Efficient eCK-secure Authenticated Key Exchange Protocols in the Standard Model

Zheng Yang

Horst Görtz Institute for IT Security
Chair for Network- and Data Security
Ruhr-University Bochum, Germany
zheng.yang@rub.de

Abstract

The extended Canetti–Krawczyk (eCK) security models, are widely used to provide security arguments for authenticated key exchange protocols that capture leakage of various kinds of secret information like the long-term private key and session-specific secret state. In this paper, we study the open problem on constructing eCK secure AKE protocol without random oracles and NAXOS like trick. A generic construction GC-KKN satisfying those requirements is first given relying on standard cryptographic primitives following the guideline of efficiency. On the second a concrete protocol is proposed which is the first eCK secure protocol in the standard model under both standard assumptions and post-specified peer setting. Both proposed schemes can be more efficiently implemented with secure device than previous eCK secure protocols in the standard model, where the secure device might be normally used to store the long-term private key and implement algorithms of protocol which require to be resilience of state leakage.

Keywords: eCK model, authenticated key exchange, key encapsulation mechanism, non-interactive key exchange

1 Introduction

Authenticated Key Exchange (AKE) is a fundamental cryptographic primitive which forms a crucial component in many network protocols. A two party AKE protocol is executed to enable both parties to end up sharing a session key with assurance that the key is only known to them. The security model for two party AKE and associated definitions have been evolved over years subjecting to increasing security requirements. Recently the Canetti-Krawczyk (CK) [8] and extended Canetti-Krawczyk (eCK) [26] models, are widely used to provide security arguments for AKE protocols. The CK model introduced a strong query StateReveal, to model the internal states leakage of any running session. In the eCK model, a new query EphemeralKeyReveal is used instead of StateReveal query which is claimed to cover almost all 'session-specific secret' information, and is allowed to be issued even on test session. Moreover there are quite a few variants of (e)CK model used in literatures (e.g., [25, 12, 16]). The CK+ model is recently used by Fujioka et al. [16], and the authors adopt StateReveal query in place of EphemeralKeyReveal query to model maximum exposure of states. Similar model has also been introduced before by Cremers [12] which was called eCK' model, where the StateReveal is used either. In this paper, we would like to call both CK+ and eCK' models as eCK model to avoid ambiguity since these models are based on the similar *freshness* restriction as the original eCK model that is distinct to the first CK model. The eCK model is known to be one of the strongest AKE models that covers the most desirable security attributes for AKE including resistance to key compromise impersonation (KCI) attacks, leakage of secret states and chosen identity and public key (CIDPK) attacks and provision of weak perfect forward secrecy (wPFS). Please note that both CK and eCK models leave out the definition of session state or ephemeral key to specific protocols. Since it is hard to define session state in a general approach, which is independent of any protocols and corresponding implementation scenarios. However the ambiguities on session state may yield a lot of potential problems in either the protocol construction or its security analysis. If any implementer realizes a specific AKE protocol in a *careless* way allowing it to leak non-trivial session state to attackers, then it would trivially invalidate the security proof in those strong models. On the other hand, to our best of knowledge, no AKE protocol is secure in the (e)CK model if 'all' session states can be revealed. Namely some session states of AKE protocols should be leakage resilience.[1]

Implementation Model vs. Session States. In order to fulfil the gap that often exists between formal models and practical security, Sarr et al. [35] introduced two implementation scenarios for the situation that at each party an untrusted host machine is used together with a secure device such as smart card. Similar modelling technique involving secure device was previously used by Bresson et al. [7]. A secure device may usually be used to store long-term cryptographic authentication keys and at least be able to fulfil a library of mathematical functions (such as addition, modulo and exponentiation) which are necessary to implement cryptographic operations or primitives. Hence based on secure device we are able to adopt a 'All-and-Nothing' strategy to define the states that can be revealed without leaving any ambiguity. General speaking we could assume that all intermediate states and ephemeral keys generated on host machine are susceptible to StateReveal attacks to model the maximum state leakage (MSL) attacks, but we treat the secure device as a black-box which is immune to leakage of internal states.[2] Of course one could distribute all protocol computations on the secure device then the security model would equal to a model without StateReveal query. However the security result of a protocol analysed with such implementation scenario must be weaker than that in a case allowing leakage of states. In contrast, our goal is to define the maximum states that can be leaked. As those secure devices might be short in both storage capacity and computational resource, the algorithm on secure device is often causing performance bottleneck of systems. In addition, the communication round between host machine and secure device (which is called *HS-round* for short) might cause another efficiency problem, since the serial I/O bus of most secure devices is too slow. Due to those facts, it is necessary to optimize AKE protocols when they are realized involving secure device.

NAXOS Trick Revisit. One of the most prominent open questions in the research field on AKE is how to securely implement the NAXOS trick [26]. Such trick is a technique that is introduced to hide the exponent (or de-facto ephemeral key) of an ephemeral public key from an adversary even if the adversary obtains the ephemeral secret key. Please first note that there might exist some variants of NAXOS trick in which the prominent example is the twisted-PRF trick recently used in FSXY scheme [16].[3] In the sequel, we are going to call all those variants as NAXOS trick. In a typical Diffie-Hellman key exchange protocol, the ephemeral secret key $\tilde{x}$ is used as the exponent of the ephemeral public key as $X = g^{\tilde{x}}$. However, in a protocol designed with NAXOS trick,

[1]Those critical session states might be, for instances, the pre-image of session key in HMQV protocol [25] and the decapsulation key of KEM in Boyd et al.'s generic construction [6].

[2]Although there might exist some side-channel attacks (such as [24]) against secure device, they are more likely to compromise the long-term key (which attracts the attackers mostly) rather than ephemeral key. Since such attacks might be very expensive.

[3]The twisted-PRF trick is first used in Okamoto's construction [33] to satisfy the eCK security in the standard model.

taking the twisted-PRF trick as example, the exponent of the ephemeral public key is generated as $x := \mathsf{PRF}(\tilde{x}', a) \oplus \mathsf{PRF}(a, \tilde{x})$. Therefore, even though the security model allows an adversary to obtain ephemeral secret key $\tilde{x}$, but the exponent of the ephemeral public key is still not exposed to the adversary. However, after applying the twisted-PRF trick, the attacker's interest may still be the exponent of Diffie-Helleman key. Then how to protect such exponent remains an open problem. Exploiting secure device might be one natural solution to protect the output of twisted-PRF trick, since the computation of such trick relies on long-term key which is always stored on secure device and should never be directly passed to host machine. Then the twisted-PRF trick can be realized as follows: (i) the host machine generates the ephemeral key $\tilde{x}$ and passes it to secure device; (ii) and then the secure device computes the de-facto ephemeral private key $x := \mathsf{PRF}(\tilde{x}, a) \oplus \mathsf{PRF}(a, \tilde{x})$ and keeps the x securely as long-term key, (iii) then compute the ephemeral public key $X = g^x$ and send it back to host machine. If the secure device sends the *new* generated ephemeral private x back to host machine where it will be used then it is also vulnerable to attacks aiming to the original $\tilde{x}$ chosen at host machine. This implementation approach can be applied to other NAXOS tricks.

Eventually, we could conclude that the NAXOS trick might be necessary if and only if the secure device is unable to generate the randomness. Otherwise the NAXOS trick can be seen identically to choosing the exponent x from a leakage-free random source, e.g. randomness generator of secure device. Besides, all other computations related to NAXOS trick might need to be done on secure device too (e.g. encapsulation algorithm of KEM in the FSXY protocol), since the ephemeral private key generated by NAXOS trick might be stored only on secure device. Those computations would dramatically increase the *burden* of secure device. On the other hand, the protocol with NAXOS trick is HS-round inefficient since it might need at least two HS-round, in which the first round is used for generate the ephemeral public key and the second round might be used to generate the final session key. In contrast, for a protocol without NAXOS trick, only one HS-round might be enough for session key generation.

Motivating problem. So far there are only few AKE protocols which are provably secure without random oracles in the eCK model. Although the protocols [33, 32, 38] were proven to be eCK secure in the standard model, they require a rather strong class of pseudo-random function family with pairwise independent random sources (which is referred to as $\pi\mathsf{PRF}$) as key derivation function (KDF). Most recently, Fujioka et al. [16] introduced a generic construction for two-message AKE from key encapsulation mechanism (KEM) which is generalized from BCNP [6]. Although the FSXY scheme [16] is shown to be eCK (CK+) secure in the standard model, it is built relying on a special twisted-PRF trick. The session states defined in FSXY scheme only include the random values used to execute the twisted-PRF trick and IND-CPA KEM. However it is not hard to see if either the ephemeral private key generated by twisted-PRF trick or the encapsulation key of test session is allowed to be leaked via $\mathsf{StateReveal}$ query (as the BCNP scheme), then the FSXY protocol is insecure in the eCK model. As discussed above, to securely implement the FSXY protocol, one might need to distribute all computations related to NAXOS trick on secure device. This would lead to inefficiencies in the implementation of the FSXY protocol with secure device. Another drawback of FSXY protocol is that it cannot be executed simultaneously by two session participants. Since, in the protocol description of FSXY, the responder cannot generate the outgoing ciphertext of IND-CPA KEM until it received the ephemeral public key sent by initiator. Hence the FSXY may lose an important feature of one-round AKE protocol.

So far, to our best of knowledge it is still an open question to construct eCK secure protocols without random oracles and without NAXOS trick under standard assumptions (e.g. without $\pi\mathsf{PRF}$). As those secure devices might be short in both storage capacity and computational resource, the algorithm on secure device is often causing performance bottleneck of systems. Thus it is necessary to seek efficient eCK secure construction which can be efficiently implemented with secure device.

Contributions. We first present an authenticated key exchange protocol (named GC-KKN) to solve the above open problem. In the construction of GC-KKN, we exploit the building blocks including strong randomness extractor (SEXT) family, and pseudo-random function (PRF) family, passively secure one-round key exchange (KE) protocols, IND-CCA secure key encapsulation mechanism (KEM) schemes, and CKS-light secure non-interactive key exchange (NIKE) schemes [15] (that is secure against chosen identity and long-term public key attacks). Each building block is required to cope with specific attacks following the guideline that the used assumption is as weak as possible. For example, to provide weak perfect forwards secrecy, the passively secure one-round key exchange is enough, whereas to deal with maximum states leakage we may need CKS-light secure non-interactive key exchange. The NIKE can be instantiated with the schemes based on factorization problem or Diffie-Hellman problem as proposed by Freire et al. [15]. Similar to FSXY protocol, the KEM in our construction could be instantiated using any IND-CCA secure scheme based on hardness of factorization problem, code-based problem, lattice-based problem and so forth. As opposed to FSXY scheme, GC-KKN does not rely on any NAXOS alike trick, that yields a more efficient solution when it is implemented with secure device. We give compact game-based proofs reducing eCK security of GC-KKN to break the used cryptographic primitives without random oracles.

On the second we present a practical AKE protocol (P1) that is eCK secure under standard assumptions (e.g. without $\pi\mathsf{PRF}$). The proposed protocol is based on bilinear pairings, target collision resistant hash function family, and pseudo-random function family. To be of independent interesting, P1 is able to run under post-specified peer setting [9] (i.e. without knowing any information of communication peer at session activation), unlike FSXY scheme and our GC-KKN scheme which might be executed under only pre-specified peer setting (i.e. they need the public key of peer to run the KEM). Our construction idea of P1 is inspired by the GC-KKN. We observe that it is possible to merge those computations in KE, KEM and NIKE schemes if they work under the same algebraic groups. In order to be secure against active attackers, each party (including the attacker) is required to construct some kind of 'tag' to encode consistency information on its chosen (either long-term or ephemeral) public keys based on specific weak Programmable Hash Function [18]. Those tags are particularly customized to be independent of any information about receivers, which enables P1 to be able to run in the post-specified peer setting. Although the *consistency check* of those tags might be relatively expensive, fortunately all pairing (including partial session key material generation) can be done on more powerful host machine rather than on computational resource-limited secure device (which is only used to generate final session key). In order to securely implement P1, only one exponentiation is required on secure device that is the more efficient than any previous eCK secure protocols without random oracles.

Related Works. In 1986, Matsumoto et al. [28] first studied the Diffie-Hellman based key exchange protocols with implicit key authentication that result in a line of research on one-round AKE. Meanwhile, the most famous and efficient one is the MQV protocol by Menezes, Qu, and Vanstone [30, 27]. On Crypto 2005 conference, Krawczyk [25] presented a hashed variant of MQV called HMQV, which is formally proven secure in a modified CK [8] model referred as $\mathsf{CH}_{\mathsf{HMQV}}$, relying on random oracles and strong assumptions (i.e. gap Diffie-Hellman assumption [34] and knowledge of exponent assumption [2]). Krawczyk also showed that HMQV protocol offers more security features than MQV, in particular for resistance to KCI attacks, UKS attacks and the leakage of secret states. However Menezes [29] pointed out that the chosen identity and long-term public key attacks on HMQV are not formally studied in [25]. Besides, the application of secure module was also suggested by HMQV protocol [25] to prevent the leakage of intermediate secrets. But the detailed implementation approach was not elaborated.

In 2007 LaMacchia et al. [26] proposed an extended Canetti-Krawczyk (eCK) model, in which the adversary is equipped with a $\mathsf{EphemeralKeyReveal}$ query to access all ephemeral private input required to carry on session key computations which is similar to the $\mathsf{StateReveal}$ query in the CK model. e.g. [35, 37, 33, 32, 38], are proposed to capture eCK security. However, most of those schemes are only provably secure in the random oracle model. Since then many AKE protocols many protocols, e.g. [33, 32, 38], are proposed to provide eCK security without random oracles. But they require the $\pi\mathsf{PRF}$ family as key derivation function that is hard to realize in practical.

As for generic two party AKE constructions, e.g. the BCNP [6] and FSXY [16] schemes based on IND-CCA secure KEM, the session states that can be revealed are never clearly defined which makes theirs security analysis incomplete. In particular, we notice that the twisted-PRF technique used in the FSXY protocol (also used in [33]) may need to be implemented with secure device to achieve leakage resilience on new generated ephemeral private key by such trick. As the above discussion regarding NAXOS trick, that might increase unnecessary workload on secure device. Basically the FSXY protocol is built in an inefficient manner. Please note that there are a lot of works such as [32, 37, 23] which are motivated to construct key exchange protocols without the NAXOS tricks. As well this is also one of the motivations of our work.

With respect to the protection of session states involving secure device, two implementation scenarios and corresponding security models have been studied in literatures [35, 39]. Basically, the implementation of the AKE protocol is divided into two parts which are respectively run at host machine (which is subject to attacks on session states) and at secure device. In corresponding security model, the adversary is able to reveal all possible states stored at host machine, but the secure device is treated as a black-box which is resilience of the leakage of intermediate values.

2 Preliminaries

Notations. We let $\kappa \in \mathbb{N}$ denote the security parameter and 1^κ the string that consists of κ ones. Let a "hat" on top of a capital letter denote an identity; without the hat the letter denotes the public key of that party. Let $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ be the set of integers between 1 and n. If S is a set, then $a \overset{\$}{\leftarrow} S$ denotes the action of sampling a uniformly random element from S. Let $\mathcal{IDS}$ be an identity space. Let $\mathcal{K}_{\mathsf{AKE}}$ be the key space of session key, and $\{\mathcal{PK}, \mathcal{SK}\}$ be key spaces for long-term public/private key respectively. Those spaces are associated with security parameter κ.

2.1 Min-entropy and Strong Randomness Extractors

Definition 1. We say that a random variable m^* distributed over domain $\mathcal{M}$ has min-entropy κ, if for all $m \in \mathcal{M}$ it holds that $\Pr[m^* = m] \leq 2^{-\kappa}$.

The min-entropy is a formal indicator for 'good' key distribution of a key (say the key of following KE scheme, KEM scheme and NIKE scheme).

Let $\mathsf{SEXT} : \mathcal{S}_{\mathsf{SEXT}} \times \mathcal{M}_{\mathsf{SEXT}} \rightarrow \mathcal{R}_{\mathsf{SEXT}}$ be a function family associated with seed space $\mathcal{S}_{\mathsf{SEXT}}$, domain $\mathcal{M}_{\mathsf{SEXT}}$, and range $\mathcal{R}_{\mathsf{SEXT}}$.

Definition 2. We say that function SEXT is a $(\kappa, \epsilon_{\mathsf{SEXT}})$-strong randomness extractor, if for any variable m distributed over $\mathcal{M}_{\mathsf{SEXT}}$ that has min-entropy κ and for any seed k_{SEXT} which is chosen uniformly at random from $\mathcal{S}_{\mathsf{SEXT}}$ and for any value R which is chosen uniformly at random from

$\mathcal{R}_{\mathsf{SEXT}}$, the two distributions $\langle k_{\mathsf{SEXT}}, \mathsf{SEXT}(k_{\mathsf{SEXT}}, m)\rangle$ and $\langle k_{\mathsf{SEXT}}, R\rangle$ have statistical distance at most ϵ_{SEXT}. i.e.

$$\frac{1}{2}\sum_{y\in\mathcal{R}_{\mathsf{SEXT}}} |\Pr[\mathsf{SEXT}(k_{\mathsf{SEXT}}, m) = y] - \Pr[R = y]| = \epsilon_{\mathsf{SEXT}}.$$

In the context where the seed k_{SEXT} is clear we will write $\mathsf{SEXT}(X)$ for $\mathsf{SEXT}(k_{\mathsf{SEXT}}, X)$. As suggested by Dodis et al. [14], one could use a pseudo-random function as a strong randomness extractor. Some good results on key derivation and randomness extraction can be also found in [10].

2.2 Target Collision-Resistant Hash Functions

Let $\mathsf{TCRHF} : \mathcal{K}_{\mathsf{TCRHF}} \times \mathcal{M}_{\mathsf{TCRHF}} \to \mathcal{Y}_{\mathsf{TCRHF}}$ be a family of keyed-hash functions associated with key space $\mathcal{K}_{\mathsf{TCRHF}}$, message space $\mathcal{M}_{\mathsf{TCRHF}}$ and hash value space $\mathcal{Y}_{\mathsf{TCRHF}}$. The public key $hk_{\mathsf{TCRHF}} \in \mathcal{K}_{\mathsf{TCRHF}}$ of a hash function $\mathsf{TCRHF}(hk_{\mathsf{TCRHF}}, \cdot)$ is generated by a PPT algorithm $\mathsf{TCRHF.KG}(1^\kappa)$ on input security parameter κ.

Definition 3. TCRHF is called $(t_{\mathsf{TCRHF}}, \epsilon_{\mathsf{TCRHF}})$-target-collision-resistant if for all t_{TCRHF}-time adversaries $\mathcal{A}$ it holds that

$$\Pr\begin{bmatrix} hk_{\mathsf{TCRHF}} \xleftarrow{\$} \mathsf{TCRHF.KG}(1^\kappa),\ m \xleftarrow{\$} \mathcal{M}_{\mathsf{TCRHF}},\ m' \leftarrow \mathcal{A}(1^\kappa, hk_{\mathsf{TCRHF}}, m), \\ m \neq m',\ m' \in \mathcal{M}_{\mathsf{TCRHF}},\ \mathsf{TCRHF}(hk_{\mathsf{TCRHF}}, m) = \mathsf{TCRHF}(hk_{\mathsf{TCRHF}}, m') \end{bmatrix} \leq \epsilon_{\mathsf{TCRHF}},$$

where the probability is over the random bits of $\mathcal{A}$.

Note that the notion of target collision resistance is weaker than the notion of collision resistance. As suggested in [11], normally target collision resistant functions can be realized with a specific cryptographic hash function such as MD5 and SHA. If the hash key hk_{TCRHF} is obvious from the context, we write $\mathsf{TCRHF}(m)$ for $\mathsf{TCRHF}(hk_{\mathsf{TCRHF}}, m)$.

2.3 Passively Secure One-round Key Exchange Protocols

A key-exchange protocol (KE) is a protocol executed between two parties, that enables those two parties to compute a shared secret key. In the following, we formally provide a technical definition of passively secure (PS) key exchange protocols within two pass.

A one-round key exchange protocol $\mathsf{KE} = (\mathsf{KE.Setup}, \mathsf{KE.EKGen}, \mathsf{KE.SKGen})$ consists of three algorithms:

- $pms^{ke} \leftarrow \mathsf{KE.Setup}(1^\kappa)$: this algorithm takes as input a security parameter κ and outputs a set of system parameters pms^{ke}, e.g. a large prime and a group generator. The parameters pms^{ke} might be implicitly used by other algorithms for simplicity.
- $(esk, epk) \xleftarrow{\$} \mathsf{KE.EKGen}(pms^{ke})$: The ephemeral key generator takes as input, the parameter pms^{ke}, and outputs an ephemeral key pair (esk, epk) that consist of the ephemeral secret key esk and the ephemeral public key epk.
- $k \leftarrow \mathsf{KE.SKGen}(esk_{\mathsf{ID}_1}, epk_{\mathsf{ID}_2})$: The session key generator KE.SKGen is a deterministic polynomial-time algorithm that given as input esk_{ID_1} and epk_{ID_2} outputs the session key k.

The ephemeral public keys are exchanged by the parties, e.g. party ID_1 sends epk_{ID_1} to party ID_2 who sends epk_{ID_2} to ID_1. We only consider the key exchange protocols with perfect correctness that is

$$Pr\begin{bmatrix} \mathsf{KE.SKGen}(esk_{\mathsf{ID}_1}, epk_{\mathsf{ID}_2}) = \mathsf{KE.SKGen}(esk_{\mathsf{ID}_2}, epk_{\mathsf{ID}_1}); \\ (esk_{\mathsf{ID}_1}, epk_{\mathsf{ID}_1}) \leftarrow \mathsf{KE.EKGen}(pms^{ke}), (esk_{\mathsf{ID}_2}, epk_{\mathsf{ID}_2}) \leftarrow \mathsf{KE.EKGen}(pms^{ke}) \end{bmatrix} = 1.$$

The KE protocol in our construction should satisfy the following two conditions: (i) the protocol is executed without any long-term key(s); (ii) all ephemeral public/secret key vector (epk, esk) are chosen freshly and randomly from corresponding key spaces for each protocol instance. Similar restrictions are made on the KE protocols which are used in the recent AKE compiler by Jager, Kohlar, Schaege, and Schwenk (JKSS) [20].

We haven chosen to restrict our attention to this class of key exchange protocols because they either allow for efficient protocols with very high security guarantees (like forward secrecy) or can efficiently be recognized. We stress that important key exchange mechanisms like ephemeral Diffie-Hellman key exchange falling into this class. In order to model passive attacks we define an $\mathsf{Execute}(\mathsf{ID}_1, \mathsf{ID}_2)$ query. The adversary can use the this query to perform passive attacks in which the attacker initiates and eavesdrops on honest executions between party ID_1 and party ID_2. Note that each identity should be uniquely chosen from space $\mathcal{IDS}$. By using this query the adversary can obtain the transcripts that were exchanged during the honest execution of the protocol, and corresponding established session key.

Security Experiment $\mathsf{EXP}^{ps}_{\mathsf{KE},\mathcal{A}}(\kappa)$: On input security parameter κ, the security experiment is proceeded as a game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ based on a key exchange protocol KE, where the following steps are performed:

1. A challenger $pms^{ke} \leftarrow \mathsf{KE.Setup}(1^\kappa)$ and generates a set of identities $\{\mathsf{ID}_1, \ldots, \mathsf{ID}_\ell\}$ for potential protocol participants where $i \in [\ell]$ and $\ell \in \mathbb{N}$. The adversary is given pms^{ke} and all identities as input and is allowed to interact with challenger via making $\mathsf{Execute}(\mathsf{ID}_i, \mathsf{ID}_j)$ query at most d times for each party where $d \in \mathbb{N}$. As response, the challenger returns (T, K_0) to adversary.

2. At some point, the adversary outputs a special symbol $\top$ for challenge. Given $\top$, the challenger runs a new protocol instance, obtaining the transcript T and key K_0, samples K_1 uniformly at random from the key space of the protocol, and tosses a fair coin $b \in \{0, 1\}$. Then it returns (T, K_b) to the adversary. After that the adversary may continue making $\mathsf{Execute}(\mathsf{ID}_i, \mathsf{ID}_j)$ queries. Finally, adversary $\mathcal{A}$ may terminate with outputting a bit b'.

3. At the end of the experiment, 1 is returned if all following conditions hold:

   - $\mathcal{A}$ has issued a Test query to an oracle π_i^s without failure,
   - π_i^s is fresh throughout the security game, and
   - $\mathcal{A}$ returned a bit b' which equals to b of Test-query;

   Otherwise 0 is returned.

Definition 4. We say that a key-exchange protocol KE satisfying the above two conditions is $(t, \epsilon_{\mathsf{KE}})$-*passively-secure* if for all probabilistic polynomial-time t adversary holds that
$|\Pr[\mathsf{EXP}^{ke}_{\Sigma,\mathcal{A}}(\kappa) = 1] - 1/2| \leq \epsilon_{\mathsf{KE}}$ for some negligible function $\epsilon_{\mathsf{KE}} = \epsilon_{\mathsf{KE}}(\kappa)$ in the security parameter κ.

2.4 Key Encapsulation Mechanism Schemes

A KEM scheme consists of four polynomial time algorithms $\mathsf{KEM} = (\mathsf{KEM.Setup}, \mathsf{KEM.Gen},$
$\mathsf{KEM.EnCap}, \mathsf{KEM.DeCap})$ with the following semantics:

- $pms^{kem} \leftarrow \mathsf{KEM.Setup}(1^\kappa)$: this algorithm takes as input a security parameter κ and outputs a set of system parameters pms^{kem}. The parameters pms^{kem} might be implicitly used by other algorithms for simplicity.

- $(pk, sk) \xleftarrow{\$} \mathsf{KEM.Gen}(pms^{kem})$: a key generation algorithm which on input parameter pms^{kem}, outputs a pair of long-term encryption/decryption keys $(pk, sk) \in (\mathcal{PK}, \mathcal{SK})$.

- $(K, C) \xleftarrow{\$} \mathsf{KEM.EnCap}(pk)$: an encryption algorithm which takes as input an encryption key pk, outputs a key $K \in \mathcal{K}_{\mathsf{KEM}}$ and ciphertext $C \in \mathcal{C}_{\mathsf{KEM}}$, where $\mathcal{K}_{\mathsf{KEM}}$ is a session key space, and $\mathcal{C}_{\mathsf{KEM}}$ is a ciphertext space.

- $(K) \leftarrow \mathsf{KEM.DeCap}(sk, C)$: a decryption algorithm which takes as input a decryption key sk and a ciphertext $C \in \mathcal{C}_{\mathsf{KEM}}$, and outputs a key $K \in \mathcal{K}_{\mathsf{KEM}}$.

All 'spaces' for the corresponding values are parametrized with security parameter κ. Here, we recall the definition of IND-CCA security for KEM as follows.

Definition 5. We say that a key encapsulation mechanism scheme KEM is $(q, t, \epsilon_{\mathsf{KEM}})$-secure (key indistinguishable) against adaptive chosen-ciphertext attacks, if it holds that $|\Pr[\mathsf{EXP}^{ind-cca}_{\mathsf{KEM},\mathcal{A}}(\kappa) = 1] - 1/2| \leq \epsilon_{\mathsf{KEM}}$ for all adversaries $\mathcal{A}$ running in probabilistic polynomial time t in the following experiment:

```
SECURITY EXPERIMENT $\mathsf{EXP}^{ind-cca}_{\mathsf{KEM},\mathcal{A}}(\kappa)$
  $pms^{kem} \leftarrow \mathsf{KEM.Setup}(1^\kappa)$
  $(pk, sk) \xleftarrow{\$} \mathsf{KEM.Gen}(pms^{kem})$
  $(K_0^*, C^*) \xleftarrow{\$} \mathsf{KEM.EnCap}(pk)$, $K_1^* \xleftarrow{\$} \mathcal{K}_{\mathsf{KEM}}$
  $b \xleftarrow{\$} \{0,1\}, b' \leftarrow \mathcal{A}^{\mathcal{DEC}(sk,\cdot)}(pk, K_b^*, C^*)$
  if $b = b'$ then return 1, otherwise return 0

$\mathcal{DEC}(sk, C)$ :
  If $C = C^*$ or $C \notin \mathcal{C}_{\mathsf{KEM}}$ then return $\perp$,
  Otherwise $K \leftarrow \mathsf{KEM.DeCap}(sk, C)$
  Return $K$
```

where $\epsilon_{\mathsf{KEM}} = \epsilon_{\mathsf{KEM}}(\kappa)$ is a negligible function in the security parameter κ and the number of $\mathcal{DEC}(sk, \cdot)$ queries is bound by time t.

2.5 Pseudo-Random Functions

The concept of pseudo-random functions is introduced by Goldreich, Goldwasser and Micali in [17]. Let $\mathsf{PRF} : \mathcal{K}_{\mathsf{PRF}} \times \mathcal{D}_{\mathsf{PRF}} \to \mathcal{R}_{\mathsf{PRF}}$ denote a family of deterministic functions, where $\mathcal{K}_{\mathsf{PRF}}$ is the key space, $\mathcal{D}_{\mathsf{PRF}}$ is the domain and $\mathcal{R}_{\mathsf{PRF}}$ is the range of PRF for security parameter κ. Let $\mathsf{RL} = \{(x_1, y_1), \ldots, (x_q, y_q)\}$ be a list which is used to record bit strings formed as tuple $(x_i, y_i) \in (\mathcal{D}_{\mathsf{PRF}}, \mathcal{R}_{\mathsf{PRF}})$ where $1 \leq i \leq q$ and $q \in \mathbb{N}$. In RL each x is associated with a y. Let $\mathsf{RF} : \mathcal{D}_{\mathsf{PRF}} \to \mathcal{R}_{\mathsf{PRF}}$ be a stateful uniform random function, which can be executed at most a polynomial number of q times and keeps a list RL for recording each invocation. On input a message $x \in \mathcal{D}_{\mathsf{PRF}}$, the function $\mathsf{RF}(x)$ is executed as follows:

- If $x \in \mathsf{RL}$, then return corresponding $y \in \mathsf{RL}$,

- Otherwise return $y \xleftarrow{\$} \mathcal{R}_{\mathsf{PRF}}$ and record (x, y) into RL.

Definition 6. We say that PRF is a $(q, t, \epsilon_{\mathsf{PRF}})$-secure pseudo-random function family, if it holds that $|\Pr[\mathsf{EXP}^{ind-cma}_{\mathsf{PRF},\mathcal{A}}(\kappa) = 1] - 1/2| \leq \epsilon_{\mathsf{PRF}}$ for all adversaries $\mathcal{A}$ that make a polynomial number of oracle queries q while running in time at most t in the following experiment:

```
$\mathsf{EXP}^{ind-cma}_{\mathsf{PRF},\mathcal{A}}(\kappa)$
  $b \xleftarrow{\$} \{0,1\}, k \xleftarrow{\$} \mathcal{K}_{\mathsf{PRF}}$;
  $b' \leftarrow \mathcal{A}^{\mathcal{F}(b,\cdot)}(\kappa)$;
  If $b = b'$ then return 1;
  Otherwise return 0;

$\mathcal{F}(b, x)$
  If $x \notin \mathcal{D}_{\mathsf{PRF}}$ then return $\perp$;
  If $b = 1$ then return $\mathsf{PRF}(k, x)$;
  Otherwise return $\mathsf{RF}(x)$;
```

where $\epsilon_{\mathsf{PRF}} = \epsilon_{\mathsf{PRF}}(\kappa)$ is a negligible function in the security parameter κ, and the number of allowed queries q is bound by t.

2.6 Secure Non-Interactive Key Exchange Protocols

Notions for Non-Interactive Key Exchange. We consider a Non-Interactive Key Exchange (NIKE) scheme in the public key setting consists of three algorithms: $\mathsf{NIKE.Setup}$, KGen and $\mathsf{NIKE.ShareKey}$ associated with an identity space $\mathcal{IDS}$ and a shared key space $\mathcal{K}_{\mathsf{NIKE}}$, in which those algorithms have following semantics:

- $pms^{nike} \leftarrow \mathsf{NIKE.Setup}(1^{\kappa})$: this algorithm takes as input a security parameter κ and outputs a set of system parameters pms^{nike}. The parameters pms^{nike} might be implicitly used by other algorithms for simplicity.
- $(sk_{\mathsf{ID}}, pk_{\mathsf{ID}}) \overset{\$}{\leftarrow} \mathsf{KGen}(pms^{nike}, \mathsf{ID})$: this algorithm takes as input pms^{nike} and a identity ID, and outputs a pair of long-term private/public key $(sk_{\mathsf{ID}}, pk_{\mathsf{ID}})$ for the party ID.
- $K \overset{\$}{\leftarrow} \mathsf{NIKE.ShareKey}(\mathsf{ID}_1, sk_{\mathsf{ID}_1}, \mathsf{ID}_2, pk_{\mathsf{ID}_2})$: this algorithm takes as input parameter pms^{nike}, an identity ID_1 and a secret key sk_{ID_1} along with another identity ID_2 and corresponding public key pk_{ID_2}, and outputs either a shared key $K \in \mathcal{K}_{\mathsf{NIKE}}$ for the two parties, or a failure symbol $\perp$. This algorithm is assumed to always output $\perp$ if input identities are not distinct.

  For correctness, we require that, for a tuple of identities $(\mathsf{ID}_1, \mathsf{ID}_2)$, and corresponding key pairs $(sk_{\mathsf{ID}_1}, pk_{\mathsf{ID}_1})$ and $(sk_{\mathsf{ID}_2}, pk_3)$, algorithm $\mathsf{NIKE.ShareKey}$ satisfies the constraint:

  – $\mathsf{NIKE.ShareKey}(\mathsf{ID}_1, sk_{\mathsf{ID}_1}, \mathsf{ID}_2, pk_{\mathsf{ID}_2}) = \mathsf{NIKE.ShareKey}(\mathsf{ID}_2, sk_{\mathsf{ID}_2}, \mathsf{ID}_1, pk_{\mathsf{ID}_1})$

Security Definition for NIKE. In this section we recall the CKS-light formal security model for a two party PKI-based non-interactive authenticated key-exchange (NIKE) protocol proposed in [15]. But we do slightly modification on modelling public key registration. Specifically, each party ID_i might be required to provide extra information (denoted by $\mathsf{pf}_{\mathsf{ID}_i}$) to prove the registered public key is sound. In practice, the concrete implementation of pf is up to the CA [1] and may be either interactive or non-interactive. Examples can be found in RFC 4210 [1] and PKCS#10. Let $\{\mathsf{Honest}, \mathsf{Dishonest}\}$ be two vector lists.

Security Experiment $\mathsf{EXP}^{cks-light}_{\mathsf{NIKE},\mathcal{A}}(\kappa)$: On input security parameter κ, the security experiment is proceeded as a game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ based on a non-interactive key exchange protocol NIKE, where the following steps are performed:

1. The $\mathcal{C}$ first run $pms^{nike} \leftarrow \mathsf{NIKE.Setup}(1^{\kappa})$ and gives pms^{nike} to adversary $\mathcal{A}$.
2. The adversary $\mathcal{A}$ may interact with challenger $\mathcal{C}$ with the following queries:
   - $\mathsf{RegisterHonest}(\mathsf{ID})$: on input an identity $\mathsf{ID} \in \mathcal{IDS}$, if $\mathsf{ID} \notin \{\mathsf{Honest}, \mathsf{Dishonest}\}$ then $\mathcal{C}$ runs
     $\mathsf{NIKE.KGen}(pms^{nike}, \mathsf{ID})$ to generate a long-term secret/public key pair $(sk_{\mathsf{ID}}, pk_{\mathsf{ID}}) \in (\mathcal{PK}, \mathcal{SK})$ and adds the tuple $(\mathsf{ID}, sk_{\mathsf{ID}}, pk_{\mathsf{ID}})$ into the list Honest, and returns pk to $\mathcal{A}$; as otherwise a failure symbol $\perp$ is returned. This queries is allowed to query at most twice. Parties established by this query are called honest.

- $\mathsf{RegisterCorrupt}(\mathsf{ID}_\tau, pk_{\mathsf{ID}_\tau}, \mathsf{pf}_{\mathsf{ID}_\tau})$: This query allows the adversary to register an identity ID_τ and a long-term public key pk_{ID_τ} on behalf of a party ID_τ, if the $\mathsf{ID}_\tau \notin \{\mathsf{Honest}, \mathsf{Dishonest}\}$ and pk_{ID_τ} is ensured to be sound by evaluating the non-interactive proof $\mathsf{pf}_{\mathsf{ID}_\tau}$. We only require that the proof is non-interactive in order to keep the model simple. Parties established by this query are called dishonest.
- $\mathsf{RevealKey}^{nike}(\mathsf{ID}_1, \mathsf{ID}_2)$: On input a tuple of registered identities $(\mathsf{ID}_1, \mathsf{ID}_2)$, $\mathcal{C}$ returns a failure symbol $\perp$ if both parties ID_1 and ID_2 are dishonest. Otherwise $\mathcal{C}$ run $\mathsf{NIKE.ShareKey}$ using the secret key of one of the honest parties in $(\mathsf{ID}_1, \mathsf{ID}_2)$ and the public key of the other party and returns the result to $\mathcal{A}$
- $\mathsf{Test}^{nike}(\mathsf{ID}_1, \mathsf{ID}_2)$: Given two identities $(\mathsf{ID}_1, \mathsf{ID}_2)$, the challenger $\mathcal{C}$ returns a failure symbol $\perp$ if either $\mathsf{ID}_1 = \mathsf{ID}_2$ or $\mathsf{ID}_1 \notin \mathsf{Honest}$ or $\mathsf{ID}_2 \notin \mathsf{Honest}$. Otherwise the challenger $\mathcal{C}$ samples a random bit $b \overset{\$}{\leftarrow} \{0,1\}$, and it answers this query in terms of the bit b. Specifically, if $b = 1$, $\mathcal{C}$ runs $\mathsf{NIKE.ShareKey}$ using the secret key of ID_1 and the public key of ID_2 to obtain the shared key K_1; else if $b = 0$, the challenger generates a random key K_1. $\mathcal{C}$ returns K_b to adversary. This query can be queried only once.

3. Eventually, the adversary may terminate with outputting a bit b'.
4. At the end of the experiment, 1 is returned if all following conditions hold:
   - $\mathcal{A}$ has issued a Test^{nike} query without failure on input identities $(\mathsf{ID}_1, \mathsf{ID}_2)$,
   - Both parties ID_1 and ID_2 are honest,
   - $\mathcal{A}$ has not issued $\mathsf{RevealKey}$ query on input identities $(\mathsf{ID}_1, \mathsf{ID}_2)$ in either order, and
   - $b = b'$;

   Otherwise 0 is returned.

Definition 7. A two party non-interactive key exchange protocol Σ is called $(t, \epsilon_{\mathsf{NIKE}})$-secure if it holds that $|\Pr[\mathsf{EXP}^{cks-light}_{\Sigma,\mathcal{A}}(\kappa) = 1] - 1/2| \leq \epsilon_{\mathsf{NIKE}}$ for all adversaries $\mathcal{A}$ running within time t in the above security experiment and for some negligible probability $\epsilon_{\mathsf{NIKE}} = \epsilon_{\mathsf{NIKE}}(\kappa)$ in the security parameter κ.

Note that if the query $\mathsf{RegisterCorrupt}(\mathsf{ID}_\tau, pk_{\mathsf{ID}_\tau}, \mathsf{pf}_{\mathsf{ID}_\tau})$ is made with $\mathsf{pf} = \emptyset$ then the above model equals the CKS-light model [15]; Otherwise it is slightly weaker than CKS-light model. The number of $\mathsf{RegisterCorrupt}$ queries is bound by the time t. In this model, we adopt a slightly variant CKS-light model for simplicity which is strong enough for our AKE construction.

2.7 Bilinear Groups

In the following, we briefly recall some of the basic properties of bilinear groups. Our AKE solution mainly consists of elements from a single group $\mathbb{G}$. We therefore concentrate on symmetric bilinear maps. The bilinear groups will be parameterized by a symmetric pairing parameter generator, denoted by $\mathsf{PG.Gen}$. This is a polynomial time algorithm that on input a security parameter 1^κ, returns the description of two multiplicative cyclic groups $\mathbb{G}$ and $\mathbb{G}_T$ of the same prime order p, generator g for $\mathbb{G}$, and a bilinear computable pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$.

Definition 8 (Symmetric Bilinear groups). We call $\mathcal{PG} = (\mathbb{G}, g, \mathbb{G}_T, p, e) \overset{\$}{\leftarrow} \mathsf{PG.Gen}(1^\kappa)$ be a set of symmetric bilinear groups, if the function e is an (admissible) bilinear map and it holds that:

1. **Bilinear:** for all $a, b \in \mathbb{G}$ and $x, y \in \mathbb{Z}_p$, we have $e(a^x, b^y) = e(a, b)^{xy}$.

2. **Non-degenerate:** $e(g,g) \neq 1_{\mathbb{G}_T}$, is a generator of group $\mathbb{G}_T$.

3. **Efficiency:** e is efficiently computable for all $(a,b) \in \mathbb{G}$.

2.8 Bilinear Decisional Diffie-Hellman Assumption

Let $\mathcal{PG} = (\mathbb{G}, g, \mathbb{G}_T, p, e)$ denote the description of symmetric bilinear groups as Definition 8. We first consider the following traditional version of Bilinear Decisional Diffie-Hellman (BDDH) problem for symmetric pairing. The Bilinear Decisional Diffie-Hellman (BDDH) problem is stated as follows: given tuple $(g^a, g^b, g^c, e(g,g)^\gamma)$ for $(a,b,c,\gamma) \in (\mathbb{Z}_p^*)^4$ as input, output 1 if $e(g,g)^\gamma = e(g,g)^{abc}$ and 0 otherwise.

Definition 9. We say that the BDDH problem relative to generator $\mathsf{PG.Gen}$ is $(t, \epsilon_{\mathsf{BDDH}})$-hard, if the probability bound $|\Pr[\mathsf{EXP}^{bddh}_{\mathsf{PG.Gen},\mathcal{A}}(\kappa) = 1] - 1/2| \leq \epsilon_{\mathsf{BDDH}}$ holds for all adversaries $\mathcal{A}$ running in probabilistic polynomial time t in the following experiment:

```
$\mathsf{EXP}^{bddh}_{\mathsf{PG.Gen},\mathcal{A}}(\kappa)$
    $\mathcal{PG} = (\mathbb{G}, g, \mathbb{G}_T, p, e) \overset{\$}{\leftarrow} \mathsf{PG.Gen}(1^\kappa)$;
    $(a,b,c,\gamma) \overset{\$}{\leftarrow} \mathbb{Z}_p^*$;
    $b \overset{\$}{\leftarrow} \{0,1\}$, if $b = 1$ $\Gamma \leftarrow e(g,g)^{abc}$, otherwise $\Gamma \leftarrow e(g,g)^\gamma$;
    $b' \leftarrow \mathcal{A}(1^\kappa, \mathcal{PG}, g^a, g^b, g^c, \Gamma)$;
    if $b = b'$ then return 1, otherwise return 0;
```

where $\epsilon_{\mathsf{BDDH}} = \epsilon_{\mathsf{BDDH}}(\kappa)$ is a negligible function in the security parameter κ.

3 Security Model

In this section we present the formal security model for two party PKI-based authenticated key-exchange (AKE) protocol. In this model, while emulating the real-world capabilities of an active adversary, we provide an 'execution environment' for adversaries following an important research line research [5, 8, 25, 26, 21, 38] which is initiated by Bellare and Rogaway [3]. In the sequel, we will use the similar framework as [21].

Execution Environment. In the execution environment, we fix a set of honest parties $\{\mathsf{ID}_1, \ldots, \mathsf{ID}_\ell\}$ for $\ell \in \mathbb{N}$, where ID_i ($i \in [\ell]$) is the identity of a party which is chosen uniquely from space $\mathcal{IDS}$. Each identity is associated with a long-term key pair $(sk_{\mathsf{ID}_i}, pk_{\mathsf{ID}_i}) \in (\mathcal{SK}, \mathcal{PK})$ for authentication. Note that those identities are also lexicographically indexed via variable $i \in [\ell]$. Each honest party ID_i can sequentially and concurrently execute the protocol multiple times with different indented partners, this is characterized by a collection of oracles $\{\pi_i^s : i \in [\ell], s \in [d]\}$ for $d \in \mathbb{N}$.[4] Oracle π_i^s behaves as party ID_i carrying out a process to execute the s-th protocol instance (session), which has access to the long-term key pair $(sk_{\mathsf{ID}_i}, pk_{\mathsf{ID}_i})$ and to all other public keys. Moreover, we assume each oracle π_i^s maintains a list of independent internal state variables with semantics listed in Table 1.

All those variables of each oracle are initialized with empty string which is denoted by the symbol $\emptyset$ in the following. At some point, each oracle π_i^s may complete the execution always with a decision state $\Phi_i^s \in \{\texttt{accept}, \texttt{reject}\}$. Furthermore, we assume that the session key is assigned to the variable K_i^s (such that $K_i^s \neq \emptyset$) iff oracle π_i^s has reached an internal state $\Phi_i^s = \texttt{accept}$.

[4] An oracle in this paper might be alternatively written as $\pi^s_{\mathsf{ID}_i}$ which is conceptually equivalent to π_i^s.

Variable	Decryption
Ψ_i^s	storing the identity of its communication partner
Φ_i^s	denoting the decision $\Phi_i^s \in \{\texttt{accept}, \texttt{reject}\}$
K_i^s	recording the session key $K_i^s \in \mathcal{K}_{\mathsf{AKE}}$ used for symmetric encryption
st_i^s	storing the maximum secret states that allow to be revealed by adversary
sT_i^s	recording the transcript of messages sent by oracle π_i^s
rT_j^t	recording the transcript of messages received by oracle π_i^s

Table 1: Internal States of Oracles

Adversarial Model. An adversary $\mathcal{A}$ in our model is a PPT Turing Machine taking as input the security parameter 1^κ and the public information (e.g. generic description of above environment), which may interact with these oracles by issuing the following queries.

- $\mathsf{Send}(\pi_i^s, m)$: The adversary can use this query to send any message m of his own choice to oracle π_i^s. The oracle will respond the next message m^* (if any) to be sent according to the protocol specification and its internal states. Oracle π_i^s would be initiated as *initiator* via sending the oracle the first message $m = (\top, \widetilde{\mathsf{ID}_j})$ consisting of a special initialization symbol $\top$ and a value $\widetilde{\mathsf{ID}_j}$. The $\widetilde{\mathsf{ID}_j}$ is either the identity ID_j of intended partner or empty string $\emptyset$. After answering a Send query, the variables $(\Psi_i^s, \Phi_i^s, K_i^s, st_i^s, sT_i^s, rT_i^s)$ will be updated depending on the specific protocol.[5]

- $\mathsf{RevealKey}(\pi_i^s)$: Oracle π_i^s responds with the contents of variable K_i^s.

- $\mathsf{StateReveal}(\pi_i^s)$: Oracle π_i^s responds with the secret state stored in variable st_i^s.

- $\mathsf{Corrupt}(\mathsf{ID}_i)$: Oracle π_i^1 responds with the long-term secret key sk_{ID_i} of party ID_i if $i \in [\ell]$.

- $\mathsf{RegisterCorrupt}(\mathsf{ID}_\tau, pk_{\mathsf{ID}_\tau}, \mathsf{pf}_{\mathsf{ID}_\tau})$: This query allows the adversary to register an identity ID_τ $(\ell < \tau < \mathbb{N})$ and a static public key pk_{ID_τ} on behalf of a party ID_τ, if ID_τ is unique and pk_{ID_τ} is ensured to be sound by evaluating the non-interactive proof $\mathsf{pf}_{\mathsf{ID}_\tau}$. We only require that the proof is non-interactive in order to keep the model simple. Parties established by this query are called dishonest.

- $\mathsf{Test}(\pi_i^s)$: If the oracle has state $\Omega = \texttt{reject}$ or $K_i^s = \emptyset$, then the oracle π_i^s returns some failure symbol $\perp$. Otherwise it flips a fair coin b, samples a random element K_0 from key space $\mathcal{K}_{\mathsf{AKE}}$, and sets $K_1 = K_i^s$. Finally the key K_b is returned. This query is allowed to be asked at most once during the following security game.

We stress that the exact meaning of the $\mathsf{StateReveal}$ must be defined by each protocol separately, and each protocol should be proven secure to resist with such kind of state leakage as its claimed, i.e., the content stored in the variable st during protocol execution. In other word, each protocol should define the protocol steps processed on secure device. Of course one could distribute all protocol computations on the secure device then the security model would equal to a model without $\mathsf{StateReveal}$ query. However the security result of a protocol analysed under such implementation scenario must be weaker than that in a case allowing leakage of states. In contrast, our goal is

[5]For example, the variable Ψ_i^s might be set as identity ID_j at some point when the oracle receives a message containing the identity of its partner; the messages m and m^* will be appended to transcript rT_i^s and sT_i^s respectively. A protocol here might be either run in pre- or post-specified peer setting [9, 31]. As for a protocol running under post-specified peer setting, we always have that $\widetilde{\mathsf{ID}_j} = \emptyset$.

to define the maximum states that can be leaked. The $\mathsf{EstablishParty}$ query is used to model the chosen identity and public key attacks. In this query, the detail form of pf should be specified by each protocol. Please note that one could specify $\mathsf{pf} = \emptyset$ to model arbitrary public key registration without checking anything.

Secure AKE Protocols. To formalize the notion that two oracles are engaged in an on-line communication, we define the partnership via *matching sessions* which was first formulated by Krawczyk [25].

Definition 10 (Matching sessions)**.** We say that an oracle π_i^s has a *matching session* to oracle π_j^t, if π_i^s has sent all protocol messages and all the following conditions hold:

- $\Psi_i^s = \mathsf{ID}_j$ and $\Psi_j^t = \mathsf{ID}_i$,
- $sT_i^s = rT_j^t$ and $rT_i^s = sT_j^t$.

Correctness. We say an AKE protocol Σ is correct, if two oracles π_i^s and π_j^t accept with matching sessions, then both oracles hold the same session key, i.e. $K_i^s = K_j^t$.

For the security definition, we need the notion of *freshness* of an oracle.

Definition 11 (Freshness)**.** Let π_i^s be an accepted oracle with intended partner ID_j. Let π_j^t be an oracle (if it exists), such that π_i^s has a matching session to π_j^t. Then the oracle π_i^s is said to be *fresh* if none of the following conditions holds:

1. $\mathcal{A}$ queried $\mathsf{RegisterCorrupt}(\mathsf{ID}_j, pk_{\mathsf{ID}_j}, \mathsf{pf}_{\mathsf{ID}_j})$.
2. $\mathcal{A}$ queried either $\mathsf{RevealKey}(\pi_i^s)$ or $\mathsf{RevealKey}(\pi_j^t)$ (if π_j^t exists).
3. $\mathcal{A}$ queried both $\mathsf{Corrupt}(\mathsf{ID}_i)$ and $\mathsf{StateReveal}(\pi_i^s)$.
4. If π_j^t exists, $\mathcal{A}$ queried both $\mathsf{Corrupt}(\mathsf{ID}_j)$ and $\mathsf{StateReveal}(\pi_j^t)$.
5. If π_j^t does not exist, $\mathcal{A}$ queried $\mathsf{Corrupt}(\mathsf{ID}_j)$.

Security Experiment $\mathsf{EXP}_{\Sigma,\mathcal{A}}^{\mathsf{AKE}}(\kappa)$: On input security parameter 1^κ, the security experiment is proceeded as a game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ based on an AKE protocol Σ, where the following steps are performed:

1. At the beginning of the game, the challenger $\mathcal{C}$ implements the collection of oracles $\{\pi_i^s : i \in [\ell], s \in [d]\}$, and generates ℓ long-term key pairs $(pk_{\mathsf{ID}_i}, sk_{\mathsf{ID}_i})$ and corresponding proof $\mathsf{pf}_{\mathsf{ID}_i}$ (if any) for all honest parties ID_i for $i \in [\ell]$ where the identity $\mathsf{ID}_i \in \mathcal{IDS}$ of each party is chosen uniquely. $\mathcal{C}$ gives adversary $\mathcal{A}$ all identities, public keys and corresponding proofs $\{(\mathsf{ID}_1, pk_{\mathsf{ID}_1}, \mathsf{pf}_{\mathsf{ID}_1}), \ldots, (\mathsf{ID}_\ell, pk_\ell, \mathsf{pf}_\ell)\}$ as input.
2. $\mathcal{A}$ may issue polynomial number of queries as aforementioned, namely $\mathcal{A}$ makes queries: Send, $\mathsf{StateReveal}$, $\mathsf{Corrupt}$, $\mathsf{EstablishParty}$ and $\mathsf{RevealKey}$.
3. At some point, $\mathcal{A}$ may issue a $\mathsf{Test}(\pi_i^s)$ query on an oracle π_i^s during the game with only once.
4. At the end of the game, the $\mathcal{A}$ may terminate with returning a bit b' as its guess for b of Test query.
5. Finally, 1 is returned if all following conditions hold:

- $\mathcal{A}$ has issued a Test query on an oracle π_i^s without failure,
- π_i^s is fresh throughout the security game, and
- $\mathcal{A}$ returned a bit b' which equals to b of Test-query;

Otherwise 0 is returned.

Definition 12 (Session Key Security)**.** A correct AKE protocol Σ is called (t, ϵ)-session-key-secure if probability bound $|\Pr[\mathsf{EXP}^{\mathsf{AKE}}_{\Sigma,\mathcal{A}}(\kappa) = 1] - 1/2| \leq \epsilon$ holds for all adversaries $\mathcal{A}$ running within time t in the above security experiment and for some negligible probability $\epsilon = \epsilon(\kappa)$ in the security parameter κ.

4 A Generic One-round AKE Construction from KE, KEM and NIKE

In this section, we present a generic one-round authenticated key exchange protocol from KE, KEM and NIKE (denoted by GC-KKN), that is more suitable to be implemented for providing eCK security than previous works. In our generic construction, the following building blocks are required in the sense of definitions in Section 2:

- Passively secure one-round key exchange scheme $\mathsf{KE} = (\mathsf{KE.Setup}, \mathsf{KE.EKGen}, \mathsf{KE.SKGen})$.
- IND-CCA secure key encapsulation mechanism scheme $\mathsf{KEM} = (\mathsf{KEM.Setup}, \mathsf{KEM.Gen}, \mathsf{KEM.EnCap}, \mathsf{KEM.DeCap})$.
- CKS-light secure non-interactive key exchange scheme $\mathsf{NIKE} = (\mathsf{NIKE.Setup}, \mathsf{NIKE.KGen}, \mathsf{NIKE.ShareKey})$.
- Strong randomness extractor $\mathsf{SEXT}(\cdot, \cdot) : \mathcal{S}_{\mathsf{SEXT}} \times \mathcal{M}_{\mathsf{SEXT}} \to \mathcal{R}_{\mathsf{SEXT}}$. We assume that the space $\mathcal{M}_{\mathsf{SEXT}}$ matches the key spaces of KE, KEM and NIKE schemes.
- Pseudo-random function family $\mathsf{PRF}(\cdot, \cdot) : \mathcal{R}_{\mathsf{SEXT}} \times \mathcal{M}_{\mathsf{PRF}} \to \mathcal{K}_{\mathsf{AKE}}$.

Design Principle. The first attack we need to cope with is the chosen ephemeral key attacks against an uncorrupted party which might be only one that is not corrupted in the AKE security experiment, i.e. the adversary breaks the eCK security of considered protocol under freshness case $C6$ (shown in Appendix 4.2). The CEK attack informally addresses the situation that the adversary tries to obtain non-trivial information about test oracle (whose states are not leaked) by injecting ephemeral keys of her own choice to manipulate the session keys of other oracles (from which he can reveal key via $\mathsf{RevealKey}$ query). In order to handle CEK attacks, we utilize the IND-CCA secure KEM scheme as both FSXY [16] and BCNP [6] schemes. Since the IND-CCA secure KEM is known as one of the most efficient and effective solutions which can withstand this kind of attack. The IND-CCA security is necessary to simulate $\mathsf{RevealKey}$ queries to oracles of *target* uncorrupted party (say the intended communication partner of test oracle) without knowing corresponding long-term private key in the proof simulation. Meanwhile, we might ask $\mathcal{DEC}$ oracle provided by KEM experiment for help.

On the second, we have to consider security of test oracle when its session states are leaked. The worst case (which might be very likely to happen) is that test oracle received an ephemeral key which is chosen by adversary, or the session states are disclosed from both test oracle and its partner session. In either case, the adversary may know all ephemeral secrets used to compute the session key of test oracle without knowing the long-term keys of session participants of test oracle. This is also why the FSXY construction needs expensive NAXOS like trick to generate certain intermediate secret that is leakage resilience. To deal with this situation without NAXOS trick, our solution is to use non-interactive key exchange scheme, namely we make use of the long-term shared key of session participants to compute part of session key. This makes sense due to the fact that the long-term keys of session participants are not corrupted. We need the CKS-light security for NIKE scheme since we have to 'appropriately' simulate the session keys of uncorrupted oracles in presence of adversary who can register identities and public keys of her own choice (i.e. via $\mathsf{RegisterCorrupt}$ query).

While considering the wPFS attack in which long-term keys of both participants might be leaked. Intuitively, we cannot make use of KEM or NIKE schemes to achieve wPFS security property, since the security of both schemes relies on uncompromised long-term keys. Unlike the FSXY [16] scheme, we do not adopt to IND-CPA secure KEM (which is called as wKEM in FSXY) in the construction since wKEM cannot be executed simultaneously by two sessions. Namely the responder cannot generate the outgoing ciphertext until it received the ephemeral public key sent by initiator. Our solution avoids this circumstance via exploiting passively secure one-round key exchange protocol KE without long-term keys. This is a generalization from the BCNP construction [6] wherein only Diffie-Hellman key exchange protocol [13] is considered. Our construction is able to be instantiated with any other passively secure one-round key exchange protocols.

4.1 Protocol Description

Set-up: To initiate the system, the public system parameters $pms := (pms^{ke}, pms^{kem}, pms^{nike}, k_{\mathsf{SEXT}})$ are firstly generated via performing $pms^{ke} \leftarrow \mathsf{KE.Setup}(1^\kappa)$, $pms^{kem} \leftarrow \mathsf{KEM.Setup}(1^\kappa)$, $pms^{nike} \leftarrow \mathsf{NIKE.Setup}(1^\kappa)$ and $k_{\mathsf{SEXT}} \overset{\$}{\leftarrow} \mathcal{S}_{\mathsf{SEXT}}$.

Long-term key Generation and Registration: A party $\hat{A}$ may run algorithms $(pk_{\hat{A}}^{kem}, sk_{\hat{A}}^{kem}) \overset{\$}{\leftarrow} \mathsf{KEM.Gen}(pms^{kem})$ and $(pk_{\hat{A}}^{nike}, sk_{\hat{A}}^{nike}, \mathsf{pf}_{\hat{A}}) \overset{\$}{\leftarrow} \mathsf{NIKE.KGen}(pms^{nike}, \hat{A})$ to generate the long-term key pair for KEM and NIKE schemes respectively. The public key $pk_{\hat{A}}^{kem}$ can be registered arbitrarily. As well the public key $pk_{\hat{A}}^{nike}$ can be registered arbitrarily if $\mathsf{pf} = \emptyset$. Otherwise the $pk_{\hat{A}}^{nike}$ might be registered if the non-interactive proof $\mathsf{pf}_{\hat{A}}$ is evaluated to be sound based on $pk_{\hat{A}}^{nike}$. As for certain NIKE scheme, the interactive zero-knowledge proof scheme might be required to ensure the registered public key is consistent (e.g. the factoring based NIKE in [15]).

Protocol Execution: On input parameters $pms := (pms^{ke}, pms^{kem}, pms^{nike}, k_{\mathsf{SEXT}})$, the protocol between party $\hat{A}$ and party $\hat{B}$ is proceeded as following which is also informally depicted in Figure 1:

1. Upon activation a session at $\hat{A}$, it performs the steps:
   (a) Choose ephemeral public/private keys $(epk_{\hat{A}}, esk_{\hat{A}}) \overset{\$}{\leftarrow} \mathsf{KE.EKGen}(pms^{ke})$;
   (b) Run $(K_{\hat{A}}, C_{\hat{A}}) \overset{\$}{\leftarrow} \mathsf{KEM.EnCap}(pk_{\hat{B}}^{kem})$ and compute $K'_{\hat{A}} := \mathsf{SEXT}(K_{\hat{A}})$;
   (c) Send $(\hat{A}, epk_{\hat{A}}, C_{\hat{A}})$ to $\hat{B}$.
2. Upon activation a session at $\hat{B}$, it performs the steps:
   (a) Choose ephemeral public/private keys $(epk_{\hat{B}}, esk_{\hat{B}}) \overset{\$}{\leftarrow} \mathsf{KE.EKGen}(pms^{ke})$;

$$\begin{array}{ccc}
\hat{A} & & \hat{B} \\
sk_{\hat{A}} := (sk_{\hat{A}}^{kem}, sk_{\hat{A}}^{nike}) & & sk_{\hat{B}} := (sk_{\hat{B}}^{kem}, sk_{\hat{B}}^{nike}) \\
pk_{\hat{A}} := (pk_{\hat{A}}^{kem}, pk_{\hat{A}}^{nike}) & & pk_{\hat{B}} := (pk_{\hat{B}}^{kem}, pk_{\hat{B}}^{nike}) \\
 & & \\
(epk_{\hat{A}}, esk_{\hat{A}}) \overset{\$}{\leftarrow} \mathsf{KE.EKGen}(pms^{ke}) & & (epk_{\hat{B}}, esk_{\hat{B}}) \overset{\$}{\leftarrow} \mathsf{KE.EKGen}(pms^{ke}) \\
(K_{\hat{A}}, C_{\hat{A}}) \overset{\$}{\leftarrow} \mathsf{KEM.EnCap}(pk_{\hat{B}}^{kem}) & & (K_{\hat{B}}, C_{\hat{B}}) \overset{\$}{\leftarrow} \mathsf{KEM.EnCap}(pk_{\hat{A}}^{kem}) \\
 & \xrightarrow{\hat{A}, epk_{\hat{A}}, C_{\hat{A}}} & \\
 & \xleftarrow{\hat{B}, epk_{\hat{B}}, C_{\hat{B}}} & \\
\mathsf{sid} := \hat{A}||\hat{B}||epk_{\hat{A}}||C_{\hat{A}}||epk_{\hat{B}}||C_{\hat{B}} & & \mathsf{sid} := \hat{A}||\hat{B}||epk_{\hat{A}}||C_{\hat{A}}||epk_{\hat{B}}||C_{\hat{B}} \\
eK := \mathsf{KE.SKGen}(esk_{\hat{A}}, epk_{\hat{B}}) & & eK := \mathsf{KE.SKGen}(esk_{\hat{B}}, epk_{\hat{A}}) \\
K_{\hat{B}} := \mathsf{KEM.DeCap}(sk_{\hat{A}}^{kem}, C_{\hat{B}}) & & K_{\hat{A}} := \mathsf{KEM.DeCap}(sk_{\hat{B}}^{kem}, C_{\hat{A}}) \\
ShK_{\hat{A},\hat{B}} := & & ShK_{\hat{A},\hat{B}} := \\
\mathsf{NIKE.ShareKey}(\hat{A}, sk_{\hat{A}}^{nike}, \hat{B}, pk_{\hat{B}}^{nike}) & & \mathsf{NIKE.ShareKey}(\hat{B}, sk_{\hat{B}}^{nike}, \hat{A}, pk_{\hat{A}}^{nike}) \\
eK' := \mathsf{SEXT}(eK) & & eK' := \mathsf{SEXT}(eK) \\
K'_{\hat{A}} := \mathsf{SEXT}(K_{\hat{A}}) & & K'_{\hat{B}} := \mathsf{SEXT}(K_{\hat{B}}) \\
K'_{\hat{B}} := \mathsf{SEXT}(K_{\hat{B}}) & & K'_{\hat{A}} := \mathsf{SEXT}(K_{\hat{A}}) \\
ShK'_{\hat{A},\hat{B}} := \mathsf{SEXT}(ShK_{\hat{A},\hat{B}}) & & ShK'_{\hat{A},\hat{B}} := \mathsf{SEXT}(ShK_{\hat{A},\hat{B}}) \\
eK'' := \mathsf{PRF}(eK', \mathsf{sid}) & & eK'' := \mathsf{PRF}(eK', \mathsf{sid}) \\
K''_{\hat{A}} := \mathsf{PRF}(K'_{\hat{A}}, \mathsf{sid}) & & K''_{\hat{B}} := \mathsf{PRF}(K'_{\hat{B}}, \mathsf{sid}) \\
K''_{\hat{B}} := \mathsf{PRF}(K'_{\hat{B}}, \mathsf{sid}) & & K''_{\hat{A}} := \mathsf{PRF}(K'_{\hat{A}}, \mathsf{sid}) \\
ShK''_{\hat{A},\hat{B}} := \mathsf{PRF}(ShK'_{\hat{A},\hat{B}}, \mathsf{sid}) & & ShK''_{\hat{A},\hat{B}} := \mathsf{PRF}(ShK'_{\hat{A},\hat{B}}, \mathsf{sid}) \\
k_e := eK'' \oplus K''_{\hat{A}} \oplus K''_{\hat{B}} \oplus ShK''_{\hat{A},\hat{B}} & & k_e := eK'' \oplus K''_{\hat{A}} \oplus K''_{\hat{B}} \oplus ShK''_{\hat{A},\hat{B}}
\end{array}$$

Figure 1: Generic One-round AKE Protocol from KE, KEM and NIKE

   (b) Run $(K_{\hat{B}}, C_{\hat{B}}) \overset{\$}{\leftarrow} \mathsf{KEM.EnCap}(pk_{\hat{A}}^{kem})$ and compute $K'_{\hat{B}} := \mathsf{SEXT}(K_{\hat{B}})$;
   (c) Send $(\hat{B}, epk_{\hat{B}}, C_{\hat{B}})$ to $\hat{A}$.

3. Upon receiving $(\hat{A}, epk_{\hat{A}}, C_{\hat{A}})$, party $\hat{B}$ does the following:
   (a) Set session identifier $\mathsf{sid} := \hat{A}||\hat{B}||epk_{\hat{A}}||C_{\hat{A}}||epk_{\hat{B}}||C_{\hat{B}}$ where the messages are ordered by round, and within each round lexicographically by the identities of the purported senders;
   (b) Compute $eK := \mathsf{KE.SKGen}(esk_{\hat{B}}, epk_{\hat{A}})$ and $eK' := \mathsf{SEXT}(eK)$ and $eK'' := \mathsf{PRF}(eK', \mathsf{sid})$;
   (c) Compute $K''_{\hat{B}} := \mathsf{PRF}(K'_{\hat{B}}, \mathsf{sid})$;
   (d) Compute $K_{\hat{A}} := \mathsf{KEM.DeCap}(sk_{\hat{B}}^{kem}, C_{\hat{A}})$ and $K'_{\hat{A}} := \mathsf{SEXT}(K'_{\hat{A}})$, $K''_{\hat{A}} := \mathsf{PRF}(K'_{\hat{A}}, \mathsf{sid})$;
   (e) Compute $ShK_{\hat{A},\hat{B}} := \mathsf{NIKE.ShareKey}(\hat{A}, sk_{\hat{A}}^{nike}, \hat{B}, pk_{\hat{B}}^{nike})$, $ShK'_{\hat{A},\hat{B}} := \mathsf{SEXT}(ShK_{\hat{A},\hat{B}})$ and $ShK''_{\hat{A},\hat{B}} := \mathsf{PRF}(ShK'_{\hat{A},\hat{B}}, \mathsf{sid})$;
   (f) Compute the final session key as $k_e := eK'' \oplus K''_{\hat{A}} \oplus K''_{\hat{B}} \oplus ShK''_{\hat{A},\hat{B}}$.

4. Upon receiving $(\hat{B}, epk_{\hat{B}}, C_{\hat{B}})$, $\hat{A}$ does the following:
   (a) Set session identifier as $\mathsf{sid} := \hat{A}||\hat{B}||epk_{\hat{A}}||C_{\hat{A}}||epk_{\hat{B}}||C_{\hat{B}}$;
   (b) Compute $eK := \mathsf{KE.SKGen}(esk_{\hat{A}}, epk_{\hat{B}})$ and $eK' := \mathsf{SEXT}(eK)$, $eK'' := \mathsf{PRF}(eK', \mathsf{sid})$;
   (c) Compute $K''_{\hat{A}} := \mathsf{PRF}(K'_{\hat{A}}, \mathsf{sid})$;

(d) Compute $K_{\hat{B}} := \mathsf{KEM.DeCap}(sk^{kem}_{\hat{A}}, C_{\hat{B}})$ and $K'_{\hat{B}} := \mathsf{SEXT}(K_{\hat{B}})$, $K''_{\hat{B}} := \mathsf{PRF}(K'_{\hat{B}}, \mathsf{sid})$;

(e) Compute $ShK_{\hat{A},\hat{B}} := \mathsf{NIKE.ShareKey}(\hat{B}, sk^{nike}_{\hat{B}}, \hat{A}, pk^{nike}_{\hat{A}})$, $ShK'_{\hat{A},\hat{B}} := \mathsf{SEXT}(ShK_{\hat{A},\hat{B}})$ and $ShK''_{\hat{A},\hat{B}} := \mathsf{PRF}(ShK'_{\hat{A},\hat{B}}, \mathsf{sid})$;

(f) Compute the final session key as $k_e := eK'' \oplus K''_{\hat{A}} \oplus K''_{\hat{B}} \oplus ShK''_{\hat{A},\hat{B}}$.

Session States and Implementaton Senario. We now define the session states in terms of implementation model with secure device. Basically, all states of KE.EKGen, KE.SKGen and KEM.EnCap algorithms would be stored in the state variable st. For instance the state of an oracle $\pi^s_{\hat{A}}$ might include values $(esk_{\hat{A}}, eK_{\hat{A}}, eK'_{\hat{A}}, eK''_{\hat{A}}, K_{\hat{A}}, K'_{\hat{A}}, K''_{\hat{A}})$ appeared in the above protocol description and other randomness or intermediate values generated within algorithms: KE.EKGen , KE.SKGen and KEM.EnCap. Namely those algorithms can be executed on host machine. However, we assume no secret states related to KEM.DeCap and NIKE.ShareKey algorithms can be revealed. This can be realized by doing all computations involving long-term private key of KEM.DeCap and NIKE.ShareKey algorithms, and final session key generation on secure device, i.e. processing the steps 3.(d,e,f) and 4.(d,e,f) on secure device. In particular KEM.DeCap and NIKE.ShareKey algorithms might involve expensive consistency check operations, say pairing operations in pairing-based NIKE [15]. But we stress that the computations of KEM.DeCap and NIKE.ShareKey algorithms without using long-term private key and without using values generated by long-term private key, can be done on host machine for efficiency consideration.

4.2 Security Analysis

We assume without loss of generality that the maximum probability for the event that two oracles output the same ciphertext C or ephemeral public key epk, is a negligible fraction $1/2^{\lambda}$ where $\lambda \in N$ is a large enough integer in terms of the security parameter κ. Let $\mathsf{MAX}(X_1, X_2, X_3)$ denote the function to obtain the maximum values from variables X_1, X_2 and X_3.

Theorem 1. *Suppose that the* SEXT *is* $(\kappa, \epsilon_{\mathsf{SEXT}})$*-strong randomness extractor, the* KEM *is* $(q_{kem}, t, \epsilon_{\mathsf{KEM}})$*-IND-CCA secure and* KE *is* $(t, \epsilon_{\mathsf{KE}})$*-passively secure, and the pseudo-random function* PRF *is* $(q_{prf}, t, \epsilon_{\mathsf{PRF}})$*-secure, and the* NIKE *is* $(t, \epsilon_{\mathsf{NIKE}})$*-secure non-interactive scheme with respect to the definitions in Section 2. And we assume that either* KE *key or* KEM *key or* NIKE *key has* κ*-min-entropy. Then the proposed protocol is* (t', ϵ)*-session-key-secure in the sense of Definition 12 with* $t' \leq t$, $q_{kem} \geq d$ *and* $q_{prf} \geq d+1$, *and* $\epsilon \leq \frac{(d\ell)^2}{2^{\lambda}} + 3(d\ell)^2 \cdot (\mathsf{MAX}(\epsilon_{\mathsf{KE}}, \epsilon_{\mathsf{KEM}}, \epsilon_{\mathsf{NIKE}}) + \epsilon_{\mathsf{SEXT}} + \epsilon_{\mathsf{PRF}})$.

The proof of this theorem is presented in the Appendix A.

5 An Efficient One-round AKE Protocol under Standard Assumptions

In this section we present an eCK secure AKE protocol P1 in the standard model based on Bilinear Decisional Diffie-Hellman assumption. The proposed protocol relies on bilinear pairings, target collision resistant hash function family, and pseudo-random function family. This protocol can be implemented more efficiently involving secure device, that only one regular exponentiation is required on secure device and other expensive operations can be done on host machine.

Design Principle. The construction of P1 can be seen as a concrete instantiation of GC-KKN scheme. We first observe that it is possible to merge those computations in KE, KEM and NIKE schemes if they work under the same algebraic groups. However the standard CKS-light secure NIKE scheme is rare so far. Our AKE construction is based on the pairing-based NIKE scheme [15] with slight modifications rather than the factoring-based NIKE scheme [15] because the latter requires an *interactive key registration protocol* to ensure the consistency of public key. Unlike the pairing-based NIKE scheme [15], we retort to target collision resistant hash function family instead of collision resistant chameleon hash function family to relax the assumption. This is possible because we could alternatively bind the identities of session participants with each session key using pseudo-random functions. In order to battle against CEK attacks, each party (including the attacker) is required to construct some kind of 'tag' based on specific weak Programmable Hash Function [18] to encode consistency information on its chosen (either long-term or ephemeral) public keys. Those tags are particularly customized to be independent of any information about receivers, which enable our protocol to be able to run in the post-specified peer setting. However we have to make use of the pairing to provide a means of consistency checking that (both long-term and ephemeral) public keys coming from the adversary are in some sense of well-formed. Fortunately, those expensive consistency checks can be done on host machine.

5.1 Protocol Description

The proposed protocol takes as input the following building blocks which are initialized respectively in terms of the security parameter κ:

- Symmetric bilinear groups $\mathcal{PG} = (\mathbb{G}, g, \mathbb{G}_T, p, e) \xleftarrow{\$} \mathsf{PG.Gen}(1^\kappa)$ and along with random values $(u_1, u_2, u_3, u_4) \xleftarrow{\$} \mathbb{G}$,
- a target collision resistant hash function $\mathsf{TCRHF}(hk_{\mathsf{TCRHF}}, \cdot) : \mathcal{K}_{\mathsf{TCRHF}} \times \mathbb{G} \to \mathbb{Z}_p$, where $hk_{\mathsf{TCRHF}} \xleftarrow{\$} \mathsf{TCRHF.KG}(1^\kappa)$, and
- a pseudo-random function family $\mathsf{PRF}(\cdot, \cdot) : \mathbb{G}_T \times \mathcal{D}_{\mathsf{PRF}} \to \mathcal{K}_{\mathsf{AKE}}$.

The variable *pms* stores the public system parameters $pms := (\mathcal{PG}, \{u_i\}_{1 \le i \le 4}, hk_{\mathsf{TCRHF}})$.

Long-term Key Generation and Registration: On input $pms := (\mathcal{PG}, \{u_i\}_{2 \le i \le 4}, hk_{\mathsf{TCRHF}})$, a party $\hat{A}$ may run an efficient algorithm $(sk_{\hat{A}}, pk_{\hat{A}}, \emptyset) \xleftarrow{\$} \mathsf{KGen}(pms, \hat{A})$ to generate the long-term key pair as: $sk_{\hat{A}} = a \xleftarrow{\$} \mathbb{Z}_p^*, pk_{\hat{A}} = (A, t_A)$ where $A = g^a$, $t_A := (u_4^{h_A^2} u_3^{h_A} u_2)^a$ and $h_A = \mathsf{TCRHF}(A)$. Please note that we allow arbitrary key registration, i.e. the adversary is able to query $\mathsf{RegisterCorrupt}(\hat{A}, pk_{\hat{A}}, \emptyset)$ with $\mathsf{pf}_{\hat{A}} = \emptyset$.

Protocol Execution : On input $pms := (\mathcal{PG}, \{u_i\}_{1 \le i \le 4}, hk_{\mathsf{TCRHF}})$, the protocol between parties $\hat{A}$ and $\hat{B}$ proceeds as following which is also depicted in the Figure 2.

1. Upon activation a new session, the party $\hat{A}$ performs the steps:
   (a) Choose an ephemeral private key $x \xleftarrow{\$} \mathbb{Z}_p^*$;
   (b) Compute ephemeral public key $X := g^x$;
   (c) Compute $h_X := \mathsf{TCRHF}(X)$, and $t_X := (u_4^{h_X^2} u_3^{h_X} u_2)^x$;
   (d) Send $(\hat{A}, A, t_A, X, t_X)$ to $\hat{B}$.
2. Upon activation a new session, the party $\hat{B}$ performs the steps:
   (a) Choose an ephemeral private key $y \xleftarrow{\$} \mathbb{Z}_p^*$;

$$\begin{array}{ccc}
\hat{A} & & \hat{B} \\
(sk_{\hat{A}} = a \xleftarrow{\$} \mathbb{Z}_p^*, & & (sk_{\hat{B}} = b \xleftarrow{\$} \mathbb{Z}_p^*, \\
pk_{\hat{A}} = (A, t_A) := (g^a, (u_4^{h_A^2} u_3^{h_A} u_2)^a)) & & pk_{\hat{B}} = (B, t_B) := (g^b, (u_4^{h_B^2} u_3^{h_B} u_2)^b)) \\
 & & \\
x \xleftarrow{\$} \mathbb{Z}_p^*, X := g^x & & y \xleftarrow{\$} \mathbb{Z}_p^*, Y := g^y \\
h_X := \mathsf{TCRHF}(X) & & h_Y := \mathsf{TCRHF}(Y) \\
t_X := (u_4^{h_X^2} u_3^{h_X} u_2)^x & & t_Y := (u_4^{h_Y^2} u_3^{h_Y} u_2)^y \\
 & \xrightarrow{\hat{A}, A, t_A X, t_X} & \\
 & \xleftarrow{\hat{B}, B, t_B, Y, t_Y} & \\
h_Y := \mathsf{TCRHF}(Y) & & h_X := \mathsf{TCRHF}(X) \\
h_B := \mathsf{TCRHF}(B) & & h_A := \mathsf{TCRHF}(A) \\
\text{reject if either} & & \text{reject if either} \\
e(t_Y, g) \neq e(u_4^{h_Y^2} u_3^{h_Y} u_2, Y) \text{ or} & & e(t_X, g) \neq e(u_4^{h_X^2} u_3^{h_X} u_2, X) \text{ or} \\
e(t_B, g) \neq e(u_4^{h_B^2} u_3^{h_B} u_2, B) & & e(t_A, g) \neq e(u_4^{h_A^2} u_3^{h_A} u_2, A) \\
\mathsf{sid} := \hat{A}||A||t_A||X||t_X||\hat{B}||B||t_B||Y||t_Y & & \mathsf{sid} := \hat{A}||A||t_A||X||t_X||\hat{B}||B||t_B||Y||t_Y \\
\text{reject if some values recorded in } \mathsf{sid} \text{ are} & & \text{reject if some values recorded in } \mathsf{sid} \text{ are} \\
\text{identical} & & \text{identical} \\
\beta_{\hat{A}} := e(u_1, BY), k := \beta_{\hat{A}}^{a+x} & & \beta_{\hat{B}} := e(u_1, AX), k := \beta_{\hat{B}}^{b+y} \\
\text{accept } k_e := \mathsf{PRF}(k, \mathsf{sid}) & & \text{accept } k_e := \mathsf{PRF}(k, \mathsf{sid})
\end{array}$$

Figure 2: Pairing-based AKE Protocol under Standard Assumptions

(b) Compute the ephemeral public key $Y := g^y$;

(c) Compute $h_Y := \mathsf{TCRHF}(Y)$, and $t_Y := (u_4^{h_Y^2} u_3^{h_Y} u_2)^y$;

(d) Send $(\hat{B}, B, t_B, Y, t_Y)$ to $\hat{A}$.

3. Upon receiving $(\hat{B}, B, t_B, Y, t_Y)$, the party $\hat{A}$ does the following:

   (a) Compute $h_Y := \mathsf{TCRHF}(Y)$ and $h_B := \mathsf{TCRHF}(B)$, and reject if either $e(t_Y, g) \neq e(u_4^{h_Y^2} u_3^{h_Y} u_2, Y)$ or $e(t_B, g) \neq e(u_4^{h_B^2} u_3^{h_B} u_2, B)$;

   (b) Set session identifier $\mathsf{sid} := \hat{A}||A||t_A||X||t_X||Y||t_Y||\hat{B}||t_B||B||Y||t_Y$, and reject the session if some values recorded in sid are identical.

   (c) Compute $\beta_{\hat{A}} := e(u_1, BY)$; (d) Compute $k := \beta_{\hat{A}}^{a+x}$ and session key as $k_e := \mathsf{PRF}(k, sid)$.

4. Upon receiving $(\hat{A}, A, t_A, X, t_X)$, the party $\hat{B}$ does the following:

   (a) compute $h_X := \mathsf{TCRHF}(X)$ and $h_A := \mathsf{TCRHF}(A)$, and reject if either $e(t_X, g) \neq e(u_4^{h_X^2} u_3^{h_X} u_2, X)$ or $e(t_A, g) \neq e(u_4^{h_A^2} u_3^{h_A} u_2, A)$;

   (b) set session identifier
   $\mathsf{sid} := \hat{A}||A||t_A||X||t_X||Y||t_Y||\hat{B}||t_B||B||Y||t_Y$, and reject the session if some values recorded in sid are identical.

   (c) compute $\beta_{\hat{B}} := e(u_1, AX)$;

   (d) compute $k := \beta_{\hat{B}}^{b+y}$, session key as $k_e := \mathsf{PRF}(k, \mathsf{sid})$.

Implementation and Session States: We assume that only the ephemeral private key x (resp. y) would be stored in the state variable st, and these states are the maximum states which can be compromised by adversary. This can be guaranteed by performing the computations of key material k and session key k_e (i.e. (steps 3.(d) and step 4.(d)) on secure device.

Remark 1. Please note that the computation cost at secure device of the above scheme is dominated by only one exponentiation and one PRF. For instance, the implementer could only compute the key material $k := \beta_{\hat{A}}^{a+x}$ and finally session key on secure device, where $\beta_{\hat{A}} := e(u_1, BY)$ is computed on host machine. We stress that all other expensive operations (such as pairing) can be done on host machine which is normally more powerful than secure device. Of course one could execute the whole protocol on secure device, but then it will be much less efficient. Instead we only assume that the most necessary computations are done on secure device without pairing operations. We treat those secure device as a black-box which might take as input the ephemeral key and prepared key materials (e.g. the x and $e(u_1, BY)$), and it outputs the generated session key using key derivation function PRF. Moreover, each session participant say $\hat{A}$ might send its certificate $cert_{\hat{A}}$ instead of the tuple $(\hat{A}, A, t_A)$ in the message flow. Those parameters used to compute those tags are particularly customized to be independent of any information about session participants. This enables the protocol to be able to perfectly run in the post-specified peer setting.

5.2 Efficiency Consideration

In this section, we consider the issue on improving the efficiency of proposed protocol P1. It is not hard to see that the consistency checks on both long-term and ephemeral keys consume much computational cost which requires four pairing operations. Thus how to reduce the cost on those consistency checks is our major concern.

We first introduce an alternative consistency checking algorithm which is derived from the similar technique in [22] used to improve the efficiency of identity-based KEM scheme. The idea is to merge consistency checks on incoming Diffie-Hellman keys. In the new consistency check algorithm, a party $\hat{A}$ on receiving
$(\hat{B}, B, t_B, Y, t_Y)$ may perform the following steps:

1. Choose two random values $\theta_1, \theta_2 \xleftarrow{\$} \mathbb{Z}_p^*$.
2. Reject the session if $e(t_B^{\theta_1} t_Y^{\theta_2}, g) \neq e((u_4^{h_B^2} u_3^{h_B} u_2)^{\theta_1}, B) e((u_4^{h_Y^2} u_3^{h_Y} u_2)^{\theta_2}, Y)$.

We claim that the combined consistency check equation implies that all received tags are consistent. To prove our argument we define functions $\Delta_1(t_Y) := \frac{e(u_2 u_3^{h_Y} u_4^{h_Y^2}, Y)}{e(t_Y, g)}$ and $\Delta_2(t_B) := \frac{e(u_2 u_3^{h_B} u_4^{h_B^2}, B)}{e(t_B, g)}$. Obviously, if t_Y, t_B, t_Z, t_C are consistent we would have the fact $\Delta_1(t_Y) = \Delta_2(t_B) = 1$. Hence for random values $\theta_1, \theta_2 \xleftarrow{\$} \mathbb{Z}_p^*$, function $\Delta_1(t_Y))^{\theta_1}(\Delta_2(t_B))^{\theta_2}$ evaluates to 1 if t_Y, t_B are consistent and to a random group value in $\mathbb{G}_T$ otherwise. This alternative consistency check algorithm roughly substitutes one multiple-exponentiation for one pairing operation. But the random values θ_1 and θ_2 would be included into the returns of $\mathsf{StateReveal}$ query since the consistency check algorithm is executed on host machine.

In addition, please notice that a party $\hat{A}$ has to check the consistency of long-term key in every sessions that might be costly and unnecessary. An alternative solution could let the Certificate Authority to do the consistency check on long-term public key during key registration protocol. In this way, it could reduce two pairing operations during key exchange execution and also the number of public key. To register a public key $pk_{\hat{A}} = A$, each party $\hat{A}$ should at least prove the consistency of long-term public key via tag t_A. Then the long-term public key A is registered if $e(t_A, g) = e(A, u_4^{h_A^2} u_5^{h_A} u_2)$. This check would be done only once at CA. In particular, the tag t_A is required while querying the $\mathsf{RegisterCorrupt}(\hat{A}, pk_{\hat{A}}, \mathsf{pf}_{\hat{A}})$ in the security game, i.e. $\mathsf{pf}_{\hat{A}} = t_A$.

5.3 Security Analysis

We show the security of proposed protocol in our strong security model.

Theorem 2. *Assume each ephemeral key chosen during key exchange has bit-size $\lambda \in \mathbb{N}$. Suppose that the* BDDH *problem is $(t, \epsilon_{\mathsf{BDDH}})$-hard in the symmetric bilinear groups $\mathcal{PG}$, the* TCRHF *is $(t, \epsilon_{\mathsf{TCRHF}})$-secure target collision resistant hash function family, and the* PRF *is $(q_{prf}, t, \epsilon_{\mathsf{PRF}})$-secure pseudo-random function family. Then the proposed protocol is (t', ϵ)-session-key-secure in the sense of Definition 12 with $t' \approx t$ and $q_{prf} \geq 2$ and $\epsilon \leq \frac{(d\ell)^2}{2^\lambda} + \epsilon_{\mathsf{TCRHF}} + 3(d\ell)^2 \cdot (\epsilon_{\mathsf{BDDH}} + \epsilon_{\mathsf{PRF}})$.*

The proof of this theorem is presented in the Appendix B.

6 Comparisons

We summarize the comparisons of some existing well-known eCK secure protocols without random oracles in the Table 2 from the following perspectives: (i) the knowledge of peer's public key (peer setting), which would affect e.g. the round efficiency as aforementioned; (ii) the special design trick used to guarantee corresponding security; (iii) the security assumptions; (iv) number of long-term (LL) and ephemeral (Eph) keys which may affect the storage requirement, computation cost and communication cost; (v) overall computation cost of considered protocol; (vi) the computation cost on secure device (SD) and (vii) the communication round between host machine and secure device (HS). In the table, 'Exp' denotes the exponentiation and 'ME' denotes multi-exponentiations, 'Pair' denotes pairing evaluation, '(T)CR' denotes (target) collision-resistant hash, 'DDH' denotes the decisional Diffie-Hellman assumption, 'FAC' denote factoring assumption, 'EXT' denotes the strong randomness extractor, and 'TPRF' denotes the twisted-PRF trick. Recall that P1 denote the protocol in the Section 5.

	Peer setting	Design trick	Security assumptions	LL (pk,sk)	Eph (pk,sk)	Overall cost	SD cost	HS round
Oka [33]	post	TPRF	DDH, CR πPRF	(2,4)	(3,2)	3.Exp, 1.ME 2.CR, 1.πPRF 4.PRF	2.Exp, 1.ME 1.πPRF, 4.PRF	2
MO [32]	post	-	DDH, CR πPRF	(4,5)	(3,2)	3.Exp, 1.ME 2.CR, 1.πPRF	1.ME, 1.πPRF	1
FSXY [16]	pre	TPRF	DDH, FAC EXT,TCR PRF	(3,1)	(3,2)	6.Exp, 2.ME 2.TCR, 3.EXT 5 PRF	3.Exp, 2.ME 2.EXT, 4.PRF	2
GC-KKN § 4	pre	-	DDH,FAC EXT,TCR PRF	(6,2)	(3,2)	7.Exp, 2.ME 2.TCR, 3.PRF	2.Exp, 1.ME 2.EXT, 2.PRF	1
P1 § 5	post	-	BDDH,TCR PRF	(2,1)	(2,1)	2.Exp, 4.ME 4.Pair, 2.TCR 1.PRF	1.Exp, 1.PRF	1

Table 2: Comparisons among one-round 2AKE protocols in the standard model.

We instantiated the FSXY protocol with the factoring-based KEM [19] (also suggested by the authors of FSXY) and with the DDH-based ElGamal KEM. For comparison, we also initiate our GC-KKN for instance with the same factoring-based KEM as FSXY, with the factoring-based

NIKE scheme [15] and with the traditional Diffie-Hellman key exchange (DHKE) [13] under DDH assumption. While considering the instantiation of FSXY protocol in post-specified peer setting, the KEM [33] by Okamoto might be a candidate but in which case the $\pi\mathsf{PRF}$ is required. Then the Okamoto protocol [33] can seen as an 'optimized' instantiation of FSXY scheme.

In addition we assume without loss of generality the FSXY, Okamoto [33] and MO [32] schemes are realized with secure device to protect critical session states. For simplicity and security, we just assume the twisted-PRF trick used by both Okamoto and FSXY schemes is implemented on secure device to avoid any leakage on its output. Similarly all computations related to twisted-PRF trick are assumed to be executed on secure device too, such as the whole IND-CCA secure KEM (both encapsulation/decapsulation algorithms) used by FSXY. Otherwise the FSXY protocol is not secure in the eCK model.

We further remark that both FXSY and Okamoto protocols need at least two rounds to exchange information between host machine and secure device. In the first round the host machine may send randomness to the secure device and get back the ephemeral key (or ciphertext of $\mathsf{KEM.EnCap}$ algorithm); and in the second round, the host machine may send intermediate values to secure device and obtain the final session key from secure device. Therefore, our GC-KKN scheme is more HS-round efficient than those schemes, since only one such round is required. Although optimized P1 requires four pairing operations, it enjoys the most succinct structure and the most efficient algorithm on secure device.

7 Conclusions

We showed a generic construction GC-KKN for eCK-secure one-round two party AKE protocols in the standard model without NAXOS trick, which can be instantiated with passively secure KE scheme, IND-CCA secure KEM schemes and CKS-light secure NIKE schemes. In contrast to previous works in the standard model, the major merit of GC-KKN is its outstanding efficiency on implementing with secure device. In other word, only a small part of algorithms of GC-KKN need to be run on secure device to achieve state leakage resistant. We also gave a concrete protocol P1 based on the construction idea of GC-KKN. Similarly P1 is motivated to further improve the efficiency on secure device wherein only one regular exponential operation is required. The P1 was proved eCK secure in the standard model under only standard assumptions including BDDH assumption and secure target collision resistant hash function family and pseudo-random function family. To our best of knowledge, P1 is the first such protocol without NAXOS trick that can run under post-specified peer setting without knowing any information of communication peer at session activation.

References

[1] C. Adams, S. Farrell, T. Kause, and T. Mononen. Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP). RFC 4210 (Proposed Standard), September 2005.

[2] Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 273–289. Springer, August 2004.

[3] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, August 1994.

[4] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, May / June 2006.

[5] Simon Blake-Wilson, Don Johnson, and Alfred Menezes. Key agreement protocols and their security analysis. In Michael Darnell, editor, *6th IMA International Conference on Cryptography and Coding*, volume 1355 of *Lecture Notes in Computer Science*, pages 30–45. Springer, December 1997.

[6] Colin Boyd, Yvonne Cliff, Juan Gonzalez Nieto, and Kenneth G. Paterson. Efficient one-round key exchange in the standard model. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *ACISP 08: 13th Australasian Conference on Information Security and Privacy*, volume 5107 of *Lecture Notes in Computer Science*, pages 69–83. Springer, July 2008.

[7] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 321–336. Springer, April / May 2002.

[8] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, May 2001.

[9] Ran Canetti and Hugo Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 143–161. Springer, August 2002. `http://eprint.iacr.org/2002/120/`.

[10] Olivier Chevassut, Pierre-Alain Fouque, Pierrick Gaudry, and David Pointcheval. Key derivation and randomness extraction. Cryptology ePrint Archive, Report 2005/061, 2005. `http://eprint.iacr.org/`.

[11] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[12] Cas J. F. Cremers. Session-state reveal is stronger than ephemeral key reveal: Attacking the NAXOS authenticated key exchange protocol. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 09: 7th International Conference on Applied Cryptography and Network Security*, volume 5536 of *Lecture Notes in Computer Science*, pages 20–33. Springer, June 2009.

[13] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[14] Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the CBC, cascade and HMAC modes. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 494–510. Springer, August 2004.

[15] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 254–271. Springer, 2013.

[16] Atsushi Fujioka, Koutarou Suzuki, Keita Xagawa, and Kazuki Yoneyama. Strongly secure authenticated key exchange from factoring, codes, and lattices. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Workshop on Theory and Practice in Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 467–484. Springer, May 2012.

[17] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. In *25th Annual Symposium on Foundations of Computer Science*, pages 464–479. IEEE Computer Society Press, October 1984.

[18] Dennis Hofheinz, Tibor Jager, and Eike Kiltz. Short signatures from weaker assumptions. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 647–666. Springer, December 2011.

[19] Dennis Hofheinz and Eike Kiltz. Practical chosen ciphertext secure encryption from factoring. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 313–332. Springer, April 2009.

[20] Tibor Jager, Florian Kohlar, Sven Schaege, and Joerg Schwenk. Generic compilers for authenticated key exchange (full version). Cryptology ePrint Archive, Report 2010/621, 2010. `http://eprint.iacr.org/`.

[21] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293. Springer, August 2012.

[22] Eike Kiltz and David Galindo. Direct chosen-ciphertext secure identity-based key encapsulation without random oracles. In Lynn Margaret Batten and Reihaneh Safavi-Naini, editors, *ACISP 06: 11th Australasian Conference on Information Security and Privacy*, volume 4058 of *Lecture Notes in Computer Science*, pages 336–347. Springer, July 2006.

[23] Minkyu Kim, Atsushi Fujioka, and Berkant Ustaoglu. Strongly secure authenticated key exchange without NAXOS' approach. In Tsuyoshi Takagi and Masahiro Mambo, editors, *IWSEC 09: 4th International Workshop on Security, Advances in Information and Computer Security*, volume 5824 of *Lecture Notes in Computer Science*, pages 174–191. Springer, October 2009.

[24] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, August 1999.

[25] Hugo Krawczyk. HMQV: A high-performance secure diffie-hellman protocol. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer, August 2005.

[26] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007: 1st International Conference on Provable Security*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16. Springer, November 2007.

[27] Laurie Law, Alfred Menezes, Minghua Qu, Jerome A. Solinas, and Scott A. Vanstone. An efficient protocol for authenticated key agreement. *Des. Codes Cryptography*, 28(2):119–134, 2003.

[28] Tsutomu MATSUMOTO, Youichi TAKASHIMA, and Hideki IMAI. On seeking smart public-key-distribution systems. *IEICE TRANSACTIONS*, E69-E No.2:pp.99–106, 1986/02/20.

[29] Alfred Menezes. Another look at hmqv. *J. Mathematical Cryptology*, 1(1):47–64, 2007.

[30] Alfred Menezes, Minghua Qu, and Scott A. Vanstone. Some new key agreement protocols providing mutual implicit authentication. *SecondWorkshop on Selected Areas in Cryptography (SAC 95)*, 1995.

[31] Alfred Menezes and Berkant Ustaoglu. Comparing the pre- and post-specified peer models for key agreement. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *ACISP 08: 13th Australasian Conference on Information Security and Privacy*, volume 5107 of *Lecture Notes in Computer Science*, pages 53–68. Springer, July 2008.

[32] Daisuke Moriyama and Tatsuaki Okamoto. An eck-secure authenticated key exchange protocol without random oracles. *TIIS*, 5(3):607–625, 2011.

[33] Tatsuaki Okamoto. Authenticated key exchange and key encapsulation in the standard model (invited talk). In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 474–484. Springer, December 2007.

[34] Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In Kwangjo Kim, editor, *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118. Springer, February 2001.

[35] Augustin P. Sarr, Philippe Elbaz-Vincent, and Jean-Claude Bajard. A new security model for authenticated key agreement. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10: 7th International Conference on Security in Communication Networks*, volume 6280 of *Lecture Notes in Computer Science*, pages 219–234. Springer, September 2010.

[36] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/`.

[37] Berkant Ustaoglu. Comparing sessionstatereveal and ephemeralkeyreveal for Diffie-Hellman protocols. In Josef Pieprzyk and Fangguo Zhang, editors, *ProvSec 2009: 3rd International Conference on Provable Security*, volume 5848 of *Lecture Notes in Computer Science*, pages 183–197. Springer, November 2009.

[38] Zheng Yang. A strongly secure authenticated key exchange protocol from bilinear groups without random oracles. Cryptology ePrint Archive, Report 2012/381, 2012. `http://eprint.iacr.org/`.

[39] Kazuki Yoneyama and Yunlei Zhao. Taxonomical security consideration of authenticated key exchange resilient to intermediate computation leakage. In Xavier Boyen and Xiaofeng Chen, editors, *ProvSec 2011: 5th International Conference on Provable Security*, volume 6980 of *Lecture Notes in Computer Science*, pages 348–365. Springer, October 2011.

A Proof of Theorem 1

It is straightforward to verify that two accepted oracles (of considered protocol) having matching sessions would generate the same session key. In the sequel, we wish to show that the adversary is unable to distinguish random value from the session key of any fresh oracle. Without loss of generality, we consider that the adversary chooses the test oracle $\pi^{s^*}_{\hat{A}}$ executed between owner $\hat{A}$ with its indented partner $\hat{B}$.

Next we introduce the notations which might be used in the proof. General speaking, we use the superscript '*' to highlight corresponding values processed in test oracle $\pi^{s^*}_{\hat{A}}$. Let party $\hat{D}$ denote the indented communication partner of oracle $\pi^{t}_{\hat{B}}$ where $\hat{D}$ could be any parties but $\hat{B}$, and let corresponding ciphertext received by oracle $\pi^{t}_{\hat{B}}$ be $C_{\hat{D}}$.

If the adversary breaks the indistinguishability security property of proposed protocol, then at least one of the following fresh related cases (related to $\mathsf{StateReveal}$ and $\mathsf{Corrupt}$ queries) must occur in terms of the Definition 11:

Event 1: There is a oracle $\pi^{t^*}_{\hat{B}}$ held by $\hat{B}$, such that $\pi^{s^*}_{\hat{A}}$ and $\pi^{t^*}_{\hat{B}}$ have matching sessions, and we have the following 'freshness' related disjoint cases:

- Case 1 ($C1$): The adversary did not issue $\mathsf{Corrupt}(\hat{A})$ and $\mathsf{Corrupt}(\hat{B})$.
- Case 2 ($C2$): The adversary did not issue $\mathsf{StateReveal}(\pi^{s^*}_{\hat{A}})$ and $\mathsf{StateReveal}(\pi^{t^*}_{\hat{B}})$.
- Case 3 ($C3$): The adversary did not issue $\mathsf{StateReveal}(\pi^{s^*}_{\hat{A}})$ and $\mathsf{Corrupt}(\hat{B})$.
- Case 4 ($C4$): The adversary did not issue $\mathsf{Corrupt}(\hat{A})$ and $\mathsf{StateReveal}(\pi^{t^*}_{\hat{B}})$.

Event 2: There is no oracle $\pi^{t^*}_{\hat{B}}$ held by $\hat{B}$, such that $\pi^{s^*}_{\hat{A}}$ and $\pi^{t^*}_{\hat{B}}$ have matching sessions, and we have the following disjoint cases:

- Case 5 ($C5$): The adversary did not issue $\mathsf{Corrupt}(\hat{A})$ and $\mathsf{Corrupt}(\hat{B})$.
- Case 6 ($C6$): The adversary did not issue $\mathsf{StateReveal}(\pi^{s^*}_{\hat{A}})$ and $\mathsf{Corrupt}(\hat{B})$.

In order to complete the proof of Theorem 1, we must provide the security proofs for above six cases. We first show three propositions to facilitate our proof. Note that the proof of these two propositions can be found in [32].

Proposition 1. *If adversary $\mathcal{A}$ breaks the session key security of the proposed protocol in case $C4$, then there exists adversary $\mathcal{A}'$ who can launch a successful attack in case $C3$.*

Proposition 2. *If adversary $\mathcal{A}$ breaks the session key security of the proposed protocol in case $C1$ ($C3$), then there exists adversary $\mathcal{A}'$ who can launch a successful attack in the case $C5$ ($C6$).*

Therefore, we prove the advantage of the adversary is negligible under the proof simulation for cases $C2$ and $C5$ and $C6$. The proof proceeds in a sequence of games, following [36, 4]. The first game is the real security experiment, as assumed that there exists an adversary $\mathcal{A}$ that breaks the session key security of the proposed protocol. We then describe several intermediate games that step-wisely modify the original game. Finally we prove that (under the stated security assumptions), no adversary can break the security of the protocol.

Let S_δ be the event that the adversary wins the security experiment under the Game δ and freshness cases in the set $\{C2, C5, C6\}$. Let $\mathsf{ADV}_\delta := \Pr[\mathsf{S}_\delta] - 1/2$ denote the advantage of $\mathcal{A}$ in Game δ. Consider the following sequence of games.

Game 0. This is the original eCK game with adversary $\mathcal{A}$ under freshness cases C_2, C_5 and C_6. Thus we have that

$$\Pr[\mathsf{S}_0] = 1/2 + \epsilon = 1/2 + \mathsf{ADV}_0.$$

Game 1. In this game, the challenger proceeds exactly like previous game, except that we add a abortion rule. The challenger raises event $\mathsf{abort}_{\mathsf{trans}}$ and aborts, if during the simulation either the ephemeral key epk_i^s (outputted by $\mathsf{KE.EKGen}$) or C_i^s (generated by KEM) replied by an oracle π_i^s but it has been sample by another oracle π_u^w or sent by adversary before. Since there are $d\ell$ such values would be sampled randomly by KEM or KE. Due to theirs security, the event $\mathsf{abort}_{\mathsf{trans}}$ occurs with probability $\Pr[\mathsf{abort}_{\mathsf{trans}}] \leq \frac{(d\ell)^2}{2^\lambda}$ where λ is a large integer as assumed before. We therefore have that

$$\mathsf{ADV}_0 \leq \mathsf{ADV}_1 + \frac{(d\ell)^2}{2^\lambda}.$$

Note that the transcript of protocol messages shared by two oracles having matching sessions is unique in this game, so that the adversary can't replay any ciphertext or ephemeral key to result in two fresh oracles generating the same session key but without matching sessions.

Game 2. This game proceeds as previous game, but $\mathcal{C}$ aborts if one of the following guesses fails: (i) the freshness case occurred to test oracle in the set $\{C2, C5, C6\}$, (ii) the test oracle $\pi_{\hat{A}}^{s^*}$, and (iii) its partner oracle $\pi_{\hat{B}}^{t^*}$ in case $C2$ or the intended communication party $\hat{B}$ in cases $C5$ and $C6$. Since there are 3 fresh related cases, ℓ parties at all and at most d oracles for each party, then the probability that all guesses of $\mathcal{C}$ are correct is at least $1/3(d\ell)^2$. Thus we have that

$$\mathsf{ADV}_1 \leq 3(d\ell)^2 \cdot \mathsf{ADV}_2.$$

In the following, we always assume that the challenger guesses correctly.

Game 3. This game is proceeded as previous game, but the challenger $\mathcal{C}$ does the following modifications:

1. In the case $C2$, replace the key $eK_{\hat{A}}^*$ of test oracle $\pi_{\hat{A}}^{s^*}$ and its partner oracle $\pi_{\hat{B}}^{t^*}$ with random value $\widetilde{eK_{\hat{A}}^*}$.
2. In the case $C5$, replace the NIKE key $ShK_{\hat{A},\hat{B}}$ with random value $\widetilde{ShK_{\hat{A},\hat{B}}}$ for all oracles having session participants $\hat{A}$ and $\hat{B}$. So that $\widetilde{ShK_{\hat{A},\hat{B}}}$ is used in place of either $\mathsf{NIKE.ShareKey}(\hat{A}, \hat{B})$ or $\mathsf{NIKE.ShareKey}(\hat{B}, \hat{A})$.
3. In the case $C6$, replace the KEM key $K_{\hat{A}}^*$ encapsulated in the ciphertext $C_{\hat{A}}^*$ generated by test oracle with random value $\widetilde{K_{\hat{A}}^*}$. Whenever $C_{\hat{A}}^*$ is received by an oracle $\pi_{\hat{B}}^t$, the key $\widetilde{K_{\hat{A}}^*}$ is used instead of $\mathsf{KEM.DeCap}(sk_{\hat{B}}, C_{\hat{A}}^*)$.

If there exists an adversary $\mathcal{A}$ can distinguish the Game 3 from Game 2 then we can use it to construct an adversary $\mathcal{D}$ to break security of KE in the case $C2$ or NIKE in the case $C5$ or KEM in the case $C6$ respectively. Specifically, $\mathcal{D}$ simulates the challenger for $\mathcal{A}$ as previous game but with the following modifications based on its correct guesses (otherwise it aborts).

1. **Case** $C2$. Given a challenge instance $(epk_1^*, epk_2^*, K_c^*)$ from KE security experiment (where K_c^* is either a random value or the truth KE key in terms of challenge public keys epk_1^* and epk_2^*), $\mathcal{D}$ does the following modifications:
   (a) Set $epk_{\hat{A}}^* := epk_1^*$ and $epk_{\hat{B}}^* := epk_2^*$.
   (b) Compute the key material for test oracle and its partner oracle as $eK^* := K_c^*$.
2. **Case** $C5$. $\mathcal{D}$ selects $(\hat{A}, \hat{B})$ as identities of NIKE-tested parties which have long-term public keys pk_1^* and pk_2^* (obtained via $\mathsf{RegisterHonest}$ query) and gets K_c^* from $\mathsf{Test}^{nike}(\hat{A}, \hat{B})$ query in the NIKE security experiment, and it simulate the game for $\mathcal{A}$ with the following modifications:
   (a) Set $pk_{\hat{A}} = pk_1^*$ and $pk_{\hat{B}} = pk_2^*$.
   (b) Replace the shared key $ShK_{\hat{A},\hat{B}}$ with K_c^* for oracles having both session participants $\hat{A}$ and $\hat{B}$.
   (c) Generate the share key $ShK_{\hat{C},\hat{D}}$ using $\mathsf{RevealKey}^{nike}(\hat{C}, \hat{D})$ from NIKE security experiment, when there is one and at most one party such that $\hat{C} \in \{\hat{A}, \hat{B}\}$ or $\hat{D} \in \{\hat{A}, \hat{B}\}$.
3. **Case** $C6$. Given a challenge instance (C^*, pk^*, K_c^*) from KEM security experiment, $\mathcal{D}$ does the following modifications:
   (a) Set $pk_{\hat{B}} = pk^*$ and the outgoing ciphertext of test oracle as $C_{\hat{A}}^* := C^*$.
   (b) Replace the KEM key $K_{\hat{A}}^*$ with K_c^* for test oracle and the oracles of $\hat{B}$ receiving the ciphertext $C_{\hat{A}}^*$.
   (c) Generate the key $K_{\hat{D}}$ of other oracles $\pi_{\hat{B}}^t$ of party $\hat{B}$ using the decryption oracle $\mathcal{DEC}(sk_{\hat{B}}^{kem}, \cdot)$, where $\hat{D}$ could be any parties (but $\hat{B}$). Specifically, while receiving a ciphertext $C_{\hat{D}}$, oracle $\pi_{\hat{B}}^t$ computes the $K_{\hat{D}} := \mathcal{DEC}(sk_{\hat{B}}^{kem}, C_{\hat{D}})$.

   As for the rest of the computations are done as the same as previous game.

To answer the $\mathsf{RevealKey}$ query and Test query for those modified oracles, $\mathcal{D}$ will use the changed key material to compute the final session key. With respect to the other queries, $\mathcal{D}$ simulates them honestly as the challenger in previous game using corresponding values chosen by herself. Without flipping the bit b, the Test-query is replied with the session key which is computed using modified key material. If K_c^* is true key, then the simulation is equivalent to Game 2; otherwise the simulation is equivalent to Game 3. Finally, $\mathcal{D}$ returns what $\mathcal{A}$ returns to KEM or KE or NIKE challenger in corresponding security experiment. If $\mathcal{A}$ can distinguish the real key from the random value, that implies $\mathcal{D}$ either break the KEM or KE or NIKE scheme under corresponding freshness case. Select the maximum advantage of adversary in each case, we therefore obtain that

$$\mathsf{ADV}_2 \leq \mathsf{ADV}_3 + \mathsf{MAX}(\epsilon_{\mathsf{KE}}, \epsilon_{\mathsf{KEM}}, \epsilon_{\mathsf{NIKE}}).$$

Game 4. We change this game from previous one by modifying the generation of output of SEXT in terms of freshness cases. Specifically, (i) in case $C2$ the value $eK^{*\prime}$ of test oracle and its partner oracle(if it exists) is replaced with a uniform random value instead of the use of $\mathsf{SEXT}(\widetilde{eK^*})$, (ii) in the case $C5$ the $ShK_{\hat{A},\hat{B}}{}'$ is replaced with a uniform random value instead of the use of $\mathsf{SEXT}(\widetilde{ShK_{\hat{A},\hat{B}}})$ for oracles having both session participants $\hat{A}$ and $\hat{B}$, (iii) and in the case $C6$, we replace ${K^*_{\hat{A}}}'$ of test oracle with a random value and we do the same replacement for the oracles of $\hat{B}$ receiving the ciphertext $C^*_{\hat{A}}$ generated by test oracle. Those changes are possible since we have the fact either key $\widetilde{eK^*}$ or key $ShK_{\hat{A},\hat{B}}$ or key $K^*_{\hat{A}}$ has been modified to be random value in previous game. By the security of the strong randomness extraction function, we have that

$$\mathsf{ADV}_3 \leq \mathsf{ADV}_4 + \epsilon_{\mathsf{SEXT}}.$$

Game 5. In this game, we change function $\mathsf{PRF}({ShK_{\hat{A},\hat{B}}}', \cdot)$ in the case $C5$ or the function $\mathsf{PRF}({K^*_{\hat{A}}}', \cdot)$ in the case $C6$ or function $\mathsf{PRF}(eK^{*\prime}, \cdot)$ in the case $C2$ to a truly random function for test oracle and its partner oracle (if it exists). We make use of the fact, that the at least one of those three secret seeds of the PRFs of test oracle is a truly random value due to the modification in previous game. If there exists a polynomial time adversary $\mathcal{A}$ can distinguish the Game 5 from Game 4. Then we can construct an algorithm $\mathcal{B}$ using $\mathcal{A}$ to break the security of PRF. Exploiting the security of PRF, we have that

$$\mathsf{ADV}_4 \leq \mathsf{ADV}_5 + \epsilon_{\mathsf{PRF}}.$$

Note that in this game the session key returned by Test-query is totally a truly random value which is independent to the bit b and any messages. Thus the advantage that the adversary wins this game is $\mathsf{ADV}_5 = 0$.

Put together all probabilities from Game 0 to Game 5, we proved this theorem.

B Proof of Theorem 2

First of all, it is straightforward to verify that two accepted oracles of considered protocol with matching sessions would generate the same session key. In the sequel, we wish to show that the adversary is unable to distinguish random value from the session key of any fresh oracle. Without loss of generality, we consider that the adversary chooses the test oracle $\pi^{s^*}_{\hat{A}}$ executed between owner $\hat{A}$ with its indented partner $\hat{B}$.

Next we introduce the notations which might be used in the proof. Let party $\hat{C}$ denote the indented partner of oracle $\pi^s_{\hat{A}}$ where $\hat{C}$ could be any parties but $\hat{A}$ in the security game and has long-term public key $C = g^c$, and let $W = g^w$ denote the ephemeral public key received by oracle $\pi^s_{\hat{A}}$. In a similar way, oracle $\pi^t_{\hat{B}}$ has indented party denoted by $\hat{D}$ with long-term public key $D = g^d$, and the received ephemeral key by $\pi^t_{\hat{B}}$ is $N = g^n$. Meantime the ephemeral keys generated by oracles $\pi^s_{\hat{A}}$ and $\pi^t_{\hat{B}}$ are $X = g^x$ and $Y = g^y$ respectively. we use the superscript '*' to highlight corresponding values processed in the test oracle and its partner oracle (if it exists), say the ephemeral key X^* generated by oracle $\pi^{s^*}_{\hat{A}}$.

Applying the propositions 1 and 2 from [32], the security proof can be given only under freshness cases $C2$, $C5$ and $C6$. The proof proceeds in a sequence of games, following [36, 4]. The first game is the real security experiment, as assumed that there exists an adversary $\mathcal{A}$ that breaks the session key security of the proposed protocol. We then describe several intermediate games that step-wisely modify the original game. Finally we prove that (under the stated security assumptions), no adversary can break the security of the protocol. Let S_δ be the event that the adversary wins the security experiment under the Game δ and freshness cases in the set $\{C2, C5, C6\}$. Let $\mathsf{ADV}_\delta := \Pr[\mathsf{S}_\delta] - 1/2$ denote the advantage of $\mathcal{A}$ in Game δ.

Game 0. This is the original eCK game with adversary $\mathcal{A}$ under freshness cases C_2, C_5 and C_6. The system parameters are chosen honestly by challenger as protocol specification. Meanwhile, the challenger chooses six uniform random values (r_1, r_2, r_3, r_4)
$\overset{\$}{\leftarrow} \mathbb{Z}_p^*$, and sets $u_1 := g^{r_1}, u_2 := g^{r_2}$, $u_3 := g^{r_3}$ and $u_4 := g^{r_4}$ as public parameters. All queries are simulated honestly in terms of protocol specification. Thus we have that

$$\Pr[\mathsf{S}_0] = 1/2 + \epsilon.$$

Game 1. In this game, the challenger proceeds exactly like previous game, except that we add an abort rule. The challenger raises event $\mathsf{abort}_{\mathsf{eph}}$ and aborts, if during the simulation an ephemeral key replied by an oracle π_i^s but it has been sampled by another oracle or sent by adversary before. Since there are $d\ell$ such ephemeral keys would be sampled uniform randomly from $\{0,1\}^\lambda$. Thus, the event $\mathsf{abort}_{\mathsf{eph}}$ occurs with probability at least $\Pr[\mathsf{abort}_{\mathsf{eph}}] \leq \frac{d^2\ell^2}{2^\lambda}$. We have that

$$\mathsf{ADV}_0 \leq \mathsf{ADV}_1 + \frac{d^2\ell^2}{2^\lambda}.$$

Note that the ephemeral key chosen by each oracle is unique in this game, so that the adversary can't replay any ephemeral key to result in two oracles generating the same session key but without matching sessions.

Game 2. In this game we want to make sure that the received ephemeral keys are correctly formed. Technically, we add an abort condition, namely the challenger proceeds exactly as before, but raises event $\mathsf{abort}_{\mathsf{hash}}$ and aborts if there exist two distinct (either ephemeral or long-term) public keys W and N such that $\mathsf{TCRHF}(W) = \mathsf{TCRHF}(N)$. Obviously the $\Pr[\mathsf{abort}_{\mathsf{hash}}] \leq \epsilon_{\mathsf{TCRHF}}$, according to the security property of underlying hash function. Thus we have

$$\Pr[\mathsf{S}_1] \leq \Pr[\mathsf{S}_2] + \epsilon_{\mathsf{TCRHF}}.$$

Game 3. This game proceeds as previous game, but $\mathcal{C}$ aborts if one of the following guesses fails: (i) the freshness case occurred to test oracle in the set $\{C2, C5, C6\}$, (ii) the test oracle $\pi_{\hat{A}}^{s^*}$, and (iii) its partner oracle $\pi_{\hat{B}}^{t^*}$ in case $C2$ or the intended communication partner $\hat{B}$ in cases $C5$ and $C6$. Since there are 3 fresh related cases, ℓ parties at all and at most d oracles for each party, then the probability that all guesses of $\mathcal{C}$ are correct is at least $1/3(d\ell)^2$. Thus we have that

$$\mathsf{ADV}_2 \leq 3(d\ell)^2 \cdot \mathsf{ADV}_3.$$

Game 4. In this game, we want to reduce the security of proposed protocol to the hardness of BDDH problem. Please first note that there are at least two uncompromised (either long-term and ephemeral) Diffie-Hellman (DH) keys which are used by test oracle to generate its key material k^*, in terms of certain freshness case. As otherwise the test oracle is no longer eCK-fresh. We call such guessed two uncompromised DH keys as *target DH keys*.

This game is proceeded as previous game, but the challenger $\mathcal{C}$ replaces the key material k_i^s with random value $\widetilde{k_i^s}$ for oracles $\{\pi_i^s : i \in [\ell], s \in [d]\}$ which satisfy the following conditions:

- The k_i^s is computed involving the two *target DH keys* which are guessed by $\mathcal{C}$ for test oracle, and
- These two *target DH keys* used by π_i^s are from two distinct parties.

The second condition is necessary, because the adversary can easily result in one oracle receiving DH keys from certain party which are all uncompromised DH keys via e.g. Send query RegisterCorrupt queries. On the other side, the first condition cannot exclude the event that the DH keys from certain party are all chosen (or revealed) by adversary. In this case, the adversary can compute the session key of such oracle and we cannot change the key material of that oracle any more. The above two conditions ensure that the changed key materials of oracles can not be trivially generated by adversary.

If there exists an adversary $\mathcal{A}$ can distinguish between Game 4 and Game 3 then we can use it to construct a distinguisher $\mathcal{D}$ to solve the BDDH problem as follows. Given a BDDH challenge instance $(g, g^\nu, g^\omega, g^\xi, \Gamma) \in \mathbb{G}^3 \times \mathbb{G}_T$, $\mathcal{D}$'s goal is to determine whether Γ equals to $e(g,g)^{\nu\omega\xi}$ or a random element, where g is a group generator of $\mathbb{G}$. More specifically, $\mathcal{D}$ simulates the challenger for $\mathcal{A}$ as previous game but with the following modifications based on its correct guesses (otherwise it aborts). Meanwhile, let $p(h) = p_0 + p_1 h + p_2 h^2$ be a polynomial of degree 2 over $\mathbb{Z}_p^*$. The detail form of this polynomial will be discussed in the simulation based on specific freshness case. Let $q(h) = q_0 + q_1 h + q_2 h^2$ be a random polynomial of degree 2 over $\mathbb{Z}_p^*$.

1. **Case** $C2$. In this case $\mathcal{D}$ does the following modifications:
   (a) Set $X^* := g^\nu, Y^* := g^\omega$ and $u_1 := g^\xi$.
   (b) Compute the key material of test oracle and its partner oracle as:
      - $k^*_{\hat{A}} = k^*_{\hat{B}} := \Gamma \cdot e(u_1, BY^*)^a \cdot e(u_1, X^*)^b$.
   (c) Compute those tags of test oracle and its partner oracle as:
      - $t^*_X := (X^*)^{r_2 (h^*_X)^2 + r_3 h^*_X + r_4}$, where $h^*_X = \mathsf{TCRHF}(X^*)$.
      - $t^*_Y := (Y^*)^{r_2 (h^*_Y)^2 + r_3 h^*_Y + r_4}$, where $h^*_Y = \mathsf{TCRHF}(Y^*)$.
2. **Case** $C5$. In this case, $\mathcal{D}$ does the following modifications:
   (a) Set $u_1 := g^\nu,\ A := g^\omega$, and $B := g^\xi$.
   (b) Set polynomial $p(h)$ to satisfy that $p(h) = (h - h_A)(h - h_B)$, where $h_A = \mathsf{TCRHF}(A)$, $h_B = \mathsf{TCRHF}(B)$.
   (c) Set $u_{i+2} = {u_1}^{p_i} g^{q_i}$ for $0 \le i \le 2$.
   (d) Compute $t_A := A^{q(h_A)}$ and $t_B = B^{q(h_B)}$.
   (e) Replace the value $e(u_1, A)^b$ with Γ when computing the key material k of oracles $\pi^s_{\hat{A}}$ and $\pi^t_{\hat{B}}$ which involve both long-term public keys A and B (including the test oracle $\pi^{s^*}_{\hat{A}}$), more specifically:
      - $k^s_{\hat{A}} := \Gamma \cdot e(u_1, BW)^x \cdot e((\frac{t_W}{W^{q(h_W)}})^{1/p(h_W)}, A)$, where $h_W = \mathsf{TCRHF}(W)$.
      - $k^t_{\hat{B}} := \Gamma \cdot e(u_1, AN)^y \cdot e((\frac{t_N}{N^{q(h_N)}})^{1/p(h_N)}, B)$, where $h_N = \mathsf{TCRHF}(N)$.
   (f) Compute the secret key material for other oracles $\pi^s_{\hat{A}}$ and $\pi^t_{\hat{B}}$ of party $\hat{A}$ and $\hat{B}$:
      - $k^s_{\hat{A}} := e(u_1, CW)^x \cdot e((\frac{t_W}{W^{q(h_W)}})^{1/p(h_W)}, A) \cdot e((\frac{t_C}{C^{q(h_C)}})^{1/p(h_C)}, A)$, where $h_W = \mathsf{TCRHF}(W)$ and $h_C = \mathsf{TCRHF}(C)$.

- $k^t_{\hat{B}} := e(u_1, ND)^y \cdot e((\frac{t_N}{N^{q(h_N)}})^{1/p(h_N)}, B) \cdot e((\frac{t_D}{D^{q(h_D)}})^{1/p(h_D)}, B)$, where $h_W = \mathsf{TCRHF}(N)$ and $h_D = \mathsf{TCRHF}(D)$.

3. **Case** $C6$. In this case, $\mathcal{D}$ does the following modifications:
   (a) Set $X^* := g^\nu, B := g^\omega$ and $u_1 := g^\xi$.
   (b) Set polynomial $p(h)$ to satisfy that $p(h) = (h - h^*_X)(h - h_B)$, where $h^*_X = \mathsf{TCRHF}(X^*)$, $h_B = \mathsf{TCRHF}(B)$.
   (c) Set $u_{i+2} = {u_1}^{p_i} g^{q_i}$ for $0 \le i \le 2$.
   (d) Compute the $t_B := B^{q(h_B)}$ and $t^*_X := B^{q(h^*_X)}$.
   (e) Compute the key material $k^*_{\hat{A}}$ of test oracle $\pi^{s^*}_{\hat{A}}$ and $k^t_{\hat{B}}$ of oracles $\pi^t_{\hat{B}}$ having session states t^*_X and X^*, and corresponding tag t^*_X as:
      - $k^*_{\hat{A}} := \Gamma \cdot e((\frac{t^*_W}{{W^*}^{q(h^*_W)}})^{1/p(h^*_W)}, X^*) \cdot e(u_1, BW^*)^a$, where $h^*_W = \mathsf{TCRHF}(W^*)$.
      - $k^t_{\hat{B}} := \Gamma \cdot e(u_1, DX^*)^y \cdot e((\frac{t_D}{D^{q(h_D)}})^{1/p(h_D)}, B)$, where $h_D = \mathsf{TCRHF}(D)$.
   (f) Change the computation of secret key material $k^t_{\hat{B}}$ of other oracles of $\hat{B}$ as
      - $k^t_{\hat{B}} := e(u_1, DN)^y \cdot e((\frac{t_D}{D^{q(h_D)}})^{1/p(h_D)} (\frac{t_N}{N^{q(h_N)}})^{1/p(h_N)}, B)$.

Those modified tags are consistent with the original form. we make use of the fact there is no collision on those hash values due to the result of previous game. To answer the $\mathsf{RevealKey}$ query for those modified oracles, the $\mathcal{D}$ will use the changed key material (e.g. $k^t_{\hat{B}}$) to compute the final session key as protocol specification. With respect to the other queries, the $\mathcal{D}$ simulates them honestly as the challenger using corresponding values chosen by herself. Without flipping the bit b, the Test-query is replied with the session key which is computed using modified key material. Based on the condition that all guesses by $\mathcal{D}$ are correct, if $\Gamma = e(g,g)^{\nu\omega\xi}$, then the simulation is equivalent to Game 3; otherwise the simulation is equivalent to Game 4. At the end of the simulation, $\mathcal{D}$ returns what $\mathcal{A}$ returns to BDDH challenger. If $\mathcal{A}$ can distinguish the real key from the random value, that implies $\mathcal{D}$ solves the BDDH problem. We therefore obtain that

$$\mathsf{ADV}_3 \le \mathsf{ADV}_4 + \epsilon_{\mathsf{BDDH}}.$$

Game 5. In this game, we change function $\mathsf{PRF}(\widetilde{k^*_{\hat{A}}}, \cdot)$ to a truly random function for test oracle and its partner oracle (if it exists). We make use of the fact, that the secret seed $\widetilde{k^*_{\hat{A}}}$ of test oracle is a truly random value. If there exists a polynomial time adversary $\mathcal{A}$ can distinguish the Game 5 from Game 4. Then we can construct an algorithm $\mathcal{B}$ using $\mathcal{A}$ to break the security of PRF. Exploiting the security of PRF, we have that

$$\mathsf{ADV}_4 \le \mathsf{ADV}_5 + \epsilon_{\mathsf{PRF}}.$$

Note that in this game the session key returned by Test-query is totally a truly random value which is independent to the bit b and any messages. Thus the advantage that $\mathcal{A}$ wins this game is $\mathsf{ADV}_5 = 0$.

Sum up the probabilities from Game 0 to Game 5, we proved this theorem.