Electronic Preprint for Journal of Information Processing Vol.24 No.1

Regular Paper

# One-Round Authenticated Key Exchange without Implementation Tricks

Kazuki Yoneyama[1,a)]

Received: February 27, 2015, Accepted: September 2, 2015

**Abstract:** Fujioka et al. proposed the first generic construction (FSXY construction) of exposure-resilient authenticated key exchange (AKE) from a key encapsulation mechanism (KEM) without random oracles. However, the FSXY construction implicitly assumes that some intermediate computation result is never exposed though other secret information can be exposed. This is a kind of physical assumption, and an implementation trick (i.e., some on-line computation is executed in a special tamper-proof module) is necessary to achieve the assumption. Such a trick is very costly and may be missed by human errors in implementation. From the viewpoint of the human factor, it is desirable to avoid using complicated implementation tricks. In this paper, we introduce a new generic construction without implementation tricks. Our construction satisfies the same security model as the FSXY construction without increasing communication complexity. Moreover, it has another advantage that the protocol can be executed in one-round while the FSXY construction is a sequential two-move protocol. Our key idea is to use KEM with public-key-independent-ciphertext, which allows parties to be able to generate a ciphertext without depending on encryption keys.

**Keywords:** authenticated key exchange, NAXOS trick, key encapsulation mechanism, exposure-resilience

## 1. Introduction

### 1.1 Background

Authenticated Key Exchange (AKE) is a cryptographic primitive for sharing a common *session key* among multiple parties through unauthenticated networks such as the Internet. In the ordinary PKI-based setting, each party locally keeps his own *static secret key* (SSK) and publishes a *static public key* (SPK) corresponding to the SSK. The validity of SPKs is guaranteed by a certificate authority. In a key exchange session, each party generates some session-specific randomness. We call such a randomness an *ephemeral secret key* (ESK) of the party. Using the ESK, each party computes and sends an *ephemeral public key* (EPK) corresponding to the ESK. A session key is derived from these keys with a *key derivation procedure*. Parties can establish a secure channel with the session key. For example, in the Diffie-Hellman (DH) key exchange [9], the ESK of party $A$ (resp. $B$) is randomness $x$ (resp. $y$) and the EPK of party $A$ (resp. $B$) is $g^x$ (resp. $g^y$). Note that the SSK and SPK are not used in the DH key exchange.

An important research target of this area is to achieve exposure-resilience. That means, even if an adversary learns some of secret keys of parties, generated session keys must be protected. For example, SSKs may be exposed if a party is corrupted, or a device itself (e.g., a smart phone) that records a SSK is physically stolen. In another scenario, ESKs may be exposed if computations inputting an ESK are executed in a memory area of a smart phone, and a malicious developer steals it via a hidden malware that is embedded in some apps. Therefore, it is desirable to guarantee exposure-resilience in a provably secure way.

There are several studies about modeling exposure-resilience in the AKE setting. Canetti and Krawczyk [4] defined the first security model of AKE capturing exposure of SSKs and session state, that is called the *Canetti-Krawczyk* (*CK*) *model*. Session state contains all unprotected session-specific information. Thus, if an intermediate computation is done in some protected area (e.g., tamper-proof module, TPM), the output of the computation is not included in session state. However, the CK model does not allow an adversary to learn any SSKs nor the session state of the target session (called the *test session*). LaMacchia et al. [17] also proposed very strong security models capturing exposure of both SSKs and ESKs, which is called the *extended CK* (*eCK*) *model*. While the eCK model allows an adversary to directly learn SSKs and ESKs of the test session, exposure of the session state is not captured and exact information contained in ESKs is ambiguity. The $CK^+$ *model* [11], [16] combines these two models; that is, an adversary can obtain SSKs and ESKs of the test session, and can learn the session state of other sessions. Note that the eCK model and the $CK^+$ model are not comparable [6], [7].

Many concrete AKE schemes that are secure in these models have been studied. HMQV [16] is one of the most efficient protocols and satisfies the $CK^+$ model. However, the security proof is given in the random oracle model (ROM) under the knowledge-of-exponent assumption [8] which is a widely criticized assumption [20]. Boyd et al. [1], [2] propose a generic construction

[1] Ibaraki University, Hitachi, Ibaraki 316–8511, Japan
[a)] kazuki.yoneyama.sec@vc.ibaraki.ac.jp

This paper is the full version of the extended abstract which appeared in Ref. [30] and this work is done at NTT Secure Platform Laboratories.

(BCGNP construction) of AKE from a key encapsulation mechanism (KEM), that is secure in the CK model in the standard model (StdM). The CK model does not capture exposure of ESKs in the test session. Thus, unfortunately, it is unclear whether the BCGNP construction is secure when the ESK of the test session is exposed. Fujioka et al, [11] show that the BCGNP construction is insecure in the $CK^+$ model, and propose another generic construction (FSXY construction) of AKE from KEM, that is secure in the $CK^+$ model in the StdM.

### 1.2 Motivation

The FSXY construction uses a technique to resist exposure of ESKs, which is called the twisted pseudo-random function (PRF) trick [23]. This trick is essentially the same as the NAXOS trick [17] except with/without random oracles (ROs). Roughly, a party uses $H$(SSK,ESK) to compute an EPK instead of using the ESK directly, where $H$ is some intractable function like ROs. Unless both the SSK and the ESK are exposed, $H$(SSK,ESK) cannot be computed by an adversary even if the ESK is exposed. Thus, the FSXY construction guarantees the security against exposure of ESKs.

However, such a trick has several problems. First, it needs some *implementation trick*, because it is assumed that exposure of $H$(SSK,ESK) never occurs while ESKs may be exposed. A typical implementation is that all computations inputting $H$(SSK,ESK) are executed in a TPM such as a smart card. Without the implementation trick (i.e., $H$(SSK,ESK) is handled by the same manner as ESKs), the twisted PRF trick is not meaningful, and it may lead to an exposure attack (i.e., $H$(SSK,ESK) and ESKs are exposed simultaneously) to the FSXY construction though it is proved in the $CK^+$ model. Since some implementer may miss setting the trick due to human errors, using the implementation trick causes potential vulnerability. The other is an efficiency problem. As discussed above, computations inputting $H$(SSK,ESK) must be executed in a TPM. In the FSXY construction, $H$(SSK,ESK) is used as randomness in generating a ciphertext of chosen ciphertext secure (IND-CCA secure) KEM. This computation must be done *on-line* (i.e., any pre-computation is not possible) because the ciphertext is generated under the public key of the peer of the session, and it can be computed *after* the peer is determined. Therefore, the TPM must process a very heavy on-line computation (i.e., an encryption algorithm of IND-CCA secure KEM) for each session. It is clearly not desirable in practice.

### 1.3 Our Contribution

First, we clarify that the FSXY construction is insecure in the $CK^+$ model if the implementation trick does not work (i.e., the outputs of the twisted PRF are handled in the same manner as ESKs) in Section 3. Specifically, we give a simple attack using exposure of the outputs of the twisted PRF. This fact shows that the FSXY construction essentially needs very heavy on-line computations in a TPM or similar implementation tricks.

Next, we introduce a new generic construction of AKE from KEM, that is secure in the $CK^+$ model without relying on the twisted PRF trick (i.e., no implementation trick is necessary) in Section 4 [*1]. Our key idea is to use *KEM with public-key-independent-ciphertext* (PKIC-KEM) [27]. PKIC-KEM allows that a ciphertext can be generated independently from an encryption key, and a KEM key can be generated with the ciphertext, the encryption key and randomness in generating the ciphertext. While the previous work [27] uses a semantically secure (IND-CPA secure) PKIC-KEM to obtain a one-round AKE scheme, we use IND-CPA secure PKIC-KEM both to resist full ESK exposure and to obtain one-round protocol [*2]. A typical example of IND-CPA secure PKIC-KEM is the ElGamal KEM (i.e., an encryption key is $g^a$, a ciphertext is $g^r$, and the KEM key is $g^{ar}$).

Furthermore, though the FSXY construction adapts a strong randomness extractor as a part of the session key derivation procedure, we can replace it with a weaker building block, a *computational extractor* (cExt). The cExt is a weaker and more efficient primitive than the strong randomness extractor; the output of the cExt is just guaranteed computationally indistinguishable from random value but the strong randomness extractor guarantees statistical indistinguishability. We can prove the security of our construction only with the computational property; thus, we can improve efficiency of the session key derivation procedure. This technique is proposed in Refs. [28], [29]

The previous scheme [27] achieves a stronger security (i.e., in the $CK^+$-$sFS^{NSR}$ model) than our construction and the FSXY construction. The $CK^+$-$sFS^{NSR}$ model contains strong forward secrecy while our construction and the FSXY construction only satisfy weak forward secrecy. Strong forward secrecy guarantees that an adversary cannot recover a session key of a completed session (i.e., a session in which the session key was already established) even if static secret keys are compromised and the adversary is active in the target session. On the other hand, weak forward secrecy only guarantees the case when the adversary is passive in the target session. However, the previous scheme [27] relies on the implementation trick as the FSXY construction. Thus, the TPM must process a very heavy on-line computation.

There are some related works [19], [26] that achieve exposure-resilient AKE schemes in the StdM without implementation tricks. However, these schemes are specific constructions (i.e., not generic construction), and rely on a strong building block, PRFs with pairwise-independent random sources ($\pi$PRF). It is not known how to construct $\pi$PRF concretely. **Table 1** shows a comparison of exposure-resilient AKE schemes without implementation tricks. HMQV is the most efficient but relies on RO. The schemes in Refs. [19], [26] are secure in the StdM but relies on $\pi$PRF. In addition, the scheme in Ref. [26] needs pairing operations. Therefore, our scheme needs less communication cost than the schemes in the StdM, and does not rely on $\pi$PRF.

In Section 4, we show the difference of implementation images of the FSXY construction and our construction in **Fig. 3**.

[*1] Though the attack to the FSXY construction does not work in our construction, the security model is the same as the FSXY construction. The reason is that exact information contained in ESKs is not clear in the model of AKE, and it depends on implementations of schemes. Thus, it is hard to define such an implementation issue in the security model.

[*2] One-round means that parties can send their EPKs independently and simultaneously in two-move protocols.

**Table 1** Comparison of exposure-resilient AKE.

| | Model | implementation tricks? | Resource | Assumption (#parings + #[multi,regular]-exp.) | Computational cost complexity | Communication | |
|---|---|---|---|---|---|---|---|
| Ref. [16] | CK$^+$ | no | ROM | gap DH & KEA1 | $0 + [2, 2]$ | $2\|p\|$ | 512 |
| Ref. [19] | eCK | no | StdM | DDH & $\pi$PRF | $0 + [2, 6]$ | $9\|p\|$ | 2304 |
| Ref. [26] | eCK | no | StdM | DBDH & DLIN & $\pi$PRF | $2 + [2, 8]$ | $12\|p\|$ | 3072 |
| Ref. [11] | CK$^+$ | yes | StdM | DDH | $0 + [4, 12]$ | $8\|p\|$ | 2048 |
| Ref. [27] | CK$^+$-sFS$^{\mathrm{NSR}}$ | yes | StdM | DDH & DBDH & $q$-SDH | $4 + [2, 14]$ | $10\|p\|$ | 2560 |
| **Ours** | CK$^+$ | no | StdM | DDH | $0 + [4, 12]$ | $8\|p\|$ | 2048 |

$|p|$ means the size of a group element. For concreteness, the expected communication complexity for a 128-bit implementation is also given. Note that computational costs are estimated without any pre-computation technique. The instantiation of our construction in this table uses the Cramer-Shoup KEM [5] as IND-CCA secure KEM and the ElGamal KEM as IND-CPA secure PKIC-KEM.

## 2. CK$^+$ Security Model

In this section, we recall the CK$^+$ model [11], [16]. We slightly modify the model to specify one-round protocols. It can be trivially extended to any round protocol.

### 2.1 Notations

Throughout this paper we use the following notations. If $\mathsf{M}$ is a set, then by $m \in_R \mathsf{M}$ we denote that $m$ is sampled uniformly from $\mathsf{M}$. If $\mathcal{R}$ is an algorithm, then by $y \leftarrow \mathcal{R}(x; r)$ we denote that $y$ is output by $\mathcal{R}$ on input $x$ and randomness $r$ (if $\mathcal{R}$ is deterministic, $r$ is empty).

We denote a party by $U_P$, and party $U_P$ and other parties are modeled as probabilistic polynomial-time (PPT) Turing machines w.r.t. security parameter $\kappa$. For party $U_P$, we denote static secret (public) key by $SSK_P$ ($SPK_P$) and ephemeral secret (public) key by $ESK_P$ ($EPK_P$). Party $U_P$ generates its own keys, $ESK_P$ and $EPK_P$, and the static public key $SPK_P$ is linked with $U_P$'s identifier in some systems like PKI [*3].

### 2.2 Session

An invocation of a protocol is called a *session*. Session activation of party $U_P$ is done by an incoming message of the form $(\Pi, U_P, U_{\bar{P}})$, where we equate $\Pi$ with a protocol identifier, and $U_{\bar{P}}$ is the party identifier of the peer. Party $U_P$ outputs $(\Pi, U_P, U_{\bar{P}}, EPK_P)$, receives an incoming message of the form $(\Pi, U_{\bar{P}}, U_P, EPK_{\bar{P}})$ from the peer $U_{\bar{P}}$, and then computes the session key $SK$.

A session of $U_P$ is identified by $\mathsf{sid} = (\Pi, U_P, U_{\bar{P}}, EPK_P)$ or $\mathsf{sid} = (\Pi, U_P, U_{\bar{P}}, EPK_P, EPK_{\bar{P}})$. We say that $U_P$ is the *owner* of session $\mathsf{sid}$, if the second coordinate of session $\mathsf{sid}$ is $U_P$. We say that $U_P$ is the *peer* of session $\mathsf{sid}$, if the third coordinate of session $\mathsf{sid}$ is $U_P$. We say that a session is *completed* if its owner computes the session key. The *matching session* of $(\Pi, U_P, U_{\bar{P}}, EPK_P, EPK_{\bar{P}})$ is session $(\Pi, U_{\bar{P}}, U_P, EPK_P, EPK_{\bar{P}})$ and vice versa.

### 2.3 Adversary

The adversary $\mathcal{A}$, which is modeled as a PPT machine, controls all communications between parties including session activation by performing the following adversary query.

- $\mathsf{Send}$(message): The message has one of the following forms: $(\Pi, U_P, U_{\bar{P}})$, or $(\Pi, U_{\bar{P}}, U_P, EPK_{\bar{P}})$. The adversary $\mathcal{A}$ obtains the response from the party.

To capture exposure of secret information, the adversary $\mathcal{A}$ is allowed to issue the following queries.

- $\mathsf{SessionKeyReveal}(\mathsf{sid})$: The adversary $\mathcal{A}$ obtains the session key $SK$ for the session $\mathsf{sid}$ if the session is completed.
- $\mathsf{SessionStateReveal}(\mathsf{sid})$: The adversary $\mathcal{A}$ obtains session state of the owner of session $\mathsf{sid}$ if the session is not completed (i.e., the session key is not established yet). Session state includes all ESKs and intermediate computation results except for immediately erased information but does not include the SSK. Concrete contents of session state is specified in each protocol.
- $\mathsf{Corrupt}(U_P)$: This query allows the adversary $\mathcal{A}$ to obtain all information of the party $U_P$. If a party is corrupted by a $\mathsf{Corrupt}(U_P)$ query issued by the adversary $\mathcal{A}$, then we call the party $U_P$ *dishonest*. If not, we call the party *honest*.

### 2.4 Freshness

For the security definition, we need the notion of freshness.

**Definition 2.1 (Freshness)** Let $\mathsf{sid}^* = (\Pi, U_P, U_{\bar{P}}, EPK_P, EPK_{\bar{P}})$ be a completed session between honest users $U_P$ and $U_{\bar{P}}$. If the matching session exists, then let $\overline{\mathsf{sid}^*}$ be the matching session of $\mathsf{sid}^*$. We say session $\mathsf{sid}^*$ is *fresh* if none of the following conditions hold:

( 1 ) The adversary $\mathcal{A}$ issues $\mathsf{SessionKeyReveal}(\mathsf{sid}^*)$, or $\mathsf{SessionKeyReveal}(\overline{\mathsf{sid}^*})$ if $\overline{\mathsf{sid}^*}$ exists,

( 2 ) $\overline{\mathsf{sid}^*}$ exists and the adversary $\mathcal{A}$ makes either of the following queries
- $\mathsf{SessionStateReveal}(\mathsf{sid}^*)$ or $\mathsf{SessionStateReveal}(\overline{\mathsf{sid}^*})$,

( 3 ) $\overline{\mathsf{sid}^*}$ does not exist and the adversary $\mathcal{A}$ makes the following query
- $\mathsf{SessionStateReveal}(\mathsf{sid}^*)$.

### 2.5 Security Experiment

For the security definition, we consider the following security experiment. Initially, the adversary $\mathcal{A}$ is given a set of honest users and makes any sequence of the queries described above. During the experiment, the adversary $\mathcal{A}$ makes the following query at once.

- $\mathsf{Test}(\mathsf{sid}^*)$: Here, $\mathsf{sid}^*$ must be a fresh session. Select random bit $b \in_R \{0, 1\}$, and return the session key held by $\mathsf{sid}^*$

[*3] Static public keys must be known to both parties in advance. They can be obtained by exchanging them before starting the protocol or by receiving them from a certification authority. This situation is common for all PKI-based AKE schemes.

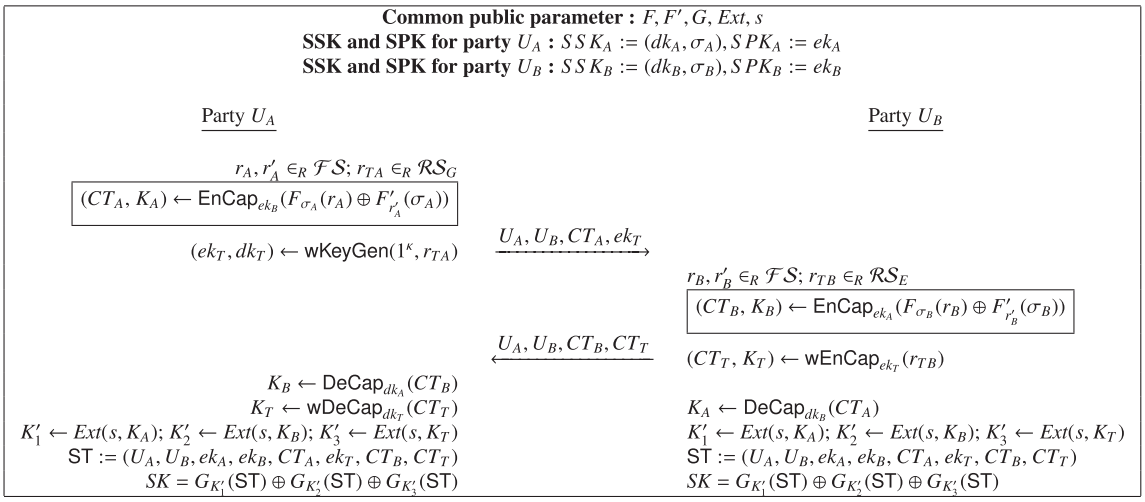


**Fig. 1** FSXY construction.

if $b = 0$, and return a random key if $b = 1$.

The experiment continues until the adversary $\mathcal{A}$ makes a guess $b'$. The adversary $\mathcal{A}$ *wins* the game if the test session $\mathsf{sid}^*$ is still fresh and if the guess of the adversary $\mathcal{A}$ is correct, i.e., $b' = b$. The advantage of the adversary $\mathcal{A}$ in the AKE experiment with the PKI-based AKE protocol $\Pi$ is defined as

$$\mathrm{Adv}_{\Pi}^{\mathrm{AKE}}(\mathcal{A}) = \Pr[\mathcal{A}\ wins] - \frac{1}{2}.$$

We define the security as follows.

**Definition 2.2 (Security)** We say that a PKI-based AKE protocol $\Pi$ is *secure in the* $\mathrm{CK}^+$ *model* if the following conditions hold:

(1) If two honest parties complete matching sessions, then, except with negligible probability, they both compute the same session key.

(2) For any PPT bounded adversary $\mathcal{A}$, $\mathrm{Adv}_{\Pi}^{\mathrm{AKE}}(\mathcal{A})$ is negligible in security parameter $\kappa$ for the test session $\mathsf{sid}^*$,

(a) if $\overline{\mathsf{sid}^*}$ does not exist, and the static secret key of the owner of $\mathsf{sid}^*$ is given to $\mathcal{A}$.

(b) if $\overline{\mathsf{sid}^*}$ does not exist, and the ephemeral secret key of $\mathsf{sid}^*$ is given to $\mathcal{A}$.

(c) if $\overline{\mathsf{sid}^*}$ exists, and the static secret key of the owner of $\mathsf{sid}^*$ and the ephemeral secret key of $\overline{\mathsf{sid}^*}$ are given to $\mathcal{A}$.

(d) if $\overline{\mathsf{sid}^*}$ exists, and the ephemeral secret key of $\mathsf{sid}^*$ and the ephemeral secret key of $\overline{\mathsf{sid}^*}$ are given to $\mathcal{A}$.

(e) if $\overline{\mathsf{sid}^*}$ exists, and the static secret key of the owner of $\mathsf{sid}^*$ and the static secret key of the peer of $\mathsf{sid}^*$ are given to $\mathcal{A}$.

(f) if $\overline{\mathsf{sid}^*}$ exists, and the ephemeral secret key of $\mathsf{sid}^*$ and the static secret key of the peer of $\mathsf{sid}^*$ are given to $\mathcal{A}$.

## 3. Exposure Attack to FSXY Construction: Case of No Implementation Trick

In this section, we show an attack to the FSXY construction if an adversary can expose the output of the twisted PRF of parties. Therefore, it is a realistic attack when the FSXY construction is implemented without a special TPM.

### 3.1 Protocol of FSXY Construction

First, we recall the protocol of the FSXY construction.

It is a general construction from IND-CCA secure KEM ($\mathsf{KeyGen}$, $\mathsf{EnCap}$, $\mathsf{DeCap}$) and IND-CPA secure KEM ($\mathsf{wKeyGen}$, $\mathsf{wEnCap}$, $\mathsf{wDeCap}$), where the randomness space of encapsulation algorithms is $\mathcal{RS}_E$, the randomness space of key generation algorithms is $\mathcal{RS}_G$ and the KEM key space is $\mathcal{KS}$. Other building blocks are PRFs and a strong randomness extractor. For a security parameter $\kappa$, let $F : \{0,1\}^* \times \mathcal{FS} \to \mathcal{RS}_E$, $F' : \{0,1\}^* \times \mathcal{FS} \to \mathcal{RS}_E$, and $G : \{0,1\}^* \times \mathcal{FS} \to \{0,1\}^\kappa$ be PRFs, where $\mathcal{FS}$ is the key space of PRFs ($|\mathcal{FS}| = \kappa$). Let $Ext : \mathcal{SS} \times \mathcal{KS} \to \mathcal{FS}$ be a strong randomness extractor with randomly chosen seed $s \in \mathcal{SS}$, where $\mathcal{SS}$ is the seed space.

Party $U_P$ randomly selects $\sigma_P \in_R \mathcal{FS}$ and $r \in_R \mathcal{RS}_G$, and runs $(ek_P, dk_P) \leftarrow \mathsf{KeyGen}(1^\kappa, r)$. Party $U_P$'s SSK and SPK are $((dk_P, \sigma_P), ek_P)$. **Fig. 1** shows the protocol.

Here, we recall the definition of security for KEM, and min-entropy of KEM keys as follows.

**Definition 3.1 (Syntax of KEM)** A KEM scheme consists of the following 3-tuple ($\mathsf{KeyGen}$, $\mathsf{EnCap}$, $\mathsf{DeCap}$):

$(ek, dk) \leftarrow \mathsf{KeyGen}(1^\kappa, r_g)$ : a key generation algorithm which on inputs $1^\kappa$ and $r_g \in \mathcal{RS}_G$, where $\kappa$ is the security parameter and $\mathcal{RS}_G$ is a randomness space, outputs a pair of keys $(ek, dk)$.

$(K, CT) \leftarrow \mathsf{EnCap}_{ek}(r_e)$ : an encryption algorithm which takes as inputs encapsulation key $ek$ and $r_e \in \mathcal{RS}_E$, outputs session key $K \in \mathcal{KS}$ and ciphertext $CT \in \mathcal{CS}$, where $\mathcal{RS}_E$ is a randomness space, $\mathcal{KS}$ is a session key space, and $\mathcal{CS}$ is a ciphertext space.

$K \leftarrow \mathsf{DeCap}_{dk}(CT)$ : a decryption algorithm which takes as inputs decapsulation key $dk$ and ciphertext $CT \in \mathcal{CS}$, and outputs session key $K \in \mathcal{KS}$.

**Definition 3.2 (IND-CCA Security for KEM)** A KEM scheme is IND-CCA secure for KEM if the following property holds for security parameter $\kappa$; For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, $\mathbf{Adv}^{\mathrm{ind-cca}} = |\Pr[r_g \leftarrow \mathcal{RS}_G; (ek, dk) \leftarrow \mathsf{KeyGen}(1^\kappa, r_g); (state) \leftarrow \mathcal{A}_1^{\mathcal{DO}(dk,\cdot)}(ek); b \leftarrow \{0,1\}; r_e \leftarrow \mathcal{RS}_E; (K_0^*, CT_0^*) \leftarrow \mathsf{EnCap}_{ek}(r_e); K_1^* \leftarrow \mathcal{K}; b' \leftarrow \mathcal{A}_2^{\mathcal{DO}(dk,\cdot)}(ek, (K_b^*, CT_0^*), state); b' = b] - 1/2| \leq negl$, where $\mathcal{DO}$

is the decryption oracle, $\mathcal{K}$ is the space of session key and *state* is state information that $\mathcal{A}$ wants to preserve from $\mathcal{A}_1$ to $\mathcal{A}_2$. $\mathcal{A}$ cannot submit the ciphertext $CT = CT_0^*$ to $\mathcal{DO}$.
We say a KEM scheme is IND-CPA secure for KEM if $\mathcal{A}$ does not access $\mathcal{DO}$.

**Definition 3.3 (Min-Entropy of KEM Key)** A KEM scheme is $k$-min-entropy KEM if for any $ek$, distribution $D_{\mathcal{KS}}$ of variable $K$ defined by $(K, CT) \leftarrow \mathsf{EnCap}_{ek}(r_e)$, distribution $D_{other}$ of public information and random $r_e \in \mathcal{RS}_E$, $H_\infty(D_{\mathcal{KS}}|D_{other}) \geq k$ holds, where $H_\infty$ denotes min-entropy.

### 3.2 Implementation Trick of FSXY Construction

The FSXY construction uses the twisted PRF trick to compute randomness in generating ciphertext $CT_A$ and $CT_B$. For instance, randomness is computed as $F_{\sigma_A}(r_A) \oplus F'_{r'_A}(\sigma_A)$ for party $U_A$. This trick allows that randomness is indistinguishable from a uniformly random value if either of SSK $\sigma_A$ and ESK $r_A$ is not exposed.

The FSXY construction assumes that the output of the twisted PRF is never exposed. Indeed, though the CK$^+$ model allows an adversary to learn ESKs, the output of the twisted PRF is not contained in the ESK (i.e., The ESK of $U_A$ is only $(r_A, r'_A, r_{TA})$.) in the security analysis. In order to implement this assumption in the real world, all computations related to the twisted PRF must be executed in a protected area such as a TPM. Specifically, party $U_A$ (resp. $U_B$) must execute the computation of $(CT_A, K_A) \leftarrow \mathsf{EnCap}_{ek_B}(F_{\sigma_A}(r_A) \oplus F'_{r'_A}(\sigma_A))$ (resp. $(CT_B, K_B) \leftarrow \mathsf{EnCap}_{ek_A}(F_{\sigma_B}(r_B) \oplus F'_{r'_B}(\sigma_B))$) in his TPM on-line. The boxed part of Fig. 1 is computed in TPM. Note that to run a complex operation like encryption algorithms in a TPM is generally very costly and should be avoided. For example, if the Cramer-Shoup KEM [5] is used as IND-CCA secure KEM, the TPM must process 4 exponentiations on-line for each session.

**Remark 3.1** In the case of the twisted PRF, all computations related to SSKs must not be exposed from the session state because SSKs are also assumed not to be learned with ESKs simultaneously. In the FSXY construction, such computations correspond to $K_B \leftarrow \mathsf{DeCap}_{dk_A}(CT_B)$ and $K_A \leftarrow \mathsf{DeCap}_{dk_B}(CT_A)$. However, it is not necessary to execute these computations in TPM. After receiving the message from the peer, all computations are executed without stopping, and the session state is immediately erased on finishing the session. Thus, the computations which must be executed in TPM on-line are only the part related to the twisted PRF and the derivation of $SK$. On the other hand, before receiving the message from the responder, the session state of the initiator contains all unprotected intermediate values. Hence, if an implementation trick is not used, $F_{\sigma_A}(r_A) \oplus F'_{r'_A}(\sigma_A)$ is contained in the session state of the initiator because it is erased just on finishing the session. Moreover, we note that exposure of ESKs is distinguished from exposure of the session state in the CK$^+$ model (i.e., ESKs are automatically given to the adversary, and the session state is obtained via the SessionStateReveal query. ). It means that ESKs may be exposed independently from the session state, and $F_{\sigma_B}(r_B) \oplus F'_{r'_B}(\sigma_B)$ of the responder can be also exposed when an implementation trick is not used even if the session state of the responder is immediately erased on finishing the session.

### 3.3 Our Attack

If an implementer misses this assumption, computations related to the twisted PRF may be executed not in a TPM. Then, $F_{\sigma_A}(r_A) \oplus F'_{r'_A}(\sigma_A)$ and $F_{\sigma_B}(r_B) \oplus F'_{r'_B}(\sigma_B)$ can be exposed as same as ESKs $(r_A, r'_A, r_{TA})$ and $(r_B, r'_B, r_{TB})$. We show an attack to the FSXY construction when we assume that any implementation trick is not used.

An adversary plays the experiment of the CK$^+$ model in the event corresponding to 2.d in Definition 2.2 (i.e., Both parties' ESKs are exposed.). In this attack, $F_{\sigma_A}(r_A) \oplus F'_{r'_A}(\sigma_A)$ and $F_{\sigma_B}(r_B) \oplus F'_{r'_B}(\sigma_B)$ is regarded as a part of ESKs. The procedure of the adversary is as follows.

(1) specify a session between $U_A$ and $U_B$ as the test session, and learn $ESK_A = (r_A, r'_A, r_{TA}, F_{\sigma_A}(r_A) \oplus F'_{r'_A}(\sigma_A))$ and $ESK_B = (r_B, r'_B, r_{TB}, F_{\sigma_B}(r_B) \oplus F'_{r'_B}(\sigma_B))$.
(2) compute $K_A$, $K_B$ and $K_T$ as $(CT_A, K_A) \leftarrow \mathsf{EnCap}_{ek_B}(F_{\sigma_A}(r_A) \oplus F'_{r'_A}(\sigma_A))$, $(CT_B, K_B) \leftarrow \mathsf{EnCap}_{ek_A}(F_{\sigma_B}(r_B) \oplus F'_{r'_B}(\sigma_B))$ and $(CT_T, K_T) \leftarrow \mathsf{wEnCap}_{ek_T}(r_{TB})$.
(3) execute the same key derivation procedure as a party with $K_A$, $K_B$ and $K_T$, and derive the session key $SK$.

Therefore, unless the output of the twisted PRF is strictly protected with an implementation trick, the FSXY construction is insecure against such an exposure attack.

## 4. One-Round AKE without Implementation Tricks

In this section, we propose a new generic construction of CK$^+$-secure AKE from KEM. Our scheme is secure against exposure of all randomness in generating ciphertexts for KEM by avoiding using the twisted PRF trick beside the FSXY construction. Moreover, while the FSXY construction is not one-round protocol, our scheme is one-round protocol by using PKIC-KEM [27].

### 4.1 Preliminaries

#### 4.1.1 Security Notions of KEM with public-key-independent-ciphertext [27]

Here, we recall the syntax for PKIC-KEM, and definitions of IND-CPA security for PKIC-KEM and min-entropy of KEM keys as follows.

**Definition 4.1 (Syntax of PKIC-KEM)** A PKIC-KEM scheme consists of the following 4-tuple (wKeyGen, wEnCapC, wEnCapK, wDeCap):

$(ek, dk) \leftarrow \mathsf{wKeyGen}(1^\kappa; r_g)$ : a key generation algorithm which on inputs $1^\kappa$, where $\kappa$ is the security parameter and $r_g$ is randomness in space $\mathcal{RS}_G$, outputs a pair of keys $(ek, dk)$.

$CT \leftarrow \mathsf{wEnCapC}(r_e)$ : a ciphertext generation algorithm which outputs ciphertext $CT \in \mathcal{CS}$ on inputs public parameters, where $r_e$ is randomness in space $\mathcal{RS}_E$, and $\mathcal{CS}$ is a ciphertext space.

$K \leftarrow \mathsf{wEnCapK}_{ek}(CT; r_e)$ : an encryption algorithm which takes as inputs encapsulation key $ek$, ciphertext $CT$, and randomness $r_e$, outputs KEM key $K \in \mathcal{KS}$, where $r_e$ is randomness used in wEnCapC, and $\mathcal{KS}$ is a KEM key space.

$K \leftarrow \mathsf{wDeCap}_{dk}(CT)$ : a decryption algorithm which takes

as inputs decapsulation key $dk$ and ciphertext $CT \in \mathcal{CS}$, and outputs KEM key $K \in \mathcal{KS}$.

**Definition 4.2 (IND-CPA Security for PKIC-KEM)** A PKIC-KEM scheme is IND-CPA secure if the following property holds for security parameter $\kappa$; For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, $\mathbf{Adv}^{\text{ind-cpa}} = |\Pr[r_g \in_R \mathcal{RS}_G; (ek, dk) \leftarrow \mathsf{wKeyGen}(1^\kappa; r_g); state \leftarrow \mathcal{A}_1(ek); b \in_R \{0,1\}; r_e \in_R \mathcal{RS}_E; CT_0^* \leftarrow \mathsf{wEnCapC}(r_e); K_0^* \leftarrow \mathsf{wEnCapK}_{ek}(CT_0^*; r_e); K_1^* \in_R \mathcal{KS}; b' \leftarrow \mathcal{A}_2(ek, (K_b^*, CT_0^*), state); b' = b] - 1/2| \leq negl$, where *state* is the state information that $\mathcal{A}$ wants to preserve from $\mathcal{A}_1$ to $\mathcal{A}_2$.

**Definition 4.3 (Min-Entropy of KEM Key)** We say a PKIC-KEM scheme is $k$-min-entropy PKIC-KEM if for any $ek$, distribution $D_{\mathcal{KS}}$ of variable $K$ defined by $CT \leftarrow \mathsf{wEnCapC}(r_e)$ and $K \leftarrow \mathsf{wEnCapK}_{ek}(CT; r_e)$, distribution $D_{other}$ of public information and random $r_e \in \mathcal{RS}_E$, $H_\infty(D_{\mathcal{KS}}|D_{other}) \geq k$ holds, where $H_\infty$ denotes min-entropy.

#### 4.1.2 Security Notion of Computational Extractor

Let $cExt : Salt \times Dom \rightarrow Rng$ be a function with finite domain $Dom$, finite range $Rng$, and a space of non-secret random salt $Salt$.

**Definition 4.4 (Computational Extractor [21])** We say a function $cExt$ is a computational extractor (cExt) if the following condition holds for a security parameter $\kappa$: For any PPT adversary $\mathcal{A}$, any salt $s \in_R Salt$ and any distribution $D_{Rng}$ over $Rng$ with $H_\infty(D_{Rng}) \geq \kappa$, $|\Pr[y \in_R Rng; 1 \leftarrow \mathcal{A}(s, y)] - \Pr[x \in_R Dom; y \leftarrow cExt(s, x); 1 \leftarrow \mathcal{A}(s, y)]| \leq negl$.

#### 4.1.3 Security Notion of Pseudo-Random Function

Let $\kappa$ be a security parameter and $\mathsf{F} = \{F_\kappa : Dom_\kappa \times \mathcal{FS}_\kappa \rightarrow Rng_\kappa\}_\kappa$ 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 $\{Rng_\kappa\}_\kappa$.

**Definition 4.5 (Pseudo-Random Function)** We say that function family $\mathsf{F} = \{F_\kappa\}_\kappa$ is the PRF family, if for any PPT distinguisher $\mathcal{D}$, $\mathbf{Adv}^{\text{prf}} = |\Pr[1 \leftarrow \mathcal{D}^{F_\kappa(\cdot)}] - \Pr[1 \leftarrow \mathcal{D}^{RF_\kappa(\cdot)}]| \leq negl$, where $RF_\kappa : Dom_\kappa \rightarrow Rng_\kappa$ is a truly random function.

### 4.2 Our Construction

#### 4.2.1 Design Principle

The goal is to avoid the twisted PRF trick. In the FSXY construction, parties share $K_A$, $K_B$ and $K_T$. $K_A$ is protected even if $SSK_A$ and $ESK_B$ are exposed, $K_B$ is protected even if $SSK_B$ and $ESK_A$ are exposed, and $K_T$ is protected even if both $SSK_A$ and $SSK_B$ are exposed. The case that both $ESK_A$ and $ESK_B$ are exposed is solved thanks to the power of the twisted PRF trick; that is, $F_{\sigma_A}(r_A) \oplus F'_{r'_A}(\sigma_A)$ and $F_{\sigma_B}(r_B) \oplus F'_{r'_B}(\sigma_B)$ look random for an adversary even in this case. To handle this case without the twisted PRF trick, parties must share an additional value that is protected even if both $ESK_A$ and $ESK_B$ are exposed.

Our solution is to change the way to generate SSKs and SPKs. In the FSXY construction, a SSK contains decryption key $dk_P$ of IND-CCA secure KEM and $\sigma_P$, and a SPK contains encryption key $ek_P$. In our construction, party $U_P$ runs $(ek_{SP}, dk_{SP}) \leftarrow \mathsf{wKeyGen}(1^\kappa, r')$ and $CT_{SP} \leftarrow \mathsf{wEnCapC}(r_{SP})$ of IND-CPA secure PKIC-KEM, where $r'$ and $r_{SP}$ are randomly chosen. $dk_{SP}$ and $r_{SP}$ are added to the SSK ($\sigma_P$ is not necessary and is removed), and $ek_{SP}$ and $CT_{SP}$ are added to the SPK. Because wEnCapC can be executed without knowing an encryption key, this key generation phase works correctly.

In each session, parties share $K_S$ in addition to $K_A$, $K_B$ and $K_T$. Party $U_A$ generates $K_S \leftarrow \mathsf{wDeCap}_{dk_{SA}}(CT_{SB})$, and party $U_B$ generates $K_S \leftarrow \mathsf{wEnCapK}_{ek_{SA}}(CT_{SB}; r_{SB})$. The lexicographic order of party identities determines which party is to run wDeCap. From the syntax of PKIC-KEM, $K_S$ is shared non-interactively, and is protected even if both $ESK_A$ and $ESK_B$ are exposed. The generation of $K_T$ corresponds to the DH key exchange if PKIC-KEM is instantiated by the ElGamal KEM.

We note that, from the definition of freshness, the adversary cannot pose SessionStateReveal query for the test session. Also, session state does not contain computation results that are immediately erased on finishing the session as the FSXY construction.

Our construction has another advantage that it is one-round protocol. The FSXY construction is not one-round because the responder's EPK depends on the initiator's EPK. We use the technique in Ref. [27] using PKIC-KEM to generate $K_T$.

Also, the session key derivation procedure is more efficient than the FSXY construction because a cExt is used instead of a strong randomness extractor. On input a value having sufficient min-entropy, a strong randomness extractor outputs a value which is *statistically indistinguishable* from a uniformly chosen random value. Indeed, such statistical indistinguishability is not necessary to prove the security of our construction. *Computational indistinguishability* is sufficient, and the cExt is suitable. Naturally, the FSXY construction is also secure with the implementation trick if the strong randomness extractor is replaced with the cExt. Indeed, in the full version of Ref. [11], the FSXY construction is proved with the cExt. [12].

#### 4.2.2 Protocol

The protocol of our generic construction is shown in **Fig. 2**.

*Public Parameters.* Let (KeyGen, EnCap, DeCap) be an IND-CCA secure KEM and (wKeyGen, wEnCapC, wEnCapK, wDeCap) be an IND-CPA secure PKIC-KEM, where the randomness space of encapsulation algorithms is $\mathcal{RS}_E$, the randomness space of key generation algorithms is $\mathcal{RS}_G$ and the KEM key space is $\mathcal{KS}$. Let $G : \{0,1\}^* \times \mathcal{FS} \rightarrow \{0,1\}^\kappa$ be a PRF, where $\mathcal{FS}$ is the key space of PRFs ($|\mathcal{FS}| = \kappa$). Let $cExt : Salt \times \mathcal{KS} \rightarrow \mathcal{FS}$ be a cExt with a non-secret random salt $s \in Salt$, where $Salt$ is the salt space.

*Static Secret and Static Public Keys.* Party $U_P$ selects $r, r' \in_R \mathcal{RS}_G$ and $r_{SP} \in_R \mathcal{RS}_E$, and generates $(ek_P, dk_P) \leftarrow \mathsf{KeyGen}(1^\kappa, r)$, $(ek_{SP}, dk_{SP}) \leftarrow \mathsf{wKeyGen}(1^\kappa, r')$ and $CT_{SP} \leftarrow \mathsf{wEnCapC}(r_{SP})$. Party $U_P$'s SSK is $(dk_P, dk_{SP}, r_{SP})$ and SPK is $(ek_P, ek_{SP}, CT_{SP})$. Note that a party does not use all contents of the SSK to generate $K_S$ in a session.

*Session State.* The session state of a session owned by $U_A$ contains ephemeral secret keys $(r_A, r_{TA})$, encapsulated KEM key $K_A$ and ad-hoc decryption key $dk_T$. Other information that is computed after receiving the message from the peer is immediately erased when the session key is established. Similarly, the session state of a session owned by $U_B$ contains ephemeral secret keys $(r_B, r_{TB})$ and encapsulated KEM key $K_B$.

Other intermediate values (e.g., decapsulated KEM keys, and outputs of cExt) are not contained in the session state because

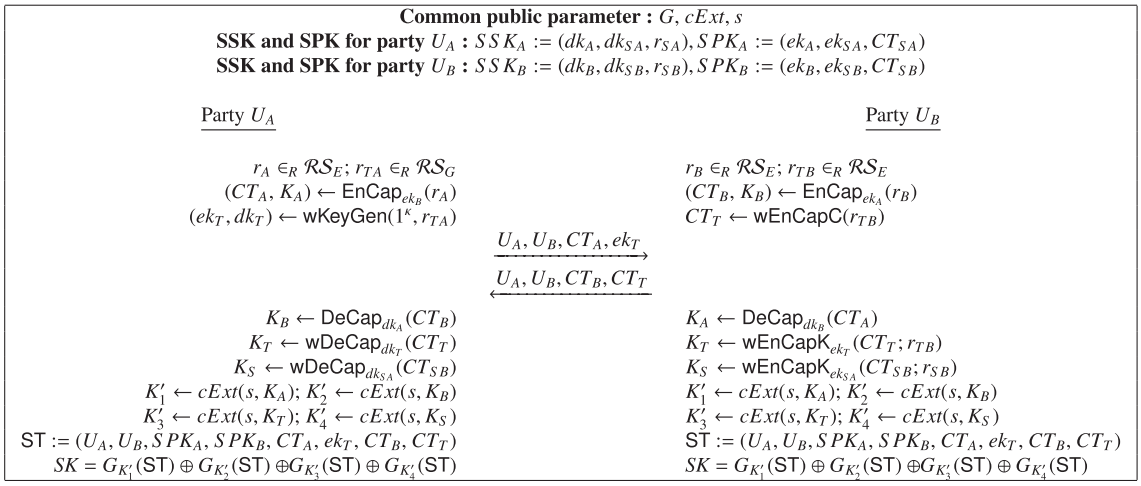


**Fig. 2** Our construction.

these values are simultaneously computed with the session key and immediately erased after completing the session.

### 4.3 Security

We show the following theorem.

**Theorem 4.1** If (KeyGen, EnCap, DeCap) is IND-CCA secure and $\kappa$-min-entropy KEM, (wKeyGen, wEnCapC, wEnCapK wDeCap) is IND-CPA secure and $\kappa$-min-entropy PKIC-KEM, $G$ is a PRF, and $cExt$ is a cExt, then our generic construction is CK$^+$-secure.

Here, we give an overview of the security proof.

We have to consider the following four exposure patterns in the CK$^+$ security model (matching cases):

**2-(c):** $SSK_A$ and $ESK_B$, **2-(d):** $ESK_A$ and $ESK_B$,

**2-(e):** $SSK_A$ and $SSK_B$, **2-(f):** $ESK_A$ and $SSK_B$.

In case 2-(c), $K_A$ is protected by the security of $CT_A$ because $r_A$ and $dk_B$ are not exposed. In case 2-(d), $K_S$ is protected by the security of $CT_S$ because $dk_{SA}$ and $r_{SB}$ are not exposed. In case 2-(e), $K_T$ is protected by the security of $CT_T$ because $dk_T$ and $r_{TB}$ are not exposed. In case 2-(f), $K_B$ is protected by the security of $CT_B$ because $r_B$ and $dk_A$ are not exposed.

Then, we transform the CK$^+$ security game, and the session key in the test session is randomly distributed in the final game. First, we change the protected KEM key into a random key for each pattern; therefore, the input of $cExt$ is randomly distributed and has sufficient min-entropy. Next, we change the output of $cExt$ into randomly chosen values. Finally, we change one of the PRFs (corresponding to the protected KEM) into a random function. Therefore, the session key in the test session is randomly distributed; thus, there is no advantage to the adversary. We can show a similar proof in non-matching cases.

*Proof.* Let $\kappa$ be the security parameter, and let $\mathcal{A}$ be a PPT (in $\kappa$) bounded adversary. $Suc$ denotes the event that $\mathcal{A}$ wins. We consider the following events that cover all cases of the behavior of $\mathcal{A}$.

- Let $E_1$ be the event that the test session $\mathsf{sid}^*$ has no matching session $\overline{\mathsf{sid}}^*$, the owner of $\mathsf{sid}^*$ is the initiator and the static secret key of the initiator is given to $\mathcal{A}$.
- Let $E_2$ be the event that the test session $\mathsf{sid}^*$ has no matching session $\overline{\mathsf{sid}}^*$, the owner of $\mathsf{sid}^*$ is the initiator and the ephemeral secret key of $\mathsf{sid}^*$ is given to $\mathcal{A}$.
- Let $E_3$ be the event that the test session $\mathsf{sid}^*$ has no matching session $\overline{\mathsf{sid}}^*$, the owner of $\mathsf{sid}^*$ is the responder and the static secret key of the responder is given to $\mathcal{A}$.
- Let $E_4$ be the event that the test session $\mathsf{sid}^*$ has no matching session $\overline{\mathsf{sid}}^*$, the owner of $\mathsf{sid}^*$ is the responder and the ephemeral secret key of $\mathsf{sid}^*$ is given to $\mathcal{A}$.
- Let $E_5$ be the event that the test session $\mathsf{sid}^*$ has matching session $\overline{\mathsf{sid}}^*$, and both static secret keys of the initiator and the responder are given to $\mathcal{A}$.
- Let $E_6$ be the event that the test session $\mathsf{sid}^*$ has matching session $\overline{\mathsf{sid}}^*$, and both ephemeral secret keys of $\mathsf{sid}^*$ and $\overline{\mathsf{sid}}^*$ are given to $\mathcal{A}$.
- Let $E_7$ be the event that the test session $\mathsf{sid}^*$ has matching session $\overline{\mathsf{sid}}^*$, and the static secret key of the owner of $\mathsf{sid}^*$ and the ephemeral secret key of $\overline{\mathsf{sid}}^*$ are given to $\mathcal{A}$.
- Let $E_8$ be the event that the test session $\mathsf{sid}^*$ has matching session $\overline{\mathsf{sid}}^*$, and the ephemeral secret key of $\mathsf{sid}^*$ and the static secret key of the owner of $\overline{\mathsf{sid}}^*$ are given to $\mathcal{A}$.

To finish the proof, we investigate events $E_i \wedge Suc$ $(i = 1, \ldots, 8)$ that cover all cases of event $Suc$. In this paper, we show the proof of event $E_6$ because it is most different from that of the FSXY construction. The proof of other events can be proved by similar ways, and we show the differences from event $E_6$.

#### 4.3.1 Event $E_6 \wedge Suc$

We change the interface of oracle queries and the computation of the session key. These instances are gradually changed over six hybrid experiments, depending on specific sub-cases. In the last hybrid experiment, the session key in the test session does not contain information of the bit $b$. Thus, the adversary can clearly only output a random guess. We denote these hybrid experiments by $\mathbf{H}_0, \ldots, \mathbf{H}_5$ and the advantage of the adversary $\mathcal{A}$ when participating in experiment $\mathbf{H}_i$ by $\mathbf{Adv}(\mathcal{A}, \mathbf{H}_i)$.

**Hybrid experiment $\mathbf{H}_0$:** This experiment denotes the real experiment for CK$^+$ security and in this experiment the environment for $\mathcal{A}$ is as defined in the protocol. Thus, $\mathbf{Adv}(\mathcal{A}, \mathbf{H}_0)$ is the same as the advantage of the real experiment.

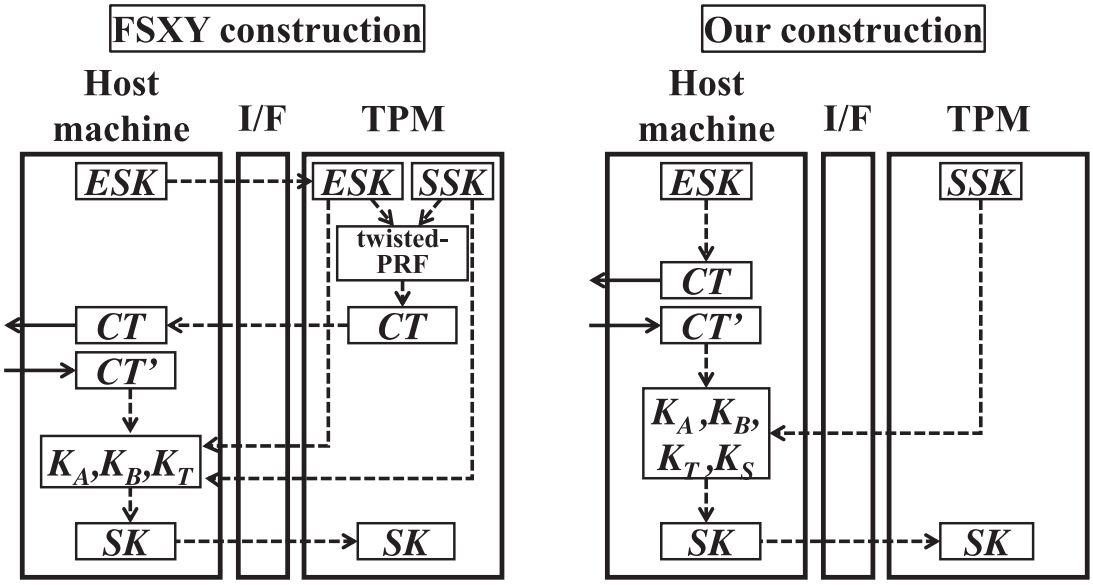


**Fig. 3** Implementation image of FSXY construction and our construction.

**Hybrid experiment $\mathbf{H}_1$:** In this experiment, if session identities in two sessions are identical, the experiment halts. When two ciphertexts from different randomness are identical and two public keys from different randomness are identical, session identities in two sessions are also identical. In any IND-CCA secure KEM and IND-CPA secure PKIC-KEM, such an event occurs with negligible probability. Thus, $|\mathbf{Adv}(\mathcal{A}, \mathbf{H}_1) - \mathbf{Adv}(\mathcal{A}, \mathbf{H}_0)| \leq negl$.

**Hybrid experiment $\mathbf{H}_2$:** In this experiment, the experiment selects a party $U_A$ and integer $i \in [1, \ell]$ randomly in advance. If $\mathcal{A}$ poses a Test query to a session except $i$-th session of $U_A$, the experiment halts. Since a guess of the test session matches with $\mathcal{A}$'s choice with probability $1/N\ell$, $\mathbf{Adv}(\mathcal{A}, \mathbf{H}_2) \geq 1/N\ell \cdot \mathbf{Adv}(\mathcal{A}, \mathbf{H}_1)$. Without loss of generality, we can suppose that the intended peer of the $i$-th session of $U_A$ is $U_B$.

**Hybrid experiment $\mathbf{H}_3$:** In this experiment, the computation of $K_S$ in the test session is changed. Instead of computing $K_S \leftarrow \mathsf{wDeCap}_{dk_{SA}}(CT_{SB})$ or $K_S \leftarrow \mathsf{wEnCapK}_{ek_{SA}}(CT_{SB}; r_{SB})$, it is changed as choosing $K_S \leftarrow \mathcal{KS}$ randomly. We construct an IND-CPA adversary $\mathcal{S}$ from $\mathcal{A}$ in $\mathbf{H}_2$ or $\mathbf{H}_3$. $\mathcal{S}$ performs the following steps. *Init*. $\mathcal{S}$ receives $(ek^*, K_b^*, CT_0^*)$ as the challenge of IND-CPA game for PKIC-KEM.

*Setup*. $\mathcal{S}$ chooses PRF $G : \{0,1\}^* \times \mathcal{FS} \to \{0,1\}^k$, where $\mathcal{FS}$ is the key space of PRFs, and cExt $cExt : Salt \times \mathcal{KS} \to \mathcal{FS}$ with random salt $s \in Salt$, where $Salt$ is the salt space. These are provided as a part of the public parameters. Also, $\mathcal{S}$ sets all $N$ users' static secret and public keys except $U_A$ and $U_B$.

For $U_P$ (except $U_A$ and $U_B$), $\mathcal{S}$ selects $r, r' \in_R \mathcal{RS}_G$ and $r_{SP} \in_R \mathcal{RS}_E$, and generates $(ek_P, dk_P) \leftarrow \mathsf{KeyGen}(1^\kappa, r)$, $(ek_{SP}, dk_{SP}) \leftarrow \mathsf{wKeyGen}(1^\kappa, r')$ and $CT_{SP} \leftarrow \mathsf{wEnCapC}(r_{SP})$. Party $U_P$'s SSK is $(dk_P, dk_{SP}, r_{SP})$ and SPK is $(ek_P, ek_{SP}, CT_{SP})$.

For $U_A$, $\mathcal{S}$ selects $r \in_R \mathcal{RS}_G$ and $r_{AP} \in_R \mathcal{RS}_E$, and generates $(ek_A, dk_A) \leftarrow \mathsf{KeyGen}(1^\kappa, r)$ and $CT_{AP} \leftarrow \mathsf{wEnCapC}(r_{AP})$. Party $U_A$'s SSK is $(dk_A, *, r_{AP})$ and SPK is $(ek_A, ek^*, CT_{AP})$, where $*$ is unknown part for $\mathcal{S}$.

For $U_B$, $\mathcal{S}$ selects $r, r' \in_R \mathcal{RS}_G$, and generates $(ek_B, dk_B) \leftarrow \mathsf{KeyGen}(1^\kappa, r)$ and $(ek_{BP}, dk_{BP}) \leftarrow \mathsf{wKeyGen}(1^\kappa, r')$. Party $U_B$'s SSK is $(dk_B, dk_{BP}, *)$ and SPK is $(ek_B, ek_{BP}, CT_0^*)$, where $*$ is unknown part for $\mathcal{S}$.

*Simulation*. $\mathcal{S}$ maintains the list $\mathcal{L}_{SK}$ that contains queries and answers of SessionKeyReveal. $\mathcal{S}$ simulates oracle queries by $\mathcal{A}$ as follows. We suppose that $P$ sorts before $\bar{P}$ lexicographically.

(1) $\mathsf{Send}(\Pi, U_P, U_{\bar{P}})$: $\mathcal{S}$ computes the ephemeral public key $(U_P, U_{\bar{P}}, CT_P, ek_T)$ obeying the protocol, returns it and records $(\Pi, U_P, U_{\bar{P}}, (U_P, U_{\bar{P}}, CT_P, ek_T))$.

(2) $\mathsf{Send}(\Pi, U_{\bar{P}}, U_P)$: $\mathcal{S}$ computes the ephemeral public key $(U_{\bar{P}}, U_P, CT_{\bar{P}}, CT_T)$ obeying the protocol, returns it and records $(\Pi, U_P, U_{\bar{P}}, (U_{\bar{P}}, U_P, CT_{\bar{P}}, CT_T))$.

(3) $\mathsf{Send}(\Pi, U_{\bar{P}}, U_P, (U_{\bar{P}}, U_P, CT_{\bar{P}}, CT_T))$: If $P = A$, $\bar{P} = B$, the session is $i$-th session of $A$, then $\mathcal{S}$ sets $K_T := K_b^*$, computes the session key $SK^*$ obeying the protocol, and records $(\Pi, U_A, U_B, (U_A, U_B, CT_A, ek_T), (U_B, U_A, CT_B, CT_T))$ as the completed session and $SK^*$ in the list $\mathcal{L}_{SK}$. Else if $(\Pi, U_P, U_{\bar{P}}, (U_P, U_{\bar{P}}, CT_P, ek_T))$ is not recorded, $\mathcal{S}$ records the session $(\Pi, U_P, U_{\bar{P}}, *, (U_{\bar{P}}, U_P, CT_{\bar{P}}, CT_T))$ and waits $\mathsf{Send}(\Pi, U_P, U_{\bar{P}})$. Otherwise, $\mathcal{S}$ computes the session key $SK$ obeying the protocol, and records $(\Pi, U_P, U_{\bar{P}}, (U_P, U_{\bar{P}}, CT_P, ek_T), (U_{\bar{P}}, U_P, CT_{\bar{P}}, CT_T))$ as the completed session and $SK$ in the list $\mathcal{L}_{SK}$.

(4) $\mathsf{Send}(\Pi, U_P, U_{\bar{P}}, (U_P, U_{\bar{P}}, CT_P, ek_T))$: If $P = A$, $\bar{P} = B$, the session is the matching session of $i$-th session of $A$, then $\mathcal{S}$ sets $K_T := K_b^*$, computes the session key $SK^*$ obeying the protocol, and records $(\Pi, U_B, U_A, (U_A, U_B, CT_A, ek_T), (U_B, U_A, CT_B, CT_T))$ as the completed session and $SK^*$ in the list $\mathcal{L}_{SK}$. Else if $(\Pi, U_{\bar{P}}, U_P, (U_{\bar{P}}, U_P, CT_{\bar{P}}, CT_T))$ is not recorded, $\mathcal{S}$ records the session $(\Pi, U_{\bar{P}}, U_P, (U_P, U_{\bar{P}}, CT_P, ek_T), *)$ and waits $\mathsf{Send}(\Pi, U_{\bar{P}}, U_P)$. Otherwise, $\mathcal{S}$ computes the session key $SK$ obeying the protocol, and records $(\Pi, U_{\bar{P}}, U_P, (U_P, U_{\bar{P}}, CT_P, ek_T), (U_{\bar{P}}, U_P, CT_{\bar{P}}, CT_T))$ as the completed session and $SK$ in the list $\mathcal{L}_{SK}$.

(5) $\mathsf{SessionKeyReveal}(\mathsf{sid})$:

(a) If the session sid is not completed, $\mathcal{S}$ returns an error

message.

(b) Otherwise, $\mathcal{S}$ returns the recorded value $SK$.

(6) SessionStateReveal(sid): $\mathcal{S}$ responds the ephemeral secret key and intermediate computation results of sid as the definition. Note that the SessionStateReveal query is not posed to the test session from the freshness definition.

(7) Corrupt($U_P$): $\mathcal{S}$ responds the static secret key and all unerased session states of $U_P$ as the definition.

(8) Test(sid): $\mathcal{S}$ responds to the query as the definition.

(9) If $\mathcal{A}$ outputs a guess $b'$, $\mathcal{S}$ outputs $b'$.

*Analysis*. For $\mathcal{A}$, the simulation by $\mathcal{S}$ is same as the experiment $\mathbf{H}_2$ if the challenge is $(K_0^*, CT_0^*)$. Otherwise, the simulation by $\mathcal{S}$ is same as the experiment $\mathbf{H}_3$. Also, both $K_T$ in two experiments have $\kappa$-min-entropy because (wKeyGen, wEnCapC, wEnCapK, wDeCap) is $\kappa$-min-entropy PKIC-KEM. Thus, if the advantage of $\mathcal{S}$ is negligible, then $|\mathbf{Adv}(\mathcal{A}, \mathbf{H}_3) - \mathbf{Adv}(\mathcal{A}, \mathbf{H}_2)| \leq negl$.

**Hybrid experiment $\mathbf{H}_4$:** In this experiment, the computation of $K_4'$ in the test session is changed. Instead of computing $K_4' \leftarrow cExt(s, K_S)$, it is changed as choosing $K_4' \in \mathcal{FS}$ randomly.

Since $K_S$ is randomly chosen in $\mathbf{H}_3$, it has sufficient min-entropy. Thus, by the definition of the cExt, $|\mathbf{Adv}(\mathcal{A}, \mathbf{H}_4) - \mathbf{Adv}(\mathcal{A}, \mathbf{H}_3)| \leq negl$.

**Hybrid experiment $\mathbf{H}_5$:** In this experiment, the computation of $SK$ in the test session is changed. Instead of computing $SK = G_{K_1'}(\mathsf{ST}) \oplus G_{K_2'}(\mathsf{ST}) \oplus G_{K_3'}(\mathsf{ST}) \oplus G_{K_4'}(\mathsf{ST})$, it is changed as $SK = G_{K_1'}(\mathsf{ST}) \oplus G_{K_2'}(\mathsf{ST}) \oplus G_{K_3'}(\mathsf{ST}) \oplus x$ where $x \in \{0,1\}^\kappa$ is chosen randomly.

We construct a distinguisher $\mathcal{D}'$ between PRF $F^* : \{0,1\}^* \times \mathcal{FS} \to \{0,1\}^k$ and a random function $RF$ from $\mathcal{A}$ in $\mathbf{H}_4$ or $\mathbf{H}_5$. $\mathcal{D}'$ performs the following steps.

*Setup*. $\mathcal{D}'$ sets $G = F^*$, and chooses cExt $cExt : \mathcal{KS} \to \mathcal{FS}$. These are provided as a part of the public parameters. Also, $\mathcal{D}'$ sets all $N$ users' static secret and public keys. $\mathcal{S}$ selects $r, r' \in_R \mathcal{RS}_G$ and $r_{SP} \in_R \mathcal{RS}_E$, and generates $(ek_P, dk_P) \leftarrow \mathsf{KeyGen}(1^\kappa, r)$, $(ek_{SP}, dk_{SP}) \leftarrow \mathsf{wKeyGen}(1^\kappa, r')$ and $CT_{SP} \leftarrow \mathsf{wEnCapC}(r_{SP})$. Party $U_P$'s SSK is $(dk_P, dk_{SP}, r_{SP})$ and SPK is $(ek_P, ek_{SP}, CT_{SP})$.

*Simulation*. $\mathcal{D}'$ maintains the list $\mathcal{L}_{SK}$ that contains queries and answers of SessionKeyReveal. $\mathcal{D}'$ simulates oracle queries by $\mathcal{A}$ as follows.

(1) Send($\Pi, U_P, U_{\bar{P}}$): $\mathcal{D}'$ computes the ephemeral public key $(U_P, U_{\bar{P}}, CT_P, ek_T)$ obeying the protocol, returns it and records $(\Pi, U_P, U_{\bar{P}}, (U_P, U_{\bar{P}}, CT_P, ek_T))$.

(2) Send($\Pi, U_{\bar{P}}, U_P$): $\mathcal{D}'$ computes the ephemeral public key $(U_{\bar{P}}, U_P, CT_{\bar{P}}, CT_T)$ obeying the protocol, returns it and records $(\Pi, U_P, U_{\bar{P}}, (U_{\bar{P}}, U_P, CT_{\bar{P}}, CT_T))$.

(3) Send($\Pi, U_{\bar{P}}, U_P, (U_{\bar{P}}, U_P, CT_{\bar{P}}, CT_T)$): If $P = A$, $\bar{P} = B$, the session is $i$-th session of $A$, then $\mathcal{D}'$ poses ST to his oracle (i.e., $F^*$ or a random function $RF$), obtains $x \in \{0,1\}^\kappa$, computes the session key $SK = G_{K_1'}(\mathsf{ST}) \oplus G_{K_2'}(\mathsf{ST}) \oplus G_{K_3'}(\mathsf{ST}) \oplus x$, and records $(\Pi, U_A, U_B, (U_A, U_B, CT_A, ek_T), (U_B, U_A, CT_B, CT_T))$ as the completed session and $SK^*$ in the list $\mathcal{L}_{SK}$. Else if $(\Pi, U_P, U_{\bar{P}}, (U_P, U_{\bar{P}}, CT_P, ek_T))$ is not recorded, $\mathcal{D}'$ records the session $(\Pi, U_P, U_{\bar{P}}, *, (U_{\bar{P}}, U_P, CT_{\bar{P}}, CT_T))$ and waits Send($\Pi, U_P, U_{\bar{P}}$). Otherwise, $\mathcal{D}'$ computes the session key $SK$ obeying the protocol, and records $(\Pi, U_P, U_{\bar{P}}, (U_P, U_{\bar{P}}, CT_P, ek_T), (U_{\bar{P}}, U_P, CT_{\bar{P}}, CT_T))$ as the completed session and $SK$ in the list $\mathcal{L}_{SK}$.

(4) Send($\Pi, U_P, U_{\bar{P}}, (U_P, U_{\bar{P}}, CT_P, ek_T)$): If $P = A$, $\bar{P} = B$, the session is the matching session of $i$-th session of $A$, then $\mathcal{D}'$ poses ST to his oracle (i.e., $F^*$ or a random function $RF$), obtains $x \in \{0,1\}^\kappa$, computes the session key $SK = G_{K_1'}(\mathsf{ST}) \oplus G_{K_2'}(\mathsf{ST}) \oplus G_{K_3'}(\mathsf{ST}) \oplus x$, and records $(\Pi, U_B, U_A, (U_A, U_B, CT_A, ek_T), (U_B, U_A, CT_B, CT_T))$ as the completed session and $SK^*$ in the list $\mathcal{L}_{SK}$. Else if $(\Pi, U_{\bar{P}}, U_P, (U_{\bar{P}}, U_P, CT_{\bar{P}}, CT_T))$ is not recorded, $\mathcal{D}'$ records the session $(\Pi, U_{\bar{P}}, U_P, (U_P, U_{\bar{P}}, CT_P, ek_T), *)$ and waits Send($\Pi, U_{\bar{P}}, U_P$). Otherwise, $\mathcal{D}'$ computes the session key $SK$ obeying the protocol, and records $(\Pi, U_{\bar{P}}, U_P, (U_P, U_{\bar{P}}, CT_P, ek_T), (U_{\bar{P}}, U_P, CT_{\bar{P}}, CT_T))$ as the completed session and $SK$ in the list $\mathcal{L}_{SK}$.

(5) SessionKeyReveal(sid):

(a) If the session sid is not completed, $\mathcal{D}'$ returns an error message.

(b) Otherwise, $\mathcal{D}'$ returns the recorded value $SK$.

(6) SessionStateReveal(sid): $\mathcal{D}'$ responds the ephemeral secret key and intermediate computation results of sid as the definition. Note that the SessionStateReveal query is not posed to the test session from the freshness definition.

(7) Corrupt($U_P$): $\mathcal{D}'$ responds the static secret key and all unerased session states of $U_P$ as the definition.

(8) Test(sid): $\mathcal{D}'$ responds to the query as the definition.

(9) If $\mathcal{A}$ outputs a guess $b' = 0$, $\mathcal{D}'$ outputs that the oracle is the PRF $F^*$. Otherwise, $\mathcal{D}'$ outputs that the oracle is a random function $RF$.

*Analysis*. For $\mathcal{A}$, the simulation by $\mathcal{D}'$ is same as the experiment $\mathbf{H}_4$ if the oracle is the PRF $F^*$. Otherwise, the simulation by $\mathcal{D}'$ is same as the experiment $\mathbf{H}_5$. Thus, if the advantage of $\mathcal{D}'$ is negligible, then $|\mathbf{Adv}(\mathcal{A}, \mathbf{H}_5) - \mathbf{Adv}(\mathcal{A}, \mathbf{H}_4)| \leq negl$.

In $\mathbf{H}_5$, the session key in the test session is perfectly randomized. Thus, $\mathcal{A}$ cannot obtain any advantage from Test query.

Therefore, $\mathbf{Adv}(\mathcal{A}, \mathbf{H}_5) = 0$ and $\Pr[E_5 \wedge Suc]$ is negligible.

□

#### 4.3.2 Event $E_1 \wedge Suc$

The proof in this case is similar to the event $E_6 \wedge Suc$. There is a difference in the experiment $\mathbf{H}_3$. In the event $E_6 \wedge Suc$, instead of computing $K_S \leftarrow \mathsf{wDeCap}_{dk_{SA}}(CT_{SB})$ or $K_S \leftarrow \mathsf{wEnCapK}_{ek_{SA}}(CT_{SB}; r_{SB})$, it is changed as choosing $K_S \leftarrow \mathcal{KS}$, where we suppose that $U_B$ is the intended partner of $U_A$ in the test session. In the event $E_1 \wedge Suc$, instead of computing $(CT_A, K_A) \leftarrow \mathsf{EnCap}_{ek_B}(r_A)$, it is changed as $K_A \leftarrow \mathcal{KS}$. Since $\mathcal{A}$ cannot obtain $r_A$ and $dk_B$ by the freshness definition in this event, we can construct an adversary $\mathcal{S}$ from $\mathcal{A}$ in the similar manner in the proof of the event $E_6 \wedge Suc$. Note that if $\mathcal{A}$ poses Send query to $U_B$ other than the test session, $\mathcal{S}$ simulates $K_A$ by posing the decryption oracle.

#### 4.3.3 Event $E_2 \wedge Suc$

The proof in this case is almost same as the event $E_6 \wedge Suc$.

#### 4.3.4 Event $E_3 \wedge Suc$

The proof in this case is similar to the event $E_6 \wedge Suc$. There

is a difference in the experiment $\mathbf{H}_3$. In the event $E_6 \wedge Suc$, instead of computing $K_S \leftarrow \mathsf{wDeCap}_{dk_{SA}}(CT_{SB})$ or $K_S \leftarrow \mathsf{wEnCapK}_{ek_{SA}}(CT_{SB}; r_{SB})$, it is changed as choosing $K_S \leftarrow \mathcal{KS}$, where we suppose that $U_B$ is the intended partner of $U_A$ in the test session. In the event $E_3 \wedge Suc$, instead of computing $(CT_B, K_B) \leftarrow \mathsf{EnCap}_{ek_A}(r_B)$, it is changed as $K_B \leftarrow \mathcal{KS}$. Since $\mathcal{A}$ cannot obtain $r_B$ and $dk_A$ by the freshness definition in this event, we can construct an adversary $\mathcal{S}$ from $\mathcal{A}$ in the similar manner in the proof of the event $E_6 \wedge Suc$. Note that if $\mathcal{A}$ poses Send query to $U_A$ other than the test session, $\mathcal{S}$ simulates $K_B$ by posing the decryption oracle.

#### 4.3.5 Event $E_4 \wedge Suc$

The proof in this case is almost same as the event $E_6 \wedge Suc$.

#### 4.3.6 Event $E_5 \wedge Suc$

The proof in this case is similar to the event $E_6 \wedge Suc$. There is a difference in the experiment $\mathbf{H}_3$. In the event $E_6 \wedge Suc$, instead of computing $K_S \leftarrow \mathsf{wDeCap}_{dk_{SA}}(CT_{SB})$ or $K_S \leftarrow \mathsf{wEnCapK}_{ek_{SA}}(CT_{SB}; r_{SB})$, it is changed as choosing $K_S \leftarrow \mathcal{KS}$, where we suppose that $U_B$ is the intended partner of $U_A$ in the test session. In the event $E_5 \wedge Suc$, instead of computing $K_T \leftarrow \mathsf{wDeCap}_{dk_T}(CT_T)$ or $K_T \leftarrow \mathsf{wEnCapK}_{ek_T}(CT_T; r_{TB})$, it is changed as $K_T \leftarrow \mathcal{KS}$. Since $\mathcal{A}$ cannot obtain $r_{TA}$ and $r_{TB}$ by the freshness definition in this event, we can construct an adversary $\mathcal{S}$ from $\mathcal{A}$ in the similar manner in the proof of the event $E_6 \wedge Suc$.

#### 4.3.7 Event $E_7 \wedge Suc$

The proof in this case is almost same as the event $E_1 \wedge Suc$.

#### 4.3.8 Event $E_8 \wedge Suc$

The proof in this case is almost same as the event $E_2 \wedge Suc$.

### 4.4 Instantiations

We can instantiate IND-CCA secure KEM by various schemes. For example, we can use efficient IND-CCA KEM schemes from the decisional DH [5] (DDH), computational DH [13], [14], hashed DH [15], bilinear DH [3], the McEliece and LPN [10], and the (ring-)LWE [18], [24] assumptions. We can easily show that these schemes have $\kappa$-min-entropy KEM keys. The KEM part of the Cramer-Shoup PKE consists of $g_1^{zr} \in G$, where $G$ is a finite cyclic group of order prime $p$, $g_1^z$ is part of $ek$, and $r$ is uniformly chosen randomness, and $|r|$ is $2\kappa$. Thus, $g_1^{zr}$ has min-entropy larger than $\kappa$. Similarly, other schemes are also $\kappa$-min-entropy KEM.

Conversely, instantiations of IND-CPA secure PKIC-KEM are limited. The ElGamal KEM is the representative instantiation from the DDH assumption. Also, we can use lattice-based KEM like the Regev's KEM [25] from the LWE assumption, or code-based KEM like the Nojima et al.'s KEM [22] from the LPN assumption. However, to ignore non-negligible error of decasulation, a very large parameter size is necessary; and thus, these schemes become inefficient.

## 5. Concluding Remark

This paper studied the exposure-resilience of AKE schemes in terms of implementation tricks. It was shown that some AKE schemes are not exposure-resilient if implementation tricks do not work. We also gave a generic construction that is exposure-resilient without implementation tricks.

A remaining problem of future researches is to clarify security models capturing the difference between security with/without implementation tricks.

### References


[1] Boyd, C., Cliff, Y., González Nieto, J.M. and Paterson, K.G.: Efficient One-Round Key Exchange in the Standard Model, *ACISP 2008*, pp.69–83 (2008).

[2] Boyd, C., Cliff, Y., González Nieto, J.M. and Paterson, K.G.: One-round key exchange in the standard model, *IJACT*, Vol.1, No.3, pp.181–199 (2009).

[3] Boyen, X., Mei, Q. and Waters, B.: Direct chosen ciphertext security from identity-based techniques, *ACM Conference on Computer and Communications Security 2005*, pp.320–329 (2005).

[4] Canetti, R. and Krawczyk, H.: Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels, *EUROCRYPT 2001*, pp.453–474 (2001).

[5] Cramer, R. and Shoup, V.: A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack, *CRYPTO 1998*, pp.13–25 (1998).

[6] Cremers, C.J.F.: Session-state Reveal Is Stronger Than Ephemeral Key Reveal: Attacking the NAXOS Authenticated Key Exchange Protocol, *ACNS 2009*, pp.20–33 (2009).

[7] Cremers, C.J.F.: Examining Indistinguishability-Based Security Models for Key Exchange Protocols: The case of CK, CK-HMQV, and eCK, *ASIACCS 2011*, pp.80–91 (2011).

[8] Damgård, I.: Towards Practical Public Key Systems Secure Against Chosen Ciphertext Attacks, *CRYPTO 1991*, pp.445–456 (1991).

[9] Diffie, W. and Hellman, M.E.: New Directions in Cryptography, *IEEE Trans. on Info. Theory*, Vol.IT-22, No.6, pp.644–654 (1976).

[10] Dowsley, R., Müller-Quade, J. and Nascimento, A.C.A.: A CCA2 Secure Public Key Encryption Scheme Based on the McEliece Assumptions in the Standard Model, *CT-RSA 2009*, pp.240–251 (2009).

[11] Fujioka, A., Suzuki, K., Xagawa, K. and Yoneyama, K.: Strongly Secure Authenticated Key Exchange from Factoring, Codes, and Lattices, *Public Key Cryptography 2012*, pp.467–484 (2012).

[12] Fujioka, A., Suzuki, K., Xagawa, K. and Yoneyama, K.: Strongly Secure Authenticated Key Exchange from Factoring, Codes, and Lattices, *Designs, Codes and Cryptography* (2014).

[13] Hanaoka, G. and Kurosawa, K.: Efficient Chosen Ciphertext Secure Public Key Encryption under the Computational Diffie-Hellman Assumption, *ASIACRYPT 2008*, pp.308–325 (2008).

[14] Haralambiev, K., Jager, T., Kiltz, E. and Shoup, V.: Simple and Efficient Public-Key Encryption from Computational Diffie-Hellman in the Standard Model, *Public Key Cryptography 2010*, pp.1–18 (2010).

[15] Kiltz, E.: Chosen-Ciphertext Secure Key-Encapsulation Based on Gap Hashed Diffie-Hellman, *Public Key Cryptography 2007*, pp.282–297 (2007).

[16] Krawczyk, H.: HMQV: A High-Performance Secure Diffie-Hellman Protocol, *CRYPTO 2005*, pp.546–566 (2005).

[17] LaMacchia, B., Lauter, K. and Mityagin, A.: Stronger Security of Authenticated Key Exchange, *ProvSec 2007*, pp.1–16 (2007).

[18] Lyubashevsky, V., Peikert, C. and Regev, O.: On Ideal Lattices and Learning with Errors over Rings, *EUROCRYPT 2010*, pp.1–23 (2010).

[19] Moriyama, D. and Okamoto, T.: An eCK-Secure Authenticated Key Exchange Protocol without Random Oracles, *ProvSec 2009*, pp.154–167 (2009).

[20] Naor, M.: On Cryptographic Assumptions and Challenges, *CRYPTO 2003*, pp.96–109 (2003).

[21] Nisan, N. and Zuckerman, D.: Randomness is Linear in Space, *J. Comput. Syst. Sci.*, Vol.52, No.1, pp.43–52 (1996).

[22] Nojima, R., Imai, H., Kobara, K. and Morozov, K.: Semantic security for the McEliece cryptosystem without random oracles, *Designs, Codes and Cryptography*, Vol.49, No.1-3, pp.289–305 (2008).

[23] Okamoto, T.: Authenticated Key Exchange and Key Encapsulation in the Standard Model, *ASIACRYPT 2007*, pp.474–484 (2007).

[24] Peikert, C. and Waters, B.: Lossy Trapdoor Functions and Their Applications, *STOC 2008*, pp.187–196 (2008).

[25] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography, *J. ACM*, Vol.56, No.6 (2009).

[26] Yang, Z. and Schwenk, J.: Strongly Authenticated Key Exchange Protocol from Bilinear Groups without Random Oracles, *ProvSec 2012*, pp.264–275 (2012).

[27] Yoneyama, K.: One-Round Authenticated Key Exchange with Strong Forward Secrecy in the Standard Model against Constrained Adversary, *IWSEC 2012*, pp.69–86 (2012).

[28] Yoneyama, K.: Generic Construction of Two-Party Round-Optimal Attribute-Based Authenticated Key Exchange without Random Oracles, *IEICE Transactions*, Vol.96-A, No.6, pp.1112–1123 (2013).

[29] Yoneyama, K.: One-Round Authenticated Key Exchange with Strong

Forward Secrecy in the Standard Model against Constrained Adversary, *IEICE Transactions*, Vol.96-A, No.6, pp.1124–1138 (2013).
[30] Yoneyama, K.: One-Round Authenticated Key Exchange without Implementation Trick, *IWSEC 2013*, pp.272–289 (2013).

**Kazuki Yoneyama** received his B.E., M.E. and Ph.D. degrees from the University of Electro-Communications, Tokyo, Japan, in 2004, 2006 and 2008, respectively. He was a researcher of NTT Secure Platform Laboratories from 2009 to 2015. He is presently engaged in research on cryptography at the Ibaraki University, since 2015. He is a member of the International Association for Cryptologic Research (IACR), IPSJ, IEICE and JSIAM.