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

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

[a]*School of Computer, South China Normal University, Guangzhou 510631, China*
[b]*State Key Laboratory of Information Security (Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093)*
[c]*Faculty of Mathematics*
Horst Görtz Institute for IT-Security
Ruhr University Bochum, Germany

Abstract

The most widely accepted models in the security proofs of Authenticated Key Exchange protocols are the Canetti-Krawczyk and extended Canetti-Krawczyk models that admit different adversarial queries with ambiguities and incomparable strength. It is desirable to incorporate specific and powerful adversarial queries into a single unified security model and establish a more practical-oriented security notion. Concerning the security of one-round implicitly authenticated Diffie-Hellman key exchange protocols, we present a unified security model that has many advantages over the previous ones. In the model, a system environment is set up, all of adversarial queries are practically interpreted and definitely characterized through physical environment, and some rigorous rules of secret leakage are also specified. To demonstrate usability of our model, a new protocol based on the OAKE protocol is proposed, which satisfies the presented strong security notion and attains high efficiency. The protocol is proven secure in random oracle model under gap Diffie-Hellman assumption.

Keywords: Authenticated Key Exchange, Provable Security, Security Model

[1]Email: weiqwen@gmail.com
[2]Corresponding author (Email: lbwang@scnu.edu.cn; Fax: +86 020 85215418; Phone: +86 020 85211352 ext. 405)
[3]Email: jiaxin.pan@rub.de

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) and extended Canetti-Krawczyk (eCK) models 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].

To objectively evaluate AKE protocols we need a security model with specific adversarial capabilities. The subtle issues that arises between the CK and eCK models 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.

It is also desirable to incorporate many powerful adversarial queries into a single unified security model and establish a strong security notion. Many works analyze the differences between the CK and eCK models 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, to encompass these queries into a single framework is urgently needed.

In this work, we focus on the security of one-round implicitly authenticated Diffie-Hellman key exchange protocols, and aim to propose a more practical-oriented 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. 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 and eCK models 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 and eCK models 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 strong notion of security for AKE protocols. The proposed PACK model is more tightly related to the actual protocol implementation environment, compared with the CK and eCK models. All of the attacks defined in the CK and eCK models are captured in the PACK model, which include unknown key share attack (UKS), key compromise impersonation attack (KCI) and maximal exposure attack (MEX) *etc.*, as well as the security notions of weak perfect forward secrecy (wPFS) and perfect forward secrecy (PFS) *etc.*.

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, 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 strong 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 the CK and eCK models [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. The detailed comparison between the seCK model and the PACK model will be presented in section 4. As suggested in paper of Yoneyama *et al.* [13], it is hard 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] is proved to be 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 is strongly secure in the CK model with more allowed secrecy exposure and achieves high efficiency as HMQV protocol. Unfortunately, the OAKE protocol is eCK insecure, and its variant T-OAKE (also named srYZ) is also shown to be eCK insecure [17], which limits their implementation. Thus, it is still left as an urgent work to design a strongly secure AKE protocol. In this paper, we improve the OAKE protocol by applying NAXOS technique, the improved OAKE protocol is provably secure in the PACK model 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 and eCK models 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 continually outputs all the intermediate secret values (denoted as `InterResult` in Fig. 1) computed or used in the host machine's memory and outputs the session key (denoted as `SessionKey` in Fig. 1) if the named session is completed.

- 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). We note that the attacker described above can also obtain the intermediate values stored in the tamper-resistant device. The leakage of long-term private key, combined with the leakage of session state, implies the leakage of intermediate values in the tamper-resistant device. Thus we provide definition of Long-Term Key Reveal and do not provide an explicit definition of the leakage query on intermediate values stored in the tamper-resistant device.

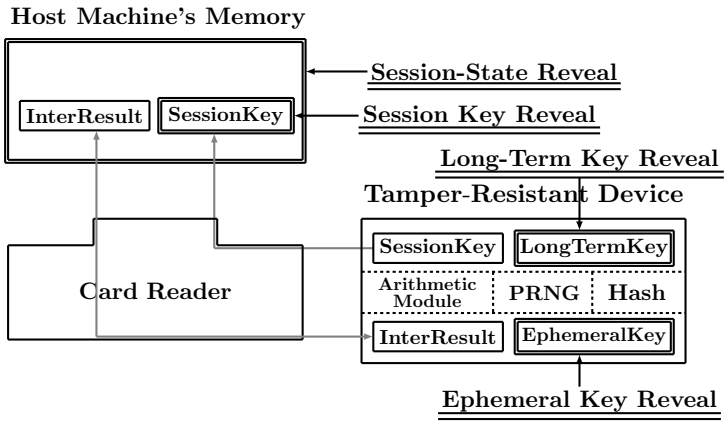

Figure 1: The system implementation environment and the adversarial queries.

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 more practical-oriented 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 continually 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. We note that the Session-State Reveal query is an important tool to analyze the security of the protocol. In the design of the protocol, in order to improve the efficiency, some of the computation processes are designed to be removed from temper resistant device to host machine. Then the presence of Session-State Reveal query is of great importance to leak the values that appear in the host machine's memory.

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 binding assurance between an identity and a public key is provided by a certification authority (CA), which is trusted by all parties. However we do not assume that the CA requires parties to prove possession of their long-term private keys.

Session. We suppose parties $\hat{P}_{i=1,\cdots,n}$ to be probabilistic polynomial time machines. A party 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. $\hat{P}_i$ is activated by a Send query with one of the following forms: (i) $\mathsf{Send}(\hat{P}_i, \hat{P}_j)$ or (ii) $\mathsf{Send}(\hat{P}_i, \hat{P}_j, out)$ where *out* is $\hat{P}_j$'s outgoing protocol message. If $\hat{P}_i$ is activated by $\mathsf{Send}(\hat{P}_i, \hat{P}_j)$ then $\hat{P}_i$ is the session *initiator* (denoted as $\mathcal{I}$) with peer $\hat{P}_j$, otherwise the session *responder* (denoted as $\mathcal{R}$)

with peer $\hat{P}_j$. A session identifier is a quadruple $(Role, \hat{P}_i, \hat{P}_j, out, in)$ where $\hat{P}_i$ is the *owner* of the session, $\hat{P}_j$ is the *peer*, *out* and *in* are respectively the concatenation of the messages $\hat{P}_i$ sends to $\hat{P}_j$, or believes to be from $\hat{P}_j$. Two sessions with identifiers $(Role, \hat{P}_i, \hat{P}_j, out, in)$ and $(Role', \hat{P}'_j, \hat{P}'_i, out', in')$ are said to be matching if $Role \neq Role'$, $\hat{P}_i = \hat{P}'_i$, $\hat{P}_j = \hat{P}'_j$, and at completion $out = in'$ and $in = out'$. A session cannot have (except with negligible probability) more than one matching session. Note that the session defined is general enough to cover different types of AKE protocols. Concretely, the PACK model can capture multi-pass protocols beyond one-round. In addition, we say a session is *completed* if and only if the session key of the session is computed and outputted. If explicit authentication is used, then a symbol "*reject*" is outputted if the authentication fails, and the session outputs "*accept*" if the authentication succeeds.

Adversary. Essentially, we define the adversary's capability of registering public key by using the same style as the eCK model [3]. The adversary can register arbitrary public key of its choice, including public keys equal to keys of some honest parties in the system, on behalf of adversary-controlled parties.

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.

- Send$(Message)$. The message has one of the following forms: $(\hat{P}_i, \hat{P}_j)$, $(\hat{P}_i, \hat{P}_j, out)$ or $(\hat{P}_i, \hat{P}_j, in, out)$. The adversary is given the session's response according to the protocol and the variables *in*, *out* are initialized and updated (by concatenation) accordingly.

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` contains `InterResult`, and contains `SessionKey` if the session is completed.

- Ephemeral Key Reveal(s): The adversary obtains the `EphemeralKey` associated with session s.
- Session-State Reveal(s): The adversary $\mathcal{A}$ obtains the `SessionState` of the owner of session s.

- Session Key Reveal(s): The adversary obtains the `SessionKey` in a completed session s.
- 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 the PACK security definition, we need a new notion of freshness to incorporate both Ephemeral Key Reveal and Session-State Reveal.

Definition 1. *(Freshness of PACK) Let s be a completed session owned by an honest party $\hat{A}$ with an honest peer party $\hat{B}$. Let $\overline{s}$ be the matching session of s. We say session sid is fresh if none of the following conditions holds:*

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

Freshness of PACK-PFS. Informally, an AKE protocol is said to be secure with PFS if disclosure of the long-term secret key of a party does not compromise the security of session established by that party with an active involvement of the attacker. The PFS security notion is not captured in the original eCK model [3], however, the PACK model can be more general. We define the freshness of PACK-PFS to make the PACK model encompass the PFS notion captured in the CK model, and to characterize the security of explicit authenticated key exchange protocol.

The freshness definition of PFS in the PACK model benefits from the work of Cremers *et al.* [21]. In order to extend the PACK model to capture PFS, we require that after the test session is completed, the Long-Term Key Reveal on the test session's owner and peer should not breach its freshness even if it has no matching session. Thus, to define the PFS security notion in the PACK model, we need a new notion of freshness.

Definition 2. *(Freshness of PACK-PFS) Let s be a completed session owned by an honest party $\hat{A}$ with an honest peer party $\hat{B}$. Let $\overline{s}$ be the matching session of s. We say session s is fresh if none of the following conditions holds:*

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

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(s^t): In this query, s^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 s^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 s^t is still fresh and if the guess of $\mathcal{A}$ is correct, i.e., $b' = \mathrm{b}$. The advantage of the adversary in the experiment with the AKE protocol Φ is defined as

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

Definition 3. (**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 and and the freshness of session is defined according to the Freshness of PACK it holds:*

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

Definition 4. (**AKE Security with Perfect Forward Secrecy**) *A key exchange protocol* Φ *is called* AKE *secure with PFS if for all PPT adversaries* $\mathcal{A}$ *with the above capabilities running against* Φ *and the freshness of session is defined according to Freshness of PACK-PFS it holds:*

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

The proposed PACK model have many advantages over previous models. Detailed analysis is presented in section 4.

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 the strong 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 strong 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, 22]. 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 Modification. 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. $\hat{A}$, A, $\hat{B}$, 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

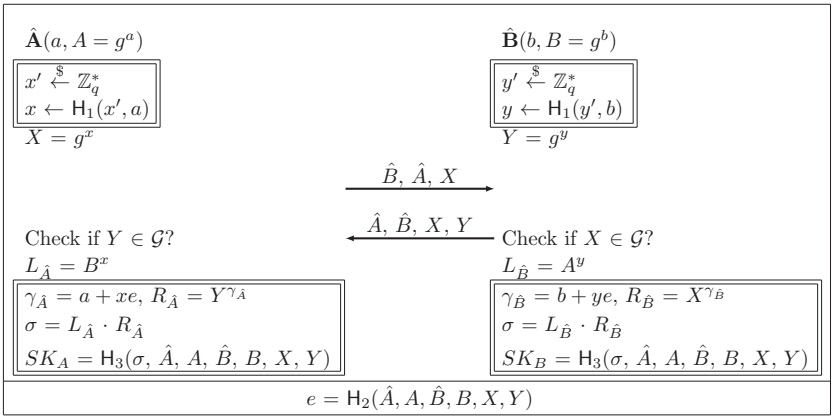

Figure 2: The description of the VOAKE protocol.

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 pre-computation. The certificates are exchanged by the higher level applications in our construction.

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 $\mathrm{CDH}(X, Y)$ with non-negligible probability. The gap Diffie-Hellman (GDH) assumption [23] 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 $W = \mathrm{CDH}(U, V)$.

Public Parameters. Let H_1: $\{0,1\}^\lambda \times \mathbb{Z}_q^* \to \mathbb{Z}_q^*$, H_2: $\{0,1\}^* \to \mathbb{Z}_q^*$, H_3 : $\{0,1\}^* \to \{0,1\}^\lambda$ 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 $e = \mathsf{H}_2(\hat{A}, A, \hat{B}, B, X, Y)$ 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 $SK_{\hat{B}} = \mathsf{H}_3(\sigma, \hat{A}, A, \hat{B}, B, X, Y)$ 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 $e = \mathsf{H}_2(\hat{A}, A, \hat{B}, B, X, Y)$ 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 $SK_{\hat{A}} = \mathsf{H}_3(\sigma, \hat{A}, A, \hat{B}, B, X, Y)$ 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 $SK_{\hat{B}}$). Note that, erasing all the session state can not prevent the adversary obtaining them by issuing Session-State Reveal, as all the values appear in the host machine's memory.

Intermediate Values in the Tamper-Resistant Device. Without loss of generality, we consider the session owned by party $\hat{A}$. If the intermediate values $\gamma_{\hat{A}}$ and $R_{\hat{A}}$ are computed in the host machine, then they will be leaked by Session-State Reveal and the protocol will suffer from attacks. For example, if the value $\gamma_{\hat{A}}$ is leaked, combining with the value x, the adversary can compute the long-term private key a. Moreover, the protocol can be trivially broken by the adversary if the value $R_{\hat{A}}$ is leaked (refer to Appendix B). Hence these values must be computed and stored in the tamper-resistant device and can not be outputted to host machine's memory. This also demonstrate that the presence of Session-State Reveal query is of great importance to analyze the security of the protocol.

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 7-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 7-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 7-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 Win $\wedge$ 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 Win $\wedge$ AskH3 $\wedge$ $\overline{\text{QH1}}$ occurs. This situation is further separated into two events according to the event 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. PACK is more practical-oriented than CK and eCK

We unify the CK and eCK models into the PACK model which is more practical-oriented than the previous ones. We establish the argument by showing that the PACK model provides richer adversarial queries and it captures more attacks than the CK and eCK models.

Models	PACK	CK	eCK
Queries	Session-State Reveal Long-Term Key Reveal Ephemeral Key Reveal	Session-State Reveal Corrupt	Long-Term Key Reveal Ephemeral Key Reveal
Attacks & Notions	PFS UKS, wPFS KCI, MEX	PFS UKS, wPFS	UKS, wPFS KCI, MEX

Table 1: Comparison among PACK, CK and eCK.

Table 1 provides a comparison between PACK and the CK and eCK models. First of all, the adversarial queries admitted in the PACK model are richer than the one defined in both the CK and eCK models. Note that the Corrupt query defined in the CK model can be simulated by Long-Term Key Reveal query in the PACK model. Secondly, all of the advanced attacks and the security properties defined in the CK and eCK models, including UKS, KCI, MEX, wPFS and PFS *etc.*, are well captured in the PACK model. Furthermore, the freshness of PACK as shown in Definition 1 can be reduced to the ones defined in the CK and eCK models.

The PACK model improve the previous queries as it provides a formal notion to make the design of protocol as well as its security more clear and unambiguous. In the PACK model, once the protocol description is determined, which values should be leaked by which queries and why are all clearly specified by the model without ambiguity. Namely, the designer decides how the protocol executes, and let the PACK model determines what values can be leaked. We emphasize this is a major improvement over previous models. The previous models do not clearly specify which values should be leaked by which qeuries or explain why, it is the designer who decides what and how values can be leaked.

Lack of notion to define exact leakage of queries will result in many discrepancies. For example, in the BCGNP protocol [24] and FSXY protocol [16], the decapsulated key is strategically excluded from session state. Because all computations of both protocols are executed in the host machine, all results appear in the host machine's memory. A malicious insider can reveal all the values in memory before the values are erased. Thus the decapsulated key should be leaked and the security of the aforementioned protocols are invalidated by a trivial attack presented in the paper [24].

Note that, the definition of matching session in the PACK model benefits from the eCK model and the CK-HMQV model [11], which is different from the one in the CK model and more self-contained. In the CK model, a session

identifier is assigned by a higher layer protocol, and the matching condition is defined at the onset of each session, which has been commented by Sarr et al in [4] and improved by Krawczyk in [11]. We claim that the matching condition in the CK model does not improve the adversarial capabilities, but makes more restrictions. Concretely, in the CK model, if a session is never completed and its freshness is breached, then its completed matching session is immediately un-fresh and can not be selected as test session. On the contrary, in the PACK model the incomplete sessions will not affect the freshness of any other sessions.

4.2. Comparison between PACK and seCK

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

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 more powerful 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.

4.3. Protocols Comparison

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

Table 2 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).

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

Table 2: 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 H-MQV 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 the security goal. We remark, to achieve PACK security one modular exponentiation in the tamper-resistant device is almost inevitable.

4.4. Concluding Remark

We conclude this work by proposing some suggestions for future investigations.

- In the VOAKE protocol, we carefully design the computation environment for the whole process of execution to achieve PACK secure. In this case, not all the intermediate values are revealed by Ephemeral Key Reveal and Session-State Reveal queries. A subtle issue is that whether we can design a provably secure protocol such that the adversary can learn all the intermediate values within a session by issuing these two queries. Such an protocol may be more efficient for implementation.

- In this work, we mainly focus on the security of the VOAKE protocol in the random oracle model. Another direction is to design a provably PACK secure protocol in the standard model with high efficiency.

- Over past few years, one-round AKE protocols with PFS have been proposed based on authenticated primitives, *e.g.*, signature and MAC *etc.*, which loss full-deniability. An urgent work is that can we design a provable AKE protocol with PFS as well as full-deniability based on weaker primitives, *e.g.*, proof of knowledge [25], in the PACK model.

References

[1] Bellare, M., Rogaway, P.: 'Entity authentication and key distribution', In Stinson, D.R. (Ed.): 'Proc. Int. Conf. Cryptology (CRYPTO 1993)' (Springer-Verlag, LNCS 773, 1993), pp 232-249.

[2] Canetti, R., Krawczyk, H.: 'Analysis of key-exchange protocols and their use for building secure channels', In Pfitzmann, B. (Ed.): 'Proc. Int. Conf. Theory and Application of Cryptographic Techniques (EUROCRYPT 2001)' (Springer-Verlag, LNCS 2045, 2001), pp 453–474.

[3] LaMacchia, B., Lauter, K., Mityagin, A.: 'Stronger security of authenticated key exchange', In Susilo, W., Liu, J. and Mu, Y. (Eds.): 'Proc. Int. Conf. Provable Security (ProvSec 2007)' (Springer-Verlag, LNCS 4784, 2007), pp 1–16.

[4] Sarr, A.P., Elbaz-Vincent, P., Bajard, J.: 'A new security model for authenticated key agreement', In Garay, J., Prisco, R. (Eds.): 'Proc. Int. Conf. Security and Cryptography for Networks (SCN 2010)' (Springer-Verlag, LNCS 6280, 2010), pp 219–234.

[5] Cremers, C.J.F.: '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.): 'Proc. Int. Conf. Applied Cryptography and Network Security (ACNS 2009)' (Springer-Verlag, LNCS 5536, 2009), pp 20–33.

[6] Cremers, C.J.F.: 'Formally and practically relating the CK, CK-HMQV, and eCK security models for authenticated key exchange', IACR Cryptology ePrint Archive, Report 2009/253, 2009.

[7] Ustaoglu, B.: 'Comparing SessionStateReveal and EphemeralKeyReveal for Diffie-Hellman protocols', In Pieprzyk, J. and Zhang, F. (Eds.): 'Proc. Int. Conf. Provable Security (ProvSec 2009)' (Springer-Verlag, LNCS 5848, 2009), pp 183–197.

[8] Ustaoglu, B.: 'Obtaining a secure and efficient key agreement protocol from (H)MQV and NAXOS', Des. Codes Cryptogr., 2008, **46**, (3), pp 329–342.

[9] Okamoto, T.: 'Authenticated key exchange and key encapsulation in the standard model', In Kurosawa, K. (ed.): 'Proc. Int. Conf. Theory and Application of Cryptology and Information Security (ASIACRYPT 2007)' (Springer-Verlag, LNCS 4833, 2007), pp 474–484.

[10] Yao, A.C., Zhao, Y.: 'OAKE: A new family of implicitly authenticated Diffie-Hellman protocols', In Sadeghi, A., Gligor, V.D., Yung, M. (Eds.): 'Proc. ACM Conf. Computer and Communications Security (CCS 2013)', pp 1113-1128.

[11] Krawczyk, H.: 'HMQV: A high-performance secure Diffie-Hellman protocol', In Shoup, V. (Ed.): 'Proc. Int. Conf. Cryptology (CRYPTO 2005)' (Springer-Verlag, LNCS 3621, 2005), pp 546–566.

[12] Cremers, C.J.F., Feltz, M.: 'Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal', In Foresti, S., Yung, M., Martinelli, F. (Eds.): 'European Symp. on Research in Computer Security (ESORICS 2012)' (Springer-Verlag, LNCS 7459, 2012), pp 734–751.

[13] Yoneyama, K., Zhao, Y.: 'Taxonomical security consideration of authenticated key exchange resilient to intermediate computation leakage', In Boyen, X. and Chen, X., (Eds.): 'Proc. Int. Conf. Provable Security (ProvSec 2011)' (Springer-Verlag, LNCS 6980, 2011), pp 348–365.

[14] Sarr, A.P., Elbaz-Vincent, P., Bajard, J.: 'A secure and efficient authenticated Diffie-Hellman protocol', In Fabio Martinelli, Bart Preneel (Eds.): 'Proc. European Workshop on Public key Infrastructures, Services and Applications (EuroPKI 2010)' (Springer-Verlag, LNCS 6391, 2010), pp 83–98.

[15] Yao, A.C., Zhao, Y.: 'A new family of practical non-malleable protocols', IACR Cryptology ePrint Archive, Report 2011/035, 2011.

[16] Fujioka, A., Suzuki, K., Xagawa, K., *et al.*: 'Strongly secure authenticated key exchange from factoring, codes, and lattices', In Fischlin, M., Buchmann, J., Manulis, M. (Eds.): 'Proc. Int. Conf. Practice and Theory in Public-Key Cryptography (PKC 2012)' (Springer-Verlag, LNCS 7293, 2012), pp 467–484.

[17] Sarr, A.P., Elbaz-Vincent, P.: 'A complementary analysis of the (s)YZ and DIKE protocols', In Mitrokotsa, A. and Vaudenay, S. (Eds.): 'Proc. Int. Conf. Cryptology (AFRICACRYPT 2012)' (Springer-Verlag, LNCS 7374, 2012), pp 203–220.

[18] Bellare, M., Brakerski, Z., Naor, M., *et al.*: 'Hedged public-key encryption: How to protect against bad randomness', IACR Cryptology ePrint Archive, Report 2012/220, 2012.

[19] Benoît, O., Peyrin, T.: 'Side-channel analysis of six SHA-3 candidates', In Mangard, S. and Standaert, F. (Eds.): 'Proc. Int. Conf. Cryptographic Hardware and Embedded Systems (CHES 2010)' (Springer-Verlag, LNCS 6225, 2010), pp 140–157.

[20] Halderman, J.A., Schoen, S.D., Heninger, N., *et al.*: 'Lest we remember: Cold-boot attacks on encryption keys', Commun. ACM, 2009, **52**, (5), pp 91–98.

[21] Cremers, C.J.F., Feltz, M.: 'One-round Strongly Secure Key Exchange with Perfect Forward Secrecy and Deniability', IACR Cryptology ePrint Archive, Report 2011/300, 2011.

[22] Moriyama, D., Okamoto, T.: 'An eCK-secure authenticated key exchange protocol without random oracles', In Pieprzyk, J., Zhang, F. (Eds.): 'Proc. Int. Conf. Provable Security (ProvSec 2009)' (Springer-Verlag, LNCS 5848, 2009), pp 154–167.

[23] Okamoto, T., Pointcheval, D.: 'The gap-problems: A new class of problems for the security of cryptographic schemes', In Kim, K. (Eds.): 'Proc. Int. Conf. Practice and Theory in Public-Key Cryptography (PKC 2001)' (Springer-Verlag, LNCS 1992, 2001), pp 104–118.

[24] Boyd, C., Cliff, Y., Nieto, J.G., *et al.*: 'Efficient one-round key exchange in the standard model', In Mu, Y., Susilo, W., Seberry, J. (Eds.): 'Information Security and Privacy' (Springer-Verlag, LNCS 5107, 2008), pp 69–83.

[25] Blum, M., Feldman, P., Micali, S.: 'Non-interactive zero-knowledge and its applications', In Simon, J. (Ed.): 'ACM Symp. on Theory of Computing (STOC 1988)', pp 103–112.

[26] Pointcheval, D., Stern, J.: 'Security arguments for digital signatures and blind signatures', J. Cryptol., 2000, **13**, (3), pp 361–396.

Appendix 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 7-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 $\delta = (\sigma, \hat{A}, A, \hat{B}, B, X, Y)$, 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 $\text{Win} \wedge \text{AskH3}$ 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 Long-Term Key Reveal($\hat{B}$) query during its execution. Note that $\mathcal{A}$ does not necessarily make a Long-Term Key Reveal($\hat{B}$) query.

$\overline{\text{QH1}}$. This event means that for every party $\hat{B}$, if the adversary $\mathcal{A}$ queries H_1 with $(*, b)$ then $\mathcal{A}$ issued Long-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 $\Pr(\text{Win}^*) \geq p(\lambda)$, it must be the case that either event $\text{QH1} \wedge \text{Win}^*$ or event $\overline{\text{QH1}} \wedge \text{Win}^*$ 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 Mat and its complementary event by $\overline{\text{Mat}}$):

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

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

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

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

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

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 $\wedge$ WIN* *occurs with probability* p_{QH1}*, we can construct a GDH solver* $\mathcal{S}$ *that can solve the* GDH *problem with probability*

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

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

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

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

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

Appendix 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 $\mathsf{H}_3(\sigma, \hat{A}, A, \hat{B}, B, X, Y)$. 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 $(\sigma, \hat{A}, A, \hat{B}, B, X, Y)$ to H_3 (for some $\sigma \in \mathcal{G}$), $\mathcal{S}$ checks if $\mathrm{DDH}(Y, AX^e, \sigma X^{-b})$ 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 $\mathrm{CDH}(U, V) = U^a$. 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

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

Appendix 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 7-tuple $\delta = (\sigma, \hat{A}, A, \hat{B}, B, X, Y)$ 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 $\overline{\text{QH1}}$ 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

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

Appendix A.3. Proof of Lemma 3

Simulation. Suppose that event $\overline{\text{TAR}}$ 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 WIN* $\wedge$ 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 7-tuple $\delta = (\sigma, \hat{A}, A, \hat{B}, B, X, Y)$ 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 [26] 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 $(\hat{A}, A, \hat{B}, B, X, Y)$ 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 $\mathsf{H}_2(\hat{A}, A, \hat{B}, B, X, Y)$ 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

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

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

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

where $\mathrm{O}(\frac{1}{h_2(\lambda)})$ comes from the use of the Forking Lemma [26].

Appendix B. Damage on VOAKE Caused by leakage of $R_{\hat{A}}$

For the benefit of the reader we present a trivial attack against VOAKE in the PACK model if the value $R_{\hat{A}}$ is included in the `SessionState` and can be leaked by issuing Session-State Reveal query.

1. The adversary $\mathcal{A}$ randomly chooses a value $x \in \mathbb{Z}_q^*$, and obtains $X = g^x$.
2. The adversary $\mathcal{A}$ pretends to be $\hat{A}$ and initiates a new session with $\hat{B}$ by sending $(\hat{B}, \hat{A}, X)$ to $\hat{B}$. Party $\hat{B}$ responses with $(\hat{A}, \hat{B}, X, Y)$.
3. The adversary $\mathcal{A}$ pretends to be any other party $\hat{C}$ and initiates a new session with $\hat{A}$ by sending $(\hat{A}, \hat{C}, Y)$ to $\hat{A}$. Party $\hat{A}$ responses with $(\hat{C}, \hat{A}, Y, W)$.
4. The adversary $\mathcal{A}$ issues a Session-State Reveal on the new session owned by $\hat{A}$ and obtains $R_{\hat{A}} = Y^{a+e'w}$ and w, where $W = g^w$ and $e' = \mathsf{H}_2(\hat{C}, C, \hat{A}, A, Y, W)$. Then the adversary $\mathcal{A}$ computes $Y^a = \frac{Y^{a+e'w}}{Y^{e'w}} = \frac{R_{\hat{A}}}{Y^{e'w}}$.

According to the definition of freshness in the PACK model, the session associated with session identifier $(\mathcal{R}, \hat{A}, \hat{B}, X, Y)$ is fresh. Note that the adversary has the values Y^a and x, the session key corresponding to the session $(\mathcal{R}, \hat{A}, \hat{B}, X, Y)$ can be computed as $SK_B = \mathsf{H}_3(\sigma, \hat{A}, A, \hat{B}, B, X, Y)$, where $\sigma = L_{\hat{B}} \cdot R_{\hat{B}} = A^y \cdot X^{b+ye} = Y^a \cdot (BY^e)^x$ and $e = \mathsf{H}_2(\hat{A}, A, \hat{B}, B, X, Y)$. Thus, the adversary $\mathcal{A}$ can perfectly distinguish the real session key from the random one and break the `AKE` security of VOAKE protocol in the PACK model.