}_{g,a}(.,.)$ oracle. If the oracle outputs $\texttt{true}$, output α as the solution to the SDH problem instance.

If any tuple is not available in any of the lists or any of the tests fails, returns *Invalid.*

Guess: At the end of ***Phase II***, $\mathcal{A}_I$ produces a bit δ' to $\mathcal{C}$, $\mathcal{C}$ performs the following to output the solution for the SDH problem instance.

- Retrieve the tuple $\langle ID^*, y_0^*, h_1^* \rangle$ from the list L_{H_1}.
- For each tuple of the form $\langle ID^*, c_1^*, m, w_1^*, w_2^*, \texttt{index}^*, \mathcal{G} \rangle$ and $\langle ID^*, c_1^*, c_2^*, w_1^*, w_2^*, \texttt{index}^*, \mathcal{H} \rangle$ in the lists L_G and L_H, compute $\alpha = (w_1^* {c_1^*}^{-s_0^*})^{{h_1^*}^{-1}}$ by taking the corresponding values from the tuples.
- Check whether $\langle g, c_1^*, g^a, \alpha \rangle$ is a valid DDH tuple using the $\mathcal{DDH}_{g,a}(.,.)$ oracle. If the oracle outputs $\texttt{true}$, output α as the solution to the SDH problem instance.

Thus, $\mathcal{C}$ obtains the solution to the SDH problem with almost the same advantage of $\mathcal{A}_I$ in the IND-stCLE-CCA2-I game. $\square$

Analysis: We now derive the advantage of $\mathcal{C}$ in solving the SDH problem using the adversary $\mathcal{A}_I$. The simulations of F, G, H, H_1 and H_2 clearly shows that the hash oracles are perfectly random. Let ϵ be the non-negligible advantage of $\mathcal{A}_I$ in winning the IND-stCLE-CCA2-I game.
The events in which $\mathcal{C}$ aborts the game and the respective probabilities are given below:

1. $\mathcal{E}_1$ - The event in which $\mathcal{A}_I$ queries the partial private key of ID_γ.
2. $\mathcal{E}_2$ - The event in which ID_γ is not chosen as the target identity by $\mathcal{A}_I$ for the challenge.

Suppose, $\mathcal{A}_I$ has asked q_{PK} queries to the $\mathcal{O}_{RequestPublicKey}$ oracle and q_P queries to the $\mathcal{O}_{PartialKeyExtract}$ oracle. Let us consider that there are a total of q_I individual identities, where $q_I \leq q_{PK} + q_P$ queried by $\mathcal{A}_I$ to these oracles, then:

$\Pr[\mathcal{E}_1]= \frac{q_P}{q_I}$ and $\Pr[\mathcal{E}_2]= 1 - \frac{1}{q_I - q_P}$.
Therefore,
$\Pr[\neg abort]=[\neg\mathcal{E}_1 \wedge \neg\mathcal{E}_2]=\left[1 - \frac{q_P}{q_I}\right] . \left[1 - 1 - \frac{1}{q_I - q_P}\right]=\frac{1}{q_I}$.
Therefore, the advantage of $\mathcal{C}$ solving the SDH problem is $\epsilon' \geq \left(\epsilon . \frac{1}{q_I}\right)$. Since ϵ is assumed to be non-negligible and $frac1q_I$ is also non-negligible, ϵ' will be non-negligible. This contradicts the assumption that there is no polynomial time algorithm to solve the SDH problem. Thus, we conclude that there does not exist a polynomial time adversary that can break the IND-stCLE-CCA2-I security of the $\texttt{stCLE}$ scheme.

Confidentiality Against Type-II Adversary

Theorem 2. *Our certificateless public key encryption scheme* `stCLE` *is IND-stCLE-CCA2-II secure in the random oracle model, if the SDH problem is intractable.*

The proof of this theorem is omitted here due to page limitation and will appear in the full version of the paper.

4 Conclusion

In this paper, we have proposed the first stateful certificateless PKE scheme. We have formally proved the scheme in the random oracle model assuming the strongest adversary. Our scheme finds straightforward application in secure sharing of encrypted cloud data with a very minimum cost. Assuming the security aspects, all the files of a user stored in the cloud are encrypted with unique symmetric keys. The existing method is to use a PKE scheme and encrypt the symmetric keys to the receiver. The receiver on receiving the encrypted keys can decrypt them using his private key and use them to decrypt the actual file from the downloaded ciphertext. Our approach reduces the cost of performing this encryption of symmetric keys from $\mathcal{O}(k)$ to $\mathcal{O}(1)$ exponentiations. This efficiency come because our scheme is stateful, which suits for resource constrained devices such as cell phones, personal digital assistants, tablets, wearables and sensors, where each exponentiation costs on the battery life of the device.

References

1. Abe, M., Kiltz, E., Okamoto, T.: Compact CCA-secure encryption for messages of arbitrary length. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 377–392. Springer, Heidelberg (2009)
2. Al-Riyami, S.S., Paterson, K.G.: Certificateless public key cryptography. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 452–473. Springer, Heidelberg (2003)
3. Baek, J., Chu, C.-K., Zhou, J.: On shortening ciphertexts: new constructions for compact public key and stateful encryption schemes. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 302–318. Springer, Heidelberg (2011)
4. Baek, J., Safavi-Naini, R., Susilo, W.: Certificateless public key encryption without pairing. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 134–148. Springer, Heidelberg (2005)
5. Bellare, M., Kohno, T., Shoup, V.: Stateful public-key cryptosystems: how to encrypt with one 160-bit exponentiation. In: ACM Conference on Computer and Communications Security - ACM-CCS 2006, pp. 380–389. ACM (2006)
6. Bendel, M.: Hackers describe ps3 security as epic fail, gain unrestricted access (2010). http://www.exophase.com/20540/hackers-describe-ps3-security-as-epic-fail-gain-unrestricted-access/
7. Castro, R., Dahab, R.: Two notes on the security of certificateless signatures. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 85–102. Springer, Heidelberg (2007)
8. Cheng, Z., Comley, R.: Efficient certificateless public key encryption. Cryptology ePrint Archive, Report 2005/012 (2005). http://eprint.iacr.org/
9. Dent, A.W.: A survey of certificateless encryption schemes and security models. Int. J. Inf. Secur. **7**(5), 349–377 (2008)
10. Huang, X., Susilo, W., Mu, Y., Zhang, F.T.: On the security of certificateless signature schemes from Asiacrypt 2003. In: Desmedt, Y.G., Wang, H., Mu, Y., Li, Y. (eds.) CANS 2005. LNCS, vol. 3810, pp. 13–25. Springer, Heidelberg (2005)
11. Lai, J., Deng, R.H., Liu, S., Kou, W.: RSA-Based Certificateless Public Key Encryption. In: Bao, F., Li, H., Wang, G. (eds.) ISPEC 2009. LNCS, vol. 5451, pp. 24–34. Springer, Heidelberg (2009)

12. Lenstra, A.K., Hughes, J.P., Augier, M., Bos, J.W., Kleinjung, T., Wachter, C.: Ron was wrong, whit is right. IACR Cryptology ePrint Archive (2012)
13. Liu, J.K., Au, M.H., Susilo, W.: Self-generated-certificate public key cryptography and certificateless signature/encryption scheme in the standard model: extended abstract. In: Proceedings of the ACM Symposium on Information, Computer and Communications Security - ASIA-CCS 2007, pp. 273–283. ACM (2007)
14. Okamoto, T., Pointcheval, D.: The gap-problems: a new class of problems for the security of cryptographic schemes. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 104–118. Springer, Heidelberg (2001)
15. Park, J.-H., Choi, K.Y., Hwang, J.Y., Lee, D.-H.: Certificateless public key encryption in the selective-ID security model (without random oracles). In: Takagi, T., Okamoto, E., Okamoto, T., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 60–82. Springer, Heidelberg (2007)
16. Phong, L.T., Matsuoka, H., Ogata, W.: Stateful identity-based encryption scheme: faster encryption and decryption. In: Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security - ASIACCS 2008, pp. 381–388. ACM (2008)
17. Shi, Y., Li, J.: Provable efficient certificateless public key encryption. Cryptology ePrint Archive, Report 2005/287 (2005). http://eprint.iacr.org/
18. Sun, Y., Zhang, F.T., Baek, J.: Strongly secure certificateless public key encryption without pairing. In: Bao, F., Ling, S., Okamoto, T., Wang, H., Xing, C. (eds.) CANS 2007. LNCS, vol. 4856, pp. 194–208. Springer, Heidelberg (2007)
19. Sree Vivek, S., Selvi, S., Rangan, C.P.: CCA secure certificateless encryption schemes based on RSA. In: SECRYPT 2011 - Proceedings of the International Conference on Security and Cryptography, pp. 208–217. SciTePress (2011)
20. Vivek, S.S., Selvi, S.S.D., Rangan, C.P.: Compact stateful encryption schemes with ciphertext verifiability. In: Hanaoka, G., Yamauchi, T. (eds.) IWSEC 2012. LNCS, vol. 7631, pp. 87–104. Springer, Heidelberg (2012)