A Unified Security Model of Authenticated Key Exchange with Specific Adversarial Capabilities

Weiqiang Wen[1,2] Libin Wang[1,2] Jiaxin Pan[3]

[1]School of Computer, South China Normal University,
Guangzhou 510631, China
[2]State Key Laboratory of Information Security (Institute of Information Engineering,
Chinese Academy of Sciences, Beijing 100093)
weiqwen@gmail.com, lbwang@scnu.edu.cn
[3]Faculty of Mathematics
Horst Görtz Institute for IT-Security
Ruhr University Bochum, Germany
jiaxin.pan@rub.de

Abstract. The most widely accepted models in the security proofs of Authenticated Key Exchange protocols are the Canetti-Krawczyk model and the extended Canetti-Krawczyk model. They are shown to be incomparable due to the subtleties that they admit different adversarial queries and the definitions of the queries are not specific and strict enough to allow a rigorous comparison be made. Concerning the security of one-round implicitly authenticated Diffie-Hellman key exchange protocols, we present a stronger security model that characterizes specific adversarial capabilities and encompass the Ephemeral Key Reveal and the Session-State Reveal simultaneously. To demonstrate the usability of our model, a new protocol based on the OAKE protocol is proposed, which satisfies the presented stronger security notion and at the same time attains high efficiency as the OAKE protocol. The protocol is proven secure in random oracle model under the gap Diffie-Hellman assumption.
Keywords: Authenticated Key Exchange; Provable Security; Security Model

1 Introduction

Motivation. Authenticated Key Exchange (AKE) is an important cryptographic primitive which enables two parties, who hold each others' public key, to establish a common secret key in an insecure network. Numerous AKE protocols have been proposed in the last decade, it is desirable that the proposed protocols can be proven secure in a security model. Indeed, many of the protocols have been formally proven secure, but some of them have been shown to be insecure. Roughly, a security model describes realistic attack capabilities in open networks, and gives the precise definition of security. After the seminal work of Bellare and Rogaway [1], a number of security models were proposed [2,3,4], among these models, the Canetti-Krawczyk (CK) model and the extended Canetti-Krawczyk (eCK) model are the most widely accepted models in security proofs of AKE protocols.

An important goal of the security model is to specify the potential capabilities of adversaries. In general, the adversary can control the communication links between parties, moreover, the adversary is allowed access to secret information used or generated in the protocol via reveal queries. In the CK model, three types of reveal queries are defined, including Session Key Reveal, Session-State Reveal and Corrupt query. The eCK model aims to define a stronger notion of security by providing significantly severer queries to the adversary. The model defines a new Ephemeral Key Reveal query, which outputs all session-specific secret information of a party, to replace the Session-State Reveal query of the CK model, and provides a new definition of freshness that permits Ephemeral Key Reveal queries on the test session. Seemingly, the eCK model is stronger than the CK model, however, this is not the case, as many work showed that they are incompatible and incomparable [5,6,7], which means some protocols can be proven secure in one model but can be proven insecure in the other, and vice versa. For example, it is showed that the NAXOS protocol is proven secure in the eCK model, but it is insecure in the CK model [5].

The subtle issues that arises between the CK model and the eCK model stem from the fact that they admit different adversarial queries and the definitions of the queries are not specific and strict enough to allow a rigorous comparison be made. In the CK model, the Session-State Reveal query models an adversary who can capture all the intermediate computation results of a given session. The ambiguity lies on the fact that what information is included in the session state will be specified by each AKE protocol [2]. In some papers, to present a proven secure protocol in the CK model, the session state is carefully defined as public values, or will be erased immediately, such that the Session-State Reveal query has a very limited power of information leakage. As for the eCK model, the original Ephemeral Key Reveal is allowed to output all session-specific secret information, but in some subsequent papers [5,6,7,8,9], it is defined that the Ephemeral Key Reveal query only outputs the ephemeral private key held by the specific session. Due to these ambiguities, it is not clearly explained why one kind of information can (or can not) be leaked. Even worse, if the output of the reveal queries can be deliberately tailored from the protocol values, the security claims based on these models will reflect more or less subjective opinion of the protocol designers. We remark that to achieve objective evaluation of AKE protocols a security model with specific adversarial capabilities is needed.

Many works analyze the differences between the CK model and the eCK model by examining the strength of the Ephemeral Key Reveal query and the Session-State Reveal query, however it is worth noting that the two queries are proposed to model two different kinds of adversary in practice and they embody different harmfulness of attacks. We observe that a significant difference between the Ephemeral Key Reveal and the Session-State Reveal is that the session which is compromised by the Ephemeral Key Reveal may keep its freshness, but the session which is compromised by the Session-State Reveal will lose its freshness.

In other words, the two queries may be subject to different security levels in effect. Thus we need not simply support one query and oppose the other, on the contrary, it is desirable to incorporate these queries into a single unified security model and establish a stronger security notion.

In this work, we focus on the security of one-round implicitly authenticated Diffie-Hellman key exchange protocols, and aim to propose a stronger security model that characterizes specific adversarial capabilities and encompass the Ephemeral Key Reveal and the Session-State Reveal simultaneously. To achieve this goal, we must solve a major problem. If the leakages of the Ephemeral Key Reveal and the Session-State Reveal are precisely defined in a determinate manner, instead of chosen strategically by the designer of the protocol, and these two queries are allowed simultaneously for a session, the security proof appears to be extremely hard since encountering such a powerful adversary, to construct a consistent simulation in the security proof is very difficult. In another word, it is highly technical to unify Session-State Reveal and Ephemeral Key Reveal into one model. Fortunately, we present a solution to the problem in this paper.

Contributions. The contribution of the paper are two-fold. Firstly, we propose a new security model of AKE with powerful and specific adversarial capabilities, which is roughly a combination of the CK model and the eCK model and named as PACK, which stands for **P**owerful and specific **A**dversarial capabilities **CK** model and also means that our model **PACK**s the CK model and the eCK model into a single model. We set up the protocol implementation environment, and from a practical perspective, we clearly define four kinds of reveal queries and determinately specify what kind of information can be accessed by each query. Subsequently, we present a new definition of freshness of protocol sessions and establish a stronger notion of security for AKE protocols, which encompasses both CK model and eCK model.

Secondly, we present a new AKE protocol called VOAKE which is a variant of the OAKE protocol [10]. The OAKE protocol is proved to be strongly secure in the CK model (reformulated in the paper [11], named as CK-HMQV) with superior efficiency, however it is insecure in the eCK model (refer to Appendix B), thus is not resistant to Ephemeral Key Reveal, which is also characterized in the PACK model. In this paper, we derive a new protocol from the OAKE protocol and prove that it is secure in the PACK model. Our results demonstrate that the VOAKE protocol is a substantial improvement of the OAKE protocol since it attains almost the same efficiency as the OAKE protocol while at the same time satisfies a strictly stronger security notion.

Related Works. The CK and eCK models are well-known formal security models for AKE protocols. In recent years, a number of variants of these models have been proposed, including CK-HMQV [11], seCK [4], eCK^w [12]. There are a trend of works devoted to investigate the relationship of strength between CK and eCK [5,6,7]. Cremers [5] analyzed the subtleties between the CK and eCK models and showed that Session-State Reveal is stronger than Ephemeral Key Reveal, thus these two models are incomparable. Our work aims to further clarify the security notion of the formal model.

The closest related work to ours is the work of Sarr *et al.*. Aiming to combine the CK and eCK models, they propose the strengthened eCK (seCK) model [4]. Two implementation environments are firstly set up in the seCK model. Then two sets of adversarial queries are defined to model leakages that may occur on either implementation environments, one set corresponds to the eCK model and the other corresponds to the CK model. It is shown that the seCK model encompasses the eCK model and is *practically* stronger than the CK model. As suggested in paper of Yoneyama *et al.* [13], it is highly subtle to design a provably secure Diffie-Hellman type protocol in the seCK model. Thus it is a major concern to design a provably secure AKE protocol in a stronger model.

A number of AKE protocols claimed to be strongly secure in a formal model have been proposed [4,14,15,10,16]. The SMQV protocol [4] achieves high efficiency as (H)MQV protocols and strongly secure in the seCK model. However, the security proof of SMQV protocol in the seCK model was shown to be incomplete as the leakage of intermediate values can not be perfectly simulated in the proof, to fix the problem appears to be hard as Yoneyama *et al.* suggested in [13]. Zhao *et al.* [15] present a new protocol family named (s)YZ family which is later reformulated and improved to be (T)OAKE family [10], which also achieves high efficiency as HMQV protocol. The OAKE protocol is eCK insecure (refer to Appendix B), and its variant T-OAKE (also named srYZ) is also shown to be eCK insecure [17], which limits their implementation. Thus, it is still an urgent work to design a strongly secure protocol with high efficiency. In this paper, we improve the OAKE protocol by applying NAXOS technique, which strengthens the security of OAKE as well as retains high efficiency as OAKE.

Organization. The rest of this paper is organized as follows. Firstly, we introduce the PACK model in section 2. In section 3, we describe the proposed VOAKE protocol as well as its security proof in the PACK model. Some comparison focusing on models and protocols as well as concluding remark are given in Section 4.

2 Security Model

In this section, we present a security model which is a combination of the CK model and the eCK model and allows more powerful and specific adversarial capabilities. Our model follows the basic notions of the previous security models [2,11,3,4], including the message-driven style description of AKE protocols and the security definition *etc.*, but strengthens the reveal queries and strictly classifies the secret information to be leaked.

2.1 Design Rationale

Before presenting the formal definition, we present the intuitive principles behind the design of the model by setting up the system implementation environment. Then we clarify what kind of the security breach happens in practice will be modeled by which adversarial query, and explicitly specify what kind of information can be accessed by which query.

System Environment. An AKE protocol if often implemented in a common system setting where a tamper-resistant device is used to store the long-term private key and do some limited computations, such as hashing message, generating pseudorandom numbers or performing some basic arithmetic or logical operations. In such setting, a break-in cracker or a malicious insider can not trivially access to the long-term private key, but we do not assume any additional properties for the build-in functions. For instance, it is not assumed the hash function can resist side-channel attacks, or the pseudorandom number generator (PRNG) always products high quality randomness [18]. To accommodate more protocol scenarios, we do not mandatorily require that some computationally expensive functions, such as modular exponentiation, are (or not) implemented in the device, it will be optionally decided by the designer of the protocol.

It should be emphasized that assuming the existence of a tamper-resistant device is a normal requirement, we rely on the assumption to let the reveal queries be natural and reasonable. If the long-term private key is not stored in a tamper-resistant device, we have no reason to believe that to compromise the long-term private key is more difficult than to reveal ephemeral random numbers, or to reveal the intermediate computation results.

Additionally, implementations of hash functions in keyed settings are potential targets of side-channel attacks when the adversary's goal is to obtain information on the key, previous works include differential power analysis on HMAC-SHA-2 and template attacks on HMAC-SHA-1 [19]. Thus, we do not assume the implementation of hash function is side-channel attack resistant. The PRNG and the key derivation function may be realized by a hash function, therefore they are also suffered the same attacks as the hash function.

Practical Interpretation. Our model defines four types of queries to model the realistic attacks, we present the practical interpretation of the queries as follows to give the intuitive insight. The relationship between the system implementation environment and the adversarial queries is depicted in Fig. 1. The double-line box indicates the system region that will be compromised by the query linking to it with an arrow. Though all the names of the queries are adopted from the former models, but the connotation may be different. We prefer to retain the used names instead of coining new names for the queries because we wish that to retain these names will help the community discuss and compare related works, since these conventional names are quite famous.

- Ephemeral Key Reveal. The query models attacks or compromises of the PRNG or the hash function used by one of the parties, it returns the secret randomness (denoted as `EphemeralKey` in Fig. 1) of a specific protocol session.
- Session-State Reveal. The query captures the attacks of crackers who temporarily break into the computer system or the malicious action of insiders, it outputs all the intermediate secret values (denoted as `InterResult` in Fig. 1) computed or used in the host machine's memory and the session key (denoted as `SessionKey` in Fig. 1).

– Session Key Reveal. The query characterizes leakage on the session key either via the misusage of the key by application, or the cryptanalysis of the encryption algorithm *etc.*, it returns the session key of a named session.
– Long-Term Key Reveal. The query describes an attacker who bypasses the tamper-resistant device, and gains read access to the device's private memory, it provides the attacker with the device owner's long-term private key (denoted as `LongTermKey` in Fig. 1).

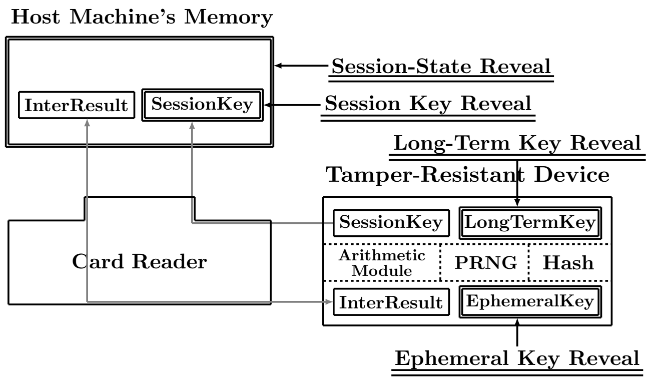

Fig. 1. The system implementation environment and the adversarial queries.

We remark that an important principle of the model is that which values should be leaked by the Ephemeral Key Reveal and the Session-State Reveal are not strategically defined by the protocol designer, instead, the computation process of the protocol will automatically determine which values should be leaked by which query. Furthermore, any declaration that the specific values will not be leaked because they are erased immediately after being used or are not stored in memory is invalid in the model.

The principle is adopted for three reasons. Firstly, to erase the specific values can not ensure they can not be leaked. For instance, a malicious insider who monitoring the system can reveal all the values in memory before they are erased. What is even worse, the Ephemeral Key Reveal attacks the underlying algorithms but not the memory. Secondly, the secrecy of session-specific values relies on the fact that in which part of the system will the values be computed, thus what the protocol designer should do is to carefully specify the computation processes, but not to optionally declare what values can be compromised. Finally, erasing data stored in memory is quite a difficult job as many research indicated [20]. Hopefully, by faithfully following the principle, an objective evaluation of the protocol under examination will be achieved.

It is also worth noting that our model allows the Ephemeral Key Reveal and the Session-State Reveal simultaneously, which makes our model much stronger than the previous ones. The queries have different adverse effects on the protocols sessions, thus we should clearly specify how the queries can be used by the adversary. Intuitively, we require that the leakage of ephemeral randomness of a session should not compromise the specified session, and the leakage of the secret values in the host machine's memory should not compromise other non-matching sessions. Formally, we need a new definition of freshness to restrict the test session that the adversary can choose.

In our model, the Session-State Reveal is defined to model the malicious action of an insider, it outputs all the secret values computed in the host machine's memory. In general, when a protocol session is completed, the session key will be generated and outputted from the tamper-resistant device to the host machine's memory. In this case, the Session-State Reveal is allowed to reveal the session key of a named session, which is also the capability of the Session Key Reveal. We incline to keep the Session Key Reveal as a separate query instead of replacing it with the Session-State Reveal for several reasons. Most of all, the two queries separately models different actions of attackers, in practice we need the Session Key Reveal to capture some kinds of attack, *e.g.*, the known-key attack.

2.2 Formal Description

In this subsection, we formally describe the security model. Our notation follows the previous models [11,3,8,4]. In the model, all the parties and the adversary are modeled as probabilistic polynomial-time (PPT) Turing machines. We assume that there is a certification authority (CA) trusted by all parties, however we do not assume that the CA requires parties to prove possession of their static private keys.

Session. A party $\hat{A}$ can be activated to run an instance of an AKE protocol, called a *session*. The adversary controls the activation of sessions via the Send query. $\hat{A}$ is activated by a Send query with one of the following forms: (i) $\mathsf{Send}(\hat{A}, \hat{B})$ or (ii) $\mathsf{Send}(\hat{A}, \hat{B}, X_B)$ where X_B is $\hat{B}$'s outgoing protocol message. If $\hat{A}$ is activated by $(\hat{A}, \hat{B})$ then $\hat{A}$ is the session *initiator* (denoted as $\mathcal{I}$) with peer $\hat{B}$, otherwise the session *responder* (denoted as $\mathcal{R}$) with peer $\hat{B}$. A session identifier is a quadruple $(Role, \hat{A}, \hat{B}, X_A, X_B)$ where $\hat{A}$ is the *owner* of the session, $\hat{B}$ is the *peer*. The session $(\mathcal{R}, \hat{B}, \hat{A}, X_B, X_A)$ (if it exists) is said to be *matching* to the session $(\mathcal{I}, \hat{A}, \hat{B}, X_A, *)$ or $(\mathcal{I}, \hat{A}, \hat{B}, X_A, X_B)$. Note that a session cannot have (except with negligible probability) more than one matching session.

Adversary. The adversary $\mathcal{A}$ presents parties with incoming messages via Send queries, obtains the outgoing messages of the parties, and makes decisions about their delivery, thereby controlling all communications between parties .

– $\mathsf{Send}(Message)$. The message has one of the following forms: $(\hat{A}, \hat{B})$, $(\hat{A}, \hat{B}, X_B)$ or $(\hat{A}, \hat{B}, X_A, X_B)$.

To capture information leakage the adversary is allowed the following queries, their output are denoted as data sets `EphemeralKey`, `SessionState`, `SessionKey` and `LongTermKey` respectively. We use `EphemeralKey` to denote the secret randomness of a session and `SessionState` to denote all the intermediate secret values of a session computed or used in the host machine's memory and the session key. The `SessionState` is a union of `InterResult` and `SessionKey`.

- Ephemeral Key Reveal(sid): The adversary obtains the `EphemeralKey` associated with session sid.
- Session-State Reveal(sid): The adversary $\mathcal{A}$ obtains the `SessionState` of the owner of session sid.
- Session Key Reveal(sid): The adversary obtains the `SessionKey` in a completed session sid.
- Long-Term Key Reveal($\hat{P}_i$): By making this query the adversary learns the `LongTermKey` of party $\hat{P}_i$.

Freshness of PACK. To define a stronger security definition, we need a new notion of freshness.

Definition 1 *(Freshness of PACK) Let sid = ($\mathcal{I}$, $\hat{A}$, $\hat{B}$, X_A, X_B) or ($\mathcal{R}$, U_A, U_B, X_A, X_B) be a completed session between honest parties $\hat{A}$ and $\hat{B}$. Let $\overline{sid}$ be the matching session of sid. We say session sid is fresh if none of the following conditions holds:*

1. *$\mathcal{A}$ issues* Session-Key Reveal(sid)*, or* Session-Key Reveal($\overline{sid}$) *if $\overline{sid}$ exists;*
2. *$\mathcal{A}$ issues* Session-State Reveal(sid)*, or* Session-State Reveal($\overline{sid}$) *if $\overline{sid}$ exists;*
3. *$\overline{sid}$ exists and the adversary $\mathcal{A}$ makes either of the following queries*
   - *both* Long-Term Key Reveal($\hat{A}$) *and* Ephemeral Key Reveal(sid)*, or*
   - *both* Long-Term Key Reveal($\hat{B}$) *and* Ephemeral Key Reveal($\overline{sid}$)*,*
4. *$\overline{sid}$ does not exist and the adversary $\mathcal{A}$ makes the following query*
   - *both* Long-Term Key Reveal($\hat{A}$) *and* Ephemeral Key Reveal(sid)*, or*
   - Long-Term Key Reveal($\hat{B}$)*.*

Security Experiment. The aim of the adversary $\mathcal{A}$ is to distinguish a session key from a random key, the experiment proceeds as follows. Initially, $\mathcal{A}$ is given a set of honest parties and makes any sequence of the queries described above. During the experiment, $\mathcal{A}$ makes the following query.

- Test(sid^t): In this query, sid^t must be a fresh session. To respond to this query, a random bit b is selected. If $b = 0$ then the session key of sid^t is output. Otherwise, a random key is output.

The experiment continues until $\mathcal{A}$ makes a guess b'. $\mathcal{A}$ *wins* the game if the test session sid^t is still fresh and if the guess of $\mathcal{A}$ is correct, i.e., $b' =$ b. The advantage of the adversary in the experiment with the AKE protocol Φ is defined as

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

Definition 2 (**AKE Security**) *A key exchange protocol Φ is called* `AKE` *secure if for all PPT adversaries $\mathcal{A}$ with the above capabilities running against Φ it holds:*

- *Parties who complete matching sessions compute the same session key.*
- *For any adversary $\mathcal{A}$ with the above capabilities, $\mathbf{Adv}_{\Phi}^{\mathsf{AKE}}(\mathcal{A})$ is negligible.*

3 VOAKE Protocol

Since many previous AKE protocols are not secure any more in the PACK model, and it also seems that to prove the security of a protocol in a strong model is quite hard [13]. One of the major goals of this work is to design an efficient AKE protocol that is provably secure in the PACK model, thereby to demonstrate the usability of the model. To fulfil the goal, we present a protocol called VOAKE which is a variant of the OAKE [15]. In this section, we present our design rationale, and describe the protocol in detail as well. The protocol is proven secure in random oracle model under the gap Diffie-Hellman (GDH) assumption.

3.1 Design Rationale

We base the design and proof of VOAKE on two techniques: the environment of all computations is specified; the NAXOS technique is applied to the computation of the ephemeral public key. By introducing the methods, we show that the VOAKE protocol satisfies strictly stronger security notions defined in the PACK model, and at the same time holds all the advantages of the original OAKE protocol with the cost of a little efficiency.
Computation Environment. To balance the tradeoff between performance and security, we set up the system implementation environment and explicitly specify the information flow between the host machine and the tamper-resistant device for the whole computing process of the protocol. Because the long-term private key should not be used in the host machine, therefore all computations involving the long-term private key will be settled in the tamper-resistant device. For instance, in the session owned by party $\hat{A}$, the value x, outputted by hashing the ephemeral and long-term private keys, is computed in the tamper-resistant device. To achieve high efficiency, our strategy is to perform as less exponentiations in the tamper-resistant device as possible. Since the values x, g^x and B^x are exposable, thus the value x can be used in the host machine to compute the values g^x and B^x. In order to ensure the master secret σ not to be leaked, the computation of $\gamma_{\hat{A}} = a + xe$ and the exponentiation of $Y^{\gamma_{\hat{A}}}$ are performed in the tamper-resistant device. In this way, the VOAKE protocol achieves stronger security in the PACK model as well as the best performance from the aspect of efficiency.
NAXOS Technique. To achieve the PACK security, the basis idea of our protocol is to apply the well-known NAXOS technique in the computation of the exponent of the ephemeral public key, which is recently criticized by some papers [5,21]. In our system setting, we give a practical interpretation why the method can be reasonably applied. In the tamper-resistant device, the ephemeral private key (x' or y') is chosen randomly and the exponent of the ephemeral public key (x or y) is computed by hashing the long-term private key and the ephemeral private key (*e.g.*, $x = \mathsf{H}_1(x', a)$). Normally, the ephemeral private key x' (or y') can be compromised by the Ephemeral Key Reveal holding the freshness of the session. Unlike the NAXOS protocol, which requires the exponent not to be leaked, the

exponent x (or y) can be revealed by the Session-State Reveal which will breach the freshness of the session, since the exponent is used in the host machine. As clarified in the security model, we specifically characterize the realistic attacks that can capture these values. The essential point is when the exponent of the ephemeral public key and some related value B^x (A^y) can be leaked, we must deliberately design the protocol to thwart the attacks amounted to the NAXOS protocol [5], concerning the efficiency at the same time. Concretely, two goals should be attained: (i) Even though the ephemeral private key is leaked out by Ephemeral Key Reveal the adversary can not compromise the secret exponent x (or y) without issuing Long-Term Key Reveal or Session-State Reveal; (ii) The leakages of the secret exponent x (or y) by the Session-State Reveal should not compromise other non-matching sessions.

Minor Modifications. We also mention some minor differences between our construction and the OAKE protocol as follows. One of the minor modifications is that the protocol transcripts (e.g. $\hat{A}$, $\hat{B}$, X and Y) are integrated into the key derivation together with σ to simplify the security proof. With the help of additional protocol transcripts, the simulator can distinguish which session the σ belongs to, thus the advantage of querying H_3 with correct master secret σ by adversary can be effectively used to solve the GDH problem (see Appendix A). Another modification is that the long-term public key is removed from the protocol messages. In the description of OAKE [10], the values B^x and A^y can be computed offline before the execution of the protocol in party $\hat{A}$ and $\hat{B}$ respectively, hence requiring to send the certificate containing the public key explicitly may invalidate the precomputation. The certificates are handled by the higher level applications in our construction.

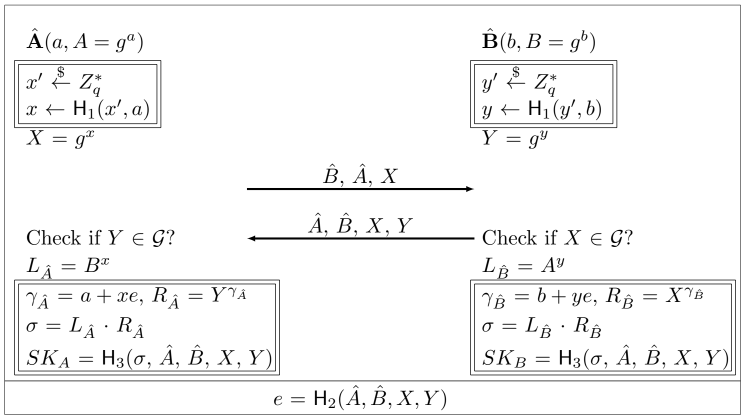
$\hat{\mathbf{A}}(a, A = g^a)$
$\hat{\mathbf{B}}(b, B = g^b)$
$x' \xleftarrow{\$} Z_q^*$
$x \leftarrow \mathsf{H}_1(x', a)$
$X = g^x$
$y' \xleftarrow{\$} Z_q^*$
$y \leftarrow \mathsf{H}_1(y', b)$
$Y = g^y$
$\hat{B}, \hat{A}, X$
$\hat{A}, \hat{B}, X, Y$
Check if $Y \in \mathcal{G}$?
$L_{\hat{A}} = B^x$
$\gamma_{\hat{A}} = a + xe$, $R_{\hat{A}} = Y^{\gamma_{\hat{A}}}$
$\sigma = L_{\hat{A}} \cdot R_{\hat{A}}$
$SK_A = \mathsf{H}_3(\sigma, \hat{A}, \hat{B}, X, Y)$
Check if $X \in \mathcal{G}$?
$L_{\hat{B}} = A^y$
$\gamma_{\hat{B}} = b + ye$, $R_{\hat{B}} = X^{\gamma_{\hat{B}}}$
$\sigma = L_{\hat{B}} \cdot R_{\hat{B}}$
$SK_B = \mathsf{H}_3(\sigma, \hat{A}, \hat{B}, X, Y)$
$e = \mathsf{H}_2(\hat{A}, \hat{B}, X, Y)$

Fig. 2. The description of the VOAKE protocol.

3.2 Protocol Description

Notation. Let $\mathcal{G}'$ be a finite Abelian group of order N, $\mathcal{G}$ be a cyclic subgroup of prime order q in $\mathcal{G}'$. Denote by g a generator of $\mathcal{G}$, by $1_{\mathcal{G}}$ the identity element, by $\mathcal{G} \setminus \{1_{\mathcal{G}}\}$ the set of elements of $\mathcal{G}$ except $1_{\mathcal{G}}$. In this work, we use multiplicative notation for the group operation in $\mathcal{G}'$. The computational Diffie-Hellman (CDH) assumption says that given $X = g^x$, $Y = g^y$, where x, $y \leftarrow \mathbb{Z}_q^*$, no efficient CDH-solver algorithm can compute CDH(X, Y) with non-negligible probability. The gap Diffie-Hellman (GDH) assumption [22] roughly says that the CDH assumption holds even if the CDH solver is equipped with a decisional Diffie-Hellman (DDH) oracle for the group $\mathcal{G}$ and the generator g, where on arbitrary input $(U, V, W) \in \mathcal{G}^3$ the DDH oracle outputs 1 if and only if $W =$ CDH(U, V).
Public Parameters. Let H_1: $\{0,1\}^\lambda \times \mathbb{Z}_q^* \rightarrow \mathbb{Z}_q^*$, H_2: $\{0,1\}^* \rightarrow \mathbb{Z}_q^*$, $\mathsf{H}_3 : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ be the functions modeled as random oracles.
Private and Public Keys. Party $\hat{U}$ randomly selects $u \in \mathbb{Z}_q^*$, and computes $U = g^u$. Party $\hat{U}$'s long-term private and public key are (u, U).
Key Exchange. Party $\hat{A}$ with private and public keys (a, A) acts as the initiator, and party $\hat{B}$ with private and public keys (b, B) acts as the responder, they perform the following two-pass key exchange protocol. The outline is presented in Fig. 2, the computations in the box of double-line are done in the tamper-resistant device, the values stored in the host machine's memory can be used freely in the tamper-resistant device, but not vice versa.

1. Upon activation $(\hat{A}, \hat{B})$, party $\hat{A}$ (the initiator) performs the steps:
   (a) Select an ephemeral private key $x' \in_R \{0,1\}^\lambda$ and compute $x = \mathsf{H}_1(x', a)$ in the tamper-resistant device, and then send x to the host machine's memory.
   (b) Compute the ephemeral public key $X = g^x$ in the host machine.
   (c) Send $(\hat{B}, \hat{A}, X)$ to party $\hat{B}$.
2. Upon receive $(\hat{B}, \hat{A}, X)$, party $\hat{B}$ (the responder) performs the steps:
   (a) Verify that $X \in \mathcal{G}$.
   (b) Select an ephemeral private key $y' \in_R \{0,1\}^\lambda$ and compute $y = \mathsf{H}_1(y', b)$ in the tamper-resistant device, and then send y to the host machine's memory.
   (c) Compute the ephemeral public key $Y = g^y$ in the host machine.
   (d) Compute $e = \mathsf{H}_2(\hat{A}, \hat{B}, X, Y)$ in the host machine.
   (e) Compute $L_{\hat{B}} = A^y$ in the host machine; compute $\gamma_{\hat{B}} = b + ye$ and $R_{\hat{B}} = X^{\gamma_{\hat{B}}}$ in the tamper-resistant device.
   (f) Compute $\sigma = L_{\hat{B}} \cdot R_{\hat{B}}$ and $SK_{\hat{B}} = \mathsf{H}_3(\sigma, \hat{A}, \hat{B}, X, Y)$ in the tamper-resistant device.
   (g) Send $(\hat{A}, \hat{B}, X, Y)$ to party $\hat{A}$.
3. Upon receive $(\hat{A}, \hat{B}, X, Y)$, party $\hat{A}$ performs the steps:
   (a) Verify that $Y \in \mathcal{G}$.
   (b) Compute $e = \mathsf{H}_2(\hat{A}, \hat{B}, X, Y)$ in the host machine.
   (c) Compute $L_{\hat{A}} = B^x$ in the host machine; compute $\gamma_{\hat{A}} = a + xe$ and $R_{\hat{A}} = Y^{\gamma_{\hat{A}}}$ in the tamper-resistant device.

(d) Compute $\sigma = L_{\hat{A}} \cdot R_{\hat{A}}$ and $SK_{\hat{A}} = \mathsf{H}_3(\sigma, \hat{A}, \hat{B}, X, Y)$ in the tamper-resistant device.

Ephemeral Key and Session State. The `EphemeralKey` of a session owned by the party $\hat{A}$ (or $\hat{B}$) is the randomly generated ephemeral private key x' (or y'). The `SessionState` of a session owned by the party $\hat{A}$ (or $\hat{B}$) is explicitly defined, including all the intermediate secret values computed or used in the host machine's memory, which are x (or y) and $L_{\hat{A}}$ (or $L_{\hat{B}}$), and the session key $SK_{\hat{A}}$ (or $SK_{\hat{B}}$). All the session state above could be erased after the session is completed, but can be obtained by issuing Session-State Reveal, as all the values above appear in the host machine's memory.

3.3 Security Proof for VOAKE

This section presents a formal security argument for the two-pass VOAKE.

Theorem 1. *If* H_1, H_2 *and* H_3 *are modeled as random oracles, and* $\mathcal{G}$ *is a group where the* GDH *assumption holds, then the advantage of any PPT adversary* $\mathcal{A}$ *to attack the* `AKE` *security of* VOAKE *in the* PACK *model is negligible.*

Proof (Sketch). We outline the proof as follows. Let $\mathcal{A}$ be any AKE adversary against VOAKE. We start by observing that since the session key of the test session is computed as $SK = \mathsf{H}_3(\delta)$ for some 5-tuple δ, the adversary has only two ways to distinguish SK from a random string:

- *Key Replication Attack.* In this attack, the adversary $\mathcal{A}$ succeeds in forcing the establishment of another session that has the same session key as the test session. Since the session key in VOAKE is computed via a random oracle, the probability that two sessions derive the same session key without using the same session string input to the random oracle is negligible.
- *Forging Attack.* The attack is valid if at some point the adversary $\mathcal{A}$ queries H_3 on the same 5-tuple δ.

To bound the advantage of forging attack, we define the following events and their complementary events:

- Win: The adversary $\mathcal{A}$ wins the security experiment described in Section 2 with probability $\frac{1}{2} + p(\lambda)$, where $p(\lambda)$ is non-negligible;
- AskH3: The adversary $\mathcal{A}$ asks random oracle H_3 with same 5-tuple δ of the test session;
- QH1: There exists a party $\hat{A}$, the adversary $\mathcal{A}$ queries H_1 with $(*, a)$ before issuing Long-Term Key Reveal$(\hat{A})$;
- Mat: The test session has matching session.

The complementary events are denoted as $\overline{\text{Win}}$, $\overline{\text{AskH3}}$, $\overline{\text{QH1}}$ and $\overline{\text{Mat}}$ respectively. The probability bounding process starts from the event Win. Because the adversary $\mathcal{A}$ has no advantage to win if the event AskH3 does not occur. Therefore, we concentrate on the event $\text{Win} \wedge \text{AskH3}$. In this case, if the event

QH1 happens, then we can construct a simulator to solve GDH problem. Furthermore, it is left to bound the probability that the events $\textsc{Win} \wedge \textsc{AskH3} \wedge \overline{\text{QH1}}$ occurs. This situation is further separated into two events according to the event $\textsc{Mat}$ happens or not. Both of the two cases occurs with negligible probability, which are also bounded by GDH assumption and the proof of the latter case relies on Forking Lemma. Detailed proofs of each of the above claims can be found in Appendix A.

4 Comparison and Concluding Remark

4.1 Models Comparison

We unify the CK model and the eCK model into the PACK model which is *practically* stronger than the previous models. Because the PACK model defines the system environment that the protocol will be implemented, which is not specified by the previous models, and some definitions of the previous models are not precisely clear, it is hard to formally establish a relationship of strength between previous models and ours.

The closest related work to ours is the seCK model proposed by Sarr *et al.* [4], we highlight some differences between the two models.

Environment. We both assume the existence of a tamper-resistant device in the system environment. In the seCK model, two implementation modes are set up, and two sets of adversarial queries are defined corresponding to the two modes. While in the PACK model, there is only one implementation mode, and only one set of adversarial queries are defined accordingly.

Reveal queries. In the PACK model, Session-State Reveal and Ephemeral Key Reveal can be issued to one same session simultaneously, which can not be achieved by seCK. In the seCK model, the adversaries can only ask Inter-Result Reveal (which has roughly the same functionality as Session-State Reveal) in mode 2 and Ephemeral Key Reveal in mode 1. That means the adversary can only reveal one kind of information (either intermediate values or ephemeral keys) for every session, therefore, it means the PACK model is stronger than the seCK model.

There are some minor differences between the reveal queries. In the seCK model, the Ephemeral Key Reveal models leakages on ephemeral Diffie-Hellman exponents; in the PACK model, the Ephemeral Key Reveal models leakages on bad randomness in general. The Inter-Result Reveal models the leakages that may occur on intermediate results in computing session keys; the Session-State Reveal of our model returns all the secret values (may include the session keys) in the host machine's memory.

4.2 Protocols Comparison

The proposed AKE protocol, VOAKE, satisfies a stronger security notion defined in the PACK model without sacrificing much efficiency.

Table 1 provides a comparison between VOAKE and some well accepted AKE protocols in the random oracle model. "#Exp-Online" and "#Exp-Total" denote the numbers of exponentiations needed to compute the input to the key derivation function H_3 with and without pre-offline computation (which can be done without incoming messages) respectively, as all the other operations are the same for all these protocols. "e" and "se" denote one modular and one simultaneous exponentiation respectively. "e_m" and "e_t" denote one modular exponentiation executed in the host machine and tamper-resistant device respectively. "#Hash" denotes the number of hash computation. Without loss of generality, we consider the session owned by party $\hat{A}$. The item of "Exposable Secret Value" refers to the secret values that can be exposed. For consistence, we denote the ephemeral private key as x', and the output of hashing long-term and ephemeral private keys as x (if any). The comparison focuses on the security (security model used, cryptographic assumption and exposable secret values) and efficiency (numbers of exponentiations in total, hash computation per session and the online computation efficiency).

Protocol	#Exp-Online	#Exp-Total	#Hash	Pre-Offline Computation	Model	Assumptions	Exposable Secret Value
HMQV	$1se$	$1se$	3	None	CK-HMQV	GDH+KEA1	(x')
NAXOS	$2e$	$3e$	2	(B^x)	eCK	GDH	(x')
CMQV	$1se$	$1se$	4	None	eCK	GDH	(x')
OAKE	$1e$	$1se$	2	$(B^{x'})$	CK-HMQV	GDL+SJKEA, or GDH	$(x',B^{x'})$
T-OAKE	$1e$	$1se$	2	$(B^{a+x'})$	CK-HMQV	GDL+SJKEA, or GDH	(x')
VOAKE	$1e_t$	$1e_t+1e_m$	3	(B^x)	PACK	GDH	(x',x,B^x)

Table 1. Comparison among VOAKE and some well-known efficient protocols.

Precisely, VOAKE is proven secure in the PACK model, while OAKE and T-OAKE are vulnerable to attacks in eCK, thus are PACK insecure. Moreover, NAXOS and CMQV are only provably secure in the eCK model, where only the ephemeral key (e.g. randomness) can be exposed. On the contrary, VOAKE remains secure even if more session states are leaked out (e.g. x and B^x *etc.*). In the security reduction, VOAKE only needs the standard GDH assumption, while HMQV requires both GDH and KEA assumptions.

The VOAKE protocol attains high efficiency as the OAKE protocol. In total, we perform one online modular exponentiation in the tamper-resistant device and one offline modular exponentiation in the host machine. In terms of online efficiency, B^x in VOAKE can be computed offline before the execution, then only one modular exponentiation is performed online, which is more efficient than one simultaneous exponentiation [8] done in the HMQV and CMQV protocol, namely, the VOAKE protocol attains the optimal online efficiency. We perform one modular exponentiation in the tamper-resistant device because some specific kind of information (e.g. $R_{\hat{A}}$) must be kept unexposed and thereby to achieve a stronger security. We remark, to achieve PACK security one modular exponentiation in the tamper-resistant device is almost inevitable.

Concluding Remark. It is hard to make a precise comparison between the protocols under different models, in the future work, we will reexamine the security and the efficiency of some well-known protocols in the PACK model. Another urgent work is to design a provably PACK secure protocol in the standard model with high efficiency, to avoid the random oracle and the Forking Lemma.

References

1. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *CRYPTO*, pages 232–249, 1993.
2. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, pages 453–474. Springer-Verlag, 2001.
3. Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, JosephK. Liu, and Yi Mu, editors, *Provable Security*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg, 2007.
4. Augustin P. Sarr, Philippe Elbaz-Vincent, and Jean-Claude Bajard. A new security model for authenticated key agreement. In JuanA. Garay and Roberto Prisco, editors, *Security and Cryptography for Networks*, volume 6280 of *Lecture Notes in Computer Science*, pages 219–234. Springer Berlin Heidelberg, 2010.
5. Cas J. F. Cremers. Session-state reveal is stronger than ephemeral key reveal: Attacking the NAXOS key exchange protocol. In *Abdalla, M., Pointcheval, D., Fouque, P. A., Vergnaud, D. (Eds.) ACNS 2009. LNCS, Vol. 5536*, pages 20–33. Springer, Heidelberg, 2009.
6. Cas J. F. Cremers. Formally and practically relating the CK, CK-HMQV, and eCK security models for authenticated key exchange. *IACR Cryptology ePrint Archive*, 2009:253, 2009.
7. Berkant Ustaoglu. Comparing SessionStateReveal and EphemeralKeyReveal for Diffie-Hellman protocols. In Josef Pieprzyk and Fangguo Zhang, editors, *Provable Security*, volume 5848 of *Lecture Notes in Computer Science*, pages 183–197. Springer Berlin Heidelberg, 2009.
8. Berkant Ustaoglu. Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS. *Designs, Codes and Cryptography*, 46(3):329–342, March 2008.
9. Tatsuaki Okamoto. Authenticated key exchange and key encapsulation in the standard model. In *Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833*, pages 474–484. Springer, Heidelberg, 2007.
10. Andrew Chi-Chih Yao and Yunlei Zhao. OAKE: A new family of implicitly authenticated Diffie-Hellman protocols. In *ACM Conference on Computer and Communications Security*, 2013. to appear.
11. Hugo Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. In *Advances in Cryptology CRYPTO 2005*, volume 3621, pages 546–566, 2005.

12. Cas J. F. Cremers and Michèle Feltz. Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *Computer Security - ESORICS 2012*, volume 7459 of *Lecture Notes in Computer Science*, pages 734–751. Springer Berlin Heidelberg, 2012.
13. 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*, volume 6980 of *Lecture Notes in Computer Science*, pages 348–365. Springer, 2011.
14. Augustin P. Sarr, Philippe Elbaz-Vincent, and Jean-Claude Bajard. A secure and efficient authenticated Diffie-Hellman protocol. In *Proceedings of the 6th European conference on Public key infrastructures, services and applications*, EuroPKI'09, pages 83–98, Berlin, Heidelberg, 2010. Springer-Verlag.
15. Andrew Chi-Chih Yao and Yunlei Zhao. A new family of practical non-malleable protocols. *IACR Cryptology ePrint Archive*, pages 35–35, 2011.
16. Atsushi Fujioka, Koutarou Suzuki, Keita Xagawa, and Kazuki Yoneyama. Strongly secure authenticated key exchange from factoring, codes, and lattices. In *Public Key Cryptography*, pages 467–484, 2012.
17. Augustin P. Sarr and Philippe Elbaz-Vincent. A complementary analysis of the (s)YZ and DIKE protocols. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *Progress in Cryptology - AFRICACRYPT 2012*, volume 7374 of *Lecture Notes in Computer Science*, pages 203–220. Springer Berlin Heidelberg, 2012.
18. Mihir Bellare, Zvika Brakerski, Moni Naor, Thomas Ristenpart, Gil Segev, Hovav Shacham, and Scott Yilek. Hedged public-key encryption: How to protect against bad randomness. *IACR Cryptology ePrint Archive*, pages 220–220, 2012.
19. Olivier Benoît and Thomas Peyrin. Side-channel analysis of six SHA-3 candidates. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 140–157. Springer Berlin Heidelberg, 2010.
20. J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98, May 2009.
21. Daisuke Moriyama and Tatsuaki Okamoto. An eCK-secure authenticated key exchange protocol without random oracles. In Josef Pieprzyk and Fangguo Zhang, editors, *Provable Security*, volume 5848 of *Lecture Notes in Computer Science*, pages 154–167. Springer Berlin Heidelberg, 2009.
22. Tatsuaki Okamoto and David Pointcheval. The gap-problems: A new class of problems for the security of cryptographic schemes. In Kwangjo Kim, editor, *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 104–118. Springer Berlin Heidelberg, 2001.
23. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13:361–396, 2000.

A Proof of Theorem 1

The following convention will be used in the security argument. Let λ denotes the security parameter, whence $q = |\mathcal{G}| = \Theta(2^\lambda)$. Let $\mathcal{A}$ be a polynomially (in λ) bounded adversary in the security experiment. If the adversary $\mathcal{A}$ can win the security experiment with non-negligible probability, then we can construct a GDH solver $\mathcal{S}$ that succeeds with non-negligible probability in the standard way. Let $(U,\ V)$ be a random GDH instance. Assume that $\mathcal{A}$ operates in an environment that involves at most $n(\lambda)$ parties, $\mathcal{A}$ activates at most $s(\lambda)$ sessions within a party, and makes at most $h_1(\lambda)$, $h_2(\lambda)$ and $h_3(\lambda)$ queries to oracles H_1, H_2 and H_3, respectively; and terminates after time at most $t_{\mathcal{A}}$. Let ν: $\mathcal{G} \times \mathcal{G} \to \mathcal{G}$ be a random function known only to $\mathcal{S}$, such that $\nu(X,\ Y) = \nu(Y,\ X)$. The algorithm $\mathcal{S}$ will use ν to "represent" $\mathrm{CDH}(X,\ Y)$ in the situation where $\mathcal{S}$ may not know $\log_g(X)$ and $\log_g(Y)$. Except with negligible probability, $\mathcal{A}$ will not detect that $\nu(X,\ Y)$ is being used instead of $\mathrm{CDH}(X,\ Y)$. A naive way to instantiate ν is to choose a random element Z in $\mathcal{G}$ as the output of $\nu(X,\ Y)$.

We start by observing that since the session key of the test session is computed as $SK = \mathsf{H}_3(\delta)$ for some 5-tuple δ, the adversary can only distinguish SK from a random string by the key replication attack or the forging attack. Note that in the key replication attack, the adversary $\mathcal{A}$ succeeds in forcing the establishment of another session that has the same session key as the test session. Since the session key in VOAKE is computed via a random oracle, the probability that two sessions derive the same session key without using the same session string input to the random oracle is negligible. Therefore, to win the security experiment, $\mathcal{A}$ must perform a forging attack in which the event querying H_3 on the same δ of the test session happens. Let the test session be $sid^t = (*,\ \hat{A},\ \hat{B},\ X,\ Y)$ and let AskH3 denote the event that $\mathcal{A}$ queries H_3 with $\delta = (\sigma,\ \hat{A},\ \hat{B},\ X,\ Y)$, where $\sigma = \mathrm{CDH}(X,\ B) \cdot \mathrm{CDH}(Y,\ AX^e)$. Let $\overline{\text{AskH3}}$ be the complementary event of AskH3 and sid^* be any other completed session owned by an honest party, such that sid^t and sid^* are non-matching. Since sid^t and sid^* are non-matching, the input to the key derivation function H_3 are different for sid^t and sid^*. And since H_3 is a random oracle it follows that $\mathcal{A}$ cannot obtain any information about the test session key from the session key of non-matching sessions. Hence $\Pr(\text{Win} \wedge \overline{\text{AskH3}}) \leq \frac{1}{2}$. The adversary $\mathcal{A}$ is said to be successful (denoted as event Win) with non-negligible probability if $\mathcal{A}$ wins in the security experiment described in Section 2 with probability $\frac{1}{2} + p(\lambda)$ and $p(\lambda)$ is non-negligible. In addition,

$$\Pr(\text{Win}) = \Pr(\text{Win} \wedge \overline{\text{AskH3}}) + \Pr(\text{Win} \wedge \text{AskH3}) \leq \frac{1}{2} + \Pr(\text{Win} \wedge \text{AskH3}),$$

whence $\Pr(\text{Win} \wedge \text{AskH3}) \geq p(\lambda)$. The event Win $\wedge$ AskH3 is denoted by Win*.

Subsequently, we define the following complementary events related to the queries of the random oracle H_1 (named as QH1):

QH1. This event means that there exists a party $\hat{B}$ and the adversary $\mathcal{A}$ queries H_1 with $(*,\ b)$ before issuing a $\mathsf{Long\text{-}Term\ Key\ Reveal}(\hat{B})$ query during its execution. Note that $\mathcal{A}$ does not necessarily make a $\mathsf{Long\text{-}Term\ Key\ Reveal}(\hat{B})$ query.

$\overline{\textbf{QH1}}$**.** This event means that for every party $\hat{B}$, if the adversary $\mathcal{A}$ queries H_1 with $(*,\ b)$ then $\mathcal{A}$ issued $\mathsf{Long\text{-}Term\ Key\ Reveal}(\hat{B})$ before the first $(*,\ b)$ query to H_1 during its execution.

If $\mathcal{A}$ succeeds with non-negligible probability, and hence $\Pr(\overline{\textsc{Win}^*}) \geq p(\lambda)$, it must be the case that either event QH1 $\wedge$ $\textsc{Win}^*$ or event $\overline{\text{QH1}}$ $\wedge$ $\textsc{Win}^*$ occurs with non-negligible probability. The former case happens with negligible probability according to Lemma 1 and the latter one is further subdivided into the following complementary events related to the test session has a matching session or not (we denote the event "the test session has a matching session owned by an honest party" by $\textsc{Mat}$ and its complementary event by $\overline{\textsc{Mat}}$):

(i) $\textsc{Tar} = (\overline{\text{QH1}} \wedge \textsc{Win}^* \wedge \textsc{Mat})$, and
(ii) $\overline{\textsc{Tar}} = (\overline{\text{QH1}} \wedge \textsc{Win}^* \wedge \overline{\textsc{Mat}})$.

Because $\textsc{Tar}$ and $\overline{\textsc{Tar}}$ are complementary, if event $\overline{\text{QH1}} \wedge \textsc{Win}^*$ occurs with non-negligible probability, then either $\textsc{Tar}$ or $\overline{\textsc{Tar}}$ occurs with non-negligible probability. Both events $\textsc{Tar}$ and $\overline{\textsc{Tar}}$ are bounded by negligible probability (refer to Lemma 2 and Lemma 3 respectively.).

Conclusion. Suppose that event $\textsc{Win}$ occurs. Combining Lemma 1, 2 and 3, the success probability of $\mathcal{S}$ is

$$\mathbf{Succ}_{\mathcal{G}}^{\text{GDH}}(\mathcal{S}) \geq max\{\frac{1}{n(\lambda)}p_{\text{QH1}}(\lambda), \frac{2}{(n(\lambda)s(\lambda))^2}p_{\textsc{Tar}}(\lambda), \frac{1}{n(\lambda)^2 s(\lambda)}\text{O}(\frac{1}{h_2(\lambda)})p_{\overline{\textsc{Tar}}}(\lambda)\},$$

which is non-negligible in λ.

If $\mathcal{A}$ is polynomially bounded, then there is a PPT algorithm $\mathcal{S}$ that succeeds in solving the GDH problem in $\mathcal{G}$ with non-negligible probability, contradicting the GDH assumption in $\mathcal{G}$. This concludes the proof of Theorem 1. □

Lemma 1. *If the event* QH1 $\wedge$ $\textsc{Win}^*$ *occurs with probability* p_{QH1}*, we can construct a GDH solver* $\mathcal{S}$ *that can solve the* GDH *problem with probability*

$$\mathbf{Succ}_{\mathcal{G}}^{\text{GDH}}(\mathcal{S}) \geq \frac{1}{n(\lambda)}p_{\text{QH1}}.$$

Lemma 2. *If the event* $\textsc{Tar}$ *occurs with probability* $p_{\textsc{Tar}}$*, we can construct a GDH solver* $\mathcal{S}$ *that can solve the* GDH *problem with probability*

$$\mathbf{Succ}_{\mathcal{G}}^{\text{GDH}}(\mathcal{S}) \geq \frac{2}{(n(\lambda)s(\lambda))^2}p_{\textsc{Tar}}.$$

Lemma 3. *If the event* $\overline{\textsc{Tar}}$ *occurs with probability* $p_{\overline{\textsc{Tar}}}$*, we can construct a GDH solver* $\mathcal{S}$ *that can solve the* GDH *problem with probability*

$$\mathbf{Succ}_{\mathcal{G}}^{\text{GDH}}(\mathcal{S}) \geq \frac{1}{n(\lambda)^2 s(\lambda)}\text{O}(\frac{1}{h_2(\lambda)})p_{\overline{\textsc{Tar}}}$$

A.1 Proof of Lemma 1

Simulation. Suppose that event QH1 $\wedge$ WIN* occurs with non-negligible probability. In this case the GDH solver $\mathcal{S}$ modifies the experiment by selecting a random party $\hat{A}$ and setting $A = V$. Note that $\mathcal{S}$ doesn't know the private key corresponding to this public key and thus it cannot properly simulate the sessions executed by $\hat{A}$. $\mathcal{S}$ handles the sessions executed by $\hat{A}$ as follows (without loss of generality, assume that $\hat{A}$ is the initiator). $\mathcal{S}$ randomly selects x', picks x at random from $\mathbb{Z}_q^*$ and sets $X = g^x$ instead of $g^{\mathsf{H}_1(x', log_g(V))}$. $\mathcal{S}$ computes $L_{\hat{A}} = X^b$ with the knowledge of b. The master secret is derived by setting $\sigma = L_{\hat{A}} \times \nu(Y, AX^e)$ and the session key SK is computed as $\mathsf{H}_3(\sigma,\ \hat{A},\ \hat{B},\ X,\ Y)$. Note that $\mathcal{S}$ can respond Session Key Reveal and Ephemeral Key Reveal with SK and x', and the Session-State Reveal can be responded by returning x, $L_{\hat{A}}$ and SK. If the event QH1 indeed happens, Long-Term Key Reveal queries need not be simulated on party $\hat{A}$.

Note that if $\hat{B}$ is a fictitious party, $\mathcal{A}$ can compute the session key on its own, reveal SK and detect that it is fake. To address this issue, $\mathcal{S}$ watches $\mathcal{A}$'s random oracle queries and if $\mathcal{A}$ queries $(\sigma,\ \hat{A},\ \hat{B},\ X,\ Y)$ to H_3 (for some $\sigma \in \mathcal{G}$), $\mathcal{S}$ checks if DDH$(Y,\ AX^e,\ \sigma X^{-b})$ equals 1, replies with the key SK. Similarly, on the computation of SK, $\mathcal{S}$ checks if SK should equal any previous response from the random oracle.

For every $\mathcal{A}$'s queries $(*,\ \alpha)$ to H_1, $\mathcal{S}$ checks if the equation $g^\alpha = V$ holds, in which case $\mathcal{S}$ stops $\mathcal{A}$ and is successful by outputting CDH$(U,\ V) = U^a$. We observe that in this simulation, $\mathcal{A}$ cannot detect that it is in the simulated AKE experiment unless it queries $(x',\ a)$ to H_1 (this way, $\mathcal{A}$ will find out that X was not computed correctly).

Analysis of event QH1 $\wedge$ WIN*. $\mathcal{S}$'s simulation of $\mathcal{A}$'s environment is perfect except with negligible probability. $\mathcal{S}$ randomly chooses an honest party $\hat{A}$ and assigns its public key to be V. With probability at least $\frac{1}{n(\lambda)}$, the event QH1 occurs at the party $\hat{A}$. If event QH1 $\wedge$ WIN* occurs with probability p_{QH1}, then the success probability of $\mathcal{S}$ is bounded by

$$\mathbf{Succ}_{\mathcal{G}}^{\text{GDH}}(\mathcal{S}) \geq \frac{1}{n(\lambda)} p_{\text{QH1}}. \qquad (1)$$

A.2 Proof of Lemma 2

Simulation. Suppose that event TAR occurs with non-negligible probability. In this case assume that $\mathcal{A}$ always selects a test session for which the matching session exists. Then the GDH solver $\mathcal{S}$ modifies the experiment as follows. $\mathcal{S}$ randomly selects two sessions executed by some honest parties $\hat{A}$ and $\hat{B}$ at random and continues only if they are matching. Denote by X and Y the ephemeral public keys sent by the respective parties in these matching sessions. When either of these sessions is activated, $\mathcal{S}$ does not follow the protocol. Instead, $\mathcal{S}$ generates x' and y' normally but sets $X = U$ (in place of $g^{\mathsf{H}_1}(x', a)$) and $Y =$

V (in place of $g^{\mathsf{H}_1}(y', b)$). Note that $\mathcal{S}$ can respond Long-Term Key Reveal query on the selected parties and Ephemeral Key Reveal query on the selected sessions with a (or b) and x' (or y'). If one of the sessions selected is indeed the test session, whose freshness should not be breached, both Session-State Reveal and Session Key Reveal queries need not be simulated on the selected sessions.

If at some point, the event AskH3 happens, the adversary queries same 5-tuple $\delta = (\sigma,\, \hat{A},\, \hat{B},\, X,\, Y)$ of test session to the random oracle H_3, in which $\mathrm{DDH}(X^e,\, Y,\, \sigma X^{-b}Y^{-a}) = 1$, then $\mathcal{S}$ aborts $\mathcal{A}$ and is successful by outputting $\mathrm{CDH}(U,\, V) = (\sigma X^{-b}Y^{-a})^{-e}$.

We observe that in this case, the only way that $\mathcal{A}$ can distinguish this simulated AKE experiment from a true AKE experiment is that if $\mathcal{A}$ queries (x', a) or (y', b) to H_1 (this way, $\mathcal{A}$ will find out that X and Y were not computed correctly). Since x' is used only in the test session, $\mathcal{A}$ must obtain it via an Ephemeral Key Reveal query before making an H_1 query that includes x'. Similarly, $\mathcal{A}$ must obtain y' from the matching session via an Ephemeral Key Reveal query before making an H_1 query that includes y'. If event $\overline{\mathrm{QH1}}$ happens, the adversary first issues a Long-Term Key Reveal query to a party before making an H_1 query that includes that party's static private key. If the session selected is indeed the test session, whose freshness should not be breached, $\mathcal{A}$ can query for at most one value in each of the pairs (x', a) and (y', b). Thus, $\mathcal{A}$ has no advantage to distinguish simulated AKE experiment from a true AKE experiment.

Analysis of event Tar. $\mathcal{S}$ simulates $\mathcal{A}$'s environment perfectly except with negligible probability. The probability that $\mathcal{A}$ selects sid^U and sid^V as the test session and its matching is at least $\frac{2}{(n(\lambda)s(\lambda))^2}$. Hence if event Tar occurs with probability p_{TAR}, then the success probability of $\mathcal{S}$ is bounded by

$$\mathbf{Succ}_{\mathcal{G}}^{\mathrm{GDH}}(\mathcal{S}) \geq \frac{2}{(n(\lambda)s(\lambda))^2} p_{\mathrm{TAR}}. \tag{2}$$

A.3 Proof of Lemma 3

Simulation. Suppose that event $\overline{\mathrm{TAR}}$ occurs with non-negligible probability. Assume that $\mathcal{A}$ always selects a test session such that the matching session does not exist. In this case the GDH solver $\mathcal{S}$ modifies the experiment as follows. $\mathcal{S}$ selects a random party $\hat{A}$ and set $A = V$, and then $\mathcal{S}$ simulates the sessions executed by $\hat{A}$ and the queries to H_3 as the simulation in event $\mathrm{WIN}^* \wedge \mathrm{QH1}$.

Besides, $\mathcal{S}$ also randomly selects an session owned by $\hat{B}$, in which $\hat{A}$ is the peer. When the session is activated, $\mathcal{S}$ follows the protocol only partially: $\mathcal{S}$ generates y' normally but sets $Y = U$ (in place of $g^{\mathsf{H}(y', b)}$). Note that $\mathcal{S}$ can respond Long-Term Key Reveal query on party $\hat{B}$ and Ephemeral Key Reveal query on the session with b and y'. If the session selected is indeed the test session, whose freshness should not be breached, the Session-State Reveal and Session Key Reveal queries on the session as well as the Long-Term Key Reveal query on party $\hat{A}$ need not be simulated.

Without loss of generality let X denote the incoming ephemeral public key selected by $\mathcal{A}$ for the test session sid^t. If at some point, the event ASKH3 happens, $\mathcal{A}$ queries H_3 with same 5-tuple $\delta = (\sigma, \hat{A}, \hat{B}, X, Y)$ of test session where $A = V$ and $Y = U$ and $\mathrm{DDH}(Y, AX^e, \sigma X^{-b}) = 1$, in which case $\mathcal{S}$ computes

$$\Omega = \sigma X^{-b} = g^{uv+xue}.$$

Without the knowledge of $x = \log_g(X)$, $\mathcal{S}$ is unable to compute $\mathrm{CDH}(U, V)$. Following the Forking Lemma [23] approach, $\mathcal{S}$ runs $\mathcal{A}$ on the same input and the same coin flips but with carefully modified answers to the H_2 queries. Note that $\mathcal{A}$ must have queried H_2 with $(\hat{A}, \hat{B}, X, Y)$ in its first run, because otherwise $\mathcal{A}$ would be unable to compute σ except with negligible probability. For the second run of $\mathcal{A}$, $\mathcal{S}$ responds to $\mathsf{H}_2(\hat{A}, \hat{B}, X, Y)$ with a new value $e' \neq e$ selected uniformly at random. If $\mathcal{A}$ succeeds in the second run, $\mathcal{S}$ computes

$$\Omega' = \sigma' X^{-b} = g^{uv+xue'}$$

and thereafter obtains

$$\mathrm{CDH}(U, V) = \left(\frac{\Omega^{\frac{e'}{e}}}{\Omega'}\right)^{(\frac{e'}{e}-1)^{-1}}.$$

We observe that in this case, $\mathcal{A}$ cannot detect that it is in the simulated AKE experiment unless it either issues a $\mathsf{Long\text{-}Term\ Key\ Reveal}(\hat{A})$ query or queries (y', b) to H_1 (this way, $\mathcal{A}$ will find out that Y was not computed correctly). Since x' is used only in the test session, $\mathcal{A}$ must obtain it via an $\mathsf{Ephemeral\ Key\ Reveal}$ query before making an H_1 query that includes x'. If event $\overline{\mathrm{QH1}}$ happens, the adversary first issues a $\mathsf{Long\text{-}Term\ Key\ Reveal}$ query to a party before making an H_1 query that includes that party's static private key. If the session selected is indeed the test session, whose freshness should not be breached, $\mathcal{A}$ is not allow to issue $\mathsf{Long\text{-}Term\ Key\ Reveal}(\hat{A})$ and can query for at most one value in the pair (y', b). Thus, $\mathcal{A}$ can not distinguish between simulated AKE experiment and a true AKE experiment.

Analysis of event $\overline{\mathrm{TAR}}$. The simulation of $\mathcal{A}$'s environment is perfect except with negligible probability. The probability that the test session has peer $\hat{A}$ (whose public key is V) and outgoing ephemeral public key U is at least $\frac{1}{n(\lambda)^2 s(\lambda)}$. Hence if event $\overline{\mathrm{TAR}}$ occurs with probability $p_{\overline{\mathrm{TAR}}}$, then the success probability of $\mathcal{S}$, excluding negligible terms, is

$$\mathbf{Succ}_{\mathcal{G}}^{\mathrm{GDH}}(\mathcal{S}) \geq \frac{1}{n(\lambda)^2 s(\lambda)} \mathrm{O}\left(\frac{1}{h_2(\lambda)}\right) p_{\overline{\mathrm{TAR}}} \qquad (3)$$

where $\mathrm{O}(\frac{1}{h_2(\lambda)})$ comes from the use of the Forking Lemma [23].

B A Trivial Attack Against OAKE in the eCK Model

For the benefit of the reader we present a trivial attack against OAKE in the eCK model demonstrating that the OAKE protocol is eCK-insecure.

1. The party $\hat{A}$ sends ($\hat{A}$, A, X) to the party $\hat{B}$.
2. The party $\hat{B}$ responds with ($\hat{B}$, B, Y) to the party $\hat{A}$.
3. The adversary $\mathcal{A}$ issues $\mathsf{Ephemeral\ Key\ Reveal}$ to the session owned by the party $\hat{A}$ (denoted as $sid_{\hat{A}}$) and another one owned by the party $\hat{B}$ (denoted as $sid_{\hat{B}}$) and learns x and y.
4. The adversary $\mathcal{A}$ issues $\mathsf{Test}(sid_{\hat{A}})$.

According to the definition of freshness in the eCK model, the session associated with session identifier $sid_{\hat{A}}$ is fresh. Note that the adversary $\mathcal{A}$ can compute the session key of the session owned by the party $\hat{A}$ as $K_{\hat{A}} = H_K(B^x(AX^e)^y)$, where $e = h(\hat{A}, A, \hat{B}, B, X, Y)$. Thus, the adversary $\mathcal{A}$ can perfectly distinguish the real session key from the random one and break the $\mathtt{SK}$ security of OAKE protocol in the eCK model.