The Exact Security of a Stateful IBE and New Compact Stateful PKE Schemes

S. Sree Vivek, S. Sharmila Deva Selvi, C. Pandu Rangan*

Theoretical Computer Science Lab,
Department of Computer Science and Engineering,
Indian Institute of Technology Madras, India.
{sharmila,svivek,prangan}@cse.iitm.ac.in

Abstract. Recently, Baek et al. proposed a stateful identity based encryption scheme with compact ciphertext and commented that the security of the scheme can be reduced to the Computational Bilinear Diffie Hellman (CBDH) problem. In this paper, we formally prove that the security of the stateful identity based encryption scheme by Baek et al. cannot be reduced to the CBDH problem. In fact, we show that the challenger will confront the Y-Computational problem while providing the decryption oracle access to the adversary. We provide the exact and formal security proof for the scheme, assuming the hardness of the Gap Bilinear Diffie Hellman (GBDH) problem. We also propose two new stateful public key encryption scheme with ciphertext verifiability. Our schemes offer more compact ciphertext when compared to all existing stateful public key encryption schemes with ciphertext verifiability. We have proved all the schemes in the random oracle model.

Keywords. Stateful Identity Based Encryption, Adaptive Chosen Ciphertext (CCA), Provable Security, Compact Ciphertext with/without Ciphertext Verification, Random Oracle model.

1 Introduction

The constraints imposed by low-powered, small-memory computing devices such as sensors and PDAs should be considered while proposing an encryption scheme. The encryption schemes for these kind of resource constrained devices must address key issues such as limited battery life, low bandwidth, small memory etc and should incur minimal computation and communication overhead. One way to reduce the computation cost is to reuse certain (computed) values and this is the key idea behind the stateful encryption schemes introduced by Bellare et al. [5]. In a stateful encryption scheme, the sender maintains state information which can be reused across different sessions while encrypting a message to the same receiver several times during a session. Typically a session can depend on the number of encryptions that can be done with the same state information with reduced number of heavy computations.

For any public key encryption scheme, the difference between the size of the ciphertext and the size of the message is referred to as its *Ciphertext Overhead*. An encryption scheme is said to generate compact ciphertext if the overhead is utmost the size of one element in the underlying group. Needless to say, compact ciphertexts are very useful in bandwidth-critical environments [3, 4]. In general, when we design encryption schemes with stronger security properties, we tend to loose compactness and often arrive at ciphertexts that have large overheads. However, in the recent past, several researchers have successfully designed CCA secure encryption schemes (stronger notion of security for encryption schemes) that result in compact ciphertexts [13, 6, 7, 3, 4]. While these schemes yield compact ciphertexts, they lack an important property which we refer as *Ciphertext Verifiability*. We briefly describe about this property and its importance below.

For the public key encryption schemes that are used in important applications such as key transport, electronic auction etc, the encryption scheme must provide a guarantee that the ciphertext (and thus the message contained in the ciphertext) was not altered during transit. If such a guarantee is not available, it may lead to unacceptable situations. For example, suppose a user A wishes to safely send a key value *key* to user B and use *key* as ephemeral/session key for some further interaction with B. A may use the public key of B and encrypt *key* and send the ciphertext c to B. If no verification mechanism is available and if c

* Currently The Head, Indian Statistical Institute, Chennai, INDIA.

is altered to c' (by the adversary or by transmission error) and if c' is decrypted to key', B would simply assume that key' is the key that A wished to send to him. This would cause further interactions between A and B impossible and this is clearly undesirable. A similar scenario can be imagined in a KEM/DEM scheme if modified ciphertexts are used to recover keys. It is not hard to imagine the possibility of change of bid values in e-auctions/e-tendering, where the altered ciphertext getting decrypted to a value different from the value actually meant by the sender.

Hence, it is important that the encryption schemes provide 'ciphertext verifiability' in addition to all the other desirable properties such as compactness and CCA security. By ciphertext verifiability we mean a testing process that is integrated in the decryption algorithm which identifies if the received ciphertext is a tweaked one or not. If the test fails, the receiver infers that the ciphertext is corrupted during transmission and rejects it. If the test passes, the receiver considers the message constructed by the decryption algorithm as a valid message. The ability to distinguish a tweaked ciphertext from a genuine ciphertext is an important property for decryption algorithm and see [15] by Pass et al. for a formal and rigorous treatment of the same.

Related Work: There are several CCA secure encryption schemes available in the literature. Some of them are customized designs [1, 6], some are based on transforming a CPA secure system to a CCA secure system [10, 9, 12], some are based on KEM/DEM (Key Encapsulation Mechanism/Data Encapsulation Mechanism) [8, 11, 13] and some are based on Tag-KEM/DEM framework [2]. However, none of them produced compact ciphertext and this prompted researchers to design afresh CCA secure encryption schemes outputting compact ciphertexts. Several new and interesting ideas emerged in the past, resulting in schemes reported in [13, 6, 7, 3, 4]. Though these schemes output compact ciphertext and CCA security, none of them offer ciphertext verifiability.

Our Contribution: There are four major contributions in this paper. First, we formally prove that the security of the $\mathcal{SIBE}$ proposed in [4] cannot be reduced to the CBDH problem as indicated in the original paper. We assume the hardness of the Y-Computational problem [12] to show this. Second, we provide the exact and formal security proof for the same $\mathcal{SIBE}$ scheme proposed in [4] assuming the hardness of the GBDH assumption. In the literature, there is one stateful identity based encryption scheme by Phong et al. in [16], whose security is based on the GBDH problem and another scheme by Yang et al. in [17], whose security is based on the CBDH problem. However, the $\mathcal{SIBE}$ in [4] offers more compact ciphertext but does not offer ciphertext verifiability. Third, we design a new PKI based stateful public key encryption scheme ($\mathcal{N}-\mathcal{SPKE}_1$), whose security is based on the SDH problem. Our fourth contribution is a stateful public key encryption scheme ($\mathcal{N}-\mathcal{SPKE}_2$), whose security is based on CDH problem but with the same ciphertext overhead as ($\mathcal{N}-\mathcal{SPKE}_1$). The ciphertext overhead of these two schemes are slightly higher than that of the $\mathcal{SPKE}$ scheme proposed in [4]. The ciphertext overhead of the $\mathcal{SPKE}$ scheme in [4] is one group element and another element with λ bits, where λ is greater than 128-bits. In our schemes we include an integer value called as `index` which represents the encryption number. That is, we index the encryptions performed during a session using an integer counter. At the start of each session, the value of `index` is initialized to 1 and incremented each time an encryption is performed during the session. If we consider that *one million* encryptions are to be done in a session, the `index` ranges from 1-bit to 20-bits utmost. This also contributes to the ciphertext overhead of the scheme. Thus, the ciphertext overhead of our scheme is one group element, one element of size 128-bits and an `index`. With this overhead, it is possible to offer ciphertext verifiability and this is the highlighting difference of our scheme. The sender has to just increment the index after each encryption and store only the incremented value (utmost 20-bits) and does not need to remember the indices that are used previously across the session. Thus this will not lead to big storage overhead. It is possible to use the folkloric construction of appending 80-bits of known value (usually 80-bits of 0's) to the plaintext while encrypting it and checking whether the decryption of the ciphertext produces a message with those 80-bits at the end to ensure ciphertext verifiability. However, the size of this value is lower bound by 80-bits, where as in our construction, the index is upper bound by 20-bits (for 2^{20} encryptions) and hence can take a value starting from $1-$bit, which is a considerable reduction for resource constrained devices. This makes our construction more attractive.

2 Preliminaries, Frameworks and Security Models

We use Computational Diffie Hellman Problem (CDH), Strong Diffie Hellman Problem (SDH) [3] and Gap Bilinear Diffie Hellman Problem (GBDH) [16] to establish the security of the schemes.

Definition 1. ***(Computational Diffie Hellman Problem (CDH))****: Let κ be the security parameter and $\mathbb{G}$ be a multiplicative group of order q, where $|q| = \kappa$. Given $(g, g^a, g^b) \in_R \mathbb{G}^4$, the computational Diffie Hellman problem is to compute $g^{ab} \in \mathbb{G}$.*

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

$$Adv_{\mathcal{A}}^{CDH} = Pr[\mathcal{A}(g, g^a, g^b) = g^{ab}]$$

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

Definition 2. ***(Strong Diffie Hellman Problem (SDH) as given in [3])****: Let κ be the security parameter and $\mathbb{G}$ be a multiplicative group of order q, where $|q| = \kappa$. Given $(g, g^a, g^b) \in_R \mathbb{G}^3$ 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].

Definition 3. ***(Gap Bilinear Diffie Hellman Problem (GBDH))****: Given $(P, aP, bP, cP) \in \mathbb{G}^4$ for unknown $a, b, c \in \mathbb{Z}_q$ and access to a Decision Bilinear Diffie Hellman (DDH) oracle $\mathcal{DBDH}_{P,a}(.,.,.)$ which on input $bP, cP \in \mathbb{G}^2$ and $\alpha \in \mathbb{G}_T$ outputs `True` if and only if $\hat{e}(P, P)^{abc} = \alpha$, the Gap Bilinear Diffie Hellman problem in $\langle \mathbb{G}, \mathbb{G}_T \rangle$ is to compute $\hat{e}(P, P)^{abc}$.*

The advantage of any probabilistic polynomial time adversary $\mathcal{A}$ in solving the GBDH problem in $\langle \mathbb{G}, \mathbb{G}_T \rangle$ is defined as:

$$Adv_{\mathcal{A}}^{GBDH} = Pr\left[\mathcal{A}(P, aP, bP, cP) = \hat{e}(P, P)^{abc} | \mathcal{DBDH}_{P,a}(.,.,.))\right]$$

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

Definition 4. ***Stateful Identity Based Encryption Scheme ($\mathcal{SIBE}$):***

A stateful identity based encryption scheme $\mathcal{SIBE}$ is a tuple of five polynomial time algorithms `Setup`,`Extract`,`New State`, `Encryption` and `Decryption` (all are randomized algorithms except the last) such that:

- The `Setup` algorithm is run by the Private Key Generator (PKG) to generate the system parameters *params*.
- The `Extract` algorithm takes an identity of a user (say identity ID_A of user A) and *params* as input, and outputs the private key (D_A) of the user. This algorithm can be denoted as $D_A \leftarrow$ `Extract`$(params, ID_A)$.
- The `New State` generation algorithm is run by any one who wants to encrypt the message to generate a fresh state information *st* by taking *params* as input.
- The `Encryption` algorithm takes as input *params*, the state information *st*, the identity of the receiver (say ID_A) and a message m, and outputs the ciphertext c. This algorithm can be denoted as $c \leftarrow$ `Encryption`$(params, st, ID_A, m)$
- The `Decryption` algorithm takes *params*, the private key ID_A and a ciphertext c as input, and outputs a message m or $\perp$ (denoting failure). This algorithm can be denoted as $\{m, \perp\} \leftarrow$ `Decryption`$(params, D_A, c)$

It is required that for a well-formed ciphertext, $Pr\,[$`Decryption`$(params, D_A,$ `Encryption`$(params, st, ID_A, m))] \neq m \leq negl(\kappa)$, where $negl(.)$ is a negligible function.

Definition 5. ***Stateful Public Key Encryption ($\mathcal{SPKE}$):***

A stateful public key encryption scheme $\mathcal{SPKE}$ is a tuple of five polynomial time algorithms `Setup`, `Key Generation`, `New State`, `Encryption` and `Decryption` (all are randomized algorithms except the last) such that:

- The `Setup` algorithm is run by an authority to generate the system parameters *params*.
- The `Key Generation` algorithm takes the system parameters *params* as input and outputs a pair of keys (sk, pk), namely the private key and the public key. This algorithm can be denoted as $(sk, pk) \leftarrow$ `Key Generation`$(params)$.
- The `New State` generation algorithm is run by any one who wants to encrypt the message to generate a fresh state information st by taking *params* as input.
- The `Encryption` algorithm takes as input *params*, the state information st, a public key pk and a message m, and outputs the ciphertext c. This algorithm can be denoted as $c \leftarrow$ `Encryption`$(params, st, pk, m)$
- The `Decryption` algorithm takes the private key sk and a ciphertext c as input and outputs a message m or $\perp$ denoting failure. This algorithm can be denoted as $\{m, \perp\} \leftarrow$ `Decryption`$(params, sk, c)$

Note: We omit the `Public Key Check` algorithm in our paper and hence our framework has one less algorithm from the actual definition in [5]. This is because public key check is concerned with all Public Key Infrastructure (PKI) based encryption schemes. It is mandatory for a sender to perform this check in order to verify whether the components of public keys are elements of the underlying group and they comply with the system. Few checks like this are sometimes required for the security of standard schemes.

Definition 6. ***Game for CCA Security of $\mathcal{SIBE}$ ($\mathcal{SIBE}_{\mathcal{A}}^{CCA}(\kappa)$):*** *The game for CCA security of a stateful identity based encryption scheme is between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. The game follows:*

Setup: $\mathcal{C}$ generates the system parameters *params* and gives it to $\mathcal{A}$.

Phase I: $\mathcal{A}$ is given oracle access to the following oracles:

- `Extract`$(params, ID)$: $\mathcal{A}$ submits an identity ID and queries the private key corresponding to ID and $\mathcal{C}$ provides $\mathcal{A}$ with the corresponding private key D.
- `Encryption`$(params, st_i, m_j)$: Encryption queries for any number of messages ($j = 1$ to $\hat{m}$) for a given state st_i ($i = 1$ 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. Note that encryption with respect to the public keys those are valid and passes the public key validity check alone are allowed.
- `Decryption`$(params, sk, c)$: Decryption for any ciphertext c can be queried by $\mathcal{A}$, irrespective of the state information, $\mathcal{C}$ should be able to provide the decryption.

Challenge: $\mathcal{A}$ gives $\mathcal{C}$ two messages m_0 and m_1 of the same length and an identity ID^*. $\mathcal{C}$ chooses a random bit $\beta \leftarrow \{0, 1\}$ and generates the challenge ciphertext $c^* \leftarrow$ `Encryption`$(params, st^*, ID^*, m_\beta)$ and gives it to $\mathcal{A}$.

Phase 2: $\mathcal{A}$ continues to get oracle access to the ciphertexts for any message including m_0 and m_1 for the state information st^* for any identity including ID^*, through the encryption oracle `Encryption`$(params, st^*, ID, m_j)$, where $j \leq \hat{m}$. $\mathcal{A}$ also gets access to the `Decryption` oracle, where it is allowed to query the decryption of any ciphertext $c \neq c^*$

Guess: $\mathcal{A}$ outputs a bit β' finally and wins the game if $\beta = \beta'$.

A stateful identity based encryption scheme $\mathcal{SIBE}$ has indistinguishable encryptions under adaptive chosen ciphertext attack (CCA) if for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function $negl(.)$ such that:

$$Pr[\mathcal{SIBE}_{\mathcal{A}}^{CCA}(\kappa) \rightarrow (\beta = \beta')] \leq \frac{1}{2} + negl(\kappa)$$

Definition 7. ***Game for CCA Security of Stateful PKE ($\mathcal{SPKE}_{\mathcal{A}}^{CCA}(\kappa)$):*** *The game for CCA security of a stateful public key encryption scheme is between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. Note that with out loss of generality we accept only the public keys that are valid, in the game. Public keys those are not well-formed will be rejected by public key check algorithm which we do not make explicit in our proofs. The game follows:*

– $\mathcal{C}$ generates the system parameters $params$, generates a key pair $(sk, pk) \leftarrow$ `Key Generation`(κ) and $prams, pk$ are given to $\mathcal{A}$. (It should be noted that since $\mathcal{A}$ knows $params$, $\mathcal{A}$ could generate any number of private key / public key pairs but $\mathcal{A}$ does not know sk which is the private key corresponding to pk).
– $\mathcal{A}$ is given oracle access to the following oracles:
  - `Encryption`$(params, st_i, m_j)$: Encryption queries for any number of messages ($j = 1$ to $\hat{m}$) for a given state st_i ($i = 1$ 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. Note that encryption with respect to the public keys those are valid and passes the public key validity check alone are allowed.
  - `Decryption`$(params, sk, c)$: Decryption for any ciphertext c can be queried by $\mathcal{A}$, irrespective of the state information, $\mathcal{C}$ should be able to provide the decryption.

– $\mathcal{A}$ gives $\mathcal{C}$ two messages m_0 and m_1 of the same length.
– $\mathcal{C}$ chooses a random bit $\beta \leftarrow \{0, 1\}$ and generates the challenge ciphertext $c^* \leftarrow$ `Encryption`$(params, st^*, pk, m_\beta)$ and gives it to $\mathcal{A}$.
– $\mathcal{A}$ continues to get oracle access to all ciphertexts for any message including m_0 and m_1 for the state information st^* through the encryption oracle `Encryption`$(params,\ st^*, pk, m_j)$, where $j \leq \hat{m}$.
– $\mathcal{A}$ also gets access to the `Decryption` oracle, where it is allowed to query the decryption of any ciphertext $c \neq c^*$ and outputs a bit β' finally.
– $\mathcal{C}$ outputs 1, if $\beta = \beta'$ and 0 otherwise.

A stateful public key encryption scheme $\mathcal{SPKE}$ has indistinguishable encryptions under adaptive chosen ciphertext attack (CCA) if for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function $negl(.)$ such that:

$$Pr[\mathcal{SPKE}_{\mathcal{A}}^{CCA}(\kappa) \rightarrow 1] \leq \frac{1}{2} + negl(\kappa)$$

3 Stateful Identity Based Encryption Scheme

In this section, we review the stateful identity based encryption scheme by Baek et al. [4]. We show that, the security proof of the scheme cannot be proved under the CBDH assumption. We prove this assuming the hardness of the Y-Computational problem. We provide the exact proof of the scheme by reducing it to a slightly stronger assumption, namely the GBDH assumption.

3.1 Review of $\mathcal{SIBE}$ in [4]:

`Setup(`κ`)`: Let $\mathbb{G}$ and $\mathbb{G}_T$ be two groups with prime order $p \approx 2^{2\kappa}$, where $\mathbb{G}$ is an additive group and $\mathbb{G}_T$ is a multiplicative group. Let $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be an admissible pairing. Pick $P \in_R \mathbb{G}$, $x \in_R \mathbb{Z}_p^*$ and compute $P_{pub} = xP$. Choose a length preserving symmetric key encryption scheme $\Pi_{sym} = (E, D)$ (eg: One-Time Pad) and three cryptographic hash functions $G : \mathbb{G} \times \{0, 1\}^\kappa \rightarrow \{0, 1\}^{\kappa_k}$, where $\kappa_k = 2\kappa$, $H_1 : \{0, 1\}^* \rightarrow \mathbb{G}$ and $H_2 : \mathbb{G} \times \mathbb{G} \times \{0, 1\}^* \times \mathbb{G}_T \rightarrow \{0, 1\}^\kappa$. The system parameters are $params = \langle \mathbb{G}, \mathbb{G}_T, p, P, P_{pub}, \Pi_{sym}, G, H_1, H_2 \rangle$ and the master secret key is $msk = x$.

`Extract(`$params, msk, ID_A$`)`: To extract the private key of a user with identity ID_A, compute $Q_A = H_1(ID_A)$ and compute $D_A = xQ_A$.

`New State(`$params$`)`: The sender generates the state information as follows:

– Choose $r \in_R \mathbb{Z}_p^*$
– Compute $U = rP$
– Compute $T = rP_{pub}$

The state information $st = \langle U, T \rangle$.

`Encryption(`$params, st, ID_A, m$`)`: To generate the ciphertext with params, state information, public key and the message as input the sender computes $Q_A = H_1(ID_A)$ and $\omega = \hat{e}(Q_A, T)$. Chooses $s \in_R \{0, 1\}^\kappa$, computes $k = G(U, s)$, $V = E_k(m)$ and $W = H_2(Q_A, U, V, \omega) \oplus s$. The ciphertext is $c = \langle U, V, W \rangle$.

`Decryption`$(params, D_A, c)$ To decrypt the ciphertext with the private key D_A the receiver with identity ID_A computes $\omega = \hat{e}(D_A, U)$, $s = H_2(Q_A, U, V, \omega) \oplus W$, $k = G(U, s)$ and $m = D_k(V)$. Outputs m as the corresponding message.

3.2 Comment on the Proof of $\mathcal{SIBE}$ in [4]:

The following is a slightly generalised version of the Y-Computational (YC) problem defined in [12].

Definition 8. *An instance generator $\mathcal{I}_{YC}(1^\kappa)$ for the Y-Computation problem outputs a description of $(S_1, S_2, S_3, f_1, f_2, t)$. Here, S_1, S_2 and S_3 are sets with $|S_1| = |S_2| = |S_3| = \kappa$; $f_1 : S_1 \rightarrow S_2$, $f_2 : S_1 \rightarrow S_3$ are functions and $t : S_2 \rightarrow S_3$ is a trapdoor function such that for all $x \in S_1, t(f_1(x)) = f_2(x)$. The functions f_1, f_2 and t should be easy to evaluate and it should be possible to sample efficiently from S_1.*

Let $\mathcal{A}$ be an adversary and define

$$Adv_{\mathcal{A},\mathcal{I}_{YC}(\kappa)} = Pr\begin{bmatrix}(S_1, S_2, S_3, f_1, f_2, t) \leftarrow \mathcal{I}_{YC}(\kappa); \\ x \leftarrow S_1; \\ f_2(x) \leftarrow \mathcal{A}(S_1, S_2, S_3, f_1, f_2, f_1(x))\end{bmatrix}$$

The advantage function is defined as:

$$Adv_{\mathcal{A},\mathcal{I}_{YC}}(\kappa, t) = max\{Adv_{\mathcal{A},\mathcal{I}_{YC}}(\kappa)\}$$

Where the maximum is taken over all adversaries that run for time $\hat{t}$. We say that Y-Computation is hard for $\mathcal{I}_{YC}(\kappa)$ if $\hat{t}$ being polynomial in κ implies that the advantage function $Adv_{\mathcal{A},\mathcal{I}_{YC}}(\kappa, \hat{t})$ is negligible in κ.

A Hard Y-Computational Problem:

The instance generator $\mathcal{I}_{YC}(\kappa)$ generates a random κ-bit prime p, an additive group $\mathbb{G}$ of order p, a multiplicative group $\mathbb{G}_T$ of order p, chooses $P \in_R \mathbb{G}$ and an admissible bilinear map $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. The sets S_1, S_2 and S_3 are defined by $\mathbb{Z}_p^*, \mathbb{G}$ and $\mathbb{G}_T$ respectively. Choose $a, b \in_R \mathbb{Z}_p^*$, compute aP, bP and $\alpha = \hat{e}(aP, bP)$. The functions f_1, f_2 and t are defined as:

- $f_1(x) = xP$
- $f_2(x) = \alpha^x$ and
- $t(Y) = \hat{e}(Y, bP)^a$.

Obviously, $t(f_1(x)) = \hat{e}(f_1(x), bP)^a = \hat{e}(xP, bP)^a = \hat{e}(aP, bP)^x = \alpha^x = f_2(x)$ holds and Y-Computation is hard if the computational Bilinear Diffie Hellman (CBDH) assumption holds. This is because, computing $\hat{e}(aP, bP)^x$ from P, aP, bP and xP is the CBDH problem, which is exactly the case here.

Fig. 1. A Hard Y-Computational Problem

Proof Sketch:

As mentioned by the authors, if the $\mathcal{SIBE}$ scheme in [4] should be reduced to the CBDH problem, the reduction should be done as follows:

Let $(P, aP, bP, cP) \in_R \mathbb{G}^4$ be an instance of the CBDH problem. The aim of the challenger $\mathcal{C}$ is to find $\hat{e}(P, P)^{abc} \in \mathbb{G}_T$. $\mathcal{C}$ sets $P_{pub} = aP$, chooses a length preserving symmetric key encryption algorithm $\Pi_{sym} : (E, D)$ and sends $params = \langle \mathbb{G}, \mathbb{G}_T, p, P, P_{pub}, \Pi_{sym} \rangle$ to the adversary $\mathcal{A}$. Models the hash functions G, H_1, H_2 as random oracles.

Let L_G, L_{H_1} and L_{H_2} be the three lists that are used to store the input and the corresponding output to the random oracle queries. A typical entry of the list L_G will be of the form $\langle U \in \mathbb{G}, s \in \{0,1\}^\kappa, k \rangle$, where k is the output of the hash function corresponding to the input U and s. An entry of the list L_{H_1} will be of the form $\langle ID_i \in \{0,1\}^*, Q_i \in \mathbb{G} \rangle$, where Q_i is the hash value corresponding to the identity ID_i. The list L_{H_2} will have entries of the form $\langle Q_i \in \mathbb{G}, U \in \mathbb{G}, V \in \{0,1\}^*, \omega \in \mathbb{G}_T, h_2 \in \{0,1\}^\kappa \rangle$, here h_2 is the corresponding output. Let ID^* be the target identity and during the H_1 query, $H_1(ID^*)$ will be set as bP by $\mathcal{C}$.

Lemma 1. *$\mathcal{C}$ has to solve the Y-Computational problem if he has to consistently answer the decryption oracle queries.*

Proof: Assume that $\mathcal{A}$ constructs a ciphertext in the following way:

- Choose $r_1, r_2 \in_R \mathbb{Z}_p^*$.
- Compute $U = r_1 P$ and $T = r_2 P_{pub}$.
- Compute $Q = H_1(ID^*)$ and $\omega = \hat{e}(Q, T)$.
- Choose $s \in_R \{0,1\}^\kappa$ and queries the G oracle with (U, s) as input and obtain k as the corresponding output.
- Compute $V = E_k(m)$, query the H_2 oracle with (Q, U, V, ω) as input, obtain h_2 as output and compute $W = h_2 \oplus s$.
- The ciphertext is $c = \langle U, V, W \rangle$.

Now, $\mathcal{A}$ queries the decryption oracle with $c = \langle U, V, W \rangle$ as input. In order to decrypt the ciphertext c, $\mathcal{C}$ performs the following:

- Checks whether a tuple of the form $\langle Q, U, V, \omega, h_2 \rangle$ is available in the list L_{H_2}. If a tuple of this form is available in the list, retrieves the corresponding ω and h_2.
- Now, $\mathcal{C}$ has the values aP, bP, r_1P, $\hat{e}(aP, bP)$ and ω. With these values $\mathcal{C}$ has to compute $\hat{e}(aP, bP)^{r_1}$.
- Then, $\mathcal{C}$ has to find out whether $\omega = \hat{e}(Q, r_1 P_{pub}) = \hat{e}(bP, r_1 aP) = \hat{e}(aP, bP)^{r_1}$. (Note that the ω values retrieved in the above step is $\hat{e}(Q, T) = \hat{e}(bP, r_2 aP) = \hat{e}(aP, bP)^{r_2}$.)
- $\mathcal{C}$ confronts the Y-Computational problem here, with the sets S_1, S_2 and S_3 as $\mathbb{Z}_p^*, \mathbb{G}$ and $\mathbb{G}_T$ respectively, and $\alpha = \hat{e}(aP, bP)$ for unknown a and b. Here, $f_1(r_1) = r_1 P$ and $f_2(r_1) = \alpha^{r_1}$. The trapdoor function is $t(r_1 P) = \hat{e}(r_1 P, bP)^a$. To compute this $\mathcal{C}$ should know a. This is because, $t(f_1(r_1)) = \hat{e}(f_1(r_1), bP)^a = \hat{e}(r_1 P, bP)^a = \hat{e}(aP, bP)^{r_1} = \alpha^{r_1} = f_2(r_1)$.

Thus, $\mathcal{C}$ has to solve the Y-Computational problem to consistently answer the decryption queries of $\mathcal{A}$. □

This shows that the $\mathcal{SIBE}$ scheme from [4] is not provable in the random oracle model under the CBDH assumption.

3.3 The Exact Security of $\mathcal{SIBE}$ from [4]:

In the previous section, we have argued why the security of the $\mathcal{SIBE}$ proposed in [4] cannot be reduced to the CBDH problem. In this section, we show that the security of the $\mathcal{SIBE}$ can be related to the Gap Bilinear Diffie Hellman Problem (GBDH).

Theorem 1. *The stateful identity based encryption scheme $\mathcal{SIBE}$ is IND-CCA secure in the random oracle model if the GBDH problem is hard in $\langle \mathbb{G}, \mathbb{G}_T \rangle$.*

Let κ be the security parameter and $\mathbb{G}$ be a multiplicative group of order P. The challenger $\mathcal{C}$ is challenged with an instance of the GBDH problem, say $(P, aP, bP, cP) \in_R \mathbb{G}^4$ and access to a Decision Bilinear Diffie Hellman (DBDH) oracle $\mathcal{DBDH}_{P,a}(.,.,.)$ which on input $bP, cP \in \mathbb{G}^2$ and $\alpha \in \mathbb{G}_T$ outputs `True` if and only if $\hat{e}(P, P)^{abc} = \alpha$. The aim of $\mathcal{C}$ is to find $\hat{e}(P, P)^{abc}$. Consider an adversary $\mathcal{A}$, who is capable of breaking the IND-CCA security of $\mathcal{SIBE}$. $\mathcal{C}$ can make use of $\mathcal{A}$ to compute $\hat{e}(P, P)^{abc}$, by playing the following interactive game with $\mathcal{A}$.

Setup: $\mathcal{C}$ begins the game by setting up the system parameters as in the $\mathcal{SIBE}$ scheme by performing the following:

- Sets the master public key $P_{pub} = aP$ (where aP is taken from the GBDH instance).
- Hence, the master private key is a implicitly.
- Choose a length preserving symmetric key encryption scheme $\Pi_{sym} = (E, D)$.

$\mathcal{C}$ gives $\mathcal{A}$ the public parameters $params = \langle \mathbb{G}, \mathbb{G}_T, p, P, P_{pub}, \Pi_{sym} \rangle$ and designs the three cryptographic hash functions G, H_1, H_2 as random oracles $\mathcal{O}_G$, $\mathcal{O}_{H_1}$ and $\mathcal{O}_{H_2}$ respectively. $\mathcal{C}$ maintains three lists L_G, L_{H_1} and L_{H_2} in order to consistently respond to the queries to the random oracles. A typical entry in lists will have the input parameters of hash functions followed by the corresponding hash value returned as the response to the hash oracle query. 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 $i = 1$ to $\hat{n}$, $\mathcal{C}$ performs the following:
  • Choose $r_i \in_R \mathbb{Z}_p^*$
  • Compute $U_i = r_i P$
  • Compute $T_i = r_i P_{pub}$
– Store the tuple $\langle r_i, U_i, T_i \rangle$ in the list L_{st}

The game proceeds as per the $\mathcal{SIBE}_{\mathcal{A}}^{CCA}(\kappa)$ game.

Phase I: $\mathcal{A}$ 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 $\mathcal{A}$ are described below:

$\mathcal{O}_{H_1}(ID_i \in \{0,1\}^*)$**:** We will make a simplifying assumption that $\mathcal{A}$ queries the $\mathcal{O}_{H_1}$ oracle with distinct identities in each query. Without loss of generality, if the oracle query is repeated with an already queried identity, by definition the oracle consults the list L_{H_1} and gives the same response. Thus, we assume that $\mathcal{A}$ asks q_{H_1} distinct queries for q_{H_1} distinct identities. Among this q_{H_1} identities, a random identity has to be selected by $\mathcal{C}$ as target identity and it is done as follows (Note that $\mathcal{A}$ should also choose this identity in the challenge phase).

$\mathcal{C}$ selects a random index γ, where $1 \leq \gamma \leq q_{H_1}$. $\mathcal{C}$ does not reveal γ to $\mathcal{A}$. When $\mathcal{A}$ puts forth the γ^{th} query on ID_γ, $\mathcal{C}$ decides to fix ID_γ as target identity for the challenge phase. Moreover, $\mathcal{C}$ responds to $\mathcal{A}$ as follows:

– If it is the γ^{th} query, then $\mathcal{C}$ sets $Q_\gamma = bP$ and stores the tuple $\langle ID_\gamma, Q_\gamma = bP, - \rangle$ in the list L_{H_1}. Here, $\mathcal{C}$ does not know b. $\mathcal{C}$ is simply using the bP value given in the instance of the GBDH problem.
– For all other queries, $\mathcal{C}$ chooses $b_i \in_R \mathbb{Z}_p^*$ and sets $Q_i = b_i P$ and stores $\langle ID_i, Q_i, b_i \rangle$ in the list L_{H_1}.

$\mathcal{C}$ returns Q_i to $\mathcal{A}$. (Note that as the identities are assumed to be distinct, for each query, we create distinct entry and add in the list L_{H_1}).

$\mathcal{O}_{H_2}(Q_i, U, V, \omega)$**:** To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle Q_i, U, V, \omega, h_2 \rangle$ exists in the list L_{H_2}. If a tuple of this form exists, $\mathcal{C}$ returns the corresponding h_2, else chooses $h_2 \in_R \{0,1\}^\kappa$, adds the tuple $\langle Q_i, U, V, \omega, h_2 \rangle$ to the list L_{H_2} and returns h_2 to $\mathcal{A}$.

$\mathcal{O}_G(U, s)$**:** To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle U, s, g \rangle$ exists in the list L_G. If a tuple of this form exists, $\mathcal{C}$ returns the corresponding g, else chooses $g \in_R \{0,1\}^{\kappa_k}$, adds the tuple $\langle U, s, g \rangle$ to the list L_G and returns g to $\mathcal{A}$.

$\mathcal{O}_{Extract}(ID_i)$: $\mathcal{A}$ submits an identity ID_i and queries the corresponding private key. $\mathcal{C}$ responds as follows:

– If $ID = ID_\gamma$ then, $\mathcal{C}$ aborts the game.
– Else, retrieve the tuple of the form $\langle ID_i, b_i P, b_i \rangle$ and compute $D_i = b_i aP$ (Where. aP is the master public key P_{pub}) and sends D_i to $\mathcal{A}$.

$\mathcal{O}_{Encryption}(st_i, m_j, ID)$: $\mathcal{A}$ may perform encryption with respect to any state information st_i, chosen by $\mathcal{C}$. $\mathcal{C}$ performs the following to encrypt the message m_j with respect to the state information st_i, where $i = 1$ to $\hat{n}$ and $\hat{n}$ is bound by the total number of states and $j = 1$ to $\hat{m}$ where $\hat{m}$ is bound by the total number of messages that can be queried per state. $\mathcal{C}$ performs the following to encrypt m_j:

– Retrieve the tuple $st_i = \langle r_i, U_i, T_i \rangle$ from the list L_{st} and set $U = U_i$.
– Retrieve the tuple of the form $\langle ID_i, Q_i, b_i \rangle$ from the list L_{H_1}, where $ID_i = ID$ and compute $\omega = \hat{e}(Q_i, T_i)$.
– Choose $s \in_R \{0,1\}^\kappa$, $g \in_R \{0,1\}^{\kappa_k}$, add the tuple $\langle U_i, s, g \rangle$ in list L_G and compute $V = E_g(m_j)$.
– Choose $h_2 \in_R \{0,1\}^\kappa$, add the tuple $\langle Q_i, U_i, V, \omega, h_2 \rangle$ in the list L_{H_2} and compute $W = h_2 \oplus s$.

The ciphertext $c = \langle U_i, V, W \rangle$ is sent to $\mathcal{A}$.

$\mathcal{O}_{Decryption}(c, ID_i)$: $\mathcal{C}$ performs the following to decrypt the ciphertext $c = \langle U, V, W \rangle$:

If $ID_i \neq ID_\gamma$ then $\mathcal{C}$ performs the decryption as per the decryption algorithm. If $ID_i = ID_\gamma$, $\mathcal{C}$ performs the following:

– Q_i corresponding to ID_i is bP.
– Retrieve the tuple of the form $\langle Q_i, U, V, \omega, h_2 \rangle$ from the list L_{H_2} and check whether $\mathcal{DBDH}_{P,a}(Q_i, U, \omega) =$ `True`. If so proceed; else, reject c.

– Compute $s = W \oplus h_2$.
– Retrieve the tuple of the form $\langle U, s, g\rangle$ from the list L_G. If there is no tuple of this form, then reject the ciphertext c; else, compute $m = D_g(V)$ and output m.

Challenge: At the end of ***Phase I***, $\mathcal{A}$ produces two messages m_0 and m_1 of equal length and an identity ID^*. If $ID^* \neq ID_\gamma$, $\mathcal{C}$ aborts. Else, $\mathcal{C}$ randomly chooses a bit $\beta \in_R \{0,1\}$ and computes a ciphertext c^* by performing the following steps:

– Set $U^* = cP$ (cP taken from the GBDH instance),
– Choose $s \in_R \{0,1\}^\kappa$.
– Choose $g \in_R \{0,1\}^{\kappa_k}$, add the tuple $\langle U^*, s, g\rangle$ in the list L_G and compute $V^* = E_g(m_\beta)$.
– Add the tuple $\langle Q_i, U^*, V^*, -, h_2\rangle$ in the list L_{H_2} and the tuple $st^* = \langle -, U^*, -\rangle$ in the list L_{st}
– Compute $W^* = h_2 \oplus s$.

Now, $c^* = \langle U^*, V^*, W^*\rangle$ is sent to $\mathcal{A}$ as the challenge ciphertext.

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

– $\mathcal{A}$ should not query the $\mathcal{O}_{Decryption}$ oracle with c^* as input.
– $\mathcal{A}$ continues to get oracle access to all ciphertexts for any message including m_0 and m_1 for the state information st^* through the encryption oracle $\texttt{Encryption}(params,\ st^*, pk, m)$.

Note that the simulation of $\mathcal{O}_{H_2}$, encryption and decryption oracle with respect to the challenge state information st^* is not trivial because the adversary himself does not know the randomness used to generate the state information st^* because U^* is set to be cP during the challenge phase and hence we describe them below:

$\mathcal{O}_{H_2}(Q_i, U, V, \omega)$**:** To respond to this query, $\mathcal{C}$ performs the following:

– If a tuple of the form $\langle -, U, -\rangle$ appears in the list L_{st} (i.e. $U = U^* = cP$)
  • If a tuple of the form $\langle Q_i, U, V, \omega, h_2\rangle$ exists in the list L_{H_2}.
    ∗ Return the corresponding h_2,
  • Else,
    ∗ If $Q_i = Q_\gamma$, check whether $\mathcal{DBDH}_{P,a}(Q_i, U, \omega) = \texttt{True}$. If so output ω as the output to the GBDH Problem.
    ∗ If $Q_i \neq Q_\gamma$, choose $h_2 \in_R \{0,1\}^\kappa$, add the tuple $\langle Q_i, U, V, \omega, h_2\rangle$ to the list L_{H_2} and return h_2 to $\mathcal{A}$.
– Else, (There exists a tuple of the form $\langle r_i, U_i, T_i\rangle$ in the list L_{st} such that $U_i = U$ and $U_i \neq cP$)
  • Check whether a tuple of the form $\langle Q_i, U, V, \omega, h_2\rangle$ exists in the list L_{H_2}. If a tuple of this form exists, return the corresponding h_2, else choose $h_2 \in_R \{0,1\}^\kappa$, add the tuple $\langle Q_i, U, V, \omega, h_2\rangle$ to the list L_{H_2} and return h_2 to $\mathcal{A}$.

$\mathcal{O}_{Encryption}(st^*, m_j, ID)$: When $\mathcal{A}$ queries this oracle for the state information st^* and when $ID \neq ID_\gamma$, $\mathcal{C}$ responds in the normal way as mentioned in Phase I. However, when $ID = ID_\gamma$, $\mathcal{C}$ performs the following to encrypt m_j:

– Set $U = U^* = cP$.
– Retrieve the tuple of the form $\langle ID, Q_\gamma, -\rangle$ from the list L_{H_1}.
– Choose $s \in_R \{0,1\}^\kappa$, $g \in_R \{0,1\}^{\kappa_k}$, add the tuple $\langle U, s, g\rangle$ in list L_G and compute $V = E_g(m_j)$.
– Choose $h_2 \in_R \{0,1\}^\kappa$, add the tuple $\langle Q_\gamma, U, V, -, h_2\rangle$ in the list L_{H_2} and compute $W = h_2 \oplus s$.

The ciphertext $c = \langle U, V, W\rangle$ is sent to $\mathcal{A}$.

$\mathcal{O}_{Decryption}(c, ID)$: When $ID \neq ID_\gamma$ or $ID = ID_\gamma$ and $U \in c = U^* \in st^*$ (i.e. the ciphertext c corresponds to the state st^*), $\mathcal{C}$ performs the following to decrypt the ciphertext $c = \langle U, V, W\rangle$:

– Retrieve Q_i corresponding to ID from the list L_{H_1}.
– Retrieve the tuple of the form $\langle Q_i, U, V, -, h_2\rangle$ from the list L_{H_2}. If there is no tuple of this form in L_{H_2}, reject c.

- Compute $s = W \oplus h_2$.
- Retrieve the tuple of the form $\langle U, s, g\rangle$ from the list L_G. If there is no tuple of this form, then reject the ciphertext c; else, compute $m = D_g(V)$ and output m.

Guess: At the end of ***Phase II***, $\mathcal{A}$ produces a bit β' to $\mathcal{C}$, but $\mathcal{C}$ ignores the response and performs the following to output the solution for the GBDH problem instance.

- Each time a query for the $\mathcal{O}_{H_2}$ oracle is made by $\mathcal{A}$ with $\langle Q_i, U, V, \omega\rangle$ as input, $\mathcal{C}$ checks whether $Q_i = Q_\gamma$ and $\mathcal{DBDH}_{P,a}(Q_i, U, \omega) = \texttt{True}$.
- If the checks hold, output ω as the solution to the GBDH problem instance.

Correctness: Below, we show that the ω value obtained through the above steps is indeed $\hat{e}(P,P)^{abc}$.

- The master public key P_{pub} of the system is set to be aP by $\mathcal{C}$; therefore, the master secret key $msk = x = a$ implicitly.
- The public key Q_γ of the target identity ID_γ is set to be bP by $\mathcal{C}$. Therefore the private key $D_\gamma = xQ_\gamma = abP$ implicitly.
- $\mathcal{C}$ has set the U^* component of the challenge ciphertext c^* as cP during the challenge phase.
- In order to decrypt the ciphertext c^*, $\mathcal{A}$ should compute a value $\omega = \hat{e}(Q_\gamma, T)$, where $T = caP$ and query the $\mathcal{O}_{H_2}$ oracle with $\langle Q_\gamma, U^*, V^*, \omega\rangle$ as input.
- Thus when $\mathcal{DBDH}_{P,a}(Q_\gamma, U^*, \omega) = \texttt{True}$, $\omega = \hat{e}(P,P)^{abc}$ which is the solution to the GBDH problem.

The events in which $\mathcal{C}$ aborts the game and the respective probabilities are given below:

1. $\mathcal{E}_1$ - The event in which $\mathcal{C}$ *aborts* when $\mathcal{A}$ queries the private key corresponding to ID_γ.
2. $\mathcal{E}_2$ - The event in which ID_γ is not chosen as the target identity by $\mathcal{A}$ for the challenge phase.

Suppose $\mathcal{A}$ has made q_{H_1} number of $\mathcal{O}_{H_1}$ queries and q_e number of $\mathcal{O}_{Extract}$ queries, then: $\Pr[\mathcal{E}_1]= \dfrac{q_e}{q_{H_1}}$ and $\Pr[\mathcal{E}_2]= 1 - \dfrac{1}{q_{H_1} - q_e}$. Therefore,

$$\begin{aligned}\Pr[\neg abort] &= [\neg\mathcal{E}_1 \wedge \neg\mathcal{E}_2]\\ &= \left[1 - \frac{q_e}{q_{H_1}}\right]\cdot\left[1 - 1 - \frac{1}{q_{H_1} - q_e}\right]\\ &= \frac{1}{q_{H_1}}.\end{aligned}$$

Therefore, the advantage of $\mathcal{C}$ solving the GBDH problem is $\epsilon' \geq \left(\epsilon.\dfrac{1}{q_{H_1}}\right)$, where ϵ is the advantage of $\mathcal{A}$ in breaking the IND-CCA security of the $\mathcal{SIBE}$ scheme.

■

4 Stateful Public Key Encryption Scheme ($\mathcal{N} - \mathcal{SPKE}_1$)

In this section, we propose a compact CCA secure public key encryption scheme which provides shorter ciphertext and is stateful, in the sense that the same randomness can be used across a session that typically comprises encrypting different messages to the same receiver during the session. The ciphertext overhead of our scheme is slightly higher than the recent stateful public key encryption scheme reported in [4] with the added advantage that the ciphertext is verifiable after the decryption process. The main thing to be noticed is that this ciphertext verifiability property comes with almost the same computational complexity as the scheme in [4] and one more exponentiation for decryption which is strictly due to the additional verifiability property of our scheme. The description of the new stateful public key encryption scheme with verifiable ciphertext follows:

$\texttt{Setup}(\kappa)$: Let κ be the security parameter and $\mathbb{G}$ be a group of prime order q. Choose a generator $g \in_R \mathbb{G}$. Let $F : \mathbb{G} \rightarrow \mathbb{Z}_q$, $G : \mathbb{G} \times \mathbb{G} \times \{0,1\}^{l_m} \times \{0,1\}^{\mu} \rightarrow \{0,1\}^{\lambda}$ and $H : \mathbb{G} \times \mathbb{G} \times \{0,1\}^{\lambda} \times \{0,1\}^{\mu} \rightarrow \{0,1\}^{l_m}$ be three cryptographic hash functions, where λ is a parameter such that any computation involving 2^{λ} or more steps is considered infeasible in practice, l_m represents the size of the message, μ is the size of the `index` used in the scheme. Typically `index` may be a number from 1 to 2^{20} (this supports one million encryption per session) and hence the size of `index` will be utmost 20-bits. Set the system parameters as $params = \langle \kappa, q, g, \mathbb{G}, F, G, H, \rangle$.

`Key Generation(`$params$`):` Choose $x \in_R \mathbb{Z}_q$ and compute $h = g^x$. The private key of the user is $sk = x$ and the public keys are $pk = \langle g, h \rangle$.

`New State(`$params$`):` 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 $u_i = F(g^{r_i}) \in \mathbb{Z}_q$
- Compute $s_i = r_i u_i \in \mathbb{Z}_q$
- Compute $v_i = g^{s_i}$

The state information $st_i = \langle u_i, v_i, s_i \rangle$.

`Encryption(`$params, st_i, pk, m$`):` Let `index` be a number which represents the invocation number of the encryption algorithm in the i^{th} session. So during the start of each session, the value of `index` is initialized to 1 and incremented each time an encryption is performed during the session. The sender generates the ciphertext with params, state information, public key and the message as follows:

- Set $c_1 = v_i$
- Compute $w = h^{s_i}$
- Compute $c_2 = G(c_1, w, m, \texttt{index}) \oplus u_i$
- Compute $c_3 = H(c_1, w, c_2, \texttt{index}) \oplus m$

The ciphertext $c = \langle c_1, c_2, c_3, \texttt{index} \rangle$. We emphasize that he 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`$(params, sk, c)$ The receiver decrypts the ciphertext with the private key by performing the following:

- Compute $w' = c_1^{sk}$
- Compute $m' = c_3 \oplus H(c_1, w', c_2, \texttt{index})$
- Compute $u' = c_2 \oplus G(c_1, w', m', \texttt{index})$

Check whether $u' \stackrel{?}{=} F(c_1^{(u')^{-1}})$. If the check holds output m', otherwise output $\perp$.

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 $u' = u_i = F(g^{r_i})$.

Theorem 2. *The compact stateful public key encryption scheme $\mathcal{N} - \mathcal{SPKE}_1$ is IND-CCA secure in the random oracle model if the SDH problem is hard in $\mathbb{G}$.*

Let κ be the security parameter and $\mathbb{G}$ be a multiplicative group of order q, where $|q| = \kappa$. The challenger $\mathcal{C}$ is challenged with an instance of the SDH problem, say $(g, g^a, g^b) \in_R \mathbb{G}^3$ and access to a DDH oracle $\mathcal{DDH}_{g,a}(.,.)$ which on input g^b and g^c outputs `True` if and only if $g^{ab} = g^c$. Consider an adversary $\mathcal{A}$, who is capable of breaking the IND-CCA security of the scheme $\mathcal{N} - \mathcal{SPKE}_1$. $\mathcal{C}$ can make use of $\mathcal{A}$ to compute g^{ab}, by playing the following interactive game with $\mathcal{A}$.

Setup: $\mathcal{C}$ begins the game by setting up the system parameters as in the $\mathcal{N} - \mathcal{SPKE}_1$ scheme by performing the following:

– Sets the public key $h = g^a$ (where g^a is taken from the SDH instance).
– Hence, the private key is a implicitly.

$\mathcal{C}$ gives $\mathcal{A}$ the public keys $pk = \langle g, h \rangle$ and $\mathcal{C}$ also designs the three cryptographic hash functions F, G and H as random oracles $\mathcal{O}_F$, $\mathcal{O}_G$ and $\mathcal{O}_H$. $\mathcal{C}$ maintains three lists L_F, L_G and L_H in order to consistently respond to the queries to the random oracles $\mathcal{O}_F$, $\mathcal{O}_G$ and $\mathcal{O}_H$ respectively. A typical entry in list $L_{\hat{h}}$ will have the input parameters of hash functions $\hat{h}$ (for $\hat{h} = F, G$ and H) followed by the corresponding hash value returned as the response to the hash oracle query. 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 $i = 1$ to $\hat{n}$, $\mathcal{C}$ performs the following:
  • Choose $r_i \in_R \mathbb{Z}_q$, compute $k_i = g^{r_i}$, choose $u_i \in_R \mathbb{Z}_q$ and adds the tuple $\langle k_i, u_i \rangle$ in the list L_F, compute $s_i = r_i u_i$ and compute $v_i = g^{s_i}$.
  • The state information $st_i = \langle u_i, v_i, s_i, \mathtt{index}_i = 1 \rangle$.
  • Store the tuple st_i in list L_{st}.

The game proceeds as per the $\mathcal{SPKE}^{CCA}_{\mathcal{A}}(\kappa)$ game.

Phase I: $\mathcal{A}$ 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 $\mathcal{A}$ are described below:

$\mathcal{O}_F(k \in \mathbb{G})$**:** To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle k, u \rangle$ exists in the list L_F. If a tuple of this form exists, $\mathcal{C}$ returns the corresponding u, else chooses $u \in_R \mathbb{Z}_q$, adds the tuple $\langle k, u \rangle$ to the list L_F and returns u to $\mathcal{A}$.

$\mathcal{O}_G(c_1 \in \mathbb{G}, w \in \mathbb{G}, m \in \{0,1\}^{l_m}, \mathtt{index} \in \{0,1\}^{\mu})$**:** To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle c_1, w, m, \mathtt{index}, h_1 \rangle$ exists in the list L_G. If a tuple of this form exists, $\mathcal{C}$ returns the corresponding h_1, else chooses $h_1 \in_R \{0,1\}^{\lambda}$, adds the tuple $\langle c_1, w, m, \mathtt{index}, h_1 \rangle$ to the list L_G and returns h_1 to $\mathcal{A}$.

$\mathcal{O}_H(c_1 \in \mathbb{G}, w \in \mathbb{G}, c_2 \in \{0,1\}^{\lambda}, \mathtt{index} \in \{0,1\}^{\mu})$**:** To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle c_1, w, c_2, \mathtt{index}, h_2 \rangle$ exists in the list L_H. If a tuple of this form exists, $\mathcal{C}$ returns the corresponding h_2, else chooses $h_2 \in_R \{0,1\}^{l_m}$, adds the tuple $\langle c_1, w, c_2, \mathtt{index}, h_2 \rangle$ to the list L_H and returns h_2 to $\mathcal{A}$.

$\mathcal{O}_{Encryption}(st_i, m_j)$: $\mathcal{A}$ may perform encryption with respect to any state information st_i, chosen by $\mathcal{C}$. $\mathcal{C}$ performs the following to encrypt the message m_j with respect to the state information st_i, where $i = 1$ to $\hat{n}$, where $\hat{n}$ is bound by the total number of states and $j = 1$ to $\hat{m}$ is bound by the number of messages that can be encrypted in one session:

– $\mathcal{C}$ retrieves the tuple st_i of the form $\langle u_i, v_i, s_i, \mathtt{index}_i \rangle$ from L_{st}, sets $c_1 = v_i$, computes $w = h^{s_i}$.
– Chooses $h_1 \in_R \{0,1\}^{\lambda}$, adds the tuple $\langle c_1, w, m_j, \mathtt{index}_i,\ h_1 \rangle$ to the list L_G and computes $c_2 = h_1 \oplus u_i$.
– Chooses $h_2 \in_R \{0,1\}^{l_m}$, adds the tuple $\langle c_1, w, c_2, \mathtt{index}_i,\ h_2 \rangle$ to the list L_H and computes $c_3 = h_2 \oplus m_j$.
– Returns $c = \langle c_1, c_2, c_3 \rangle$ as the ciphertext, increments $\mathtt{index}_i$ and updates the state information st_i.

$\mathcal{O}_{Decryption}(c)$: $\mathcal{C}$ performs the following to decrypt the ciphertext $c = \langle c_1, c_2, c_3, \mathtt{index} \rangle$:

– Retrieve the tuple $\langle c_1, w, c_2, \mathtt{index}, h_2 \rangle$ from list L_H such that the output of the DDH oracle query $\mathcal{DDH}_{g,a}(w, c_1)$ is $\mathtt{True}$ and compute $m' = c_3 \oplus h_2$.
– Check whether a tuple of the form $\langle c_1, w, m, \mathtt{index}, h_1 \rangle$, where w is the same as the w value retrieved from the tuple in the list L_H and m is equal to m' computed in the above step appears in the list L_G. If such a tuple appears, retrieve h_1 and compute $u' = c_2 \oplus h_1$.
– Check whether a tuple of the form $\langle k, u \rangle$, where $k = c_1^{u'^{-1}}$ and $u = u'$ appears in list L_F,
– If any of the required tuples did not appear in the lists L_F, L_G or L_H return $\perp$.

Challenge: At the end of ***Phase I***, $\mathcal{A}$ produces two messages m_0 and m_1 of equal length. $\mathcal{C}$ randomly chooses a bit $\beta \in_R \{0,1\}$ and computes a ciphertext c^* by performing the following steps:

– Choose $u \in_R \{0,1\}^{\lambda}$ and add the tuple $\langle g^b, u \rangle$ to the list L_F.
– Set $\mathtt{index}^* = 1$.
– Compute $c_1^* = g^{bu}$
– Choose $h_1 \in_R \{0,1\}^{\lambda}$ and add the tuple $\langle c_1^*, -, m_{\beta}, \mathtt{index}^*,\ h_1 \rangle$ in the list L_G.

– Compute $c_2^* = h_1 \oplus u$.
– Choose $h_2 \in_R \{0,1\}^{l_m}$ and add the tuple $\langle c_1^*, -, c_2, \texttt{index}^*, h_2\rangle$ in the list L_H.
– Compute $c_3^* = h_2 \oplus m_\beta$.
– Here the state information $st^* = \langle u^* = u, v^* = g^{bu}, s^* = -, \texttt{index}^*\rangle$

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

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

– $\mathcal{A}$ should not query the $\mathcal{O}_{Decryption}$ oracle with c^* as input.
– $\mathcal{A}$ continues to get oracle access to all the oracles. It can also get the encryption for any message including m_0 and m_1 for the state information st^* through the encryption oracle $\texttt{Encryption}(params, st^*, pk, m_j)$.

The simulation of the $\mathcal{O}_G$, $\mathcal{O}_H$, $\mathcal{O}_{Encryption}$ and $\mathcal{O}_{Decryption}$ oracles are not same as in Phase I and hence we provide the details below:

$\mathcal{O}_G(c_1 \in \mathbb{G}, w \in \mathbb{G}, m \in \{0,1\}^{l_m}, \texttt{index} \in \{0,1\}^\mu)$**:** To respond to this query, $\mathcal{C}$ performs the following:

– Check whether a tuple of the form $\langle c_1, w, m, \texttt{index}, h_1\rangle$ exists in the list L_G. If a tuple of this form exists, return the corresponding h_1, else,
  - If $c_1 = c_1^*$ then check with the DDH oracle whether $\mathcal{DDH}_{g,a}(w, c_1)$ is $\texttt{True}$. If the output is $\texttt{True}$, return $w^{u^{*-1}}$ as the solution to the SDH problem instance.
  - Else, choose $h_1 \in_R \{0,1\}^\lambda$, add the tuple $\langle c_1, w, m, \texttt{index}, h_1\rangle$ to the list L_G and return h_1 to $\mathcal{A}$.

$\mathcal{O}_H(c_1 \in \mathbb{G}, w \in \mathbb{G}, c_2 \in \{0,1\}^\lambda, \texttt{index} \in \{0,1\}^\mu)$**:** To respond to this query, $\mathcal{C}$ performs the following:

– Check whether a tuple of the form $\langle c_1, w, c_2, \texttt{index}, h_2\rangle$ exists in the list L_H. If a tuple of this form exists, $\mathcal{C}$ returns the corresponding h_2, else,
  - If $c_1 = c_1^*$ then check with the DDH oracle whether $\mathcal{DDH}_{g,a}(w, c_1)$ is $\texttt{True}$. If the output is $\texttt{True}$, return $w^{u^{*-1}}$ as the solution to the SDH problem instance.
  - Else, choose $h_2 \in_R \{0,1\}^{l_m}$, add the tuple $\langle w, c_2, \texttt{index}, h_2\rangle$ to the list L_H and return h_2 to $\mathcal{A}$.

$\mathcal{O}_{Encryption}(st_i, m_j)$: $\mathcal{A}$ may perform encryption with respect to any state information st_i including st^*, chosen by $\mathcal{C}$. $\mathcal{C}$ performs the following to encrypt the message m_j with respect to the state information st_i:

– If $st_i \neq st^*$ then encryption is done as in Phase I
– If $st_i = st^*$ then perform the following:
  - Retrieve the tuple st^* of the form $st^* = \langle u^* = u, v^* = g^{bu}, s^* = -, \texttt{index}^*\rangle$ from L_{st} and set $c_1 = v^*$.
  - Choose $h_1 \in_R \{0,1\}^\lambda$, add the tuple $\langle c_1, -, m_j, \texttt{index}^*, h_1\rangle$ to the list L_G and compute $c_2 = h_1 \oplus u^*$.
  - Choose $h_2 \in_R \{0,1\}^{l_m}$, add the tuple $\langle c_1, -, c_2, \texttt{index}^*, h_2\rangle$ to the list L_H and compute $c_3 = h_2 \oplus m_j$.
  - Return $c = \langle c_1, c_2, c_3\rangle$ as the ciphertext, increment $\texttt{index}^*$ and update the state information st^*.

$\mathcal{O}_{Decryption}(c)$: $\mathcal{C}$ performs the following to decrypt the ciphertext $c = \langle c_1, c_2, c_3, \texttt{index}\rangle$:

– If $c_1 \neq c_1^*$ then decryption is done as in Phase - I
– If $c_1 = c_1^*$ then perform the following:
  - Retrieve the tuple of the form $\langle c_1, w, c_2, \texttt{index}, h_2\rangle$ from list L_H, such that the output of the DDH oracle query, $\mathcal{DDH}_{g,a}(w, c_1)$ is $\texttt{True}$. If the retrieved tuple is of the form $\langle c_1, -, c_2, \texttt{index}, h_2\rangle$ then it was the tuple generated by $\mathcal{C}$ during an encryption oracle query in the phase II. Note that $\mathcal{C}$ can even work consistently with the tuple of this form. In this case, $\mathcal{C}$ chooses the value h_2 without consulting the DDH oracle. Compute $m' = c_3 \oplus h_2$.
  - Check whether a tuple of the form $\langle c_1, w, m, \texttt{index}, h_1\rangle$, where w is the same as the w value retrieved from the tuple in the list L_H and m is equal to m' computed in the above step appears in the list L_G. If such a tuple appears, retrieve h_1 and compute $u' = c_2 \oplus h_1$. (Note that even in this case $\mathcal{C}$ can work consistently with the tuple of the form $\langle c_1, -, m, \texttt{index}, h_1\rangle$)
  - Check whether a tuple of the form $\langle k, u\rangle$, where $k = c_1^{u'^{-1}}$ and $u = u'$ appears in list L_F,
  - If any of the required tuples did not appear in the lists L_F, L_G or L_H return $\perp$.

- If in the process a tuple of the form $\langle c_1, w, c_2, \texttt{index}, h_2\rangle$ appeared in the list L_G and a tuple of the form $\langle c_1, w, m, \texttt{index}, h_1\rangle$ appeared in the list L_H with $\mathcal{DDH}_{g,a}(w, c_1)$ is $\texttt{True}$, then output w as the output to the SDH problem.

Lemma 2. *The decryption oracle responds correctly to well-formed ciphertexts and rejects invalid ciphertexts.*

Proof: Let us consider $c = \langle c_1, c_2, c_3, \texttt{index}\rangle$ is a well-formed ciphertext. In order to construct c, $\mathcal{A}$ should have done the following:

– Chosen $r \in_R \mathbb{Z}_q$ and queried the $\mathcal{O}_F$ oracle with $k = g^r$. Thus a tuple of the form $\langle k, u\rangle$ should appear in L_F.
– $\mathcal{A}$ should have computed $c_1 = g^{ru}$, $w = h^{ru}$ and queried the $\mathcal{O}_G$ oracle with $\langle c_1, w, m, \texttt{index}\rangle$ as input and received h_1 corresponding to this input.
– $\mathcal{A}$ should have computed $c_2 = h_1 \oplus u$ and queried the $\mathcal{O}_H$ oracle with $\langle c_1, w, c_2, \texttt{index}\rangle$ as input and received h_2 corresponding to this input.

During the decryption, $\mathcal{C}$ retrieves the corresponding tuples, one from the lists L_G and L_H for which both the w values are same and checks whether the output of the DDH oracle query, $\mathcal{DDH}_{g,a}(w, c_1)$ is $\texttt{True}$. For a well formed ciphertext, this check holds because,

$$c_1 = g^{ru} \tag{1}$$

$$w = h^{ru} = g^{aru} \tag{2}$$

From equations (1) and (2) it is clear that for a well formed ciphertext, this check holds and working with the corresponding h_1 and h_2 will properly yield the message during decryption. Else, the ciphertext will be rejected. □

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

– Each time a query for the $\mathcal{O}_G$ oracle is made by $\mathcal{A}$ with $(c_1, w, m, \texttt{index})$ as input, $\mathcal{C}$ computes $g' = w^{u^{*-1}}$ and checks whether $\mathcal{DDH}_{g,a}(g', g^b) \stackrel{?}{=} \texttt{True}$. Alternatively, $\mathcal{C}$ can also perform the same with $\mathcal{O}_H$ oracle queries.
– Outputs the corresponding g' value for which the above check holds as the solution for the SDH problem instance.

Correctness: Below, we show that the g' value obtained through the above steps is indeed g^{ab}.

– The public key h of the target user is set to be $pk = \langle g, h = g^a\rangle$ by $\mathcal{C}$. Therefore the private key $sk = x = a$ implicitly.
– $\mathcal{C}$ has set the c_1^* component of the challenge ciphertext c^* as g^{bu} (where, $u = F(g^b)$) during the challenge phase.
– In order to decrypt the ciphertext c^*, $\mathcal{A}$ should compute a value $w = g^{abu}$ and query the $\mathcal{O}_G$ oracle with w as the input.
– $\mathcal{C}$ computes $g' = w^{u^{-1}} = (g^{abu})^{u^{-1}} = g^{ab}$, for each value of u from the list L_F when ever a query is made to the $\mathcal{O}_G$ oracle with w as one of the inputs. $\mathcal{C}$ checks whether $\mathcal{DDH}_{g,a}(g', g^b) \stackrel{?}{=} \texttt{True}$, if so returns $g' = g^{ab}$ as the output to the SDH problem.

Thus, $\mathcal{C}$ obtains the solution to the SDH problem with almost the same advantage of $\mathcal{A}$ in the IND-CCA game. ■

5 Stateful Public Key Encryption Scheme ($\mathcal{N}-\mathcal{SPKE}_2$)

In this section, we propose a compact CCA secure public key encryption scheme whose security is based on the CDH problem. The ciphertext overhead and computational complexity of this scheme is same as that of the previous scheme and the ciphertext is verifiable after the decryption process. The description of this stateful public key encryption scheme follows:

`Setup`(κ): Same as the `Setup(.)` algorithm of $\mathcal{N}-\mathcal{SPKE}_1$.

`Key Generation`($params$): Choose $x, y \in_R \mathbb{Z}_q$, compute $g_1 = g^x$ and $g_2 = g^y$. The private key of the user is $sk = \langle x, y\rangle$ and the public keys are $pk = \langle g, g_1, g_2\rangle$.

`New State`($params$): Same as the `New State(.)` algorithm of $\mathcal{N}-\mathcal{SPKE}_1$.

`Encryption`($params, st_i, pk, m$): Let `index` be a number which represents the invocation number of the encryption algorithm in the i^{th} session. So during the start of each session, the value of `index` is initialized to 1 and incremented each time an encryption is performed during the session. The sender generates the ciphertext with params, state information $st_i = \langle u_i, v_i, s_i\rangle$, public key and the message as follows:

- Set $c_1 = v_i$
- Compute $w_1 = g_1^{s_i}$ and $w_2 = g_2^{s_i}$
- Compute $c_2 = G(c_1, w_1, m, \texttt{index}) \oplus u_i$
- Compute $c_3 = H(c_1, w_2, c_2, \texttt{index}) \oplus m$

The ciphertext $c = \langle c_1, c_2, c_3, \texttt{index}\rangle$. It should be noted that `index` is an integer such that $1 \leq \texttt{index} \leq 2^{20}$. So `index` may typically be of size less than 20-bits.

`Decryption`($params, sk, c$) The receiver decrypts the ciphertext with the private key by performing the following:

- Compute $w_1' = c_1^x$ and $w_2' = c_1^y$
- Compute $m' = c_3 \oplus H(c_1, w_2', c_2, \texttt{index})$
- Compute $u' = c_2 \oplus G(c_1, w_1', m', \texttt{index})$

Check whether $u' \stackrel{?}{=} F(c_1^{(u')^{-1}})$. If the check holds output m', otherwise output $\perp$.

Theorem 3. *The compact stateful public key encryption scheme $\mathcal{N}-\mathcal{SPKE}_2$ is IND-CCA secure in the random oracle model if the CDH problem is hard in $\mathbb{G}$.*

Let κ be the security parameter and $\mathbb{G}$ be a multiplicative group of order q, where $|q| = \kappa$. The challenger $\mathcal{C}$ is challenged with an instance of the CDH problem, say $(g, g^a, g^b) \in_R \mathbb{G}^3$. Consider an adversary $\mathcal{A}$, who is capable of breaking the IND-CCA security of the scheme $\mathcal{N}-\mathcal{SPKE}_2$. $\mathcal{C}$ can make use of $\mathcal{A}$ to compute g^{ab}, by playing the following interactive game with $\mathcal{A}$. The proof revolves around the technique of [7].

Setup: $\mathcal{C}$ begins the game by setting up the system parameters as in the $\mathcal{N}-\mathcal{SPKE}_2$ scheme by performing the following:

- Choose $z_1, z_2 \in_R \mathbb{Z}_q$.
- Set the public key $g_1 = g^a$ (where g^a is taken from the CDH instance).
- Compute $g_2 = g^{z_1}/g^{az_2}$
- Hence, the private keys are a and $(z_1 - az_2)$ implicitly.

$\mathcal{C}$ gives $\mathcal{A}$ the public keys $pk = \langle g, g_1, g_2\rangle$ and $\mathcal{C}$ also designs the three cryptographic hash functions F, G and H as random oracles $\mathcal{O}_F$, $\mathcal{O}_G$ and $\mathcal{O}_H$. $\mathcal{C}$ maintains three lists L_F, L_G and L_H in order to consistently respond to the queries to the random oracles $\mathcal{O}_F$, $\mathcal{O}_G$ and $\mathcal{O}_H$ respectively. A typical entry in list $L_{\hat{h}}$ will have the input parameters of hash functions $\hat{h}$ (for $\hat{h} = F, G$ and H) followed by the corresponding hash value returned as the response to the hash oracle query. 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 $i = 1$ to $\hat{n}$, $\mathcal{C}$ performs the following:

- Choose $r_i \in_R \mathbb{Z}_q$, compute $k_i = g^{r_i}$, choose $u_i \in_R \mathbb{Z}_q$ and adds the tuple $\langle k_i, u_i \rangle$ in the list L_F, compute $s_i = r_i u_i$ and compute $v_i = g^{s_i}$.
- The state information $st_i = \langle u_i, v_i, s_i, \mathtt{index}_i = 1 \rangle$.
- Store the tuple st_i in list L_{st}.

The game proceeds as per the $\mathcal{SPKE}_{\mathcal{A}}^{CCA}(\kappa)$ game.

Phase I: $\mathcal{A}$ 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 $\mathcal{A}$ are described below:

$\mathcal{O}_F(k \in \mathbb{G})$**:** To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle k, u \rangle$ exists in the list L_F. If a tuple of this form exists, $\mathcal{C}$ returns the corresponding u, else chooses $u \in_R \mathbb{Z}_q$, adds the tuple $\langle k, u \rangle$ to the list L_F and returns u to $\mathcal{A}$.

$\mathcal{O}_G(c_1 \in \mathbb{G}, w_1 \in \mathbb{G}, m \in \{0,1\}^{l_m}, \mathtt{index} \in \{0,1\}^{\mu})$**:** To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle c_1, w_1, m, \mathtt{index}, h_1 \rangle$ exists in the list L_G. If a tuple of this form exists, $\mathcal{C}$ returns the corresponding h_1, else chooses $h_1 \in_R \{0,1\}^{\lambda}$, adds the tuple $\langle c_1, w_1, m, \mathtt{index}, h_1 \rangle$ to the list L_G and returns h_1 to $\mathcal{A}$.

$\mathcal{O}_H(c_1 \in \mathbb{G}, w_2 \in \mathbb{G}, c_2 \in \{0,1\}^{\lambda}, \mathtt{index} \in \{0,1\}^{\mu})$**:** To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle c_1, w_2, c_2, \mathtt{index}, h_2 \rangle$ exists in the list L_H. If a tuple of this form exists, $\mathcal{C}$ returns the corresponding h_2, else chooses $h_2 \in_R \{0,1\}^{l_m}$, adds the tuple $\langle c_1, w_2, c_2, \mathtt{index}, h_2 \rangle$ to the list L_H and returns h_2 to $\mathcal{A}$.

$\mathcal{O}_{Encryption}(st_i, m_j)$: $\mathcal{A}$ may perform encryption with respect to any state information st_i, chosen by $\mathcal{C}$. $\mathcal{C}$ performs the following to encrypt the message m_j with respect to the state information st_i, where $i = 1$ to $\hat{n}$, where $\hat{n}$ is bound by the total number of states and $j = 1$ to $\hat{m}$ is bound by the number of messages that can be encrypted in one session:

– Retrieve the tuple st_i of the form $\langle u_i, v_i, s_i, \mathtt{index}_i \rangle$ from L_{st}, set $c_1 = v_i$, compute $w_1 = g_1^{s_i}$ and $w_2 = g_2^{s_i}$.
– Choose $h_1 \in_R \{0,1\}^{\lambda}$, add the tuple $\langle c_1, w_1, m_j, \mathtt{index}_i, h_1 \rangle$ to the list L_G and compute $c_2 = h_1 \oplus u_i$.
– Choose $h_2 \in_R \{0,1\}^{l_m}$, add the tuple $\langle c_1, w_2, c_2, \mathtt{index}_i, h_2 \rangle$ to the list L_H and compute $c_3 = h_2 \oplus m_j$.
– Return $c = \langle c_1, c_2, c_3 \rangle$ as the ciphertext, increment $\mathtt{index}_i$ and update the state information st_i.

$\mathcal{O}_{Decryption}(c)$: $\mathcal{C}$ performs the following to decrypt the ciphertext $c = \langle c_1, c_2, c_3, \mathtt{index} \rangle$:

– Retrieve the tuples of the form $\langle c_1, w_1, m, \mathtt{index}, h_1 \rangle$ from the list L_G. Consider that there are $\hat{n}_G$ such tuples. Choose the corresponding (w_{1i}, h_{1i}) values, for $i = 1$ to $\hat{n}_G$.
– Retrieve the tuples of the form $\langle c_1, w_2, c_2, \mathtt{index}, h_2 \rangle$ from the list L_H. Consider that there are $\hat{n}_H$ such tuples. Choose the corresponding (w_{2j}, h_{2j}) values, for $j = 1$ to $\hat{n}_H$.
– For $i = 1$ to $\hat{n}_G$
  - For $j = 1$ to $\hat{n}_H$
    * Check whether $w_{2j} \stackrel{?}{=} c_1^{z_1}/w_{1i}^{z_2}$.
    * If the check holds for some index $\hat{i}$ and $\hat{j}$, choose the corresponding $h_{1\hat{i}}$ and $h_{2\hat{j}}$. If the check does not hold for any tuple then reject the ciphertext c.
– Compute $m' = c_3 \oplus h_{2\hat{j}}$.
– Retrieve the value m from the tuple $\langle c_1, w_{1\hat{i}}, m, \mathtt{index}, h_{1\hat{i}} \rangle$ in the list L_G.
– If $(m = m')$, then compute $u' = c_2 \oplus h_{1\hat{i}}$, else reject the ciphertext c..
– Check whether a tuple of the form $\langle k, u \rangle$, where $k = c_1^{u'^{-1}}$ and $u = u'$ appears in list L_F. If it appears accept m' and return it as the message corresponding to c.
– If any of the required tuples did not appear in the lists L_F, L_G or L_H return $\perp$.

Lemma 3. *The decryption oracle responds correctly to well-formed ciphertexts and rejects invalid ciphertexts.*

Proof: Let us consider $c = \langle c_1, c_2, c_3, \mathtt{index} \rangle$ is a well-formed ciphertext. In order to construct c, $\mathcal{A}$ should have done the following:

– Chosen $r \in_R \mathbb{Z}_q$ and queried the $\mathcal{O}_F$ oracle with $k = g^r$. Thus a tuple of the form $\langle k, u \rangle$ should appear in L_F.
– $\mathcal{A}$ should have computed $c_1 = g^{ru}$, $w_1 = g_1^{ru}$, $w_2 = g_2^{ru}$ and queried the $\mathcal{O}_G$ oracle with $\langle c_1, w_1, m, \mathtt{index} \rangle$ as input and received h_1 corresponding to this input.

– $\mathcal{A}$ should have computed $c_2 = h_1 \oplus u$ and queried the $\mathcal{O}_H$ oracle with $\langle c_1, w_2, c_2, \texttt{index}\rangle$ as input and received h_2 corresponding to this input.

During the decryption, $\mathcal{C}$ retrieves the corresponding tuples, one from the list L_G and the other from the list L_H for which

$$w_2 = c_1^{z_1}/w_1^{z_2} \tag{3}$$

For a well formed ciphertext, this check holds because,

$$w_1 = g_1^{ru} = g^{aru} \tag{4}$$

$$w_2 = g_2^{ru} = g^{(z_1 - z_2 a)ru} \tag{5}$$

$$c_1^{z_1} = g^{z_1 ru} \tag{6}$$

From equations (4), (5) and (6), we have

$c_1^{z_1}/w_1^{z_2} = g^{z_1 ru}/g^{aruz_2} = g^{(z_1-z_2a)ru} = w_2$

This clearly shows that for a well formed ciphertext, this check holds and working with the corresponding h_1 and h_2 will properly yield the message during decryption. Else, the ciphertext will be rejected. □

Challenge: At the end of ***Phase I***, $\mathcal{A}$ produces two messages m_0 and m_1 of equal length. $\mathcal{C}$ randomly chooses a bit $\beta \in_R \{0,1\}$ and computes a ciphertext c^* by performing the following steps:

– Choose $u \in_R \{0,1\}^\lambda$ and add the tuple $\langle g^b, u\rangle$ to the list L_F.
– Set $\texttt{index}^* = 1$.
– Compute $c_1^* = g^{bu}$
– Choose $h_1 \in_R \{0,1\}^\lambda$ and add the tuple $\langle c_1^*, -, m_\beta, \texttt{index}^*, h_1\rangle$ in the list L_G.
– Compute $c_2^* = h_1 \oplus u$.
– Choose $h_2 \in_R \{0,1\}^{l_m}$ and add the tuple $\langle c_1^*, -, c_2, \texttt{index}^*, h_2\rangle$ in the list L_H.
– Compute $c_3^* = h_2 \oplus m_\beta$.
– Here the state information $st^* = \langle u^* = u, v^* = g^{bu}, s^* = -, \texttt{index}^*\rangle$

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

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

– $\mathcal{A}$ should not query the $\mathcal{O}_{Decryption}$ oracle with c^* as input.
– $\mathcal{A}$ continues to get oracle access to all the oracles. It can also get the encryption for any message including m_0 and m_1 for the state information st^* through the encryption oracle $\texttt{Encryption}(params, st^*, pk, m_j)$.

The simulation of the $\mathcal{O}_G$, $\mathcal{O}_H$, $\mathcal{O}_{Encryption}$ and $\mathcal{O}_{Decryption}$ oracles are not same as in Phase I and hence we provide the details below:

$\mathcal{O}_G(c_1 \in \mathbb{G}, w_1 \in \mathbb{G}, m \in \{0,1\}^{l_m}, \texttt{index} \in \{0,1\}^\mu)$**:** To respond to this query, $\mathcal{C}$ performs the following:

– If $c_1 \neq c_1^*$ then
  - If a tuple of the form $\langle c_1, w_1, m, \texttt{index}, h_1\rangle$ exists in the list L_G, return the corresponding h_1.
  - Else, choose $h_1 \in_R \{0,1\}^\lambda$, add the tuple $\langle c_1, w_1, m, \texttt{index}, h_1\rangle$ to the list L_G and return h_1 to $\mathcal{A}$.
– If $c_1 = c_1^*$ then
  - If a tuple of the form $\langle c_1, w_2, c_2, \texttt{index}, h_2\rangle$ exists in the list L_H, check whether $w_2 \stackrel{?}{=} c_1^{z_1}/w_1^{z_2}$. If the check holds then return $w_1^{u^{*-1}}$ as the solution to the CDH problem instance.
  - If a tuple of the form $\langle c_1, w_2, c_2, \texttt{index}, h_2\rangle$ does not exist in the list L_H, choose $h_1 \in_R \{0,1\}^\lambda$, add the tuple $\langle c_1, w_1, m, \texttt{index}, h_1\rangle$ to the list L_G and return h_1 to $\mathcal{A}$.

$\mathcal{O}_H(c_1 \in \mathbb{G}, w_2 \in \mathbb{G}, c_2 \in \{0,1\}^\lambda, \texttt{index} \in \{0,1\}^\mu)$**:** To respond to this query, $\mathcal{C}$ performs the following:

- If $c_1 \neq c_1^*$ then
  - If a tuple of the form $\langle c_1, w_2, c_2, \texttt{index}, h_2\rangle$ exists in the list L_H, return the corresponding h_2.
  - Else, choose $h_2 \in_R \{0,1\}^{l_m}$, add the tuple $\langle c_1, w_2, c_2, \texttt{index}, h_2\rangle$ to the list L_H and return h_2 to $\mathcal{A}$.
- If $c_1 = c_1^*$ then
  - If a tuple of the form $\langle c_1, w_1, m, \texttt{index}, h_1\rangle$ exists in the list L_G, check whether $w_2 \stackrel{?}{=} c_1^{z_1}/w_1^{z_2}$. If the check holds then return $w_1^{u^{*-1}}$ as the solution to the CDH problem instance.
  - If a tuple of the form $\langle c_1, w_1, m, \texttt{index}, h_2\rangle$ does not exist in the list L_G, choose $h_2 \in_R \{0,1\}^{l_m}$, add the tuple $\langle c_1, w_2, c_2, \texttt{index}, h_2\rangle$ to the list L_H and return h_2 to $\mathcal{A}$.

$\mathcal{O}_{Encryption}(st_i, m_j)$: $\mathcal{A}$ may perform encryption with respect to any state information st_i including st^*, chosen by $\mathcal{C}$. $\mathcal{C}$ performs the following to encrypt the message m_j with respect to the state information st_i:

- If $st_i \neq st^*$ then encryption is done as in Phase I
- If $st_i = st^*$ then perform the following:
  - Retrieve the tuple st^* of the form $st^* = \langle u^* = u, v^* = g^{bu}, s^* = -, \texttt{index}^*\rangle$ from L_{st} and set $c_1 = v^*$.
  - Choose $h_1 \in_R \{0,1\}^{\lambda}$, add the tuple $\langle c_1, -, m_j, \texttt{index}^*, h_1\rangle$ to the list L_G and compute $c_2 = h_1 \oplus u^*$.
  - Choose $h_2 \in_R \{0,1\}^{l_m}$, add the tuple $\langle c_1, -, c_2, \texttt{index}^*, h_2\rangle$ to the list L_H and compute $c_3 = h_2 \oplus m_j$.
  - Return $c = \langle c_1, c_2, c_3, \texttt{index}^*\rangle$ as the ciphertext, increment $\texttt{index}^*$ and update the state information st^*.

$\mathcal{O}_{Decryption}(c)$: In the case where $(c_1 \neq c_1^*)$, $\mathcal{C}$ responds as in phase I. If $(c_1 = c_1^*)$, $\mathcal{C}$ performs the following to decrypt the ciphertext $c = \langle c_1, c_2, c_3, \texttt{index}\rangle$:

- Retrieve the tuples of the form $\langle c_1, w_1, m, \texttt{index}, h_1\rangle$ from the list L_G. Consider that there are $\hat{n}_G$ such tuples. Choose the corresponding (w_{1i}, h_{1i}) values, for $i = 1$ to $\hat{n}_G$. (If the retrieved tuple is of the form $\langle c_1, -, m, \texttt{index}, h_1\rangle$ then it was the tuple generated by $\mathcal{C}$ during an encryption oracle query in phase II. Note that $\mathcal{C}$ can even work consistently with the tuple of this form without performing the test mentioned below. Further note that for a fixed c_1 and $\texttt{index}$, there will be only one such tuple in the list L_G.)
- Retrieve the tuples of the form $\langle c_1, w_2, c_2, \texttt{index}, h_2\rangle$ from the list L_H. Consider that there are $\hat{n}_H$ such tuples. Choose the corresponding (w_{2j}, h_{2j}) values, for $j = 1$ to $\hat{n}_H$. (Even in this case, if the retrieved tuple is of the form $\langle c_1, -, c_2, \texttt{index}, h_2\rangle$, the tuple was generated by $\mathcal{C}$ during an encryption oracle query in phase II. $\mathcal{C}$ can even work consistently with the tuple of this form without performing the test mentioned below. This is because for a fixed c_1, c_2 and $\texttt{index}$ there will be only one tuple of this form available in the list L_H.)
- For $i = 1$ to $\hat{n}_G$
  - For $j = 1$ to $\hat{n}_H$
    - Check whether $w_{2j} \stackrel{?}{=} c_1^{z_1}/w_{1i}^{z_2}$.
    - If the check holds for some index $\hat{i}$ and $\hat{j}$, choose the corresponding $h_{1\hat{i}}$ and $h_{2\hat{j}}$ and return $w_{1\hat{i}}^{u^{*-1}}$ as the solution to the CDH problem instance. If the check does not hold for any tuple then reject the ciphertext c.
- Compute $m' = c_3 \oplus h_{2\hat{j}}$.
- Retrieve the value m from the tuple of the form $\langle c_1, w_{1\hat{i}}, m, \texttt{index}, h_{1\hat{i}}\rangle$ from the list L_G.
- If $(m = m')$, then compute $u' = c_2 \oplus h_{1\hat{i}}$, else reject the ciphertext c.
- Check whether a tuple of the form $\langle k, u\rangle$, where $k = c_1^{u'^{-1}}$ and $u = u'$ appears in list L_F. If it appears accept m' and return it as the message corresponding to c.
- If any of the required tuples did not appear in the lists L_F, L_G or L_H return $\perp$.

Guess: At the end of ***Phase II***, $\mathcal{A}$ produces a bit β' to $\mathcal{C}$, but $\mathcal{C}$ ignores the response and performs the following to output the solution for the CDH problem instance.

- Retrieves the tuples of the form $\langle c_1^*, w_1, m, \texttt{index}\rangle$ from the list L_G and checks whether a tuple of the form $\langle c_1^*, w_2, c_2^*, \texttt{index}, h_2\rangle$ is available in list L_H. If a tuple of this form exists in the list L_H, $\mathcal{C}$ checks whether $w_2 \stackrel{?}{=} c_1^{z_1}/w_1^{z_2}$. If the check holds, compute $g' = w_1^{u^{*-1}}$ as the solution to the CDH problem.
- Alternatively, retrieves the tuples of the form $\langle c_1^*, w_2, c_2^*, \texttt{index}\rangle$ and checks whether a tuple of the form $\langle c_1^*, w_1, m_\beta, \texttt{index}, h_1\rangle$ is available in list L_G. If a tuple of this form exists in the list L_G, $\mathcal{C}$ checks whether $w_2 \stackrel{?}{=} c_1^{z_1}/w_1^{z_2}$. If the check holds, compute $g' = w_1^{u^{*-1}}$ as the solution to the CDH problem.

here

Scheme	Encryption Cost	Decryption Cost	Ciphertext Expansion	Assumption	Ciphertext Verifiability
PMO$_{sibe}$ [16]	$1H+1MAC$	$1B+1H+1MAC$	$\|\|\mathbb{G}\|\|+\|MAC\|+\|R\|$	GBDH	YES
YZM$_{sibe}$ [17]	$1H+1MAC$	$2BE+1H+1MAC$	$\|\|\mathbb{G}\|\|+\|MAC\|+\|R\|$	CBDH	YES
$\mathcal{SIBE}$ [4]	$2H$	$1B+2H$	$\|\|\mathbb{G}\|\|+\lambda$	GBDH	NO

Table - 1. Stateful Identity Based Encryption Schemes

here

Scheme	Encryption Cost	Decryption Cost	Ciphertext Expansion	Assumption	Ciphertext Verifiability
BKS$_{st}$ [5]	$1H+1MAC$	$1E+1H+1MAC$	$\|\|\mathbb{G}\|\|+\|MAC\|+\|R\|$	GDH	YES
BCZ$_{st}$ [4]	$2H$	$1E+2H$	$\|\|\mathbb{G}\|\|+\lambda$	GDH	NO
$\mathcal{N}-\mathcal{SPKE}_1$	$2H$	$2E+2H$	$\|\|\mathbb{G}\|\|+\lambda+\mu$	SDH	YES
$\mathcal{N}-\mathcal{SPKE}_2$	$2H$	$3E+2H$	$\|\|\mathbb{G}\|\|+\lambda+\mu$	CDH	YES

Table - 2. Stateful Public Key Encryption Schemes with Short Ciphertext

Correctness: Below, we show that the g' value obtained through the above steps is indeed g^{ab}.

- The public key pk of the target user is set to be $pk = \langle g, g_1 = g^a, g_2 = g^{z_1/g^{az_2}}\rangle$ by $\mathcal{C}$, where $z_1, z_2 \in \mathbb{Z}_q$ are known to $\mathcal{C}$. Therefore the private key $sk = x = a$ implicitly.
- $\mathcal{C}$ has set the c_1^* component of the challenge ciphertext c^* as g^{bu^*} (where, $u^* = F(g^b)$) during the challenge phase.
- In order to decrypt the ciphertext c^*, $\mathcal{A}$ should have computed the values

$$w_1 = (c_1^*)^x = (c_1^*)^a = (g^{bu^*})^a = g^{abu^*} \tag{7}$$

  and $w_2 = (c_1^*)^y = (c_1^*)^{z_1-xz_2} = (c_1^*)^{z_1-az_2} = (g^{bu^*})^{z_1-az_2} = (g^{bu^*z_1})(g^{-abu^*z_2}) = c_1^{*z_1}/w_1^{z_2}$. Therefore,

$$w_2 = c_1^{*z_1}/w_1^{z_2} \tag{8}$$

- $\mathcal{C}$ should have queried the $\mathcal{O}_G$ oracle with $(c_1^*, w_1, m_\delta, \texttt{index}^*)$ as input and $\mathcal{O}_H$ oracle with $(c_1^*, w_2, c_2^*, \texttt{index}^*)$ as input.
- From equation (8), it is clear that the check $w_2 \stackrel{?}{=} c_1^{z_1}/w_1^{z_2}$ holds good.
- $\mathcal{C}$ computes $g' = w_1^{u^{-1}} = (g^{abu^*})^{u^{*-1}} = g^{ab}$ and returns g' as the solution to the CDH problem.

Thus, $\mathcal{C}$ obtains the solution to the CDH problem with almost the same advantage of $\mathcal{A}$ in the IND-CCA game. ■

6 Comparison With Existing Schemes

In this section, we compare the $\mathcal{SIBE}$ scheme in [4] and new stateful public key encryption scheme ($\mathcal{N}-\mathcal{SPKE}_1$), proposed in section 4 with the existing schemes related to them respectively. In all the tables below, the legends are E - Exponentiation, B - Bilinear Pairing, H - Hash computation, $|\mathbb{G}|$ - Cardinality of the group $\mathbb{G}$, $||\mathbb{G}|| = log|\mathbb{G}|$ - The size of a single group element, MAC - Computation of a MAC value, $|MAC|$ - Length of a MAC value, $|R|$ - Size of a random string usually λ, $CBDH$ - Computational Bilinear Diffie Hellman Problem, $GBDH$ - Gap Bilinear Diffie Hellman Problem, GDH - Gap Diffie Hellman Problem and SDH - Strong Diffie Hellman Problem.

Table - 1 compares the stateful identity based encryption scheme $\mathcal{SIBE}$ (by Baek et al. [4]) with the schemes by Phong et al. (PMO$_{sibe}$ [16]) and Yang et al. (YZM$_{sibe}$ [17]). We consider the situation where a sender encrypts different messages to a fixed receiver during a state and this situation maximizes the functionality of stateful encryption schemes. Hence, we don't consider the computations done during the state initialization. YZM$_{sibe}$ scheme makes use of an IND-CCA secure symmetric key encryption scheme, which adds one MAC value and a random number to the ciphertext expansion of the scheme. From the table it is clear that Baek et al.'s $\mathcal{SIBE}$ offers compact ciphertext when compared to the other two schemes but does not offer ciphertext verifiability.

Table - 2 summarizes the computation complexity and ciphertext overhead of the stateful public key encryption schemes by Bellare et al. (BKS_{st} [5]), Baek et al. (BCZ_{st} [4]), $\mathcal{N}-\mathcal{SPKE}_1$ and $\mathcal{N}-\mathcal{SPKE}_2$. In this table, μ is the size of the index used in our scheme. As mentioned above, it is possible to append 80-bits of known value (usually 80-bits of 0's) to the plaintext while encrypting it and checking whether the decryption of the ciphertext produces those 80-bits at the end along with the message to ensure ciphertext verifiability. If this technique is used to ensure ciphertext verifiability, in the BCZ_{st} scheme, the ciphertext expansion will be $||\mathbb{G}|| + \lambda +$ '80−bits'. However, in the new schemes $\mathcal{N}-\mathcal{SPKE}_1$ and $\mathcal{N}-\mathcal{SPKE}_2$, the size of the `index` ($\mu$), is upper bound by 20-bits and hence can take a value starting from 1−bit, which is a considerable reduction for resource constrained devices like sensors, PDAs and mobile devices. The ciphertext overhead is also smaller than that of the BKS_{st} scheme, that offers ciphertext verifiability.

7 Conclusion

We have formally prove that the security of the stateful identity based encryption scheme by Baek et al. [4] does not reduce to the CBDH problem. We have shown that the challenger will confront the Y-Computational problem while providing the decryption oracle access to the adversary. We have provided a formal security proof for the same scheme, assuming the hardness of the Gap Bilinear Diffie Hellman (GBDH) problem. Two new stateful public key encryption schemes with ciphertext verifiability were proposed and the security of these schemes were supported by a formal proof. Our first stateful public key encryption scheme is proved to be secure assuming the SDH problem and the second assuming the CDH problem. However, the ciphertext overhead of both the schemes turns out to be the same. We have proved both the schemes in the random oracle model. An interesting open issue that can be looked at is designing a public key encryption scheme which offers compact ciphertext (ciphertext overhead of one group element) with ciphertext verifiability.

References

1. Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle diffie-hellman assumptions and an analysis of dhies. In *Topics in Cryptology - CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2001.
2. Masayuki Abe, Rosario Gennaro, Kaoru Kurosawa, and Victor Shoup. Tag-kem/dem: A new framework for hybrid encryption and a new analysis of kurosawa-desmedt kem. In *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 128–146. Springer, 2005.
3. Masayuki Abe, Eike Kiltz, and Tatsuaki Okamoto. Compact cca-secure encryption for messages of arbitrary length. In *Public Key Cryptography - PKC 2009*, volume 5443 of *Lecture Notes in Computer Science*, pages 377–392. Springer, 2009.
4. Joonsang Baek, Cheng-Kang Chu, and Jianying Zhou. On shortening ciphertexts: New constructions for compact public key and stateful encryption schemes. In *Topics in Cryptology - CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 302–318. Springer, 2011.
5. Mihir Bellare, Tadayoshi Kohno, and Victor Shoup. Stateful public-key cryptosystems: how to encrypt with one 160-bit exponentiation. In *ACM Conference on Computer and Communications Security*, pages 380–389. ACM, 2006.
6. Xavier Boyen. Miniature cca2 pk encryption: Tight security without redundancy. In *Advances in Cryptology - ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 485–501. Springer, 2007.
7. David Cash, Eike Kiltz, and Victor Shoup. The twin diffie-hellman problem and applications. *Journal of Cryptology*, Vol.22(No.4):470–504, 2009.
8. Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against chosen ciphertext attack. *SIAM Journal on Computing*, Vol-33(No-1):167–226, 2003.
9. Eiichiro Fujisaki and Tatsuaki Okamoto. How to enhance the security of public-key encryption at minimum cost. In *Public Key Cryptography, Second International Workshop on Practice and Theory in Public Key Cryptography*, volume 1560 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 1999.
10. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999.

11. Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 482–499. Springer, 2003.
12. Eike Kiltz and John Malone-Lee. A general construction of ind-cca2 secure public key encryption. In *Cryptography and Coding, 9th IMA International Conference*, volume 2898 of *Lecture Notes in Computer Science*, pages 152–166. Springer, 2003.
13. Kaoru Kurosawa and Toshihiko Matsuo. How to remove mac from dhies. In *Information Security and Privacy, ACISP - 2004*, volume 3108 of *Lecture Notes in Computer Science*, pages 236–247. Springer, 2004.
14. Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In *Public Key Cryptography, PKC - 2001*, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118. Springer, 2001.
15. Rafael Pass, Abhi Shelat, and Vinod Vaikuntanathan. Relations among notions of non-malleability for encryption. In *Advances in Cryptology - ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 519–535. Springer, 2007.
16. Le Trieu Phong, Hiroto Matsuoka, and Wakaha Ogata. Stateful identity-based encryption scheme: Faster encryption and decryption. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2008*, pages 381–388. ACM, 2008.
17. Peng Yang, Rui Zhang, and Kanta Matsuura. Stateful public key encryption: How to remove gap assumptions and maintaining tight reductions. In *International Symposium on Information Theory and Its Applications, 2008. ISITA 2008*, pages 1–6. IEEE, 2008.