Post-Quantum Anonymous One-Sided Authenticated Key Exchange without Random Oracles

Ren Ishibashi[1] and Kazuki Yoneyama[1]

Ibaraki University, 4-12-1, Nakanarusawacho, Hitachi-shi, Ibaraki 316-8511, Japan.
21nm706r@vc.ibaraki.ac.jp
kazuki.yoneyama.sec@vc.ibaraki.ac.jp

Abstract. Authenticated Key Exchange (AKE) is a cryptographic protocol to share a common session key among multiple parties. Usually, PKI-based AKE schemes are designed to guarantee secrecy of the session key and mutual authentication. However, in practice, there are many cases where mutual authentication is undesirable such as in anonymous networks like Tor and Riffle, or difficult to achieve due to the certificate management at the user level such as the Internet. Goldberg et al. formulated a model of anonymous one-sided AKE which guarantees the anonymity of the client by allowing only the client to authenticate the server, and proposed a concrete scheme. However, existing anonymous one-sided AKE schemes are only known to be secure in the random oracle model. In this paper, we propose generic constructions of anonymous one-sided AKE in the random oracle model and in the standard model, respectively. Our constructions allow us to construct the first post-quantum anonymous one-sided AKE scheme from isogenies in the standard model.

Keywords: authenticated key exchange · one-sided secure · anonymity · post-quantum · isogenies.

1 Introduction

Authenticated Key Exchange (AKE) is a cryptographic protocol to share a common session key among multiple parties through an unauthenticated channel such as the Internet. In ordinary PKI-based AKE, each party locally keeps its own static secret key (SSK) and issues a static public key (SPK) corresponding to the SSK. The validity of the SPK is guaranteed by a certificate issued by the certification authority. In a key exchange session, each party generates an ephemeral secret key (ESK) and sends an ephemeral public key (EPK) corresponding to the ESK to the other party. The session key is derived from these keys and the key derivation function. Ordinary AKE is intended for session key secrecy and mutual authentication, and provable security is formulated by security models such as CK model [8] and eCK model [34].

On the other hand, there are situations that the mutual authentication is undesirable such as anonymous networks like Tor [14] and Riffle [33]. In addition, in HTTPS transactions, it is common for an unauthenticated client to communicate with an authenticated server. In these cases, it is desirable for the client to be anonymous, and mutual authentication is not necessary. The ordinary security models of AKE cannot cover such one-sided authentication and anonymity.

Anonymous one-sided AKE (OS-AKE) is a cryptographic protocol that guarantees the anonymity of the client with one-sided authentication. In OS-AKE, there are a client and a server, and only the server locally keeps a SSK and publishes a certified SPK. In a key exchange session, both the client and the server generate ESK and EPK to share a common session key. Since the client does not have any static secret, OS-AKE is AKE without authentication to the client. Also, in OS-AKE, it is required that the client and the server can generate ESK/EPK offline (i.e., before starting a session). Goldberg et al. [24] formulated a security model for OS-AKE (GSU model). The GSU model captures the anonymity of clients and exposure resilience for non-trivial leakage patterns, and they proposed a concrete scheme satisfying their model.

One of the main objectives of this paper is to construct post-quantum OS-AKE because known OS-AKE schemes in the GSU model are not (fully) post-quantum.

1.1 Related Work

One-sided AKE. The notion of one-sided AKE has been studied in many kinds of literature. For example, to capture the security of SSL/TLS, various flavors of security models [13, 23, 29, 32, 31, 12, 38, 15] are introduced. In these models, the application to the setting of anonymous networks is not considered and the anonymity is not focused.

Anonymous AKE. The notion of anonymous AKE has been studied in contexts of the symmetric key (including password) setting [3, 1, 35, 44] or the group setting [42, 10]. These models cannot be simply applied to (asymmetric key-based client-server) one-sided AKE.

OS-AKE. There are three existing OS-AKE schemes secure in the GSU model or its variant: ntor [24] by Goldberg et al., Ace [6] by Backes et al., and HybridOR [22] by Ghosh and Kate. These schemes are based on Diffie-Hellman (DH) problems, and HybridOR is also based on lattices. There are three problems in these schemes. First, these schemes are proved in the random oracle model. Random oracles do not exist, and cannot always be instantiated by real hash functions. Indeed, Canetti et al. [7] show that there are primitives which are secure in the random oracle model but insecure if random oracles are instantiated by real hash functions. Second, Ace and HybridOR are not proved to be secure in the original GSU model. The security of these schemes is guaranteed under a

weaker freshness setting [6] than the original one. Finally, though these schemes use MAC for explicit authentication, implicit authentication is enough to satisfy the GSU model. Thus, removing such a MAC can make OS-AKE schemes more simple and efficient. For more details on the security of existing schemes, please see Section 4.

Isogeny-based AKE. Recently, many post-quantum AKE schemes are proposed from isogenies. Isogeny-based AKE schemes are classified into two settings: SIDH-based [26] and CSIDH-based [9]. There are several SIDH-based AKE schemes [19, 21, 36] from specific SIDH-related assumptions. Also, some generic constructions [17, 25, 43] of AKE can be instantiated from SIDH-based KEM. On the other hand, CSIDH-based AKE schemes [20, 28, 27] are also proposed. However, there is no known isogeny-based OS-AKE scheme.

1.2 Our Contribution

In this paper, we achieve the first post-quantum OS-AKE scheme without random oracles. Specifically, we propose a generic construction (GC-Std) for OS-AKE secure in the GSU model in the standard model from an IND-CCA secure KEM and an IND-CPA secure KEM with public-key-independent-ciphertext (PKIC-KEM) [45]. PKIC-KEM allows that a ciphertext can be generated independently from the public key, and a KEM session key can be generated with the ciphertext, the public key, and randomness in generating the ciphertext. By instantiating GC-Std with CSIDH-based KEM schemes, we can obtain CSIDH-based anonymous OS-AKE in the standard model. Moreover, we also propose a generic construction (GC-RO) for OS-AKE secure in the GSU model in the random oracle model from an OW-CCA secure KEM and an OW-CPA secure PKIC-KEM.

Compared with existing DH-based OS-AKE schemes [24, 6], an instantiation of GC-Std with DH-based KEM is secure in the standard model though existing schemes are secure in the random oracle model. For the DH-based instantiations, please see Section 6.

Also, the existing (partially) post-quantum OS-AKE scheme [22] is secure in the weaker model than the GSU model, and its post-quantum security is guaranteed only in a partial adversarial scenario. On the other hand, an instantiation of our generic constructions with isogeny-based KEM schemes guarantees the security in the original GSU model and is fully post-quantum for any adversarial scenario. For the isogeny-based instantiations, please see Section 7.

1.3 Key Technique

We start from the FSXY generic construction [18] of AKE (with the mutual authentication) from KEM (see Fig.3). Since a difference between AKE and OS-AKE is static keys for clients, it seems that the FSXY construction removing static keys for clients works as OS-AKE. However, there are several problems in

such a strategy. The eCK security model [34] and CK+ security model [30, 18] for AKE allow leakage of the ESK of the test session, and the TPRF trick [18] and the NAXOS trick [34] are known as techniques to guarantee security against such leakage. However, since the client does not have the SSK in OS-AKE such tricks cannot be used on the client-side. Hence, we need another solution to prove the security.

We focus on the definition of session freshness in the GSU model. Since the secrecy of the session key is trivially broken if all secret values of the client are revealed, the adversary cannot reveal at least a secret value of the client. Thus, if there is only one ESK used on the client-side, there is no need to consider leakage on the client-side. However, the FSXY construction uses two types of KEMs, and two randomnesses are necessary as ESKs. Our solution is to generate two types of randomness from an ESK with a pseudo-random function (PRF), and generate the ciphertext of each KEM from these output values of the PRF as randomness. Then, by erasing the two randomnesses used to generate the ciphertexts after sending the ciphertexts, the client only keeps a single ESK. Therefore, the number of secret values used in our scheme is one on the client-side (ESK) and two on the server-side (ESK, SSK), and thus we only need to consider the case where (1) the SSK on the server-side is revealed and (2) the ESK on the server-side is revealed.

There is another problem to be solved. In the GSU model, both the client and the server need to be able to generate EPKs offline (i.e., before starting a session). However, in the FSXY construction, the server cannot generate a ciphertext of KEM in advance because it depends on the session-specifically generated public key sent from the client. We solve this problem by using PKIC-KEM. Since, in PKIC-KEM, the ciphertext can be generated independently from the public key, the server can generate the ciphertext offline. Finally, for reducing the computational cost of the client, we reverse the procedures of the client and the server to generate such a ciphertext (i.e., the client generates the ciphertext of PKIC-KEM). For more details, please see Section 5.1.

2 Security Model for OS-AKE

In this section, we introduce the GSU security model [24] by Goldberg et al. Their model consists of the definitions of OS-AKE security and OS-anonymity, which cover the secrecy of session keys in one-sided authentication and the anonymity of clients, respectively.

As the notation, $x \in_R X$ denotes that the element x is sampled uniformly randomly from the set X.

2.1 System Model and Adversarial Capacity

Parties, key pairs, and certificates. Parties are modeled as probabilistic polynomial-time Turing machines. Each party is activated by receiving an initialization message and returns a message defined by the protocol.

A key pair that each party keeps is denoted in the form (x, X), where x is the secret value and X is the public value. The key pair includes, for example, the secret key and public key in public key cryptography, the ciphertext, and the randomness used for encryption. There are two types of key pairs: ephemeral key pairs that are used in a specific session and static key pairs that are used through all sessions.

Each server owns a certificate $cert_X = (ID_S, X)$ that combines a public value X and an identifier ID_S as SPK, and uses it for the server authentication. When a party owns the secret value x corresponding to the public value X, the party is said to be the owner of the public value X.

Protocol and sessions. Each execution of the protocol is called a session, and each session has a session identifier *sid* assigned to the party, where each *sid* must be unique within the party. Each session is associated with a session state containing intermediate values, and the session state of *sid* by party U_P is denoted by $M^P_{state}[sid]$. If a session *sid* is executed within a party, the party is called the owner of *sid*. Also, if the owner of a session completes the session by computing the session key *sk*, the session is called a completed session.

Session execution. When the session *sid* in which party U_P is the owner is completed, the ephemeral key pair (x, X) used in the session is deleted, and U_P outputs $\perp$ or $(sk, pid, \vec{v})$ as the output $M^P_{out}[sid]$ of the session, where *sk* is the session key in the keyspace $\mathcal{SK}$, *pid* is the peer's identifier or anonymous symbol "$\circledast$", Each vector $\vec{v_i}$ in $\vec{v} = (\vec{v_0}, \vec{v_1}, ...)$ is a vector of the public values of the static and the ephemeral keys used in the session. For example, $\vec{v_1}$ is a set of values consisting of the public values sent by party U_1. By including the public values used as part of the output, each session can be uniquely determined. If necessary, we use the notation $M^P_{out}[sid].sk$ to denote the session key of session *sid*. Other output values are denoted in the same way.

Adversary. Let *params* be a public parameter. The adversary $\mathcal{A}$ is modeled as a probabilistic polynomial-time Turing machine, which takes *params* as input and has oracle access to parties $P_1, \ldots, P_n$. $\mathcal{A}$ controls all communication between users including session activation. $\mathcal{A}$ can interfere in party U_P to execute a specific action using the following adversary's queries.

- $\mathsf{Send}(params, pid) \rightarrow (sid, msg)$: Let a party activate a session. The party activates a new session and returns a message according to the protocol. The input value *params* is defined by the protocol and includes the following. (1) the protocol to be executed, (2) the certificate used by the party to authenticate itself if the party is a server, (3) the certificate used by the peer *pid* in the session. The *pid* is the identifier of the intended peer establishing the session. When the session is intended to be with an unauthenticated anonymous peer, the *pid* is a special symbol "$\circledast$".

- $\mathsf{Send}(sid, msg) \to msg'$: The party executes the session sid with msg and returns the message msg' according to the protocol.
- $\mathsf{RevealNext} \to X$: $\mathcal{A}$ obtains a public value that is precomputed offline. The party generates a new key pair (x, X), records it as unused, and returns the public value X.
- $\mathsf{Partner}(X) \to x$: $\mathcal{A}$ obtains the secret value x corresponding to the public value X used in the session. If the key pair (x, X) is recorded in the party's memory, it returns the secret value x.
- $\mathsf{SessionKeyReveal}(sid) \to sk$: $\mathcal{A}$ obtains the session key of sid. It returns the session key $M_{out}^{P}[sid].sk$ of sid if the session is completed.

In addition, $\mathcal{A}$ can generate public keys and certificates using the following query.

- $\mathsf{EstablishCertificate}(ID_i, X)$: $\mathcal{A}$ registers a certificate containing the public value X of an unused identifier ID_i to all parties. $\mathcal{A}$ becomes the owner of the certificate as ID_i. If a party is registered by this query, we call the party *dishonest*, otherwise we call it *honest*.

Where necessary to avoid ambiguity, we use the superscript to indicate the party to whom the query is posed, such as $\mathsf{Send}^{P_i}(sid, msg)$.

Partnering. Unless a value X is the output of a Send query or the output of a RevealNext query to some party P_i, and the adversary $\mathcal{A}$ has not issued a $\mathsf{Partner}(X)$ query to P_i, then the adversary $\mathcal{A}$ is called a partner of X. If a party generates a key pair (x, X) by a query from $\mathcal{A}$ or by executing a session, we call the party a partner of X. Also, if different public values X and X' are corresponding to the same secret value x, then if $\mathcal{A}$ is a partner of X, then $\mathcal{A}$ is also regarded to be a partner of X'.

Correctness. If a two-party key exchange protocol Π satisfies the following conditions, Π is said to be *correct*.

- The adversary $\mathcal{A}$ relays all messages in the protocol running between the two parties without any modification.
- If a party is activated with a Send query with $pid \neq \circledast$, it will have the correct certificate for pid.
- Both parties output the same session key sk and the same vector $\vec{v}$.
- The value pid in the output of each party matches the pid in the Send query that was used to activate the party.

2.2 One-Sided AKE Security

For defining OS-AKE security, we need the notion of freshness.

Definition 1 (Freshness). *If the following conditions are satisfied, the session sid by party P_i is said to be OS-AKE fresh.*

1. *For each vector* $\vec{v_j}$ *in* $M_{out}^{P_i}$*, there is at least one public value* X *in* $\vec{v_j}$ *such that* $\mathcal{A}$ *is not a partner, where* $j \geq 1$*.*
2. $\mathcal{A}$ *does not issue a* SessionKeyReveal(sid) *query to any party* P_j*, where* P_j *is the owner of the certificate of* $M_{out}^{P_i}[sid].pid$ *such that it is* $M_{out}^{P_i}[sid].\vec{v} = M_{out}^{P_j}[sid].\vec{v}$*.*

The goal of the adversary $\mathcal{A}$ in the OS-AKE security game is to distinguish between the true session key and a random key. Initially, $\mathcal{A}$ is given a set of honest parties and makes any sequence of the queries described above. During the experiment, $\mathcal{A}$ makes the following query.

– Test$(i, sid^*) \rightarrow \mathcal{SK}$: Here, sid^* must be OS-AKE fresh. If $M_{out}^{P_i}[sid^*].sk = \perp$ or $M_{out}^{P_i}[sid^*].pid = \circledast$, an error symbol is returned. Otherwise, it chooses $b \in_R \{0, 1\}$. If $b = 0$, then it returns $M_{out}^{P_i}[sid^*].sk$. Otherwise, it returns a random element of $\mathcal{SK}$. This query can be issued only once.

Since OS-AKE provides the one-sided authentication, the test session sid^* is only for the session of the client-side that performs the authentication to the server. The adversary $\mathcal{A}$ obtains either the session key of sid^* or a random key with probability $1/2$ respectively. After issuing the Test query, the game continues until $\mathcal{A}$ outputs b' as a result of guessing whether the received key is random or not. If sid^* is OS-AKE fresh by the end and the guess of $\mathcal{A}$ is correct (i.e., $b = b'$), then it defines $\mathcal{A}$ wins the game.

Definition 2 (One-sided AKE security). *The advantage of the adversary* $\mathcal{A}$ *in the above game with the OS-AKE protocol* Π *is defined as follows.*

$$Adv_{\Pi,\kappa}^{OS-AKE}(\mathcal{A}) = \Pr[b = b'] - 1/2$$

Let κ *be a security parameter. For all probabilistic polynomial-time adversaries* $\mathcal{A}$*,* Π *is one-sided AKE-secure if* $Adv_{\Pi,\kappa}^{OS-AKE}$ *is negligible in* κ*.*

Remark 1. Due to the RevealNext query, this model requires the offline generation of ephemeral keys. Hence, the secret values may be stored in different locations for each generation. For example, a static key is stored in the database, an ephemeral key used for offline generation is stored in the storage, and another ephemeral key used for online generation is stored in the cache. In order to cover such a case, the leakage of each secret value is considered in OS-AKE fresh (Definition 1).

Remark 2. As described in the second condition of the Definition 1, the test session to be tested is the session in which *sid* matches between the two parties and the server's SSK can be revealed. Thus, the model captures weak forward secrecy which the adversary who does not modify the messages in the test session cannot break the security even if SSK is revealed. It also captures the adversarial arbitrary key registrant because of the EstablishCertificate query, which allows the adversary to establish a new party with registering arbitrary certified keys. Furthermore, it also captures the known-key security because of the SessionKeyReveal query, in which no information about the session key of the test session is revealed if other session keys are revealed.

2.3 One-sided Anonymity

The goal of the adversary $\mathcal{A}$ in the OS-anonymity game is to distinguish which of the two clients is participating in the session. Here, instead of $\mathcal{A}$ querying directly to the target party, the challenger $\mathcal{C}$ relays its communication. $\mathcal{A}$ gives the indices i_0 and i_1 of the parties it is the target to identify to $\mathcal{C}$. $\mathcal{C}$ chooses $i^* \in_R \{i_0, i_1\}$ randomly and relays the message between $\mathcal{A}$ and P_{i^*}. $\mathcal{A}$ guesses i^*.

In the game, in addition to the normal queries, $\mathcal{A}$ can issue the following special queries to $\mathcal{C}$. The first two queries are for the activation and the communication of the test session.

- $\mathsf{Start}^{\mathcal{C}}(i_0, i_1, params, pid) \to msg'$: If $i_0 = i_1$, an error symbol is returned. Otherwise, it sets $i^* \in_R \{i_0, i_1\}$, and it poses $\mathsf{Send}^{P_{i^*}}(params, pid) \to (sid^*, msg')$. Then it returns msg'. This query can be issued only once.
- $\mathsf{Send}^{\mathcal{C}}(msg) \to msg'$: It poses $\mathsf{Send}^{P_{i^*}}(sid^*, msg) \to msg'$ and returns msg'.

The other queries that $\mathcal{A}$ can query to $\mathcal{C}$ are to leak information about the test session sid^*.

- $\mathsf{RevealNext}^{\mathcal{C}} \to X$: It queries $\mathsf{RevealNext}^{P_{i^*}}$ and returns the public value, under restriction that the returned public value is not used in any session other than the test session, and the public value generated by the adversary's direct queries to $\mathsf{RevealNext}^{P_{i^*}}$ is not used in the test session.
- $\mathsf{SessionKeyReveal}^{\mathcal{C}}() \to sk$: It poses $\mathsf{SessionKeyReveal}^{P_{i^*}}(sid^*)$ and returns the session key sk.
- $\mathsf{Partner}^{\mathcal{C}} \to x$: It poses $\mathsf{Partner}^{P_{i^*}}(X)$ and returns the secret value x, where X is the value returned by the $\mathsf{Send}^{\mathcal{C}}$ query.

Definition 3 (One-sided anonymity). *Let κ be a security parameter and $n \geq 1$. For all probabilistic polynomial-time adversaries $\mathcal{A}$, the protocol Π is one-sided anonymous if the advantage $Adv^{OS-anon}_{\Pi,\kappa}(\mathcal{A}) = \Pr[i^* = i'] - 1/2$ of $\mathcal{A}$ wins the following game is negligible in κ.*

- $\mathsf{Expt}^{OS-anon}_{\Pi,\kappa,n}(\mathcal{A})$:
  - Initialize $params$ and parties $P_1, \ldots, P_n$.
  - Sets $i' \leftarrow \mathcal{A}^{P_1,\ldots,P_n,\mathcal{C}}(params)$.
  - Suppose that $\mathcal{A}$ poses a $\mathsf{Start}^{\mathcal{C}}(i_0, i_1, params, pid)$ query and the challenger $\mathcal{C}$ chooses i^*. If $i^* = i'$ and the query of $\mathcal{A}$ satisfies the following restrictions, then $\mathcal{A}$ wins the game.
    - There is no $\mathsf{SessionKeyReveal}(sid^*)$ query to P_{i_0} nor P_{i_1}.
    - There is no $\mathsf{Partner}(X)$ query to P_{i_0} nor P_{i_1} for any public value X returned by $\mathcal{C}$.
    - There is no $\mathsf{Send}(sid^*, \cdot)$ query to P_{i_0} nor P_{i_1}.
    - Both P_{i_0} and P_{i_1} had the same certificate for pid during the run of the protocol for sid^*.

The restrictions in definition 3 are to prevent $\mathcal{A}$ from knowing P_{i^*} trivially by obtaining information about the test session. For example, if $i^* = i_0$, then the $\mathsf{SessionKeyReveal}^{P_{i_0}}(sid^*)$ query returns the true session key, and the $\mathsf{SessionKeyReveal}^{P_{i_1}}(sid^*)$ query returns $\perp$ because P_{i_1} is not participating in sid^*. Therefore, $\mathcal{A}$ can determine $i^* = i_0$ trivially. Thus, the main restrictions in the OS-anonymity game are that queries for P_{i_0} and P_{i_1} must be posed through the challenger $\mathcal{C}$, and the public values used in the test session must not be used in any other session.

3 Building Blocks

3.1 Key Encapsulation Mechanism (KEM)

In this section, we show the definition of KEM.

Definition 4 (KEM). *KEM consist of algorithms* $(\mathsf{KeyGen}, \mathsf{EnCap}, \mathsf{DeCap})$ *as follows.*

- $(ek, dk) \leftarrow \mathsf{KeyGen}(1^\kappa; r_g)$ *: The key generation algorithm takes* 1^κ *and* $r_g \in \mathcal{RS}_\mathcal{G}$ *as input and outputs a key pair of public and secret key* (ek, dk)*, where* κ *is a security parameter and* $\mathcal{RS}_\mathcal{G}$ *is the randomness space of the key generation algorithm.*
- $(K, C) \leftarrow \mathsf{EnCap}(ek; r_e)$ *: The encapsulation algorithm takes the public key* ek *and* $r_e \in \mathcal{RS}_\mathcal{E}$ *as input and outputs the session key* $K \in \mathcal{KS}$ *and the ciphertext* $C \in \mathcal{CS}$*, where* $\mathcal{RS}_\mathcal{E}$ *is the randomness space of the encapsulation algorithm,* $\mathcal{KS}$ *is the session key space, and* $\mathcal{CS}$ *is the ciphertext space.*
- $K \leftarrow \mathsf{DeCap}(dk, C)$ *: The decapsulation algorithm takes the secret key* dk *and the ciphertext* $C \in \mathcal{CS}$ *as input and outputs the session key* $K \in \mathcal{KS}$*.*

Here, for any $\kappa \in \mathbb{N}$, any public and secret key $(ek, dk) \leftarrow \mathsf{KeyGen}(1^\kappa; r_g)$, and any session key and ciphertext $(K, C) \leftarrow \mathsf{EnCap}(ek; r_e)$, it is satisfied that $K \leftarrow \mathsf{DeCap}(dk, C)$.

The definition of security for KEM is as follows.

Definition 5 (IND-CCA security for KEM). *For any probabilistic polynomial time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the KEM scheme is IND-CCA secure if the advantage* $Adv^{ind-cca}_{KEM,\kappa} = |\Pr[(ek, dk) \leftarrow \mathsf{KeyGen}(1^\kappa; r_g); state \leftarrow \mathcal{A}_1^\mathcal{O}(ek); b \leftarrow_R \{0,1\}; (K_0^*, C_0^*) \leftarrow \mathsf{EnCap}(ek; r_e); K_1^* \in_R \mathcal{KS}; b' \leftarrow \mathcal{A}_2^\mathcal{O}(ek, (K_b^*, C_0^*), state); b' = b] - 1/2|$ *is negligible in* κ*, where* $\mathcal{O}$ *is the decryption oracle.*

Definition 6 (OW-CCA security for KEM). *For any probabilistic polynomial time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the KEM scheme is OW-CCA secure if the advantage* $Adv^{ow-cca}_{KEM,\kappa} = |\Pr[(ek, dk) \leftarrow \mathsf{KeyGen}(1^\kappa; r_g); state \leftarrow \mathcal{A}_1^\mathcal{O}(ek); (K^*, C^*) \leftarrow \mathsf{EnCap}(ek; r_e); K'^* \leftarrow \mathcal{A}_2^\mathcal{O}(ek, C^*, state); K'^* = K^*]|$ *is negligible in* κ*, where* $\mathcal{O}$ *is the decryption oracle.*

A KEM scheme is a κ-min-entropy KEM if for any secret key, the distribution $\mathcal{D}_{\mathcal{KS}}$ for K defined by $(K, C) \leftarrow \mathsf{EnCap}(ek; r_e)$, the distribution $\mathcal{D}_{pub}$ for public information, and a randomness $r_e \in \mathcal{RS}_\mathcal{E}$, it holds that $H_\infty(\mathcal{D}_{\mathcal{KS}}|\mathcal{D}_{pub}) \geq \kappa$. Here, H_∞ denotes the min-entropy function.

3.2 PKIC-KEM

In this section, we show the definition of PKIC-KEM [45] that can generate the ciphertext independently of the public key.

Definition 7 (PKIC-KEM). *PKIC-KEM consist of algorithms* $(\mathsf{wKeyGen}, \mathsf{wEnCapC}, \mathsf{wEnCapK}, \mathsf{wDeCap})$ *as follows.*

- $(ek, dk) \leftarrow \mathsf{wKeyGen}(1^\kappa; r_g)$ *: The key generation algorithm takes* 1^κ *and* $r_g \in \mathcal{RS}_\mathcal{G}$ *as input and outputs a key pair of public and secret key* (ek, dk)*, where* κ *is a security parameter and* $\mathcal{RS}_\mathcal{G}$ *is the randomness space of the key generation algorithm.*
- $C \leftarrow \mathsf{wEnCapC}(r_e)$ *: The ciphertext generation algorithm takes* $r_e \in \mathcal{RS}_\mathcal{E}$ *as input and outputs a ciphertext* $C \in \mathcal{CS}$*, where* $\mathcal{RS}_\mathcal{E}$ *is the randomness space of the encapsulation algorithm and* $\mathcal{CS}$ *is the ciphertext space.*
- $K \leftarrow \mathsf{wEnCapK}(ek, C, r_e)$ *: The encapsulation algorithm takes the public key* ek*, the ciphertext* $C \in \mathcal{CS}$ *and a randomness* $r_e \in \mathcal{RS}_\mathcal{E}$ *as input, and outputs the session key* $K \in \mathcal{KS}$*, where* $\mathcal{KS}$ *is the session key space.*
- $K \leftarrow \mathsf{DeCap}(dk, C)$*: The decapsulation algorithm takes the secret key* dk *and the ciphertext* $C \in \mathcal{CS}$ *as input and outputs the session key* $K \in \mathcal{KS}$*.*

For any $\kappa \in \mathbb{N}$, any public and secret key $(ek, dk) \leftarrow \mathsf{wKeyGen}(1^\kappa; r_g)$, and any ciphertext $C \leftarrow \mathsf{wEnCapC}(r_e)$, it is satisfied that $K \leftarrow \mathsf{wEnCapK}(ek, C, r_e)$ and $K \leftarrow \mathsf{wDeCap}(dk, C)$.

The definition of security for PKIC-KEM is as follows.

Definition 8 (IND-CPA security for PKIC-KEM). *For any probabilistic polynomial time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the PKIC-KEM scheme is IND-CPA secure if the advantage* $Adv^{ind-cpa}_{PKIC-KEM,\kappa} = |\Pr[(ek, dk) \leftarrow \mathsf{wKeyGen}(1^\kappa; r_g); state \leftarrow \mathcal{A}_1(ek); b \leftarrow_R \{0,1\}; C^*_0 \leftarrow \mathsf{wEnCapC}(r_e); K^*_0 \leftarrow \mathsf{wEnCapK}(ek, C^*_0, r_e); K^*_1 \in_R \mathcal{KS}; b' \leftarrow \mathcal{A}_2(ek, (K^*_b, C^*_0), state); b' = b] - 1/2|$ *is negligible in* κ*.*

Definition 9 (OW-CPA security for PKIC-KEM). *For any probabilistic polynomial time adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the PKIC-KEM scheme is OW-CPA secure if the advantage* $Adv^{ow-cpa}_{PKIC-KEM,\kappa} = |\Pr[(ek, dk) \leftarrow \mathsf{wKeyGen}(1^\kappa; r_g); state \leftarrow \mathcal{A}_1(ek); C^* \leftarrow \mathsf{wEnCapC}(r_e); K^* \leftarrow \mathsf{wEnCapK}(ek, C^*, r_e); K'^* \leftarrow \mathcal{A}_2(ek, C^*, state); K'^* = K^*]|$ *is negligible in* κ*.*

Also, the κ-min-entropy of PKIC-KEM can be defined in the same way as KEM.

3.3 Pseudo-Random Function

We show the definition of Pseudo-Random Function (PRF). Let κ be a security parameter and $\mathbf{F} = \{\mathsf{F}_\kappa : Dom_\kappa \times \mathcal{FS}_\kappa \to Rng_\kappa\}_\kappa$ be a function family with a family of domains $\{Dom_\kappa\}_\kappa$, a family of key spaces $\{\mathcal{FS}_\kappa\}_\kappa$ and a family of ranges $\{Rng_\kappa\}_\kappa$.

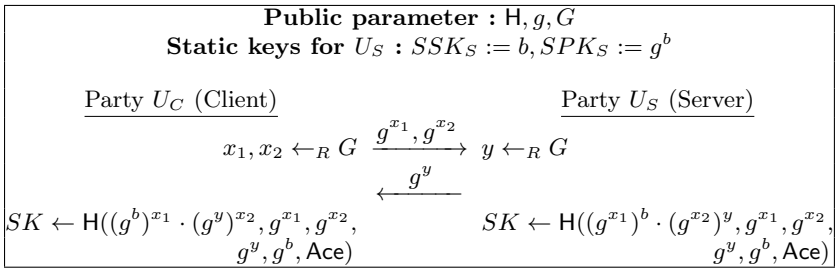

Fig. 1. Overview of Ace

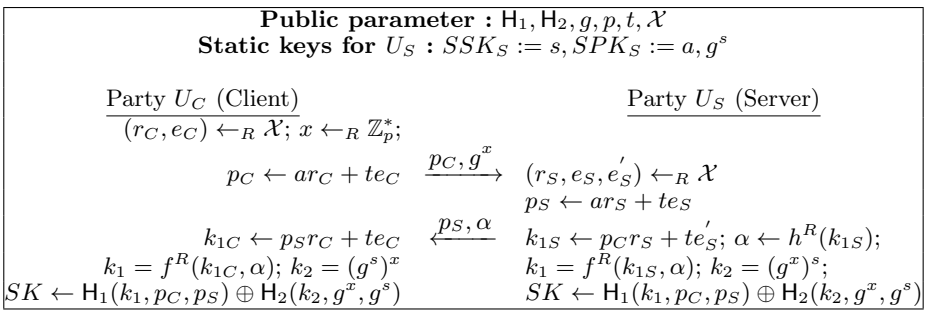

Fig. 2. Overview of HybridOR

Definition 10 (Pseudo-Random Function). *We say that function family* $\mathbf{F} = \{\mathsf{F}_\kappa\}_\kappa$ *is a PRF family if for any probabilistic polynomial time distinguisher* $\mathcal{D}$, $Adv^{PRF} = |\Pr[1 \leftarrow \mathcal{D}^{\mathsf{F}_\kappa(\cdot,k)}] - \Pr[1 \leftarrow \mathcal{D}^{\mathsf{RF}_\kappa(\cdot)}]| \leq negl$, *where* $\mathsf{RF}_\kappa : Dom_\kappa \rightarrow Rng_\kappa$ *is a truly random function.*

3.4 Key-Derivation Function

Let κ be a security parameter and $\mathsf{KDF} : Salt \times Dom \rightarrow Rng$ be a function with finite domain Dom, finite range Rng, and a space of non-secret random salt $Salt$.

Definition 11 (Key-Derivation Function). *We say that function* KDF *is a KDF if the following condition holds for a security parameter* κ. *For any probabilistic polynomial time adversary* $\mathcal{A}$ *and any distribution* $\mathcal{D}_{Dom}$ *over Dom with* $H_\infty(\mathcal{D}_{Dom}) \geq \kappa$, $|\Pr[y \in_R Rng, s \in_R Salt; 1 \leftarrow \mathcal{A}(s, y)] - \Pr[x \in_R Dom; s \in_R Salt; y \leftarrow \mathsf{KDF}(s, x); 1 \leftarrow \mathcal{A}(s, y)]| \leq negl$.

4 Security of Ace and HybridOR in GSU Model

In this section, we revisit the security of existing OS-AKE schemes. While ntor [24] is proved in the GSU model, other two schemes Ace [6] and HybridOR [22] are proved in an weaker model. Specifically, the security of Ace and

HybridOR are proved under a weaker freshness setting [6] than the original one. The weak freshness is called the double value freshness, and it requires that if the client and the server have two secret values (I_1, I_2) and (J_1, J_2) respectively, then the adversary cannot reveal (I_1, J_2) or (I_2, J_1). In the OS-AKE freshness in the GSU model, the adversary is allowed to reveal such secret values. Hence, the model that Ace and HybridOR are proved to be weaker than the GSU model. Here, we show the definition of the double value freshness.

Definition 12 (Double value freshness [6]). *We say that a session is double value OS-AKE fresh if it is OS-AKE fresh and the following condition does not hold.*

If $\vec{v_i}$ is (I_1, I_2) and $\vec{v_j}$ is (J_1, J_2), $\mathcal{A}$ is not a partner of (I_1, J_2) nor (I_2, J_1).

We show that Ace is not secure in the GSU model. An overview of Ace is shown in Fig. 1, where G is the exponent group and H is a random oracle. It uses two ESKs x_1 and x_2 on the client-side, and a SSK b and an ESK y on the server-side. By the OS-AKE freshness definition of the GSU model, the adversary can reveal (x_2, b) or (x_1, y). For example, If (x_2, b) is revealed, the adversary can compute the session key as follows.

1. Obtain (x_2, b) by Partner queries.
2. Obtain the EPKs (g^{x_1}, g^{x_2}, g^y) from the communication channel.
3. Compute the session key $SK \leftarrow \mathsf{H}((g^{x_1})^b \cdot (g^y)^{x_2}, g^{x_1}, g^{x_2}, g^y, g^b, \mathsf{Ace})$.

Next, we show that HybridOR is not secure in the GSU model. An overview of HybridOR is shown in Fig. 2, where $f^R(\cdot)$ is a robust extractor, $h^R(\cdot)$ is a randomized algorithm used to generate the signal value α, $\mathcal{X}$ is the error distribution of the ring-LWE problem, and H_1 and H_2 are random oracles. It uses two ESKs (r_C, e_C) and x on the client side, and a SSK s and an ESK (e_S, e'_S) on the server side. By the OS-AKE freshness definition of the GSU model, the adversary can reveal $((r_C, e_C), s)$ or $(x, (r_S, e'_S))$. For example, If $((r_C, e_C), s)$ is revealed, the adversary can compute the session key as follows.

1. Obtain $((r_C, e_C), s)$ by Partner queries..
2. Obtain the EPKs (g^x, p_C, p_S, α) from the communication channel.
3. Compute the session key as follows.
   (a) $k_{1C} \leftarrow p_S r_C + t e_C$
   (b) $k_1 = f^R(k_{1C}, \alpha)$
   (c) $k_2 = (g^x)^s$
   (d) $SK \leftarrow \mathsf{H}_1(k_1, p_C, p_S) \oplus \mathsf{H}_2(k_2, g^x, g^s)$

Therefore, Ace and HybridOR are insecure in the GSU model.

Remark 3. By applying our technique of using single randomness to produce two randomnesses via PRFs to these schemes, we can obtain secure schemes in the GSU model.

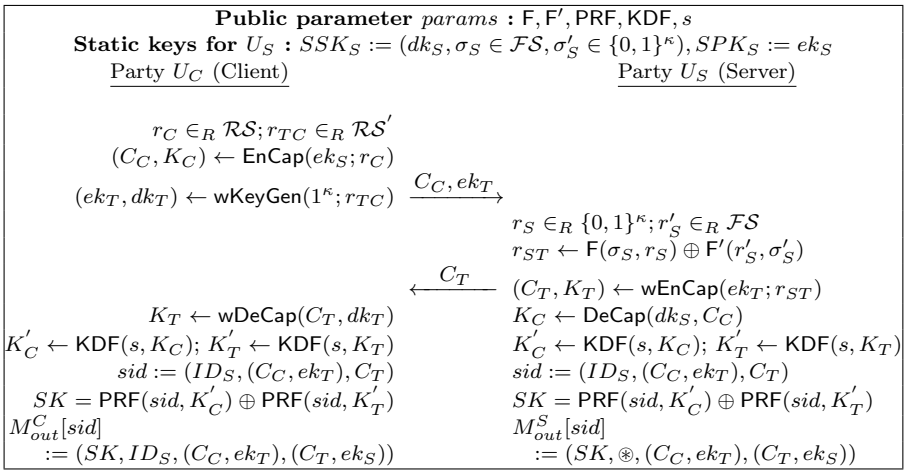

Fig. 3. FSXY-based OS-AKE scheme

5 Our Generic Constructions

In this section, we propose two generic constructions of OS-AKE from KEM in the standard model (GC-Std) and the random oracle model (GC-RO). GC-Std is based on IND-CCA secure KEM and IND-CPA secure PKIC-KEM, and GC-RO is based on OW-CCA secure KEM and OW-CPA secure PKIC-KEM. Our constructions are secure in the GSU model. The protocols of GC-Std and GC-RO are shown in Fig. 4 and 5, respectively.

5.1 Construction Idea

As discussed in Section 1.3, our generic construction are based on the FSXY construction [18] which is CK+ secure AKE scheme. Since a client does not have any static keys in OS-AKE, we show a naive FSXY-based OS-AKE protocol in Fig. 3 by simply removing static keys and related computations of the client, where $(\mathsf{KeyGen}, \mathsf{EnCap}, \mathsf{DeCap})$ is an IND-CCA secure KEM and $(\mathsf{wKeyGen}, \mathsf{wEnCap}, \mathsf{wDeCap})$ is an IND-CPA secure KEM. The CK+ security model allows leakage of the ephemeral key of the test session, and the TPRF trick is used to guarantee security against such a leakage such that $r_{ST} \leftarrow \mathsf{F}(r_S, \sigma_S) \oplus \mathsf{F}'(\sigma'_S, r'_S)$. Naturally, OS-AKE provides one-sided authentication, and clients that need to guarantee anonymity cannot have static key pairs, and thus the TPRF trick is not available on the client-side. Furthermore, though all ESKs (r_C, r_{TC}) are revealed at once by a query to the client in the CK+ model and the freshness definition prohibits leakage of ESKs if there is no SSK, the $\mathsf{Partner}$ query in the GSU model reveals the secret value x for the public value X and the OS-AKE freshness definition allows leakage of one of ESKs. For example, the session key can be computed if the adversary reveals an ESK r_{TC} of the client and the SSK dk_S of the server (such a leakage is allowed in the

GSU model) because the adversary can compute dk_T from r_{TC} and then can decrypt both C_C and C_T. Therefore, it is not trivial to construct an OS-AKE secure scheme from the FSXY construction.

Also, in the GSU model, for $\mathsf{RevealNext}$ queries, the ephemeral keys used by both parties in each session must be able to be generated offline in advance. On the other hand, in the FSXY-based construction, the server needs to generate the EPK after receiving the client's message, and thus the IND-CPA secure KEM is not sufficient for OS-AKE.

For the problem of leakage on the client-side, we propose a technique such that two types of randomness are generated from a single ESK. According to the definition of OS-AKE freshness, if there is only one ESK used on the client-side, there is no need to consider leakage on the client-side. However, the FSXY-based construction requires the generation of ciphertext of a session-specific public key of IND-CCA KEM and IND-CPA KEM at the client-side, thus two types of randomness are required. We generate two types of randomness from a single ESK through a PRF, and generate the ciphertext of each KEM from these randomnesses. Concretely, we construct the PRF F to obtain two outputs from one randomness by using a PRF F' and two PRFs F_0', F_1' having each range is each randomness space of KEMs. Then, two randomness $(r_0||r_1) \leftarrow \mathsf{F}(ID_S, r)$ is computed as $(r_0||r_1) = (\mathsf{F}_0'(ID_S, \mathsf{F}'(0,r))||\mathsf{F}_1'(ID_S, \mathsf{F}'(1,r)))$. In this way, two types of randomness are generated from one randomness. Here, if only F' is used in this technique, the OS-AKE security cannot be reduced to the CCA security or the CPA security. For example, in a game of the reduction to the CCA security, $r_1'^*$ is masked first, but the simulator needs to simultaneously input the correct value of r_C^* into F' to generate r_0'. This case cannot be simulated correctly because the simulator does not have r_C^*. Therefore, the output of F' is passed through F_0' and F_1' to be enabled for these reductions. We prove that our constructions are still secure under such a randomness generation in Section 5.2. Then, by erasing the two randomnesses used to generate the ciphertext and the session-specific public key after sending the client's message, the target of the $\mathsf{Partner}$ query can be one ESK that was generated first. Therefore, the number of secret values can be one on the client-side (ESK) and two on the server-side (ESK, SSK).

Next, for the problem of the offline generation of EPKs, we use an IND-CPA secure PKIC-KEM instead of IND-CPA secure KEM. Since the PKIC-KEM can generate ciphertexts independently of the public key, it is possible to generate the EPK for each session before starting the session. Specifically, the server can generate C_T before starting the session by using $\mathsf{wEnCapC}$ algorithm of PKIC-KEM.

Finally, we reverse the procedures of the client and the server to generate the public key ek_T and the ciphertext C_T of PKIC-KEM. If the client generates ek_T and the server generates C_T as the FSXY construction, the client must compute $\mathsf{wKeyGen}$ again before decrypting C_T because the client must erase dk_T after sending the client's message. Since the computational cost for the client is increased by $\mathsf{wKeyGen}$, and it is not efficient, we reverse the procedures.

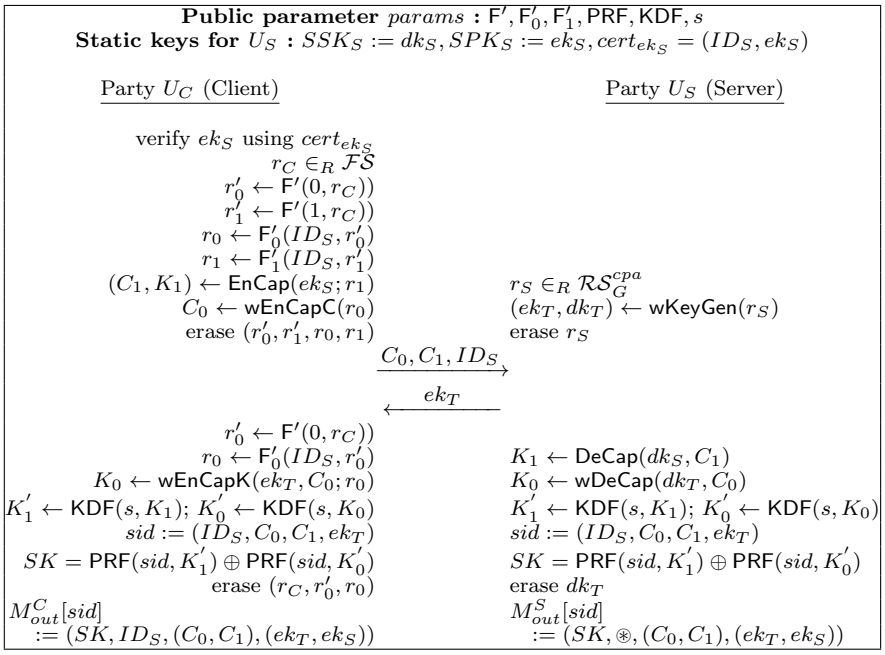

Fig. 4. Generic construction in the standard model (GC-Std)

If the client generates C_T and the server generates ek_T, then the client does not need to compute $\mathsf{wKeyGen}$ again.

In the proof of the proposed construction, by the definition of freshness, the ESK on the client-side is not revealed, and thus we need to consider the case where (1) the SSK on the server-side is revealed and (2) the ESK on the server-side is revealed. In (1), since the ESK at the server-side is not compromised, the adversary cannot compute K_0 which is the session key of the IND-CPA secure PKIC-KEM. Similarly, in (2), since the SSK at the server-side is not compromised, the adversary cannot compute K_1 which is the session key of the IND-CCA secure KEM. Thus, the proposed construction satisfies the OS-AKE security. Moreover, since the ESK used by the client-side in each session is only one randomness independent to the client's ID, no information about the client can be obtained from the ciphertext. Hence, the proposed construction satisfies the OS-anonymity.

5.2 OS-AKE in Standard Model

The protocol in the standard model consists of an IND-CCA secure KEM ($\mathsf{KeyGen}, \mathsf{EnCap}, \mathsf{DeCap}$) and an IND-CPA secure PKIC-KEM ($\mathsf{wKeyGen}$, $\mathsf{wEnCapC}, \mathsf{wEnCapK}, \mathsf{wDeCap}$) as follows.

Protocol.

Public Parameters : Let κ be a security parameter, $\mathsf{F}^{'} : \{0,1\}^{\kappa} \times \mathcal{FS} \to \mathcal{FS}$, $\mathsf{F}_0^{'} : \{0,1\}^{\kappa} \times \mathcal{FS} \to \mathcal{RS}_E^{cpa}$, $\mathsf{F}_1^{'} : \{0,1\}^{\kappa} \times \mathcal{FS} \to \mathcal{RS}_E^{cca}$, and $\mathsf{PRF} : \{0,1\}^{*} \times \mathcal{FS} \to \{0,1\}^{\kappa}$ be pseudo-random functions. Also, let $\mathsf{KDF} : \mathit{Salt} \times \mathcal{KS} \to \mathcal{FS}$ be a key derivation function and it chooses $s \in_R \mathit{Salt}$, where $\mathcal{RS}_E^{cpa}$ and $\mathcal{RS}_G^{cpa}$ are randomness spaces of the encapsulation algorithm and the key generation algorithm of IND-CPA secure PKIC-KEM, $\mathcal{RS}_E^{cca}$ and $\mathcal{RS}_G^{cca}$ are randomness spaces of the encapsulation algorithm and the key generation algorithm of IND-CCA secure KEM, $\mathcal{FS}$ is a key space of the pseudo-random functions ($|\mathcal{FS}| = \kappa$), $\mathcal{KS}$ is a session key space of KEM, and Salt is a salt space of the key derivation functions. These are provided as part of the public parameters.

Secret and Public Keys : Party U_S selects a randomness $r \in_R \mathcal{RS}_G^{cca}$, computes $(ek_S, dk_S) \leftarrow \mathsf{KeyGen}(1^{\kappa}; r)$ and sets $\mathit{cert}_{ek_S} = (ID_S, ek_S)$ as a certificate for U_S. The static key pair for party U_S is (ek_S, dk_S).

Key Exchange : Let U_S which has a static key pair (ek_S, dk_S) be a server, and U_C be a client. When U_C is initialized as a client, it obtains the certificate $\mathit{cert}_{ek_S} = (ID_S, ek_S)$ of U_S.

1. U_C verifies the server using $\mathit{cert}_{ek_S} = (ID_S, ek_S)$. U_C chooses an unused ephemeral key pair $((C_0, C_1), r_C)$ or chooses a ephemeral secret key $r_C \in_R \mathcal{FS}$ and sets $r_0' \leftarrow \mathsf{F}'(0, r_C))$, $r_1' \leftarrow \mathsf{F}'(1, r_C))$, $r_0 \leftarrow \mathsf{F}_0'(ID_S, r_0')$, and $r_1 \leftarrow \mathsf{F}_1'(ID_S, r_1')$. Also, U_C computes $(C_1, K_1) \leftarrow \mathsf{EnCap}(ek_S; r_1)$, $C_0 \leftarrow \mathsf{wEnCapC}(r_0)$, and erases (r_0, r_1). Then, U_C sends (C_0, C_1, ID_S) to U_S.
2. Upon receiving (C_0, C_1, ID_S), U_S chooses an unused ephemeral key pair (ek_T, dk_T) or chooses a randomness $r_S \in_R \mathcal{RS}_G^{cpa}$ and computes $(ek_T, dk_T) \leftarrow \mathsf{wKeyGen}(r_S)$ to generate a key pair, and sends ek_T to U_C. Also, U_S computes $K_1 \leftarrow \mathsf{DeCap}(dk_S, C_1)$, $K_0 \leftarrow \mathsf{wDeCap}(dk_T, C_0)$, $K_1^{'} \leftarrow \mathsf{KDF}(s, K_1)$, and $K_0^{'} \leftarrow \mathsf{KDF}(s, K_0)$. U_S sets $\mathit{sid} = (ID_S, C_0, C_1, ek_T)$ and computes the session key $SK = \mathsf{PRF}(\mathit{sid}, K_1^{'}) \oplus \mathsf{PRF}(\mathit{sid}, K_0^{'})$. Then, U_S erases (r_S, dk_T) and outputs $(SK, \circledast, (C_0, C_1), (ek_T, ek_S))$.
3. Upon receiving ek_T, U_C sets $r_0' \leftarrow \mathsf{F}'(0, r_C))$, $r_0 \leftarrow \mathsf{F}_0'(ID_S, r_0')$, computes $K_0 \leftarrow \mathsf{wEnCapK}(ek_T, C_0, r_0)$, $K_1^{'} \leftarrow \mathsf{KDF}(s, K_1)$, and $K_0^{'} \leftarrow \mathsf{KDF}(s, K_0)$. U_C sets $\mathit{sid} = (ID_S, C_0, C_1, ek_T)$ and computes the session key $SK = \mathsf{PRF}(\mathit{sid}, K_1^{'}) \oplus \mathsf{PRF}(\mathit{sid}, K_0^{'})$. Then, U_C erases (r_C, r_0, r_1) and outputs $(SK, ID_S, (C_0, C_1), (ek_T, ek_S))$.

Remark 4. Existing OS-AKE schemes contain the explicit authentication of the server with the key confirmation by MAC. As discussed in Section 1.1, implicit authentication is sufficient to satisfy the security in the GSU model. It is trivial to be able to add the explicit authentication to our construction by the same key confirmation step.

Security. We show the security of the proposed scheme in the standard model. An intuition of the proof is shown in Section 5.1.

Theorem 1. *If* $(\mathsf{KeyGen}, \mathsf{EnCap}, \mathsf{DeCap})$ *is an IND-CCA secure and* κ*-min-entropy KEM,* $(\mathsf{wKeyGen}, \mathsf{wEnCapC}, \mathsf{wEnCapK}, \mathsf{wDeCap})$ *is an IND-CPA secure and* κ*-min-entropy PKIC-KEM,* F'*,* F'_0*,* F'_1*, and* PRF *are pseudo-random functions, and* KDF *is a key derivation function, GC-Std is OS-AKE secure.*

Proof. *Suc* denotes the event that $\mathcal{A}$ wins. We consider the following events that cover all cases of the behavior of $\mathcal{A}$.

-E_1**:** The ESK dk_T^* of the server is revealed.
-E_2**:** The SSK dk_S^* of the server is revealed.

Let κ be a security parameter. In the OS-AKE security game, sid^* is a session ID of the test session, and the maximum number of parties is n and the maximum ℓ sessions are activated. Let the adversary $\mathcal{A}$ be a probabilistic polynomial-time adversary in κ, and construct the IND-CCA or IND-CPA adversary $\mathcal{S}$ and a distinguisher $\mathcal{D}$ from $\mathcal{A}$ that performs the OS-AKE game.

To finish the proof, we investigate events $E_i \wedge Suc$ $(i = 1, 2)$ that cover all cases of event *Suc*. Due to the page limitation, we give the proof of event $E_1 \wedge Suc$, and the proof of the other event is given in Appendix A.

Event $E_1 \wedge Suc$ *:* We change the interface of oracle queries and the computation of the session key. These instances are gradually changed over eight hybrid experiments, depending on specific subcases. In the last hybrid experiment, the session key in the test session does not contain information of the bit b. Thus, the adversary clearly only outputs a random guess. We denote these hybrid experiments by $\mathbf{H}_0, \ldots, \mathbf{H}_7$, and the advantage of the adversary $\mathcal{A}$ when participating in experiment $\mathbf{H_i}$ by $Adv(\mathcal{A}, \mathbf{H_i})$.

Hybrid experiment $\mathbf{H_0}$: This experiment denotes the real experiment for OS-AKE security and in this experiment, the environment for $\mathcal{A}$ is as defined in the protocol. Thus, $Adv(\mathcal{A}, \mathbf{H_0})$ is the same as the advantage of the real experiment.

Hybrid experiment $\mathbf{H_1}$: This experiment aborts when a session ID is matched with multiple sessions.

Because of κ-min entropy KEM, the probability of outputting the same ciphertext from different randomness in each session is negligible. Thus, $|Adv(\mathcal{A}, \mathbf{H_1}) - Adv(\mathcal{A}, \mathbf{H_0})| \leq negl$.

Hybrid experiment $\mathbf{H_2}$: This experiment chooses a party U_S^* and a party U_C^*, an integer $i^* \in [1, \ell]$ in advance, and fixes parties and the session for the Test query. If $\mathcal{A}$ queries a session other than the i^*-th of client U_C^* (partner is U_S^*) in Test query, it aborts the experiment.

The probability that the guess of the test session is correct is $1/n^2\ell$, thus $Adv(\mathcal{A}, \mathbf{H_2}) \geq 1/n^2\ell \cdot Adv(\mathcal{A}, \mathbf{H_1})$.

Hybrid experiment $\mathbf{H_3}$: This experiment changes the way of the computation of $r_0'^*$ and $r_1'^*$ in the i^*-th session of U_C^* (partner is U_S^*). Instead of $r_0'^* \leftarrow \mathsf{F}'(0, r_C^*)$ and $r_1'^* \leftarrow \mathsf{F}'(1, r_C^*)$, it is changed as $r_0'^* \in_R \mathcal{FS}$ and $r_1'^* \in_R \mathcal{FS}$.

We construct a distinguisher $\mathcal{D}_0$ that distinguishes if F^* is either a pseudo-random function F' or a random function RF from $\mathcal{A}$ in $\mathbf{H_2}$ or $\mathbf{H_3}$. $\mathcal{D}_0$ performs the following steps.

[setup]

$\mathcal{D}_0$ is given a pseudo-random function $\mathsf{F}^{'} : \{0,1\}^\kappa \times \mathcal{FS} \to \mathcal{FS}$. Then, $\mathcal{D}_0$ chooses pseudo-random functions $\mathsf{F}_0^{'} : \{0,1\}^\kappa \times \mathcal{FS} \to \mathcal{RS}_E^{cpa}$, $\mathsf{F}_1^{'} : \{0,1\}^\kappa \times \mathcal{FS} \to \mathcal{RS}_E^{cca}$, $\mathsf{PRF} : \{0,1\}^* \times \mathcal{FS} \to \{0,1\}^\kappa$, a key derivation function $\mathsf{KDF} : Salt \times \mathcal{KS} \to \mathcal{FS}$, and $s \in_R Salt$.

$\mathcal{D}_0$ generates (ek_i, dk_i) for each server U_i including (ek_S^*, dk_S^*) of U_S^* according to the protocol, publishes ek_i, and sets $cert_{ek_i} = (ID_i, ek_i)$ as a certificate for each server U_i. $\mathcal{D}_0$ poses 0 and 1 to the oracle F^*, receives $r_1'^*$ and $r_0'^*$ as a challenge, and computes $r_0^* \leftarrow \mathsf{F}_0'(ID_S, r_0'^*)$, $r_1^* \leftarrow \mathsf{F}_1'(ID_S, r_1'^*)$, $(C_1^*, K_1^*) \leftarrow \mathsf{EnCap}(ek_S^*; r_1^*)$, and $C_0^* \leftarrow \mathsf{wEnCapC}(r_0^*)$ for the i^*-th session of U_C^*.

[simulation]

$\mathcal{D}_0$ keeps the list L_{SK} that contains queries and answers of $\mathsf{SessionKeyReveal}$. $\mathcal{D}_0$ simulates oracle queries by $\mathcal{A}$ as follows.

1. $\mathsf{Send}(params, pid)$: If the session is the i^*-th session of U_C^*, then $\mathcal{D}_0$ sets $K_1 = K_1^*$, returns (C_0^*, C_1^*, ID_S^*), and records $(\Pi, ID = pid, (C_0^*, C_1^*), (*, *), *, K_1)$ in L_{SK}. Otherwise, $\mathcal{D}_0$ chooses $((C_0, C_1), r_C)$ from the unused key pairs and returns it, or computes $((C_0, C_1), r_C)$ according to the protocol and returns it, and records $(\Pi, ID = pid, (C_0, C_1), (*, *), *, K_1)$ in L_{SK}.
2. $\mathsf{Send}(sid, msg = (C_0, C_1, id))$: If $msg = (C_0^*, C_1^*, ID_S^*)$, then $\mathcal{D}_0$ sets $K_1 = K_1^*$, chooses (ek_T^*, dk_T^*) from the unused key pairs and returns it, or generates (ek_T^*, dk_T^*) according to the protocol and return it, computes SK, and records $(\Pi, ID = id, (C_0^*, C_1^*), (ek_T^*, ek_S^*), K_0, K_1)$ and SK as a completed session in L_{SK}. Otherwise, $\mathcal{D}_0$ chooses (ek_T, dk_T) from the unused key pairs and returns it, or generates ek_T according to the protocol and returns it, computes SK, and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}.
3. $\mathsf{Send}(sid, msg = ek_T)$: If the session is the i^*-th session of U_C^*, $\mathcal{D}_0$ computes $K_0^* \leftarrow \mathsf{wEnCapK}(ek_T, C_0^*, r_0^*)$, sets $K_0 = K_0^*$, computes SK according to the protocol, and records $(\Pi, ID = id, (C_0^*, C_1^*), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}. Otherwise, $\mathcal{D}_0$ computes SK according to the protocol and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}.
4. $\mathsf{SessionKeyReveal}(sid)$:
   (a) If sid is not completed, then $\mathcal{D}_0$ returns error.
   (b) Otherwise, $\mathcal{D}_0$ returns SK as recorded in L_{SK}.
5. $\mathsf{Partner}(X)$: $\mathcal{D}_0$ returns the secret value x of the public value X as defined.
6. $\mathsf{RevealNext}()$: $\mathcal{D}_0$ generates a key pair (ESK,EPK), keeps it as unused, and returns the EPK to $\mathcal{A}$ as defined.
7. $\mathsf{EstablishCertificate}(ID_i, X)$: $\mathcal{D}_0$ registers the public key of ID_i as X according to the protocol, and marks U_i as a dishonest party.

8. $\mathsf{Test}(sid) : \mathcal{D}_0$ returns as defined.
9. $\mathcal{A}$ outputs a guess $b^{'} \in \{0,1\}$. If $\mathcal{A}$ outputs $b^{'} = 0$, then $\mathcal{D}_0$ outputs that $\mathsf{F}^* = \mathsf{F}'$, otherwise $\mathcal{D}_0$ outputs that $\mathsf{F}^* = \mathsf{RF}$.

[Analysis]

For $\mathcal{A}$, the simulation by $\mathcal{D}_0$ is the same as the experiment $\mathbf{H_2}$ if $\mathsf{F}^* = \mathsf{F}'$. Otherwise, the simulation by $\mathcal{D}_0$ is the same as the experiment $\mathbf{H_3}$. Thus, since the advantage of $\mathcal{D}_0$ is negligible due to the security of the PRF, $|Adv(\mathcal{A}, \mathbf{H_3}) - Adv(\mathcal{A}, \mathbf{H_2})| \leq negl$.

Hybrid experiment $\mathbf{H_4}$: This experiment changes the way of the computation of r_1^* in the i^*-th session of U_C^* (partner is U_S^*). Instead of $r_1^* \leftarrow \mathsf{F}_1'(ID_S^*, r_1')$, it is changed as $r_1^* \in_R \mathcal{RS}_E^{cca}$.

We construct a distinguisher $\mathcal{D}_1$ that distinguishes if F^* is either a pseudo-random function F_1' or a random function RF from $\mathcal{A}$ in $\mathbf{H_3}$ or $\mathbf{H_4}$. $\mathcal{D}_1$ performs the following steps.

[setup]

$\mathcal{D}_1$ is given a pseudo-random function $\mathsf{F}_1^{'} : \{0,1\}^\kappa \times \mathcal{FS} \rightarrow \mathcal{RS}_E^{cca}$. Then, $\mathcal{D}_1$ chooses pseudo-random functions $\mathsf{F}^{'} : \{0,1\}^\kappa \times \mathcal{FS} \rightarrow \mathcal{FS}$, $\mathsf{F}_0^{'} : \{0,1\}^\kappa \times \mathcal{FS} \rightarrow \mathcal{RS}_E^{cpa}$, $\mathsf{PRF} : \{0,1\}^* \times \mathcal{FS} \rightarrow \{0,1\}^\kappa$, a key derivation function $\mathsf{KDF} : Salt \times \mathcal{KS} \rightarrow \mathcal{FS}$, and $s \in_R Salt$.

$\mathcal{D}_1$ generates (ek_i, dk_i) for each server U_i including (ek_S^*, dk_S^*) of U_S^* according to the protocol, publishes ek_i, and sets $cert_{ek_i} = (ID_i, ek_i)$ as a certificate for each server U_i. $\mathcal{D}_1$ poses ID_S to the oracle F^*, receives r_1^* as a challenge, and computes $(C_1^*, K_1^*) \leftarrow \mathsf{EnCap}(ek_S^*; r_1^*)$, and $C_0^* \leftarrow \mathsf{wEnCapC}(r_0^*)$ by using $r_0'^* \in_R FS$ according to the protocol for the i^*-th session of U_C^*.

[simulation]

$\mathcal{D}_1$ keeps the list L_{SK} that contains queries and answers of $\mathsf{SessionKeyReveal}$. $\mathcal{D}_1$ simulates oracle queries by $\mathcal{A}$ as follows.

1. $\mathsf{Send}(params, pid)$: If the session is the i^*-th session of U_C^*, then $\mathcal{D}_1$ sets $K_1 = K_1^*$, returns (C_0^*, C_1^*, ID_S^*), and records $(\Pi, ID = pid, (C_0^*, C_1^*), (*, *), *, K_1)$ in L_{SK}. Otherwise, $\mathcal{D}_1$ chooses $((C_0, C_1), r_C)$ from the unused key pairs and returns it, or computes $((C_0, C_1), r_C)$ according to the protocol and returns it, and records $(\Pi, ID = pid, (C_0, C_1), (*, *), *, K_1)$ in L_{SK}.
2. $\mathsf{Send}(sid, msg = (C_0, C_1, id))$: If $msg = (C_0^*, C_1^*, ID_S^*)$, then $\mathcal{D}_1$ sets $K_1 = K_1^*$, chooses (ek_T^*, dk_T^*) from the unused key pairs and returns it, or generates (ek_T^*, dk_T^*) according to the protocol and return it, computes SK, and records $(\Pi, ID = id, (C_0^*, C_1^*), (ek_T^*, ek_S^*), K_0, K_1)$ and SK as a completed session in L_{SK}. Otherwise, $\mathcal{D}_1$ chooses (ek_T, dk_T) from the unused key pairs and returns it, or generates ek_T according to the protocol and returns it, computes SK, and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}.

3. $\mathsf{Send}(sid, msg = ek_T)$: If the session is the i^*-th session of U_C^*, $\mathcal{D}_1$ computes $K_0^* \leftarrow \mathsf{wEnCapK}(ek_T^*, C_0^*, r_0^*)$, sets $K_0 = K_0^*$, computes SK according to the protocol, and records $(\Pi, ID = id, (C_0^*, C_1^*), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}. Otherwise, $\mathcal{D}_1$ computes SK according to the protocol and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}.
4. $\mathsf{SessionKeyReveal}(sid)$:
   (a) If sid is not completed, then $\mathcal{D}_1$ returns error.
   (b) Otherwise, $\mathcal{D}_1$ returns SK as recorded in L_{SK}.
5. $\mathsf{Partner}(X)$: $\mathcal{D}_1$ returns the secret value x of the public value X as defined.
6. $\mathsf{RevealNext}()$: $\mathcal{D}_1$ generates a key pair (ESK,EPK), keeps it as unused, and returns the EPK to $\mathcal{A}$ as defined.
7. $\mathsf{EstablishCertificate}(ID_i, X)$: $\mathcal{D}_1$ registers the public key of ID_i as X according to the protocol, and marks U_i as a dishonest party.
8. $\mathsf{Test}(sid)$: $\mathcal{D}_1$ returns as defined.
9. $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$. If $\mathcal{A}$ outputs $b' = 0$, then $\mathcal{D}_1$ outputs that $\mathsf{F}^* = \mathsf{F}'_1$, otherwise $\mathcal{D}_1$ outputs that $\mathsf{F}^* = \mathsf{RF}$.

[Analysis]

For $\mathcal{A}$, the simulation by $\mathcal{D}_1$ is the same as the experiment $\mathbf{H_3}$ if $\mathsf{F}^* = \mathsf{F}'_1$. Otherwise, the simulation by $\mathcal{D}_1$ is the same as the experiment $\mathbf{H_4}$. Thus, since the advantage of $\mathcal{D}_1$ is negligible due to the security of the PRF, $|Adv(\mathcal{A}, \mathbf{H_4}) - Adv(\mathcal{A}, \mathbf{H_3})| \leq negl$.

Hybrid experiment $\mathbf{H_5}$: This experiment changes the way of computation of K_1^* on the client side in the i^*-th session of U_C^*. Instead of computing $(C^*, K_1^*) \leftarrow \mathsf{EnCap}(ek_S^*, r_1^*)$, it is changed as $K_1^* \in_R \mathcal{KS}_{cca}$.

We construct an IND-CCA adversary $\mathcal{S}$ from $\mathcal{A}$ in $\mathbf{H_4}$ or $\mathbf{H_5}$. The $\mathcal{S}$ performs the following steps.

[init]

$\mathcal{S}$ receives ek_S^* from the challenger as a challenge.

[setup]

$\mathcal{S}$ chooses pseudo-random functions $\mathsf{F}' : \{0,1\}^\kappa \times \mathcal{FS} \to \mathcal{FS}$, $\mathsf{F}'_0 : \{0,1\}^\kappa \times \mathcal{FS} \to \mathcal{RS}_E^{cpa}$, $\mathsf{F}'_1 : \{0,1\}^\kappa \times \mathcal{FS} \to \mathcal{RS}_E^{cca}$, $\mathsf{PRF} : \{0,1\}^* \times \mathcal{FS} \to \{0,1\}^\kappa$, a key derivation function $\mathsf{KDF} : Salt \times \mathcal{KS} \to \mathcal{FS}$, and $s \in_R Salt$.

$\mathcal{S}$ receives (K_b^*, C_1^*) as a challenge and sets $C_1 = C_1^*$ for the i^*-th session of U_C^*. Also, $\mathcal{S}$ generates (ek_i, dk_i) for each server U_i other than U_S, publishes ek_i, and sets $cert_{ek_i} = (ID_i, ek_i)$ as a certificate for each server U_i.

[simulation]

$\mathcal{S}$ keeps the list L_{SK} that contains queries and answers of $\mathsf{SessionKeyReveal}$. $\mathcal{S}$ simulates oracle queries by $\mathcal{A}$ as follows.

1. $\mathsf{Send}(params, pid)$: If the session is the i^*-th session of U_C^*, then $\mathcal{S}$ computes $C_0^* \leftarrow \mathsf{wEnCapC}(r_0^*)$ where $r_0^* \in_R \mathcal{RS}_E^{cpa}$, sets $K_1 = K_b^*$, $C_1 = C_1^*$, and $C_0 =$

C_0^*, returns (C_0, C_1, ID_S^*), and records $(\Pi, ID = pid, (C_0, C_1), (*, *), *, K_1)$ in L_{SK}. Otherwise, $\mathcal{S}$ chooses $((C_0, C_1), r_C)$ from the unused key pairs and returns (C_0, C_1), or computes $((C_0, C_1), r_C)$ according to the protocol and returns (C_0, C_1), and records $(\Pi, ID = pid, (C_0, C_1), (*, *), *, K_1)$ in L_{SK}.

2. $\mathsf{Send}(sid, msg = (C_0, C_1, id))$: If $id = ID_S^*$ and $C_1 \neq C_1^*$, $\mathcal{S}$ poses C_1 to the decryption oracle to obtain K_1, chooses (ek_T, dk_T) from the unused key pairs and returns ek_T, or generates (ek_T, dk_T) and returns ek_T, computes SK, and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}. Also, else if $id = ID_S^*$ and $C_1 = C_1^*$, $\mathcal{S}$ sets $K_1 = K_b^*$, chooses (ek_T^*, dk_T^*) from the unused key pairs and returns it, or generates (ek_T^*, dk_T^*) according to the protocol and returns it, computes SK, and records $(\Pi, ID = id, (C_0, C_1), (ek_T^*, ek_S^*), K_0, K_1)$ and SK as a completed session in L_{SK}. Otherwise, $\mathcal{S}$ chooses (ek_T, dk_T) from the unused key pairs and returns it, or generates (ek_T, dk_T) according to the protocol and returns it, computes SK, and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}.
3. $\mathsf{Send}(sid, msg = ek_T)$: $\mathcal{S}$ computes SK according to the protocol and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}.
4. $\mathsf{SessionKeyReveal}(sid)$:
   (a) If sid is not completed, then $\mathcal{S}$ returns error.
   (b) Otherwise, $\mathcal{S}$ returns SK as recorded in L_{SK}.
5. $\mathsf{Partner}(X) : \mathcal{S}$ returns the secret value x of the public value X as defined.
6. $\mathsf{RevealNext}() : \mathcal{S}$ generates a key pair (ESK,EPK), keeps it as unused, and returns the EPK to $\mathcal{A}$ as defined.
7. $\mathsf{EstablishCertificate}(ID_i, X) : \mathcal{S}$ registers the public key of ID_i as X according to the protocol, and marks U_i as a dishonest party.
8. $\mathsf{Test}(sid) : \mathcal{S}$ returns as defined.
9. $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$. If $\mathcal{A}$ outputs b', then $\mathcal{S}$ outputs b'.

[Analysis]

For $\mathcal{A}$, the simulation by $\mathcal{S}$ is same the as the experiment $\mathbf{H_4}$ if the challenge is (C_1^*, K_0^*). Otherwise, the simulation by $\mathcal{S}$ is same the as the experiment $\mathbf{H_5}$. Thus, since the advantage of $\mathcal{S}$ is negligible due to the security of the IND-CCA secure KEM, $|Adv(\mathcal{A}, \mathbf{H_5}) - Adv(\mathcal{A}, \mathbf{H_4})| \leq negl$.

Hybrid experiment $\mathbf{H_6}$: This experiment changes the way of the computation of the $K_1^{'*}$ in the i^*-th session of U_C^*. Instead of computing $K_1^{'*} \leftarrow \mathsf{KDF}(s, K_1^*)$, it is changed as choosing $K_1^{'*} \in_R \mathcal{FS}$.

Since K_1^* is randomly chosen in $\mathbf{H_5}$, it has sufficient min-entropy because the KEM is κ-min-entropy KEM. Thus, by the definition of the KDF, $|Adv(\mathcal{A}, \mathbf{H_6}) - Adv(\mathcal{A}, \mathbf{H_5})| \leq negl$.

Hybrid experiment $\mathbf{H_7}$: This experiment changes the way of the computation of SK in the i^*-th session of U_C^*. Instead of computing $SK = \mathsf{PRF}(sid, K_1) \oplus \mathsf{PRF}(sid, K_0)$, it is changed as $SK = x \oplus \mathsf{PRF}(sid, K_0)$, where $x \in_R \{0, 1\}^\kappa$.

We construct a distinguisher $\mathcal{D}_2$ that distinguishes if F^* is either a pseudo-random function PRF or a random function RF from $\mathcal{A}$ in $\mathbf{H_6}$ or $\mathbf{H_7}$. $\mathcal{D}_2$ performs the following steps.

[setup]

$\mathcal{D}_2$ is given a pseudo-random function $\mathsf{PRF} : \{0,1\}^* \times \mathcal{FS} \to \{0,1\}^\kappa$. Then, $\mathcal{D}_2$ chooses pseudo-random functions $\mathsf{F}' : \{0,1\}^\kappa \times \mathcal{FS} \to \mathcal{FS}$, $\mathsf{F}'_0 : \{0,1\}^\kappa \times \mathcal{FS} \to \mathcal{RS}_E^{cpa}$, $\mathsf{F}'_1 : \{0,1\}^\kappa \times \mathcal{FS} \to \mathcal{RS}_E^{cca}$, $\mathsf{PRF} : \{0,1\}^* \times \mathcal{FS} \to \{0,1\}^\kappa$, a key derivation function $\mathsf{KDF} : Salt \times \mathcal{KS} \to \mathcal{FS}$, and $s \in_R Salt$.

$\mathcal{D}_2$ generates (ek_i, dk_i) for each server U_i including (ek_S^*, dk_S^*) of U_S^* according to the protocol, publishes ek_i, and sets $cert_{ek_i} = (ID_i, ek_i)$ as a certificate for each server U_i.

[simulation]

$\mathcal{D}_2$ keeps the list L_{SK} that contains queries and answers of $\mathsf{SessionKeyReveal}$. $\mathcal{D}_2$ simulates oracle queries by $\mathcal{A}$ as follows.

1. $\mathsf{Send}(params, pid)$: If the session is the i^*-th session of U_C^*, then $\mathcal{D}_2$ computes $(C_1^*, K_1^*) \leftarrow \mathsf{EnCap}(ek_S^*; r_1^*)$ and $C_0^* \leftarrow \mathsf{wEnCapC}(r_0^*)$, where $r_0^* \leftarrow \mathsf{F}'_0(ID_S^*, r_0'^*)$ and $r_1^* \in_R \mathcal{RS}_E^{cca}$, returns (C_0^*, C_1^*, ID_S^*), and records $(\Pi, ID = id, (C_0^*, C_1^*), (*, *), *, K_1^*)$ in L_{SK}. Otherwise, $\mathcal{D}_2$ chooses $((C_0, C_1), r_C)$ from the unused key pairs and returns (C_0, C_1), or computes $((C_0, C_1), r_C)$ according to the protocol and returns (C_0, C_1), and records $(\Pi, ID = id, (C_0, C_1), (*, *), *, K_1^*)$ in L_{SK}.
2. $\mathsf{Send}(sid, msg = (C_0, C_1, id))$: If $msg = (C_0^*, C_1^*, ID_S^*)$, then $\mathcal{D}_2$ chooses (ek_T^*, dk_T^*) from the unused key pairs and returns it, or generates (ek_T^*, dk_T^*) according to the protocol and returns it. Also, $\mathcal{D}_2$ sets sid according to the protocol, poses it to the oracle (PRF or RF), obtains $x \in \{0,1\}^\kappa$, computes $SK^* = x \oplus \mathsf{PRF}(sid, K_0)$, and records $(\Pi, ID = id, (C_0^*, C_1^*), (ek_T^*, ek_S^*))$ and SK^* as a completed session in L_{SK}. Otherwise, $\mathcal{D}_2$ chooses (ek_T, dk_T) from the unused key pairs and returns ek_T, or generates (ek_T, dk_T) according to the protocol and returns ek_T. Also, $\mathcal{D}_2$ computes SK and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S))$ and SK as a completed session in L_{SK}.
3. $\mathsf{Send}(sid, msg = ek_T)$: If the session is the i^*-th session of U_C^*, then $\mathcal{D}_2$ sets sid according to the protocol, poses it to the oracle (PRF or RF), obtains $x \in \{0,1\}^\kappa$, computes $SK^* = x \oplus \mathsf{PRF}(sid, k_0)$, and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S))$ and SK^* as a completed session in L_{SK}. Otherwise, $\mathcal{D}_2$ computes SK according to the protocol and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S))$ and SK as a completed session in L_{SK}.
4. $\mathsf{SessionKeyReveal}(sid)$:
   (a) If sid is not completed, then $\mathcal{D}_2$ returns error.
   (b) Otherwise, $\mathcal{D}_2$ returns SK as recorded in L_{SK}.
5. $\mathsf{Partner}(X)$: $\mathcal{D}_2$ returns the secret value x of the public value X as defined.
6. $\mathsf{RevealNext}()$: $\mathcal{D}_2$ generates a key pair (ESK,EPK), keeps it as unused, and returns the EPK to $\mathcal{A}$ as defined.

7. $\mathsf{EstablishCertificate}(ID_i, X) : \mathcal{D}_2$ registers the public key of ID_i as X according to the protocol, and marks U_i as a dishonest party.
8. $\mathsf{Test}(sid) : \mathcal{D}_2$ returns as defined.
9. $\mathcal{A}$ outputs a guess $b^{'} \in \{0, 1\}$. If $\mathcal{A}$ outputs $b^{'}$, then $\mathcal{D}_2$ outputs $b^{'}$.

[Analysis]

For $\mathcal{A}$, the simulation by $\mathcal{D}_2$ is the same as the experiment $\mathbf{H_6}$ if $\mathsf{F}^* = \mathsf{PRF}$. Otherwise, the simulation by $\mathcal{D}_2$ is the same as the experiment $\mathbf{H_7}$. Thus, since the advantage of $\mathcal{D}_2$ is negligible due to the security of PRF, $|Adv(\mathcal{A}, \mathbf{H_7}) - Adv(\mathcal{A}, \mathbf{H_6})| \leq negl$.

In $\mathbf{H_7}$, the session key in the test session is perfectly randomized. This gives $\mathcal{A}$ no information from the Test query, therefore $Adv(\mathcal{A}, \mathbf{H_7}) = 0$ and $\Pr[E_1 \wedge Sec] = negl$.

□

Theorem 2. *In the standard model, GC-Std is one-sided anonymous.*

Proof. We proceed by introducing another experiment, in which cannot win more than random guessing. In this new experiment, the choice of i^* is independent of the behavior of the rest of the system. Then, we show that no adversary can distinguish this new experiment from the original experiment, thereby showing the OS-anonymity of the protocol.

$\mathsf{Expt}^{OS-anon'}_{GC-Std}(\mathcal{A})$ is the same experiment as $\mathsf{Expt}^{OS-anon}_{GC-Std}(\mathcal{A})$ except for the following oracle used by the challenger $\mathcal{C}$.

- $\mathsf{Start}'(i_0, i_1, params, pid = ID^*_S) \rightarrow msg'$:
  1. If $i_0 = i_1$, then abort.
  2. Set $i^* \leftarrow_R \{i_0, i_1\}$.
  3. Set $ID^* \leftarrow ID^*_S$.
  4. Choose $((C^*_0, C^*_1), r^*_C)$ from the unused key pairs and returns (C^*_0, C^*_1, ID^*).
- $\mathsf{Send}'(sid, msg = ek^*_T)$:
  1. Compute $r'^*_0 \leftarrow \mathsf{F}'(0, r^*_C))$ and $r^*_0 \leftarrow \mathsf{F}'_0(ID_S, r'^*_0)$
  2. Compute $K^*_0 \leftarrow \mathsf{wEnCapK}(ek^*_T, C^*_0, r^*_0)$.
  3. Compute SK according to the protocol.
- $\mathsf{SessionKeyReveal}'() \rightarrow SK$: If the test session is a completed session, return SK.
- $\mathsf{Partner}'(C^*) \rightarrow r^*_C$: Return the secret value r^*_C corresponding to C^*.
- $\mathsf{RevealNext}' \rightarrow X$: Return the future public value X and record it as unused.

Since all messages computed in $\mathsf{Expt}^{OS-anon'}_{GC-Std}(\mathcal{A})$ are independent of the choice of i^*, the adversary $\mathcal{A}$ has no advantage, thus the probability that $\mathcal{A}$ wins the game is as follows.

$$\Pr[\mathsf{Expt}^{OS-anon'}_{GC-Std}(\mathcal{A}) = win] = 1/2 \tag{1}$$

Also, the distribution of messages returned by the challenger in $\mathsf{Expt}^{OS-anon'}_{GC-Std}(\mathcal{A})$ is the same as that returned in $\mathsf{Expt}^{OS-anon}_{GC-Std}(\mathcal{A})$. Furthermore, messages from

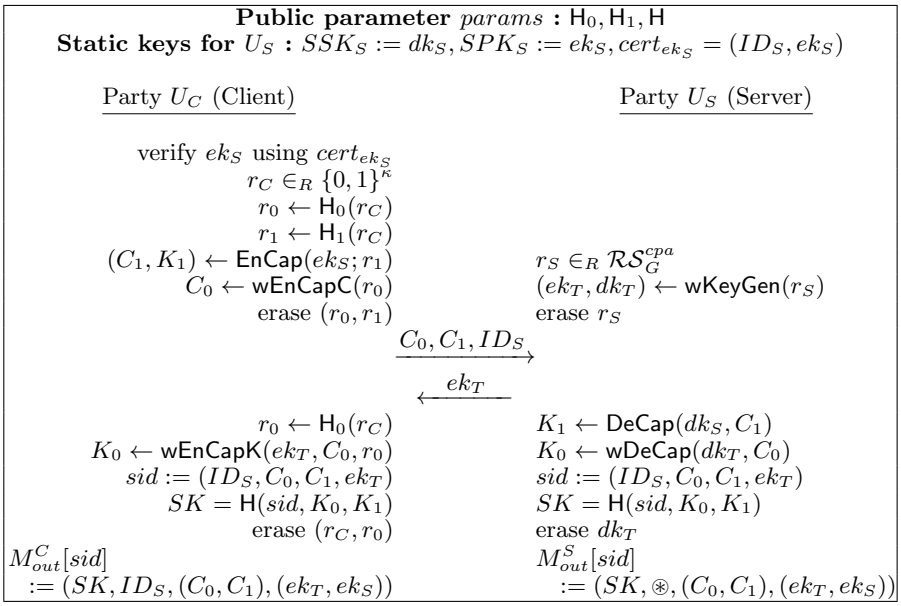

Fig. 5. Generic construction in the random oracle model (GC-RO)

all parties except P_{i_0} and P_{i_1} are unchanged. For messages from P_{i_0} and P_{i_1} in $\mathsf{Expt}_{GC-Std}^{OS-anon'}(\mathcal{A})$, all queries return messages of the same distribution as in $\mathsf{Expt}_{GC-Std}^{OS-anon}(\mathcal{A})$.

Here, queries that reveal information about whether P_{i_0} or P_{i_1} participated in the test session are prohibited by the definition. For example, $\mathcal{A}$ is prohibited from using a $\mathsf{SessionKeyReveal}(sid)$ query to P_{i_0} to find out if P_{i_0} has the session key for the test session.

Thus, $\mathcal{A}$ cannot distinguish between the two games.

$$\Pr[\mathsf{Expt}_{GC-Std}^{OS-anon'}(\mathcal{A}) = win] = \Pr[\mathsf{Expt}_{GC-Std}^{OS-anon}(\mathcal{A}) = win] \tag{2}$$

From equations (1) and (2), the scheme has one-sided anonymity. □

5.3 OS-AKE in Random Oracle Model

The protocol in the random oracle model consists of an OW-CCA secure KEM $(\mathsf{KeyGen}, \mathsf{EnCap}, \mathsf{DeCap})$ and an OW-CPA secure PKIC-KEM $(\mathsf{wKeyGen}, \mathsf{wEnCapC}, \mathsf{wEnCapK}, \mathsf{wDeCap})$ as follows.

Protocol.

Public Parameters : Let κ be a security parameter, and $\mathsf{H}_0 : \{0,1\}^* \to \mathcal{RS}_E^{cpa}$, $\mathsf{H}_1 : \{0,1\}^* \to \mathcal{RS}_E^{cca}$, $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$ be hash functions, where $\mathcal{RS}_E^{cpa}$ and $\mathcal{RS}_G^{cpa}$ are randomness spaces of the encapsulation algorithm and the key generation algorithm of OW-CPA secure PKIC-KEM, $\mathcal{RS}_E^{cca}$ and $\mathcal{RS}_G^{cca}$ are randomness spaces of the encapsulation algorithm and the key generation algorithm of OW-CCA secure KEM. These are provided as part of the public parameters.

Secret and Public Keys : Party U_S selects a randomness $r \in_R \mathcal{RS}_G^{cca}$, computes $(ek_S, dk_S) \leftarrow \mathsf{KeyGen}(1^\kappa; r)$ and sets $cert_{ek_S} = (ID_S, ek_S)$ as a certificate for U_S. The static key pair for party U_S is (ek_S, dk_S).

Key Exchange : Let U_S which has a static key pair (ek_S, dk_S) be a server, and U_C be a client. When U_C is initialized as a client, it obtains the certificate $cert_{ek_S} = (ID_S, ek_S)$ of U_S.

1. U_C verifies the server using $cert_{ek_S} = (ID_S, ek_S)$. U_C chooses an unused key pair $((C_0, C_1), r_C)$ or chooses a ehemeral secret key $r_C \in_R \{0,1\}^\kappa$ and sets $r_0 \leftarrow \mathsf{H}_0(r_C)$, and $r_1 \leftarrow \mathsf{H}_1(r_C)$. Also, U_C computes $(C_1, K_1) \leftarrow \mathsf{EnCap}(ek_S; r_1)$, and $C_0 \leftarrow \mathsf{wEnCapC}(r_0)$, and erases (r_0, r_1). Then, U_C sends (C_0, C_1, ID_S) to U_S.
2. Upon receiving (C_0, C_1, ID_S), U_S chooses an unused ephemeral key pair (ek_T, dk_T), or chooses a randomness $r_S \in_R \mathcal{RS}_G^{cpa}$ and computes $(ek_T, dk_T) \leftarrow \mathsf{wKeyGen}(r_S)$, and sends ek_T to U_C. Then, U_S computes $K_1 \leftarrow \mathsf{DeCap}(dk_S, C_1)$ and $K_0 \leftarrow \mathsf{wDeCap}(dk_T, C_0)$, sets $sid = (ID_S, (C_0, C_1), ek_T)$, and computes the session key $SK = \mathsf{H}(sid, K_0, K_1)$. U_S erases (r_S, dk_T) and outputs $(SK, \circledast, (C_0, C_1), (ek_T, ek_S))$.
3. Upon receiving ek_T, U_C sets $r_0 \leftarrow \mathsf{H}_0(r_C)$ and computes $K_0 \leftarrow \mathsf{wEnCapK}(ek_T, C_0, r_0)$. Also, U_C sets $sid = (ID_S, (C_0, C_1), ek_T)$ and computes the session key $SK = \mathsf{H}(sid, K_0, K_1)$. Then, U_C erases (r_C, r_0), and outputs $(SK, ID_S, (C_0, C_1), (ek_T, ek_S))$.

Security. We show the security of the proposed scheme in the random oracle model. An intuition of the proof is shown in Section 5.1.

Theorem 3. *If* $(\mathsf{KeyGen}, \mathsf{EnCap}, \mathsf{DeCap})$ *is an OW-CCA secure KEM,* $(\mathsf{wKeyGen}, \mathsf{wEnCapC}, \mathsf{wEnCapK}, \mathsf{wDeCap})$ *is an OW-CPA secure PKIC-KEM, and* $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}$ *are random oracles, GC-RO is OS-AKE secure.*

Theorem 4. *In the random oracle model, GC-RO scheme is one-sided anonymous.*

We show the proof of Theorem 3 and 4 in Appendix B and C.

6 Instantiations based on DH Problems

A comparison of the efficiency among our instantiations and existing schemes is shown in Table 1.

Table 1. Comparison among existing DH-based schemes and our instantiations

Protocol	Model	Resource	Assumption	Exp. (client)		Exp. (server)		Communication complexity
				Off-line	On-line	Off-line	On-line	
ntor [24]	GSU	RO	gap DH	1	2	1	1.33	$\|ID\| + 2\|G\|$
Ace [6]	weak GSU	RO	gap DH	2	1.08	1	1.08	$\|ID\| + 3\|G\|$
Ours1[6.1]	GSU	RO	CDH	3	1	1	3	$\|ID\| + 3\|G\| + \kappa$
Ours2[6.2]	GSU	Std	DDH	5.08	1	1	3.16	$\|ID\| + 5\|G\|$

For exponentiation costs, we apply the parallel computation technique [39] for two exponentiations using the same base, which costs 1.33 exponentiations for κ, and Avanzi's algorithm [4] for multi-exponentiations in the elliptic curve setting, which costs 1.08 exponentiations for κ. $|ID|$ is the length of server's ID and $|G|$ is the size of a group element.

6.1 Random Oracle Model

We can obtain an OS-AKE scheme in the random oracle model by instantiating GC-RO using the PSEC-KEM [40] which is an OW-CCA secure KEM, and the ElGamal KEM which is an OW-CPA secure PKIC-KEM. It is shown that the ElGamal KEM can be PKIC-KEM [45], and the PSEC-KEM and the ElGamal KEM are obviously κ-min-entropy KEM. Since these KEM schemes are based on the computational DH (CDH) assumption, the instantiation is also secure under the CDH assumption though ntor and Ace rely on the gap DH assumption. Also, the online computational cost of a client is smaller than existing schemes.

6.2 Standard Model

We can obtain an OS-AKE scheme in the standard model by instantiating GC-Std using CS3 [11] which is an IND-CCA secure KEM, and the ElGamal KEM which is an IND-CPA secure PKIC-KEM. CS3 is obviously κ-min-entropy KEM. Since these KEM schemes are based on the decisional DH (DDH) assumption, the instantiation is also secure under the DDH assumption. This scheme is the first DH-based anonymous OS-AKE scheme in the standard model. Moreover, the online computational cost of a client is smaller than existing schemes even in the standard model.

7 Instantiations based on Isogeny Problems

7.1 Random Oracle Model

SIDH-based. We can obtain a SIDH-based OS-AKE scheme in the random oracle model by instantiating GC-RO using the SIKE-KEM [5] which is an IND-CCA secure KEM, and an OW-CPA PKIC-KEM which is obtained by a transformation of SIKE-PKE [5]. In order to transform the SIKE-PKE to PKIC-KEM, we remove the generation of the ciphertext $C_1 = F(j) \oplus m$ (i.e., masking of plaintext m) in the encapsulation algorithm and the decryption procedure

Public parameter : $\mathcal{K}_3, \mathcal{K}_2, \mathsf{isogen}_3, \mathsf{isogen}_2, \mathsf{isoex}_2, \mathsf{isoex}_3$			
$\underline{\mathsf{wKeyGen}(1^\kappa)}$	$\underline{\mathsf{wEnCapC}}$	$\underline{\mathsf{wEnCapK}(pk_3, C_0, sk_2)}$	$\underline{\mathsf{wDeCap}(sk_3, C_0)}$
$sk_3 \in_R \mathcal{K}_3$	$sk_2 \in_R \mathcal{K}_2$	$j = \mathsf{isoex}_2(pk_3, sk_2)$	$j = \mathsf{isoex}_3(C_0, sk_3)$
$pk_3 = \mathsf{isogen}_3(sk_3)$	$C_0 = \mathsf{isogen}_2(sk_2)$	$K = j$	$K = j$
return: (pk_3, sk_3)	**return:** C_0	**return:** K	**return:** K

Fig. 6. PKIC-KEM scheme based on SIKE-PKE [5]

Public parameter : $X, G, E_0 \in G, H$			
$\underline{\mathsf{wKeyGen}(1^\kappa)}$	$\underline{\mathsf{wEnCapC}}$	$\underline{\mathsf{wEnCapK}(pk, C, \mathfrak{r})}$	$\underline{\mathsf{wDeCap}(sk, C)}$
$\mathfrak{s} \in_R G$	$\mathfrak{r} \in_R G$	$S = [\mathfrak{r}] * pk$	$S = [sk] * C$
$sk = \mathfrak{s}$	$C = [\mathfrak{r}] * E_0$	$K = H(S)$	$K = H(S)$
$pk = [\mathfrak{s}] * E_0$	**return:** C	**return:** K	**return:** K
return $:(pk, sk)$			

Fig. 7. Hashed CSIDH-KEM scheme

$m = F(j) \oplus C_1$ in the decryption algorithm, and use $j = \mathsf{isoex}_2(pk_3, sk_2)$ as the session key of PKIC-KEM. Such a PKIC-KEM based on SIKE-PKE is shown in Fig. 6. SIKE-KEM and PKIC-KEM in Fig. 6 are obviously κ-min-entropy KEM. Note that PKIC-KEM in Fig. 6 is regarded as a SIDH version of the ElGamal KEM and it is pointed out that it is OW-CPA secure under the supersingular decisional DH (SSDDH) assumption [37]. Since SIKE-KEM is based on the supersingular computational DH (SSCDH) assumption, the instantiation is secure under the SSDDH assumption.

CSIDH-based. We can obtain a CSIDH-based OS-AKE scheme in the random oracle model by instantiating GC-RO using the CSIDH-PSEC-KEM [46] which is an IND-CCA secure KEM, and CSIDH-KEM [9] which is an OW-CPA secure KEM. Note that CSIDH-KEM can be used as PKIC-KEM in the same way as Fig. 6. CSIDH-PSEC-KEM and CSIDH-KEM are obviously κ-min-entropy KEM. Note that CSIDH-KEM is pointed that it is OW-CPA secure under the commutative supersingular decisional DH (CSSDDH) assumption [37]. Since CSIDH-PSEC-KEM is based on the commutative supersingular computational DH (CSSCDH) assumption, the instantiation is secure under the CSSDDH assumption.

7.2 Standard Model

We can obtain a CSIDH-based OS-AKE scheme in the standard model by instantiating GC-Std using the KEM from smooth projective hashing [2] which is an IND-CCA secure KEM based on the hash proof system under the existence of weak pseudorandom effective group action (wPR-EGA) (a generalization of CSIDH assumptions), and a hashed CSIDH-KEM. The hashed CSIDH-KEM is a variant of CSIDH-KEM such that the session key is computed as the output of the entropy-smoothing hash function H on inputting the result of the group action of the randomness and the public key ($K = H([\mathfrak{r}] * pk)$) or the

secret key and the ciphertext ($K = H([\mathfrak{s}] * C)$). We can use the hashed CSIDH-KEM as PKIC-KEM as Fig. 6. The protocol of hashed CSIDH-KEM is shown in Fig. 7. As the same as the hashed ElGamal KEM [41], it is pointed out that the hashed CSIDH-KEM is IND-CPA secure under the CSSDDH assumption [37]. This instantiation is the first post-quantum anonymous OS-AKE scheme in the standard model under the wPR-EGA and the CSSDDH assumption.

Also, very recently, a KEM scheme called SimS [16] was proposed as a CSIDH-based IND-CCA secure KEM in the standard model. By using SimS as the instantiation of IND-CCA secure KEM, we can also construct the OS-AKE scheme from knowledge of exponent-type assumption and the CSSDDH assumption.

References

1. Abdalla, M., Izabachène, M., Pointcheval, D.: Anonymous and Transparent Gateway-Based Password-Authenticated Key Exchange. In: CANS 2008. pp. 133–148 (2008)
2. Alamati, N., Feo, L.D., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: ASIACRYPT 2020. pp. 411–439 (2020)
3. Alwen, J., Hirt, M., Maurer, U., Patra, A., Raykov, P.: Anonymous Authentication with Shared Secrets. In: LATINCRYPT 2014. pp. 219–236 (2014)
4. Avanzi, R.M.: The Complexity of Certain Multi-Exponentiation Techniques in Cryptography. J. Cryptology pp. 357–373 (2005)
5. Azarderakhsh, R., Campagna, M., Costello, C., Feo, L.D., Hess, B., Hutchinson, A., Jalali, A., Karabina, K., Koziel, B., LaMacchia, B., Longa, P., Naehrig, M., Pereira, G., Renes, J., Soukharev, V., Urbanik, D.: Supersingular isogeny key encapsulation. NIST Post-Quantum Cryptography Standardization (2020)
6. Backes, M., Kate, A., Mohammadi, E.: Ace: An Efficient Key-Exchange Protocol for Onion Routing. In: 11th ACM WPES. pp. 55–64 (2012)
7. Canetti, R., Goldreich, O., Halevi, S.: The Random Oracle Methodology, Revisited. J. ACM pp. 557–594 (2004)
8. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Eurocrypt 2001. pp. 453–474 (2001)
9. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: An efficient post-quantum commutative group action. In: ASIACRYPT 2018. pp. 395–427 (2018)
10. Chow, S.S.M., Choo, K.R.: Strongly-Secure Identity-Based Key Agreement and Anonymous Extension. In: ISC 2007. pp. 203–220 (2007)
11. Cramer, R., Shoup, V.: Design and Analysis of Practical Public-key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack. SIAM J. Comput pp. 167–226 (2003)
12. Dagdelen, O., Fischlin, M., Gagliardoni, T., Marson, G.A., Mittelbach, A., Onete, C.: A Cryptographic Analysis of OPACITY - (Extended Abstract). In: ESORICS 2013. pp. 345–362 (2013)
13. Diemert, D., Jager, T.: On the Tight Security of TLS 1.3: Theoretically Sound Cryptographic Parameters for Real-World Deployments. J. Cryptol. p. 30 (2021)
14. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: 13th USENIX Security Symposium. pp. 303–320 (2004)

15. Dodis, Y., Fiore, D.: Unilaterally-Authenticated Key Exchange. In: Financial Cryptography and Data Security 2017. pp. 542–560 (2017)
16. Fouotsa, T.B., Petit, C.: SimS: a Simplification of SiGamal. In: PQcrypt 2021. pp. 277–295 (2021)
17. Fujioka, A., Suzuki, K., Xagawa, K., Yoneyama, K.: Practical and post-quantum authenticated key exchange from one-way secure key encapsulation mechanism. In: AsiaCCS 2013. pp. 83–94 (2013)
18. Fujioka, A., Suzuki, K., Xagawa, K., Yoneyama, K.: Strongly secure authenticated key exchange from factoring, codes, and lattices. Designs, Codes and Cryptography pp. 469–504 (2015)
19. Fujioka, A., Takashima, K., Terada, S., Yoneyama, K.: Supersingular Isogeny Diffie-Hellman Authenticated Key Exchange. In: ICISC 2018. pp. 177–195 (2018)
20. Fujioka, A., Takashima, K., Yoneyama, K.: One-Round Authenticated Group Key Exchange from Isogenies. In: ProvSec 2019. pp. 330–338 (2019)
21. Galbraith, S.D.: Authenticated key exchange for SIDH. IACR Cryptology ePrint Archive, Report 2018/266 (2018)
22. Ghosh, S., Kate, A.: Post-Quantum Forward-Secure Onion Routing - (Future Anonymity in Today's Budget). In: ACNS 2015. pp. 263–286 (2015)
23. Giesen, F., Kohlar, F., Stebila, D.: On the security of TLS renegotiation. In: ACM CCS 2013. pp. 387–398 (2013)
24. Goldberg, I., Stebila, D., Ustaoglu, B.: Anonymity and one-way authentication in key exchange protocols. Designs, Codes and Cryptography pp. 245–269 (2013)
25. Guilhem, C.D.S., Smart, N.P., Warinschi, B.: Generic Forward-Secure Key Agreement Without Signatures. In: ISC 2017. pp. 114–133 (2017)
26. Jao, D., Feo, L.D.: Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies. In: PQCrypt 2011. pp. 19–34 (2011)
27. Kawashima, T., Takashima, K., Aikawa, Y., Takagi, T.: An Efficient Authenticated Key Exchange from Random Self-reducibility on CSIDH. In: ICISC 2020. pp. 58–84 (2020)
28. d. Kock, B., Gjøsteen, K., Veroni, M.: Practical Isogeny-Based Key-exchange with Optimal Tightness. In: SAC 2020. pp. 451–479 (2020)
29. Kohlar, F., Schäge, S., Schwenk, J.: On the Security of TLS-DH and TLS-RSA in the Standard Model. IACR Cryptology ePrint Archive, Report 2013/367 (2013)
30. Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. In: CRYPTO 2005. pp. 546–566 (2005)
31. Krawczyk, H., Paterson, K.G., Wee, H.: On the security of the TLS protocol: A systematic analysis. In: Crypto 2013. pp. 429–448 (2013)
32. Krawczyk, H., Wee, H.: The OPTLS Protocol and TLS 1.3. In: EuroS&P 2016. pp. 81–96 (2016)
33. Kwon, A., Lazar, D., Devadas, S., Ford, B.: Riffle: An Efficient Communication System With Strong Anonymity. 16th PETS pp. 115–134 (2016)
34. LaMacchia, B., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: ProvSec 2007. pp. 1–16 (2007)
35. Lee, M., Smart, N.P., Warinschi, B., Watson, G.J.: Anonymity guarantees of the UMTS/LTE authentication and connection protocol. Int. J. Inf. Sec. pp. 513–527 (2013)
36. Longa, P.: A Note on Post-Quantum Authenticated Key Exchange from Supersingular Isogenies. IACR Cryptology ePrint Archive, Report 2018/267 (2018)
37. Moriya, T., Onuki, H., Takagi, T.: Sigamal: A supersingular isogeny-based PKE and its application to a PRF. In: ASIACRYPT 2020. pp. 551–580 (2020)

38. Morrissey, P., Smart, N., Warinschi, B.: A Modular Security Analysis of the TLS Handshake Protocol. In: Asiacrypt 2008. pp. 55–73 (2008)
39. M'Raíhi, D., Naccache, D.: Batch Exponentiation: A Fast DLP-Based Signature Generation Strategy. In: ACM CCS 1996. p. 58–61 (1996)
40. Shoup, V.: A Proposal for an ISO Standard for Public Key Encryption. IACR Cryptology ePrint Archive, Report 2001/112 (2001)
41. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. IACR Cryptology ePrint Archive, Report 2004/332 (2004)
42. Walker, J., Li, J.: Key Exchange with Anonymous Authentication Using DAA-SIGMA Protocol. In: INTRUST 2010. pp. 108–127 (2010)
43. Xu, X., Xue, H., Wang, K., Au, M.H., Tian, S.: Strongly Secure Authenticated Key Exchange from Supersingular Isogenies. In: ASIACRYPT 2019. pp. 278–308 (2019)
44. Yang, X., Jiang, H., Hou, M., Zheng, Z., Xu, Q., Choo, K.R.: A Provably-Secure Two-Factor Authenticated Key Exchange Protocol with Stronger Anonymity. In: NSS 2018. pp. 111–124 (2018)
45. Yoneyama, K.: One-Round Authenticated Key Exchange with Strong Forward Secrecy in the Standard Model against Constrained Adversary. IWSEC 2012 pp. 1124–1138 (2012)
46. Yoneyama, K.: Post-Quantum Variants of ISO/IEC Standards: Compact Chosen Ciphertext Secure Key Encapsulation Mechanism from Isogenies. IEICE Transactions on Fundamentals of Electronics pp. 69–78 (2021)

A Proof of Event $E_2 \wedge Suc$ of Theorem 1

Proof. We change the interface of oracle queries and the computation of the session key. These instances are gradually changed over eight hybrid experiments, depending on specific subcases. In the last hybrid experiment, the session key in the test session does not contain information of the bit b. Thus, the adversary clearly only outputs a random guess. We denote these hybrid experiments by $\mathbf{H}_0, \ldots, \mathbf{H}_7$, and the advantage of the adversary $\mathcal{A}$ when participating in experiment $\mathbf{H_i}$ by $Adv(\mathcal{A}, \mathbf{H_i})$.

Hybrid experiment $\mathbf{H_0}$: This experiment denotes the real experiment for OS-AKE security and in this experiment, the environment for $\mathcal{A}$ is as defined in the protocol. Thus, $Adv(\mathcal{A}, \mathbf{H_0})$ is the same as the advantage of the real experiment.

Hybrid experiment $\mathbf{H_1}$: This experiment aborts when a session ID is matched with multiple sessions.

Because of κ-min entropy KEM, the probability of outputting the same ciphertext from different randomness in each session is negligible. Thus, $|Adv(\mathcal{A}, \mathbf{H_1}) - Adv(\mathcal{A}, \mathbf{H_0})| \leq negl$.

Hybrid experiment $\mathbf{H_2}$: This experiment chooses a party U_S^* and a party U_C^*, an integer $i^* \in [1, \ell]$ in advance, and fixes parties and the session for the Test query. If $\mathcal{A}$ queries a session other than the i^*-th of client U_C^* (partner is U_S^*) in Test query, it aborts the experiment.

The probability that the guess of the test session is correct is $1/n^2\ell$, thus $Adv(\mathcal{A}, \mathbf{H_2}) \geq 1/n^2\ell \cdot Adv(\mathcal{A}, \mathbf{H_1})$.

Hybrid experiment $\mathbf{H_3}$: This experiment changes the way of the computation of $r_0'^*$ and $r_1'^*$ in the i^*-th session of U_C^* (partner is U_S^*). Instead of $r_0'^* \leftarrow \mathsf{F}'(0, r_C^*)$ and $r_1'^* \leftarrow \mathsf{F}'(1, r_C^*)$, it is changed as $r_0'^* \in_R \mathcal{FS}$ and $r_1'^* \in_R \mathcal{FS}$.

We construct a distinguisher $\mathcal{D}_0$ that distinguishes if F^* is either a pseudo-random function F' or a random function RF from $\mathcal{A}$ in $\mathbf{H_3}$ or $\mathbf{H_2}$. The simulation is the same as $\mathbf{H_3}$ in event $E_1 \wedge Suc$.

Hybrid experiment $\mathbf{H_4}$: This experiment changes the way of the computation of r_0^* in the i^*-th session of U_C^* (partner is U_S^*). Instead of $r_0^* \leftarrow \mathsf{F}_0'(ID_S^*, r_0'^*)$, it is changed as $r_0^* \in_R \mathcal{RS}_E^{cpa}$.

We construct a distinguisher $\mathcal{D}_1$ that distinguishes if F^* is either a pseudo-random function F_0' or a random function RF from $\mathcal{A}$ in $\mathbf{H_3}$ or $\mathbf{H_4}$. $\mathcal{D}_1$ performs the following steps.

[setup]

$\mathcal{D}_1$ is given a pseudo-random function $\mathsf{F}_0^{'} : \{0,1\}^\kappa \times \mathcal{FS} \rightarrow \mathcal{RS}_E^{cpa}$. Then, $\mathcal{D}_1$ chooses pseudo-random functions $\mathsf{F}^{'} : \{0,1\}^\kappa \times \mathcal{FS} \rightarrow \mathcal{FS}$, $\mathsf{F}_1^{'} : \{0,1\}^\kappa \times \mathcal{FS} \rightarrow \mathcal{RS}_E^{cca}$, $\mathsf{PRF} : \{0,1\}^* \times \mathcal{FS} \rightarrow \{0,1\}^\kappa$, a key derivation function $\mathsf{KDF} : Salt \times \mathcal{KS} \rightarrow \mathcal{FS}$, and $s \in_R Salt$.

$\mathcal{D}_1$ generates (ek_i, dk_i) for each server U_i including (ek_S^*, dk_S^*) of U_S^* according to the protocol, publishes ek_i, gives dk_S^* to $\mathcal{A}$, and sets $cert_{ek_i} = (ID_i, ek_i)$ as a certificate for each server U_i. $\mathcal{D}_1$ poses ID_S^* to the oracle F^*, receives r_0^* as a challenge, and computes $(C_1^*, K_1^*) \leftarrow \mathsf{EnCap}(ek_S^*; r_1^*)$ by using $r_1'^* \in_R FS$, and $C_0^* \leftarrow \mathsf{wEnCapC}(r_0^*)$ according to the protocol for the i^*-th session of U_C^*.

[simulation]

$\mathcal{D}_1$ keeps the list L_{SK} that contains queries and answers of $\mathsf{SessionKeyReveal}$. $\mathcal{D}_1$ simulates oracle queries by $\mathcal{A}$ as follows.

1. $\mathsf{Send}(params, pid)$: If the session is the i^*-th session of U_C^*, then $\mathcal{D}_1$ sets $K_1 = K_1^*$, returns (C_0^*, C_1^*, ID_S^*), and records $(\Pi, ID = pid, (C_0^*, C_1^*), (*, *), *, K_1)$ in L_{SK}. Otherwise, $\mathcal{D}_1$ chooses $((C_0, C_1), r_C)$ from the unused key pairs and returns it, or computes $((C_0, C_1), r_C)$ according to the protocol and returns it, and records $(\Pi, ID = pid, (C_0, C_1), (*, *), *, K_1)$ in L_{SK}.
2. $\mathsf{Send}(sid, msg = (C_0, C_1, id))$: If $msg = (C_0^*, C_1^*, ID_S^*)$, then $\mathcal{D}_1$ sets $K_1 = K_1^*$, chooses (ek_T^*, dk_T^*) from the unused key pairs and returns ek_T^*, or generates (ek_T^*, dk_T^*) according to the protocol and return ek_T^*, computes SK, and records $(\Pi, ID = id, (C_0^*, C_1^*), (ek_T^*, ek_S^*), K_0, K_1)$ and SK as a completed session in L_{SK}. Otherwise, $\mathcal{D}_1$ chooses (ek_T, dk_T) from the unused key pairs and returns it, or generates ek_T according to the protocol and returns it, computes SK, and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}.
3. $\mathsf{Send}(sid, msg = ek_T)$: If the session is the i^*-th session of U_C^*, $\mathcal{D}_1$ computes $K_0^* \leftarrow \mathsf{wEnCapK}(ek_T, C_0^*, r_0^*)$, sets $K_0 = K_0^*$, computes SK according to the protocol, and records $(\Pi, ID = id, (C_0^*, C_1^*), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}. Otherwise, $\mathcal{D}_1$ computes SK according to the protocol and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}.
4. $\mathsf{SessionKeyReveal}(sid)$:
   (a) If sid is not completed, then $\mathcal{D}_1$ returns error.
   (b) Otherwise, $\mathcal{D}_1$ returns SK as recorded in L_{SK}.
5. $\mathsf{Partner}(X)$: $\mathcal{D}_1$ returns the secret value x of the public value X as defined.
6. $\mathsf{RevealNext}()$: $\mathcal{D}_1$ generates a key pair (ESK,EPK), keeps it as unused, and returns the EPK to $\mathcal{A}$ as defined.
7. $\mathsf{EstablishCertificate}(ID_i, X)$: $\mathcal{D}_1$ registers the public key of ID_i as X according to the protocol, and marks U_i as a dishonest party.
8. $\mathsf{Test}(sid)$: $\mathcal{D}_1$ returns as defined.
9. $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$. If $\mathcal{A}$ outputs $b' = 0$, then $\mathcal{D}_1$ outputs that $\mathsf{F}^* = \mathsf{F}_0'$, otherwise $\mathcal{D}_1$ outputs that $\mathsf{F}^* = \mathsf{RF}$.

[Analysis]

For $\mathcal{A}$, the simulation by $\mathcal{D}_1$ is the same as the experiment $\mathbf{H_3}$ if $\mathsf{F}^* = \mathsf{F}_0'$. Otherwise, the simulation by $\mathcal{D}_1$ is the same as the experiment $\mathbf{H_4}$. Thus, since the advantage of $\mathcal{D}_1$ is negligible due to the security of the PRF, $|Adv(\mathcal{A}, \mathbf{H_4}) - Adv(\mathcal{A}, \mathbf{H_3})| \leq negl$.

Hybrid experiment $\mathbf{H_5}$: This experiment changes the way of the computing K_0^* on the client side in the i^*-th session of U_C^*. Instead of computing $K_0^* \leftarrow \mathsf{wEnCapK}(ek_T^*, C_0^*; r_0^*)$, it is changed as $K_0^* \in_R \mathcal{KS}_{cpa}$.

We construct an IND-CPA adversary $\mathcal{S}$ from $\mathcal{A}$ in $\mathbf{H_4}$ or $\mathbf{H_5}$. The $\mathcal{S}$ performs the following steps.

[init]

$\mathcal{S}$ receives ek_T^* from the challenger as a challenge.

[setup]

$\mathcal{S}$ chooses pseudo-random functions $\mathsf{F}' : \{0,1\}^\kappa \times \mathcal{FS} \to \mathcal{FS}$, $\mathsf{F}_0' : \{0,1\}^\kappa \times \mathcal{FS} \to \mathcal{RS}_E^{cpa}$, $\mathsf{F}_1' : \{0,1\}^\kappa \times \mathcal{FS} \to \mathcal{RS}_E^{cca}$, $\mathsf{PRF} : \{0,1\}^* \times \mathcal{FS} \to \{0,1\}^\kappa$, a key derivation function $\mathsf{KDF} : Salt \times \mathcal{KS} \to \mathcal{FS}$, and $s \in_R Salt$.

$\mathcal{S}$ generates (ek_i, dk_i) for each server U_i including (ek_S^*, dk_S^*) of U_S^* according to the protocol, publishes ek_i, gives dk_S^* to $\mathcal{A}$, and sets $cert_{ek_i} = (ID_i, ek_i)$ as a certificate for each server U_i. $\mathcal{S}$ receives (K_b^*, C_0^*) as a challenge and sets $C_0 = C_0^*$ for the i^*-th session of U_C^*.

[simulation]

$\mathcal{S}$ keeps the list L_{SK} that contains queries and answers of $\mathsf{SessionKeyReveal}$. $\mathcal{S}$ simulates oracle queries by $\mathcal{A}$ as follows.

1. $\mathsf{Send}(params, pid)$: If the session is the i^*-th session of U_C^*, then $\mathcal{S}$ sets $C_0 = C_0^*$, computes $(C_1, K_1) \leftarrow \mathsf{EnCap}(ek_S^*; r_1^*)$, where $r_1^* \in_R \mathcal{RS}_E^{cca}$. Also $\mathcal{S}$ returns (C_0, C_1, ID_S^*) according to the protocol, and records $(\Pi, ID = id, (C_0, C_1), (*, *), *, K_1)$ in L_{SK}. Otherwise, $\mathcal{S}$ chooses $((C_0, C_1), r_C)$ from the unused key pairs and returns (C_0, C_1), or computes $((C_0, C_1), r_C)$ according to the protocol and returns (C_0, C_1), and records $(\Pi, ID = id, (C_0, C_1), (*, *), *, K_1)$ in L_{SK}.
2. $\mathsf{Send}(sid, msg = (C_0, C_1, id))$: If $msg = (C_0^*, C_1^*, ID_S^*)$, then $\mathcal{S}$ returns ek_T^*, sets $K_0 = K_b^*$, computes SK, and records $(\Pi, ID = id, (C_0^*, C_1^*), (ek_T^*, ek_S^*), K_0, K_1)$ and SK as a completed session in L_{SK}. Otherwise, $\mathcal{S}$ chooses (ek_T, dk_T) from the unused key pairs and returns ek_T, or generates (ek_T, dk_T) according to the protocol and returns ek_T, computes SK, and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}.
3. $\mathsf{Send}(sid, msg = ek_T)$: If the session is the i^*-th session of U_C^*, $\mathcal{S}$ sets $K_0 = K_b^*$, computes SK according to the protocol, and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}. Otherwise, $\mathcal{S}$ computes SK according to the protocol and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}.
4. $\mathsf{SessionKeyReveal}(sid)$:
5. $\mathsf{SessionKeyReveal}(sid)$:
   (a) If sid is not completed, then $\mathcal{S}$ returns error.
   (b) Otherwise, $\mathcal{S}$ returns SK as recorded in L_{SK}.
6. $\mathsf{Partner}(X)$: $\mathcal{S}$ returns the secret value x of the public value X as defined.

7. $\mathsf{RevealNext}()$: $\mathcal{S}$ generates a key pair (ESK,EPK), keeps it as unused, and returns the EPK to $\mathcal{A}$ as defined.
8. $\mathsf{EstablishCertificate}(ID_i, X)$: $\mathcal{S}$ registers the public key of ID_i as X according to the protocol, and marks U_i as a dishonest party.
9. $\mathsf{Test}(sid)$: $\mathcal{S}$ returns as defined.
10. $\mathcal{A}$ outputs a guess $b^{'} \in \{0,1\}$. If $\mathcal{A}$ outputs $b^{'}$, then $\mathcal{S}$ outputs $b^{'}$.

[Analysis]

For $\mathcal{A}$, the simulation by $\mathcal{S}$ is same the as the experiment $\mathbf{H_4}$ if the challenge is (C_0^*, K_0^*). Otherwise, the simulation by $\mathcal{S}$ is same the as the experiment $\mathbf{H_5}$. Thus, since the advantage of $\mathcal{S}$ is negligible due to the security of the IND-CPA secure KEM, $|Adv(\mathcal{A}, \mathbf{H_5}) - Adv(\mathcal{A}, \mathbf{H_4})| \leq negl$.

Hybrid experiment $\mathbf{H_6}$: This experiment changes the way of the computation of the $K_0^{'*}$ in the i^*-th session. Instead of computing $K_0^{'*} \leftarrow \mathsf{KDF}(s, K_0^*)$, it is changed as choosing $K_0^{'*} \in_R \mathcal{FS}$.

Since K_0^* is randomly chosen in $\mathbf{H_5}$, it has sufficient min-entropy because the PKIC-KEM is κ-min-entropy KEM. Thus, by the definition of the KDF, $|Adv(\mathcal{A}, \mathbf{H_6}) - Adv(\mathcal{A}, \mathbf{H_5})| \leq negl$.

Hybrid experiment $\mathbf{H_7}$: This experiment changes the way of the computation of SK in the i^*-th session of U_C^*. Instead of computing $SK = \mathsf{PRF}(sid, K_1) \oplus \mathsf{PRF}(sid, K_0)$, it is changed as $SK = \mathsf{PRF}(sid, K_1) \oplus x$, where $x \in_R \{0,1\}^\kappa$.

We construct a distinguisher $\mathcal{D}_2$ that distinguishes if F^* is either a pseudo-random function PRF or a random function RF from $\mathcal{A}$ in $\mathbf{H_6}$ or $\mathbf{H_7}$. $\mathcal{D}_2$ performs the following steps.

[setup]

$\mathcal{D}_2$ is given a pseudo-random function $\mathsf{PRF} : \{0,1\}^* \times \mathcal{FS} \to \{0,1\}^\kappa$. Then, $\mathcal{D}_2$ chooses pseudo-random functions $\mathsf{F}^{'} : \{0,1\}^\kappa \times \mathcal{FS} \to \mathcal{FS}$, $\mathsf{F}_0^{'} : \{0,1\}^\kappa \times \mathcal{FS} \to \mathcal{RS}_E^{cpa}$, $\mathsf{F}_1^{'} : \{0,1\}^\kappa \times \mathcal{FS} \to \mathcal{RS}_E^{cca}$, a key derivation function $\mathsf{KDF} : Salt \times \mathcal{KS} \to \mathcal{FS}$, and $s \in_R Salt$.

$\mathcal{D}_2$ generates (ek_i, dk_i) for each server U_i including (ek_S^*, dk_S^*) of U_S^* according to the protocol, publishes ek_i, gives dk_S^* to $\mathcal{A}$, and sets $cert_{ek_i} = (ID_i, ek_i)$ as a certificate for each server U_i^*.

[simulation]

$\mathcal{D}_2$ keeps the list L_{SK} that contains queries and answers of $\mathsf{SessionKeyReveal}$. $\mathcal{D}_2$ simulates oracle queries by $\mathcal{A}$ as follows.

1. $\mathsf{Send}(params, pid)$: If the session is the i^*-th session of U_C^*, then $\mathcal{D}_2$ computes $(C_1^*, K_1^*) \leftarrow \mathsf{EnCap}(ek_S^*; r_1^*)$ and $C_0^* \leftarrow \mathsf{wEnCapC}(r_0^*)$ where $r_1^* \leftarrow \mathsf{F}_1^{'}(ID_S^*, r_1^{'*})$ and $r_0^* \in_R \mathcal{RS}_E^{cpa}$, returns (C_0^*, C_1^*, ID_S^*), and records $(\Pi, ID = id, (C_0^*, C_1^*), (*,*), *, K_1^*)$ in L_{SK}. Otherwise, $\mathcal{D}_2$ chooses $((C_0, C_1), r_C)$ from the unused key pairs and returns (C_0, C_1), or computes $((C_0, C_1), r_C)$ according to the protocol and returns (C_0, C_1), and records $(\Pi, ID = id, (C_0, C_1), (*,*), *, K_1)$ in L_{SK}.

2. $\mathsf{Send}(sid, msg = (C_0, C_1, id))$: If $msg = (C_0^*, C_1^*, ID_S^*)$, then $\mathcal{D}_2$ chooses (ek_T^*, dk_T^*) from the unused key pairs and returns ek_T^*, or generates (ek_T^*, dk_T^*) and return ek_T^*, sets sid according to the protocol, poses it to the oracle (PRF or RF), obtains $x \in \{0,1\}^\kappa$. Also, $\mathcal{D}_2$ sets $SK^* = \mathsf{PRF}(sid, K_1) \oplus x$ and records $(\Pi, ID = id, (C_0^*, C_1^*), (ek_T^*, ek_S^*), K_0, K_1)$ and SK^* as a completed session in L_{SK}. Otherwise, $\mathcal{D}_2$ chooses (ek_T, dk_T) from the unused key pairs and returns ek_T, or generates (ek_T, dk_T) according to the protocol and returns ek_T, computes SK, and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}.
3. $\mathsf{Send}(sid, msg = ek_T)$: If the session is the i^*-th session of U_C^*, $\mathcal{D}_2$ sets sid according to the protocol, poses it to the oracle (PRF or RF), and obtains $x \in \{0,1\}^{\ kappa}$. Also, $\mathcal{D}_2$ computes $SK^* = \mathsf{PRF}(sid, K_1) \oplus x$ and records $(\Pi, ID = id, (C_0^*, C_1^*), (ek_T, ek_S))$ and SK^* as a completed session in L_{SK}. Otherwise, $\mathcal{D}_2$ computes SK according to the protocol and records $(\Pi, ID = id, (C_0, C_1), (ek_T, ek_S), K_0, K_1)$ and SK as a completed session in L_{SK}.
4. $\mathsf{SessionKeyReveal}(sid)$:
   (a) If sid is not completed, then $\mathcal{D}_2$ returns error.
   (b) Otherwise, $\mathcal{D}_2$ returns SK as recorded in L_{SK}.
5. $\mathsf{Partner}(X)$: $\mathcal{D}_2$ returns the secret value x of the public value X as defined.
6. $\mathsf{RevealNext}()$: $\mathcal{D}_2$ generates a key pair (ESK,EPK), keeps it as unused, and returns the EPK to $\mathcal{A}$ as defined.
7. $\mathsf{EstablishCertificate}(ID_i, X)$: $\mathcal{D}_2$ registers the public key of ID_i as X according to the protocol, and marks U_i as a dishonest party.
8. $\mathsf{Test}(sid)$: $\mathcal{D}_2$ returns as defined.
9. $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$. If $\mathcal{A}$ outputs b', then $\mathcal{D}_2$ outputs b'.

[Analysis]

For $\mathcal{A}$, the simulation by $\mathcal{D}_2$ is the same as the experiment $\mathbf{H_6}$ if $\mathsf{F}^* = \mathsf{PRF}$. Otherwise, the simulation by $\mathcal{D}_2$ is the same as the experiment $\mathbf{H_7}$. Thus, since the advantage of $\mathcal{D}_2$ is negligible due to the security of PRF, $|Adv(\mathcal{A}, \mathbf{H_7}) - Adv(\mathcal{A}, \mathbf{H_6})| \leq negl$.

In $\mathbf{H_7}$, the session key in the test session is perfectly randomized. This gives $\mathcal{A}$ no information from the Test query, therefore $Adv(\mathcal{A}, \mathbf{H_7}) = 0$ and $\Pr[E_2 \wedge Sec] = negl$.

□

B Proof of Theorem 3

Proof. *Suc* denotes the event that $\mathcal{A}$ wins. We consider the following events that cover all cases of the behavior $\mathcal{A}$.

-E_1: The ephemeral secret key dk_T^* of the server is revealed.
-E_2: The static secret key dk_S^* of the server is revealed.

Let κ be a security parameter. In the OS-AKE security game, sid^* is a sid of the test session, the maximum number of parties is n, and the maximum

ℓ session is activated. Let the adversary $\mathcal{A}$ be a probabilistic polynomial-time adversary in κ, and construct the OW-CCA or OW-CPA adversary $\mathcal{S}$ from $\mathcal{A}$ that performs the OS-AKE security game. We construct $\mathcal{S}$ with a non-negligible probability of a successful attack using the adversary $\mathcal{A}$. Also, let $A_{SK}H$ be an event in which $\mathcal{A}$ queries H for (sid^*, K_0^*, K_1^*), and let $\overline{A_{SK}H}$ be a complement of $A_{SK}H$.

Let sid be a completed session such as $sid \neq sid^*$ owned by the honest party. Since $sid \neq sid^*$, the input to the key derivation function H is also different. Since H is a random oracle, $\mathcal{A}$ cannot obtain any information from the session key of sid^* about the test session. Thus, $\Pr[Suc \wedge \overline{A_{SK}H}] \leq 1/2$ and $\Pr[Suc] = \Pr[Suc \wedge A_{SK}H] + \Pr[Suc \wedge \overline{A_{SK}H}] \leq \Pr[Suc \wedge A_{SK}H] + 1/2$. Henceforth, the event $Suc \wedge A_{SK}H$ is denoted by Suc^*.

We consider the following events.

-$A_{SK}S$: When U_C^* is the owner of sid^*, $\mathcal{A}$ queries H_0 and H_1 for r_C^*.
-$\overline{A_{SK}S}$: Complement of $A_{SK}S$.

To finish the proof, we investigate events $A_{SK}S \wedge Suc^*$ and $E_i \wedge \overline{A_{SK}S} \wedge Suc^* (i = 1, 2)$ that cover all cases of event Suc^*.

Event $A_{SK}S \wedge Suc^*$ *:* $\mathcal{S}$ receives the challenge ciphertext (C_0^*, C_1^*) corresponding to K_0^* and K_1^*. In this event, $\mathcal{A}$ queries H_0 and H_1 for the ephemeral secret key r_C^*. $\mathcal{S}$ obtains the corresponding K^*. Since $\mathcal{A}$ cannot reveal r_C^* by the freshness and the probability of $\mathcal{A}$ querying r_C^* is $1/2^\kappa$, the probability of occurring this event is $1/2^\kappa$.

Event $E_1 \wedge \overline{A_{SK}S} \wedge Suc^*$ *:* In event E_1, it gives $\mathcal{A}$ the ephemeral secret key dk_T^* of the test session. In event $E_1 \wedge \overline{A_{SK}S} \wedge Suc$, we construct the OW-CCA adversary $\mathcal{S}$.

[init]

$\mathcal{S}$ receives ek_S^* from the challenger as a challenge. Also, $\mathcal{S}$ receives C_1^* as a challenge.

[setup]

$\mathcal{S}$ chooses party U_S^* and party U_C^* randomly, integer $i^* \in [1, \ell]$, and fixes the session and the party of the client-server that is the target of the Test query. The probability that the guess of this test session matches is $1/n^2\ell$.

$\mathcal{S}$ sets $ek_S = ek_S^*$ of U_S and $C_1 = C_1^*$ for the i^*-th session of U_C^* (partner is U_S^*). Also, $\mathcal{S}$ generates (ek_i, dk_i) for each server U_i other than U_S^*, and publishes ek_i. Then, $\mathcal{S}$ sets $cert_{ek_i} = (ID_i, ek_i)$ as the certificate for each server U_i.

[simulation]

$\mathcal{S}$ has lists $L_{\mathsf{H}_0}, L_{\mathsf{H}_1}, L_{\mathsf{H}}$ that keep the queries and answers of each $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}$ oracle, and a list L_{SK} that holds the answers of $\mathsf{SessionKeyReveal}$. For some $sid = (ID, (C_0, C_1), ek_T)$ and SK, it records (r_C, r_0) in L_{H_0}, (r_C, r_1) in L_{H_1}, (sid, K_0, K_1, SK) in L_{H}, and (Π, sid, K_0, K_1, SK) in L_{SK}.

$\mathcal{S}$ simulates oracle queries by $\mathcal{A}$ as follows.

1. $H_1(r_C)$: If $(r_C, *)$ is recorded in L_{H_1}, then $\mathcal{S}$ returns the recorded value. Otherwise, $\mathcal{S}$ chooses $r_1 \in_R \mathcal{RS}_E^{cca}$, returns it, and records it in L_{H_1}.
2. $H_0(r_C)$: If $(r_C, *)$ is recorded in L_{H_0}, then $\mathcal{S}$ returns the recorded value. Otherwise, $\mathcal{S}$ chooses $r_0 \in_R \mathcal{RS}_E^{cpa}$, returns it, and records it in L_{H_0}.
3. $H(sid, K_0, K_1)$:
   (a) If $ID = ID_S^*, C_1 = C_1^*$ and it is the i^*-th session of U_C^*, then $\mathcal{S}$ outputs K_1 as an answer K^* of the OW-CCA game.
   (b) If (sid, K_0, K_1) is recorded in L_{H}, then $\mathcal{S}$ returns the recorded value SK.
   (c) If sid is recorded in L_{SK}, then $\mathcal{S}$ returns the recorded value SK and records (sid, K_0, K_1, SK) in L_{H}.
   (d) Otherwise, $\mathcal{S}$ returns a random value $SK \in_R \{0,1\}^\kappa$ and records (sid, K_0, K_1, SK) in L_{H}.
4. $\mathsf{Send}(params, pid)$: If the session is the i^*-th session of U_C^*, $\mathcal{S}$ computes C_0^* according to the protocol, sets $C_1 = C_1^*, C_0 = C_0^*$, and returns (C_0, C_1, ID_S^*). Otherwise, $\mathcal{S}$ chooses $((C_0, C_1), r_C)$ from the unused key pairs and returns (C_0, C_1), or computes $((C_0, C_1), r_C)$ according to the protocol and returns (C_0, C_1), and records $(\Pi, sid = (ID = pid, (C_0, C_1), *), *, K_1, *)$ in L_{SK}.
5. $\mathsf{Send}(sid, msg = (C_0, C_1, id))$: If $msg = (C_0^*, C_1^*, ID_S^*)$, then $\mathcal{S}$ chooses (ek_T^*, dk_T^*) from the unused key pair and returns it, or generates (ek_T^*, dk_T^*) according to the protocol and returns it. Otherwise, $\mathcal{S}$ chooses (ek_T, dk_T) from the unused key pairs and returns ek_T, or computes (ek_T, dk_T) according to the protocol and returns ek_T. Also, the session key SK is computed as follows.
   (a) If $id = ID_S^*$, $\mathcal{S}$ decrypts C_0 with dk_T, computes K_0, and poses C_1 to the decryption oracle to obtain K_1. If $(sid = (ID_S^*, (C_0, C_1), ek_T), K_0, K_1, SK)$ is recorded in L_{H}, $\mathcal{S}$ sets SK as the session key.
   (b) $\mathcal{S}$ decrypts (C_0, C_1) with (dk_T, dk_S), computes (K_0, K_1) respectively. If $(sid = (id, (C_0, C_1), ek_T), K_0, K_1, SK)$ is recorded in L_{H}, then $\mathcal{S}$ sets SK as the session key. Otherwise, $\mathcal{S}$ chooses $SK \in_R \{0,1\}^\kappa$ and sets it as the session key.

   Finally, $\mathcal{S}$ records $(\Pi, sid = (ID = id, (C_0, C_1), ek_T), K_0, K_1, SK)$ as a completed session in L_{SK}.
6. $\mathsf{Send}(sid, msg = ek_T)$: $\mathcal{S}$ computes K_0 according to the protocol. If $(sid = (pid, (C_0, C_1), ek_T), K_0, K_1, SK)$ is recorded in L_{H}, then SK is set as the session key. Otherwise, $\mathcal{S}$ chooses $SK \in_R \{0,1\}^\kappa$. Then, $\mathcal{S}$ records $(\Pi, sid = (ID = id, (C_0, C_1), ek_T), K_0, K_1, SK)$ as a completed session in L_{SK}.
7. $\mathsf{SessionKeyReveal}(sid)$:
   (a) If sid is not completed, then $\mathcal{S}$ returns error.
   (b) If sid is recorded, then $\mathcal{S}$ returns SK recorded in L_{SK}.
   (c) Otherwise, $\mathcal{S}$ chooses $SK \in_R \{0,1\}^\kappa$, returns it, and records it in L_{SK}.
8. $\mathsf{Partner}(X)$: $\mathcal{S}$ returns the secret value x of the public value X according to the definition.
9. $\mathsf{RevealNext}()$: $\mathcal{S}$ generates a key pair (ESK,EPK) as defined, keeps it as unused, and returns the EPK to $\mathcal{A}$.

10. $\mathsf{EstablishCertificate}(ID_i, X) : \mathcal{S}$ registers the public key of ID_i as X according to the protocol, and marks U_i as a dishonest party.
11. $\mathsf{Test}(sid)$: It aborts except for the i^*-th session of U_C. Otherwise, $\mathcal{S}$ returns as defined.
12. If $\mathcal{A}$ outputs the guess $b' \in \{0, 1\}$, then $\mathcal{S}$ aborts.

[Analysis]

The simulation for $\mathcal{S}$ is perfect except with negligible probability. The probability that $\mathcal{A}$ selects the session as the test session sid^* at least $1/n^2\ell$.

In event Suc^*, $\mathcal{A}$ queries H correctly for (sid^*, K_0^*, K_1^*). Therefore, $\mathcal{S}$ is succesful and does not abort.

Thus, $\mathcal{S}$ is successful with non-negligible probability.

Event $E_2 \wedge \overline{A_{SK}S} \wedge Suc^*$ *:* In the event E_2, it gives $\mathcal{A}$ the static secret key dk_S^* of the test session. In event $E_2 \wedge \overline{A_{SK}S} \wedge Suc^*$, we construct the OW-CPA adversary $\mathcal{S}$.

[init]

$\mathcal{S}$ receives ek_T^* from the challenger as a challenge. Also, $\mathcal{S}$ receives C_0^* as a challenge.

[setup]

$\mathcal{S}$ chooses party U_S^* and party U_C^* randomly, integer $i^* \in [1, \ell]$, and fixes the session and the party of the client-server that is the target of the Test query. The probability that the guess of this test session matches is $1/n^2\ell$.

$\mathcal{S}$ sets $ek_T = ek_T^*$ and $C_0 = C_0^*$ for the i^*-th session of U_C^* (partner is U_S^*). Also, $\mathcal{S}$ generates (ek_i, dk_i) for all servers U_i and publishes ek_i. $\mathcal{S}$ sets $cert_{ek_i} = (ID_i, ek_i)$ as the certificate for each server U_i and gives dk_S^* of U_S^* to $\mathcal{A}$.

[simulation]

$\mathcal{S}$ has lists $L_{\mathsf{H}_0}, L_{\mathsf{H}_1}, L_{\mathsf{H}}$ that keep the queries and answers of each $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}$ oracle, and a list L_{SK} that holds the answers of $\mathsf{SessionKeyReveal}$. For some $sid = (ID, (C_0, C_1), ek_T)$ and SK, it records (r_C, r_0) in L_{H_0}, (r_C, r_1) in L_{H_1}, (sid, K_0, K_1, SK) in L_{H}, and (Π, sid, K_0, K_1, SK) in L_{SK}.

$\mathcal{S}$ simulates oracle queries by $\mathcal{A}$ as follows.

1. $H_1(r_C)$: If $(r_C, *)$ is recorded in L_{H_1}, then $\mathcal{S}$ returns the recorded value. Otherwise, $\mathcal{S}$ chooses $r_1 \in_R \mathcal{RS}_E^{cca}$, returns it, and records it in L_{H_1}.
2. $H_0(r_C)$: If $(r_C, *)$ is recorded in L_{H_0}, then $\mathcal{S}$ returns the recorded value. Otherwise, $\mathcal{S}$ chooses $r_0 \in_R \mathcal{RS}_E^{cpa}$, returns it, and records it in L_{H_0}.
3. $H(sid, K_0, K_1)$:
   (a) If $ID = ID_S^*, C_0 = C_0^*$ and it is the i^*-th session of U_C^*, then $\mathcal{S}$ outputs K_0 as an answer K^* of the OW-CPA game.
   (b) If (sid, K_0, K_1) is recorded in L_{H}, then $\mathcal{S}$ returns the recorded value SK.
   (c) If sid is recorded in L_{SK}, then $\mathcal{S}$ returns the recorded value SK and records (sid, K_0, K_1, SK) in L_{H}.

(d) Otherwise, $\mathcal{S}$ returns a random value $SK \in_R \{0,1\}^\kappa$ and records (sid, K_0, K_1, SK) in L_{H}.

4. $\mathsf{Send}(params, pid)$: If the session is the i^*-th session of U_C^*, $\mathcal{S}$ computes (C_1^*, K_1^*) according to the protocol, sets $C_0 = C_0^*$ and $C_1 = C_1^*$, returns (C_0, C_1, ID_S^*), and records $(\Pi, sid = (ID = pid, (C_0, C_1), *), *, K_1^*, *)$ in L_{SK}. Otherwise, $\mathcal{S}$ chooses $((C_0, C_1), r_C)$ from the unused key pairs and returns (C_0, C_1), or computes $((C_0, C_1), r_C)$ according to the protocol and returns (C_0, C_1), and records $(\Pi, sid = (ID = pid, (C_0, C_1), *), *, K_1, *)$ in L_{SK}.
5. $\mathsf{Send}(sid, msg = (C_0, C_1, id))$: If $msg = (C_0^*, C_1^*, ID_S^*)$, then $\mathcal{S}$ returns ek_T^*. Otherwise, $\mathcal{S}$ chooses (ek_T, dk_T) from the unused key pairs and returns ek_T, or computes (ek_T, dk_T) according to the protocol and returns ek_T. Then, $\mathcal{S}$ computes (K_0, K_1) according to the protocol. If $(sid = (id, (C_0, C_1), ek_T), K_0, K_1, SK)$ is recorded in L_{H}, then S sets SK as the session key. Otherwise, $\mathcal{S}$ chooses $SK \in_R \{0,1\}^\kappa$ and sets it as the session key. Then, $\mathcal{S}$ records $(\Pi, sid = (ID = id, (C_0, C_1), ek_T), K_0, K_1, SK)$ as a completed session in L_{SK}.
6. $\mathsf{Send}(sid, msg = ek_T)$: $\mathcal{S}$ computes K_0 according to the protocol. If $(sid = (pid, (C_0, C_1), ek_T), K_0, K_1, SK)$ is recorded in L_{H}, then $\mathcal{S}$ sets SK as the session key. Otherwise, $\mathcal{S}$ chooses $SK \in_R \{0,1\}^\kappa$. Then, $\mathcal{S}$ records $(\Pi, sid = (ID = id, (C_0, C_1), ek_T), K_0, K_1, SK)$ as a completed session in L_{SK}.
7. $\mathsf{SessionKeyReveal}(sid)$:
   (a) If sid is not completed, then $\mathcal{S}$ returns error.
   (b) If sid is recorded, then $\mathcal{S}$ returns SK recorded in L_{SK}.
   (c) Otherwise, $\mathcal{S}$ chooses $SK \in_R \{0,1\}^\kappa$, returns it, and records it in L_{SK}.
8. $\mathsf{Partner}(X)$: $\mathcal{S}$ returns the secret value x of the public value X as defined.
9. $\mathsf{RevealNext}()$: $\mathcal{S}$ generates a key pair (ESK,EPK) , keeps it as unused as defined, and returns the EPK to $\mathcal{A}$.
10. $\mathsf{EstablishCertificate}(ID_i, X)$: $\mathcal{S}$ registers the public key of ID_i as X according to the protocol, and marks U_i as a dishonest party.
11. $\mathsf{Test}(sid)$: It aborts except for the i^*-th session of U_C. Otherwise, $\mathcal{S}$ returns as defined.
12. If $\mathcal{A}$ outputs the guess $b' \in \{0,1\}$, then $\mathcal{S}$ aborts.

[Analysis]

The simulation for $\mathcal{S}$ is perfect except with negligible probability. The probability that $\mathcal{A}$ selects the session as the test session sid^* at least $1/n^2\ell$.

In event Suc^*, $\mathcal{A}$ queries H correctly for (sid^*, K_0^*, K_1^*). Therefore, $\mathcal{S}$ is succesful and does not abort.

Thus, $\mathcal{S}$ is successful with non-negligible probability.

□

C Proof of Theorem 4

Proof. We proceed by introducing another experiment, in which the adversary cannot win more than random guessing. In this new experiment, the choice of

i^* is independent of the behavior of the rest of the system. Then, we show that no adversary can distinguish this new experiment from the original experiment, thereby showing the OS-anonymity of the protocol.

$\mathsf{Expt}^{OS-anon'}_{GC-ROM}(\mathcal{A})$ is the same experiment as $\mathsf{Expt}^{OS-anon}_{GC-ROM}(\mathcal{A})$ except for the following oracle used by the challenger $\mathcal{C}$.

- $\mathsf{Start}'(i_0, i_1, params, pid = ID^*_S) \to msg'$:
  1. If $i_0 = i_1$, then abort.
  2. Set $i^* \leftarrow_R \{i_0, i_1\}$.
  3. Set $ID^* \leftarrow ID^*_S$.
  4. Choose $((C^*_0, C^*_1), r^*_C)$ from the unused key pairs and return (C^*_0, C^*_1, ID^*).
- $\mathsf{Send}'(sid, msg = ek^*_T)$:
  1. Compute $r^*_0 \leftarrow \mathsf{H}_0(r^*_C)$.
  2. Compute $K^*_0 \leftarrow \mathsf{wEnCapK}(ek^*_T, C^*_0, r^*_0)$.
  3. Compute SK according to the protocol.
- $\mathsf{SessionKeyReveal}'() \to SK$: If the test session is a completed session, return SK.
- $\mathsf{Partner}'(C^*) \to r^*_C$: Return the secret value r^*_C corresponding to C^*.
- $\mathsf{RevealNext}' \to X$: Return the future public value X and record it as unused.

Since all messages computed in the OS-anon' game are independent of the choice of i^*, the adversary $\mathcal{A}$ has no advantage, thus the probability that $\mathcal{A}$ wins the game is as follows.

$$\Pr[\mathsf{Expt}^{OS-anon'}_{GC-ROM}(\mathcal{A}) = win] = 1/2 \tag{3}$$

Also, the distribution of messages returned by the challenger in the OS-anon' game is the same as that returned in the OS-anon game. Furthermore, messages from all parties except P_{i_0} and P_{i_1} are unchanged. For messages from P_{i_0} and P_{i_1} in the OS-anon' game, all queries return messages of the same distribution as in the OS-anon game.

Here, queries that reveal information about whether P_{i_0} or P_{i_1} participated in the test session are prohibited by the definition. For example, $\mathcal{A}$ is prohibited from using a $\mathsf{SessionKeyReveal}(sid)$ query to P_{i_0} to find out if P_{i_0} has the session key for the test session.

Thus, $\mathcal{A}$ cannot distinguish between the two games.

$$\Pr[\mathsf{Expt}^{OS-anon'}_{GC-ROM}(\mathcal{A}) = win] = \Pr[\mathsf{Expt}^{OS-anon}_{GC-ROM}(\mathcal{A}) = win] \tag{4}$$

From equations (3) and (4), the scheme has one-sided anonymity. □