Cryptanalysis of a Generic One-round Key Exchange Protocol with Strong Security

Journal:	*IET Information Security*
Manuscript ID	IFS-2017-0055.R2
Manuscript Type:	Research Paper
Date Submitted by the Author:	27-Jun-2017
Complete List of Authors:	Yang, Zheng; University of Helsinki, Computer Science; Chongqing University of Technology, Computer Science and Engineering Lai, Junyu; University of Electronic Science and Technology of China Li, Guoyuan; Norwegian University of Science and Technology, Department of Ocean Operations and Civil Engineering
Keyword:	Cryptanalysis, Key Agreement, Key Exchange

Dear Editors/Reviews

On behalf of my co-authors, we appreciate editor and reviewers very much for their positive and constructive comments and suggestions on our manuscript. After carefully studying the comments, we have revised the typos and made corresponding changes which are listed in the following with label '**AW**' (after each reviewer's comment starting with '**CM**').

1. Flawed definitions: Definition 1 (Signature), Definition 2 (PRF).
   AW: Thanks again for pointing out those flaws in our definitions. We have carefully modified these two definitions according to the reviewer's comments. Specifically, we have done the following modifications:
   (a) In Definition 1:
   CM: "(m*, sigma*) is not among the previously submitted to the signing oracle":
   AW: Meanwhile, we may use a list $Slist$ to record each signing oracle query in a form (m_i, σ_i), i.e., the input and output of the i-th signing oracle query. For defining SEUF-CMA security, we require that: (m^*, σ^*) is not any tuple recorded in $Slist$.
   (b) In Definition 2:
   CM: In the PRF experiment, the value x* is undefined during execution of B_1. B_1 has access to an oracle FN, and FN uses x*. So this is not well-defined. Please correct the security definition.
   AW: The oracle FN does not check x* now. Instead we add a restriction after B_1 outputs the challenge message x*.
2. 2. Games in the Proof:
   AW: Thanks again for pointing out those flaws in our proofs. We have carefully modified the proofs according to the reviewer's comments. Specifically, we have done the following modifications:
   (a) Game 1:
   CM: Here it is claimed that the forger F breaks the security of the signature scheme, "if $sigma'_{id_j}$ was not returned by any singing oracle query". This is true, but it is not the case that you have to consider here.
   AW: We changed the arguments following the reviewers suggestion, i.e.: - When the test oracle $\pi_{id_i}^{s^*}$ receives a tuple $(epk'_{id_j}, \sigma'_{ID_j})$ such that σ'_{id_j} is a valid signature for epk'_{id_j} with respect to $vk_{id_j}^{sig}$ but $(epk'_{idj}, \sigma'_{id_j})$ is not any tuple recorded in $SList$.
   (b) Game 2:
   CM: Instead, I'd propose to argue completely differently. There is a straightforward attack on a NIKE scheme for which it is likely that the same epk is generated twice: In the NIKE security experiment,

proceed as follows:
- The adversary asks for n keys
- If there exist *any* two indices i,j such that the NIKE challenger generates keys epk_i, epk_j with $epk_i = epk_j$, then "corrupt" epk_i and use the secret key to break the security w.r.t. epk_j

AW: We would like to thank the reviewer for such good suggestion. We modify the proof in this game accordingly. Briefly speaking, we change Game 2 by asking the challenger to: (i) generate all $(\ell + d\ell)$ NIKE key pairs (which will be later used as either long-term or ephemeral key) at the beginning of the game, (ii) and abort if: there are two public keys are equivalent. The first change could enable us to check the abort rule (in the second change). If the challenger aborts with non-negligible probability, then we could break the NIKE security as suggested.

(c) Game 5:
CM: In particular, I was not able to follow the reduction to the PRF security, because everything is extremely sketchy and the considered PRF security model seems inconsistent with the security definition given in Section 2.2. For instance, in Section 2.2 there are adversaries B_1, B_2, which are not reflected in the proof, but it is just claimed that "B could ask queries....", without clarifying in which stage.
AW: We enriched the proof in Game 5. We now show how adversaries B_1, B_2 are run to simulate the AKE security game. Roughly speaking, the adversary B_1 would generate the protocol messages recorded in the session identifier of the test oracle. Those protocol messages will be used as the PRF challenge message submitted to the PRF challenger. Then the adversary B_2 would continue to simulate the AKE game. We also wrote a few sentences (in conjunction of our proof) to illustrate why our new scheme can resist with our PFS attack against the BJS scheme.

If there is still any place that we need to improve, please dont hesitate to contact us. We would like to express our great appreciation again to editor and reviewers for comments on our paper.

Best Regards Zheng Yang

Best Regards
Zheng Yang

Cryptanalysis of a Generic One-round Key Exchange Protocol with Strong Security

Zheng Yang[1,2,*], Junyu Lai[3], Guoyuan Li[4]

[1]Department of Computer Science, University of Helsinki, 00014, Finland
[2]School of Computer Science and Engineering, Chongqing University of Technology, Chongqing 400054, China
[3]School of Aeronautics and Astronautics, University of Electronic Science and Technology of China,Chengdu 611731, China
[4]Department of Ocean Operations and Civil Engineering, Norwegian University of Science and Technology, Aalesund, Norway
[*]zheng.yang@helsinki.fi

Abstract: In PKC 2015, Bergsma et al. introduced an interesting one-round key exchange protocol (which will be referred to as BJS scheme) with strong security in particular for perfect forward secrecy (PFS). In this paper, we unveil a PFS attack against the BJS scheme. This would simply invalidate its security proof. An improvement is proposed to fix the problem of the BJS scheme with minimum changes.

1. Introduction

Perfect forward secrecy (PFS) has been one of the most important security properties for authenticated key exchange (AKE). This property has a long research history that can date back to the work [1] by Diffie et al. in 1992. Nowadays (perfect) forward secrecy is often considered as a desired fundamental security property for protecting the confidentiality and implicit authentication of session key, which has been incorporated into various formal AKE security models [2, 3, 4, 5, 6]. Roughly speaking, protocols with PFS should guarantee that the compromise of long-term key would not affect the confidentiality of previously established session keys.

However, it is often the case that protocols satisfying PFS are much harder to build than those which dispense with it. This fact is particularly prominent in the construction of an one-round key exchange (ORKE) protocol. Before the work by Boyd and Nieto [7] in 2011, achieving PFS for two-message key exchange was once thought to be an impossible mission (e.g. in [5]). Therefore, an alternative notion called weak perfect forward secrecy (wPFS) is defined in [5]. In the later, a very strong security model called as eCK [6] was proposed to cover various leakage combinations of long-term and ephemeral secret keys. However, only wPFS is formulated in the eCK model. The problem of modeling PFS for two-message key exchange in an eCK like model was overcome by Cremers and Feltz [4] who proposed a so-called eCK-PFS model. In PKC 2015, Bergsma et al. [2] particularly aimed to construct an one-round key exchange protocol (which is a special class of two-message key exchange) in the eCK-PFS model. A generic protocol (which will be referred to as BJS scheme) is proposed based on non-interactive key exchange (NIKE) [8], digital

signature (SIG) and pseudo-random function (PRF). As claimed by Bergsma et al. that achieving the PFS security attribute is one of the primary motivations of the BJS scheme. In this paper, we are interested in the validity of the security proof of the BJS scheme in the eCK-PFS model. Note that a security proof is useful if and only if it has no attack under the security model wherein it is analyzed. The problem of verifying the correctness of computational complexity proof for a protocol itself is non-trivial, especially under a strong security model. Many works (e.g. [9, 10, 11]) have shown that any active attack overlooked during the security analysis may trivially invalidate the corresponding security proofs.

OUR RESULTS. We revisit the security result of the BJS scheme. We discover that the BJS scheme is actually not secure in the eCK-PFS model under which it was proved. The main problem here is that the session key material which is expected to provide PFS is not specifically bound to each session. In this case, the adversary can result in two sessions without matching sessions (non-partnered), e.g., denoted by $\pi^s_{\mathsf{id}_1}$ and $\pi^t_{\mathsf{id}_2}$, having the same intermediate keying secret $k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$ which is computed based on the ephemeral public keys epk_{id_1} and epk_{id_2}. Unfortunately, we figure out that, after compromising the long-term keys of session participants, the secret $k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$ can be extracted from the session key of $\pi^t_{\mathsf{id}_2}$. This would enable the adversary to break the PFS property of the BJS scheme. We will describe the concrete attack in Section 4. An improved scheme is proposed to circumvent our PFS attack. In the improvement, we change the key derivation function (KDF) of $k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$ (and other important keying materials) by putting all protocol messages and identities of communication partners into it. Namely, the intermediate key $k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$ is generated based on the ephemeral secrets together with the protocol messages and identities. We stress that binding identities to $k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$ is important as well. Since it could resist with the unknown key share attacks [14]. We hope that our security analysis would be helpful for avoiding such mistakes in the future similar works.

2. Preliminaries

General Notations. We let $\lambda \in \mathbb{N}$ be the security parameter and 1^λ be a string that consists of λ ones. We write $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ to denote the set of integers between 1 and n. The notation $a \xleftarrow{\$} S$ denotes the operation which samples a uniformly random element from a set S. Assume $\mathcal{IDS}$ be an identity space. Let $\mathcal{K}_{\mathsf{AKE}}$ be the key space of session key. Those spaces are associated with security parameter λ.

2.1. Digital Signature Schemes

We consider a digital signature scheme SIG that consists of three probabilistic polynomial time (PPT) algorithms $\mathsf{SIG} = (\mathsf{SIG.Gen}, \mathsf{SIG.Sign}, \mathsf{SIG.Vfy})$ which associate with public and secret key spaces $\{\mathcal{PK}_{\mathsf{SIG}}, \mathcal{SK}_{\mathsf{SIG}}\}$, message space $\mathcal{M}_{\mathsf{SIG}}$, a secret randomness space $\mathcal{RS}_{\mathsf{SIG}}$ and signature space $\mathcal{S}_{\mathsf{SIG}}$ in the security parameter λ. These algorithms are defined as follows:

- $(sk, vk) \leftarrow \mathsf{SIG.Gen}(1^\lambda, rs)$: This algorithm takes as input the security parameter λ and a randomness $rs \in \mathcal{RS}_{\mathsf{SIG}}$, and outputs a (public) verification key $vk \in \mathcal{PK}_{\mathsf{SIG}}$ and a secret signing key $sk \in \mathcal{SK}_{\mathsf{SIG}}$;
- $\sigma \xleftarrow{\$} \mathsf{SIG.Sign}(sk, m)$: This is the signing algorithm that generates a signature $\sigma \in \mathcal{S}_{\mathsf{SIG}}$ for a

message $m \in \mathcal{M}_{\mathsf{SIG}}$ with signing key sk;

- $\{0,1\} \leftarrow \mathsf{SIG.Vfy}(vk, m, \sigma)$: This is the verification algorithm that on input a verification key vk, a message m and the corresponding signature σ, and outputs 1 if σ is a valid signature for m under key vk, and 0 otherwise.

Let $\mathcal{SIG}(sk, \cdot)$ be a signing oracle which takes as input a message m and returns a signature $\sigma \leftarrow \mathsf{SIG.Sign}(sk, m)$. Meanwhile, we may use a list Slist to record each signing oracle query in a form (m_i, σ_i), i.e., the input and output of the i-th signing oracle query.

Definition 1. We say that SIG is $(q_s, t, \epsilon_{\mathsf{SIG}})$-secure against *strong existential forgeries under adaptive chosen-message attacks*, if $\Pr[\mathsf{EXP}^{seuf\text{-}cma}_{\mathsf{SIG},\mathcal{A}}(\lambda) = 1] \leq \epsilon_{\mathsf{SIG}}$ for all adversaries $\mathcal{A}$ running in time at most t in the following experiment:

$\mathsf{EXP}^{seuf\text{-}cma}_{\mathsf{SIG},\mathcal{A}}(\lambda)$

$rs \in \mathcal{RS}_{\mathsf{SIG}}$; $(sk, pk) \leftarrow \mathsf{SIG.Gen}(1^\lambda, rs)$;
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{SIG}(sk,\cdot)}$, which can make up to q_s queries to the signing oracle $\mathcal{SIG}(sk, \cdot)$ with arbitrary messages m;
return 1, if the following conditions are held:

1. $\mathsf{SIG.Vfy}(pk, m^*, \sigma^*) = 1$, and
2. (m^*, σ^*) is not any tuple recorded in Slist;

output 0, otherwise;

where ϵ_{SIG} is a negligible function in λ.

2.2. Pseudo-Random Functions

A pseudo-random function family is denoted by $\mathsf{PRF} : \mathcal{K}_{\mathsf{PRF}} \times \mathcal{D}_{\mathsf{PRF}} \rightarrow \mathcal{R}_{\mathsf{PRF}}$, where $\mathcal{K}_{\mathsf{PRF}}$ is the key space, $\mathcal{D}_{\mathsf{PRF}}$ is the domain and $\mathcal{R}_{\mathsf{PRF}}$ is the range of PRF for security parameter λ. Let PList be a list to store the messages queried in the following PRF oracle $\mathcal{FN}$.

Definition 2. The pseudo-random function family PRF is said to be a $(q_f, t, \epsilon_{\mathsf{PRF}})$-secure, if the probability $|\Pr[\mathsf{EXP}^{ind\text{-}cma}_{\mathsf{PRF},\mathcal{B}}(\lambda) = 1] - 1/2| \leq \epsilon_{\mathsf{PRF}}$ holds for all adversaries $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ that make a polynomial number q_f of oracle queries while running within time t in the following experiment without failure:

$\mathsf{EXP}^{\mathsf{ind\text{-}cma}}_{\mathsf{PRF},\mathcal{B}}(q_f, \lambda)$:	$\mathcal{FN}(k, x)$:
$b \xleftarrow{\$} \{0,1\}, k \xleftarrow{\$} \mathcal{K}_{\mathsf{PRF}}$;	append $x \rightarrow \mathsf{PList}$;
$(x^*, st) \leftarrow \mathcal{B}_1^{\mathcal{FN}(k,\cdot)}$;	return $\mathsf{PRF}(k, x)$;
if $x^* \in \mathsf{PList}$ then return a failure $\perp$;	
$V_1^* := \mathsf{PRF}(k, x^*), V_0^* \xleftarrow{\$} \mathcal{R}_{\mathsf{PRF}}$;	
$b' \leftarrow \mathcal{B}_2^{\mathcal{FN}(k,\cdot)}(st, V_b^*)$;	
If $b = b'$ then return 1;	
Otherwise, return 0;	

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

2.3. Non-Interactive Key Exchange Protocols

We first review the simplified notion of NIKE presented in [12, Appendix G]. Let $\mathcal{IDS}_{\mathsf{NIKE}}$ denote an identity space and $\mathcal{K}_{\mathsf{NIKE}}$ denote a shared key space in association with security parameter. A PKI-based Non-Interactive Key Exchange (NIKE) scheme consists of three algorithms: NIKE.Setup, NIKE.Gen and ShKey, in which those algorithms are defined as follows:

- $pms^{nike} \leftarrow \mathsf{NIKE.Setup}(1^\lambda)$: This algorithm takes as input a security parameter λ, and outputs system parameters pms^{nike}. The parameters pms^{nike} might be implicitly used by other algorithms for simplicity.
- $(sk, pk) \leftarrow \mathsf{NIKE.Gen}(er)$: This algorithm takes as input a randomness $er \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$ chosen from a space $\mathcal{R}_{\mathsf{NIKE}}$, and outputs a pair of long-term secret and public key (sk, pk).
- $K \leftarrow \mathsf{ShKey}(sk_i, pk_j)$: This algorithm takes as input a secret key sk_i and a public key pk_j, and outputs a shared key $K_{i,j} \in \mathcal{K}_{\mathsf{NIKE}}$. For a tuple of key pairs (sk_i, pk_i) and (sk_j, pk_j), the algorithm ShKey must hold the following correctness condition:
  - $\mathsf{ShKey}(sk_i, pk_j) = \mathsf{ShKey}(sk_j, pk_i)$

Security Notion of NIKE. We here recall the CKS-light formal security model [8] for two party PKI-based non-interactive key-exchange (NIKE). We first describe the security experiment in the following.
$\mathsf{EXP}^{cks\text{-}light}_{\mathsf{NIKE},\mathcal{A}}(\lambda)$: On input security parameter λ, the security experiment is executed between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ based on a non-interactive key exchange protocol NIKE. Let HID and DID be two lists recording information (e.g. secret and public key pair) of honest users and dishonest users respectively. In the security experiment, the following steps are proceeded:

1. The challenger $\mathcal{C}$ first generates the system parameters pms^{nike} according to the protocol specification and gives pms^{nike} to the adversary $\mathcal{A}$.
2. The adversary $\mathcal{A}$ may ask $\mathcal{C}$ with the following queries:
   - $\mathsf{RegH}(i)$: If $i \in \{\mathsf{HID}, \mathsf{DID}\}$ a failure symbol $\perp$ is returned; Otherwise, $\mathcal{C}$ generates a long-term key pair $(sk_i, pk_i) \in (\mathcal{PK}, \mathcal{SK})$ by running $\mathsf{NIKE.Gen}(er)$ and adds the tuple (i, sk_i, pk_i) into the list HID. $\mathcal{C}$ gives pk_i to $\mathcal{A}$.
   - $\mathsf{RegC}^{nike}(j, pk_j)$: If $j \notin \mathsf{HID}$, $\mathcal{C}$ allows the adversary to register an dishonest long-term public key pk_j, and records the tuple (j, pk_j) into the list DID. If the adversary makes multiple queries for a particular index j, in which case $\mathcal{C}$ only keeps the most recent record.
   - $\mathsf{EXT}(i)$: If $i \in \mathsf{HID}$, then $\mathcal{C}$ returns the secret key sk_i.
   - $\mathsf{RevealKey}^{nike}(i, j)$: If both i and j are recorded in the list DID, $\mathcal{C}$ returns a failure symbol $\perp$; Otherwise, $\mathcal{C}$ runs ShKey using one of the honest secret keys in (sk_i, sk_j) and the public key of the other party, and returns the generated key to $\mathcal{A}$.
   - $\mathsf{Test}^{nike}(i^*, j^*)$: $\mathcal{C}$ returns a failure symbol $\perp$ if either $i^* = j^*$ or $i^* \notin \mathsf{HID}$ or $j^* \notin \mathsf{HID}$; Otherwise, $\mathcal{C}$ samples a random bit $b \xleftarrow{\$} \{0, 1\}$ and answers this query according to the bit b. Specifically, $\mathcal{C}$ obtains the real shared key $K_1 := \mathsf{ShKey}(sk_{i^*}, pk_{j^*})$ and chooses a random key $K_0 \xleftarrow{\$} \mathcal{K}_{\mathsf{NIKE}}$. Then $\mathcal{C}$ returns K_b to the adversary. This query can be queried at most one time.

3. Meantime, $\mathcal{A}$ is not allowed to ask either $\mathsf{EXT}(i^*)$ or $\mathsf{EXT}(j^*)$ where (i^*, j^*) are chosen by $\mathcal{A}$ in the Test^{nike} query; and $\mathcal{A}$ is also not allowed to ask any $\mathsf{RevealKey}^{nike}$ query with input (i^*, j^*) in either order.

4. Finally, the experiment returns 1 if $\mathcal{A}$ has issued a $\mathsf{Test}^{nike}(i^*, j^*)$ query without failure, and $b = b'$; Otherwise, 0 is returned.

Definition 3. A two party non-interactive key exchange protocol NIKE is called $(t, \epsilon_{\mathsf{NIKE}})$-adaptively-secure if for all adversaries $\mathcal{A}$ running within time t in the above security experiment, it holds that $|\Pr[\mathsf{EXP}^{cks\text{-}light}_{\mathsf{NIKE},\mathcal{A}}(\lambda) = 1] - 1/2| \leq \epsilon_{\mathsf{NIKE}}$, where $\epsilon_{\mathsf{NIKE}} = \epsilon_{\mathsf{NIKE}}(\lambda)$ is some negligible probability in the security parameter λ.

3. Security Model

In this section, we describe the eCK-PFS model as [4, 2] for two-message authenticated key-exchange (AKE). In order to emulate the protocol executions in the real world, we provide active adversaries with an 'execution environment' as in [2, 15]. Specifically, the protocol executions of a set of *honest* users (that an adversary may attack) are represented by a collection of oracles $\{\pi^s_{\mathsf{id}_i} : i \in [\ell], s \in [d]\}$ for $d \in \mathbb{N}$, where each oracle $\pi^s_{\mathsf{id}_i}$ behaves as the party id_i carrying out a process to execute the s-th protocol instance (session) and all oracles can be run sequentially and concurrently. We assume that all identities and corresponding public keys are stored in a public directory $\mathcal{PD}$ that can be accessed by all oracles. In the real world, such public directory could be like certificate authority. In order to keep track of the execution status, we assume each oracle $\pi^s_{\mathsf{id}_i}$ has a list of internal state variables: (i) $\mathsf{pid}^s_{\mathsf{id}_i}$ – storing the identity and public key of its intended communication partner, e.g. $(\mathsf{id}_j, pk_{\mathsf{id}_j})$; (ii) $\Phi^s_{\mathsf{id}_i}$ – denoting the decision $\Phi^s_{\mathsf{id}_i} \in \{\mathtt{accept}, \mathtt{reject}\}$; (iii) $K^s_{\mathsf{id}_i}$ – recording the session key $K^s_{\mathsf{id}_i} \in \mathcal{K}_{\mathsf{AKE}}$; (iv) $sT^s_{\mathsf{id}_i}$ – recording the transcript of messages sent by the oracle $\pi^s_{\mathsf{id}_i}$; (v) $rT^s_{\mathsf{id}_i}$ – recording the transcript of messages received by the oracle $\pi^s_{\mathsf{id}_i}$.

In literature [2], the authors particularly use the public key as identity. However, in the real life, the adversary may simply register a party who has the same public key with that of another honest party. In this case, a party may not be uniquely identified. Therefore, in our model, we only require that the identity id is some unique string (e.g., some sequential index). The identity of an honest party should not be manipulated by an adversary. In order to describe the problem of the BJS scheme, we try to keep our model to be consistent with the model of [2] as much as possible.

An active adversary $\mathcal{A}$ in the model is formulated as a PPT Turing Machine taking as input the security parameter 1^λ and public information (e.g. generic description of the above environment and all public keys of honest parties). The capabilities of active adversaries are formulated by allowing them to issue the following queries:

- $\mathsf{Send}(\mathsf{id}_i, s, m)$: The adversary can use this query to send any message m of his own choice to the oracle $\pi^s_{\mathsf{id}_i}$. The oracle $\pi^s_{\mathsf{id}_i}$ will respond with the next message m^* (if any) to be sent according to the protocol specification and its internal states. The oracle $\pi^s_{\mathsf{id}_i}$ would be initiated via sending it the first message $m = (\top, \widetilde{\mathsf{id}_j})$ consisting of a special initialization symbol $\top$ and a value $\widetilde{\mathsf{id}_j}$. The $\widetilde{\mathsf{id}_j}$ is either the identity id_j of the intended partner or an empty string $\emptyset$. After answering a Send query, the variables $(\mathsf{pid}^s_{\mathsf{id}_i}, \Phi^s_{\mathsf{id}_i}, K^s_{\mathsf{id}_i}, sT^s_{\mathsf{id}_i}, rT^s_{\mathsf{id}_i})$ will be updated depending on the specific protocol.

- $\mathsf{RevealKey}(\mathsf{id}_i, s)$: The oracle $\pi^s_{\mathsf{id}_i}$ responds with the contents of the variable $K^s_{\mathsf{id}_i}$ if and only if the oracle $\pi^s_{\mathsf{id}_i}$ has reached an internal state $\Phi^s_{\mathsf{id}_i} = \texttt{accept}$.
- $\mathsf{RevealEphKey}(\mathsf{id}_i, s)$: The oracle $\pi^s_{\mathsf{id}_i}$ responds with the per-session randomness used to generate ephemeral public key.
- $\mathsf{Corrupt}(\mathsf{id}_i, pk^*)$: If $i \in [\ell]$ this query responds with the honest long-term secret key sk_{id_i} (corresponding to the original sk_{id_i}) of the party id_i, and the current public key pk_{id_i} stored in the public directory $\mathcal{PD}$ is replaced with the new pk^* of adversary's choice; otherwise a failure symbol $\perp$ is returned. After this query, the party id_i is called corrupted and all unstopped oracles of id_i can answer other queries using its old public/secret key pair. But no more oracle of id_i can be initiated since then.
- $\mathsf{Test}(\mathsf{id}_i, s)$: If the oracle $\pi^s_{\mathsf{id}_i}$ has state $\Phi^s_{\mathsf{id}_i} \neq \texttt{accept}$ or $K^s_{\mathsf{id}_i} = \emptyset$, then this query returns some failure symbol $\perp$. Otherwise, it flips a fair coin $b \in \{0, 1\}$, samples a random key $K_0 \xleftarrow{\$} \mathcal{K}_{\mathsf{AKE}}$, and sets $K_1 := K^s_{\mathsf{id}_i}$. Finally, the key K_b is returned. The oracle $\pi^s_{\mathsf{id}_i}$ selected by the adversary in this query is called as *test oracle*.

Secure AKE Protocols: We here first review the notion regarding partnership of two sessions, i.e. *matching sessions* and *origin session* [2].

We first recall the notion of *origin session* [2] which is useful to formulate the relationship between the message generator and its receiver.

Definition 4 (Origin Session)**.** We say that an oracle $\pi^s_{\mathsf{id}_i}$ has an *origin session* to an oracle $\pi^t_{\mathsf{id}_j}$, if $\pi^s_{\mathsf{id}_i}$ has sent all protocol messages and $sT^s_{\mathsf{id}_i} = rT^t_{\mathsf{id}_j}$. The oracle $\pi^s_{\mathsf{id}_i}$ is said to be the origin-oracle of $\pi^t_{\mathsf{id}_j}$.

Note that if an oracle $\pi^s_{\mathsf{id}_i}$ received a protocol message which does not include any information about its generator, then such message may not come from its intended partner. We now recall the matching sessions defined in [2].

Definition 5 (Matching Sessions)**.** We say that an oracle $\pi^s_{\mathsf{id}_i}$ has a *matching session* to an oracle $\pi^t_{\mathsf{id}_j}$, if $\pi^s_{\mathsf{id}_i}$ has sent all protocol messages and all the following conditions are held:

- $\pi^s_{\mathsf{id}_i}$ is an origin-oracle of $\pi^t_{\mathsf{id}_j}$;
- $\pi^t_{\mathsf{id}_j}$ is an origin-oracle of $\pi^s_{\mathsf{id}_i}$.

CORRECTNESS. We say an AKE protocol Π is correct, if two oracles $\pi^s_{\mathsf{id}_i}$ and $\pi^t_{\mathsf{id}_j}$ accept with matching sessions, then both oracles hold the same session key.

In order to define security, we review the notion of *oracle freshness* (which is called *eCK-PFS rules* in [2]) that describes the active attacks allowed in the eCK-PFS model.

Definition 6 (Oracle Freshness)**.** Let $\pi^s_{\mathsf{id}_i}$ be an accepted oracle with the intended partner id_j such that $j \in [\ell]$. And let $\pi^t_{\mathsf{id}_j}$ be an oracle (if it exists) with the intended partner id_i, such that $\pi^s_{\mathsf{id}_i}$ has a matching session to $\pi^t_{\mathsf{id}_j}$. Let $\pi^z_{\mathsf{id}_v}$ be an oracle (if it exists), such that $\pi^z_{\mathsf{id}_v}$ has an origin session to $\pi^s_{\mathsf{id}_i}$. Then the oracle $\pi^s_{\mathsf{id}_i}$ is said to be fresh if none of the following conditions is held:

1. $\mathcal{A}$ queried $\mathsf{RevealKey}(\mathsf{id}_i, s)$.

2. If $\pi^t_{\mathsf{id}_j}$ exists, $\mathcal{A}$ queried $\mathsf{RevealKey}(\mathsf{id}_j, t)$.
3. $\mathcal{A}$ queried both $\mathsf{Corrupt}(\mathsf{id}_i, pk_{\mathsf{id}_i})$ and $\mathsf{RevealEphKey}(\mathsf{id}_i, s)$.
4. If $\pi^z_{\mathsf{id}_v}$ exists, $\mathcal{A}$ queried both $\mathsf{Corrupt}(\mathsf{id}_j, pk_{\mathsf{id}_j})$ and $\mathsf{RevealEphKey}(\mathsf{id}_v, l)$.
5. If $\pi^z_{\mathsf{id}_v}$ does not exist, $\mathcal{A}$ queried $\mathsf{Corrupt}(\mathsf{id}_j, pk_{\mathsf{id}_j})$ prior to the acceptance of $\pi^s_{\mathsf{id}_i}$.

Remark 1. The oracle freshness definition always plays a very important and fundamental role in the security model, which is used to not only formulate the queries that are allowed in the following security experiment but also exclude some trivial attacks. However, some trivial attacks might be obscure and easy to be ignored. Overlooking any of them may lead the underlying security model to be obsolete. Unfortunately we notice that the *eCK-PFS rules* defined in [2] is such a negative example. And no protocol can be secure under their definition. We here recall one of the eCK-PFS rules that: If $\pi^s_{\mathsf{id}_i}$ has an origin session $\pi^t_{\mathsf{id}_j}$, then it does not hold that both $\mathsf{Corrupted}_j \leq \tau$ and $\mathcal{A}$ asked $\mathsf{RevealEphKey}(\mathsf{id}_j, t)$, where $\tau \in \mathbb{N}$ is a variable recording the τ-query when $\pi^s_{\mathsf{id}_i}$ is accepted, and $\mathsf{Corrupted}_j \in \mathbb{N}$ is a similar variable recording the $\mathsf{Corrupted}_j$-th query when id_j is corrupted. This rule implies that the adversary is allowed to ask both $\mathsf{Corrupt}(\mathsf{id}_j)$ and $\mathsf{RevealEphKey}(\mathsf{id}_j, t)$ after the τ-th query. The RevealEphKey query is performed exactly the same as RevealRand defined in [2]. However, after querying both $\mathsf{Corrupt}(\mathsf{id}_j)$ and $\mathsf{RevealEphKey}(\mathsf{id}_j, t)$ the adversary can obtain all necessary secrets for generating the final session key of the test oracle. Actually, if $\pi^s_{\mathsf{id}_i}$ has an origin session $\pi^t_{\mathsf{id}_j}$, the adversary should be forbidden to ask both $\mathsf{Corrupt}(\mathsf{id}_j)$ and $\mathsf{RevealEphKey}(\mathsf{id}_j, t)$ throughout the security experiment (nor just before some point).

Remark 2. Furthermore, the eCK-PFS model defined in [2] is not totally consistent with the original one introduced in [4], in particularly for the use of identity. Unlike [4], the identity is not considered in the definition of matching sessions. This difference may hinder us to examine the important attacks that are relevant to identity (such as the unknown key share attacks [14]) under such a variant of eCK-PFS model [2].

SECURITY EXPERIMENT $\mathsf{EXP}^{\mathsf{AKE}}_{\Pi,\mathcal{A}}(\lambda)$: On input security parameter 1^λ, the security experiment is proceeded as a game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ based on a AKE protocol Π, where the following steps are performed:

1. At the beginning of the game, the challenger $\mathcal{C}$ implements the collection of oracles $\{\pi^s_{\mathsf{id}_i} : i \in [\ell], s \in [d]\}$, and generates ℓ long-term key pairs $(pk_{\mathsf{id}_i}, sk_{\mathsf{id}_i})$ for all honest parties id_i for $i \in [\ell]$ where the identity $\mathsf{id}_i \in \mathcal{IDS}$ of each party is chosen uniquely. $\mathcal{C}$ gives the adversary $\mathcal{A}$ all identities and public keys $\{(\mathsf{id}_1, pk_{\mathsf{id}_1}), \ldots, (\mathsf{id}_\ell, pk_{\mathsf{id}_\ell})\}$ as input.
2. $\mathcal{A}$ may issue a polynomial number of queries regarding Send, RevealEphKey, Corruptand RevealKey.
3. At some point, $\mathcal{A}$ may issue a $\mathsf{Test}(\mathsf{id}_i, s)$ query during the game at most once.
4. At the end of the game, $\mathcal{A}$ may terminate and output a bit b' as its guess for b of $\mathsf{Test}(\mathsf{id}_i, s)$ query. Then the experiment would return a failure symbol $\perp$ if one of the following conditions is satisfied: (i) $\mathcal{A}$ has not issued a $\mathsf{Test}(\mathsf{id}_i, s)$ query; (ii) the $\mathsf{Test}(\mathsf{id}_i, s)$ query returns a failure symbol $\perp$; (iii) the test oracle is not fresh.
5. Finally, the experiment returns 1 if $b = b'$; Otherwise, 0 is returned.

Definition 7 (eCK-PFS Security)**.** We say that an adversary $\mathcal{A}$ (t, ϵ)-breaks the eCK-PFS security of a correct AKE protocol Π, if for any $\mathcal{A}$ who runs the AKE security experiment $\mathsf{EXP}^{\mathsf{AKE}}_{\Pi,\mathcal{A}}(\lambda)$ within time t and without failure, it holds that $|\Pr[\mathsf{EXP}^{\mathsf{AKE}}_{\Pi,\mathcal{A}}(\lambda) = 1] - 1/2| > \epsilon$. We say that a correct AKE protocol Π is (t, ϵ)-session-key-secure, if there exists no adversary that (t, ϵ)-breaks the eCK-PFS security of Π.

4. The Insecurity and Improvement of the BJS scheme

In this section, we describe the insecurity of the BJS scheme [2]. We first review the BJS scheme in Figure 1. In the presentation slides [16] of the BJS scheme in PKC 2015, the exchanged signatures are included within the transcript T. However, this change would not change the insecurity of the BJS scheme. Furthermore, [2, equation (4)] (i.e., $k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{ShKey}(esk_{\mathsf{id}_1}, epk_{\mathsf{id}_2})$) is not consistent with its counterpart (i.e., $k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_1}, epk_{\mathsf{id}_2}))$) in [2, Fig.2.]. However, neither of them is secure. We here just recall [2, Fig.2.] for simplicity.

id_1		id_2
Key Generation:		
$rs_{\mathsf{id}_1} \xleftarrow{\$} \mathcal{RS}_{\mathsf{SIG}}$		$rs_{\mathsf{id}_2} \xleftarrow{\$} \mathcal{RS}_{\mathsf{SIG}}$
$(sk^{sig}_{\mathsf{id}_1}, vk^{sig}_{\mathsf{id}_1}) \leftarrow \mathsf{SIG.Gen}(1^\lambda, rs_{\mathsf{id}_1})$		$(sk^{sig}_{\mathsf{id}_2}, vk^{sig}_{\mathsf{id}_2}) \leftarrow \mathsf{SIG.Gen}(1^\lambda, rs_{\mathsf{id}_2})$
$r_{\mathsf{id}_1} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$		$r_{\mathsf{id}_2} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$
$(sk^{nike}_{\mathsf{id}_1}, pk^{nike}_{\mathsf{id}_1}) \leftarrow \mathsf{NIKE.Gen}(r_{\mathsf{id}_1})$		$(sk^{nike}_{\mathsf{id}_2}, pk^{nike}_{\mathsf{id}_2}) \leftarrow \mathsf{NIKE.Gen}(r_{\mathsf{id}_2})$
$sk_{\mathsf{id}_1} = (sk^{nike}_{\mathsf{id}_1}, sk^{sig}_{\mathsf{id}_1})$		$sk_{\mathsf{id}_2} = (sk^{nike}_{\mathsf{id}_2}, sk^{sig}_{\mathsf{id}_2})$
$pk_{\mathsf{id}_1} = (pk^{nike}_{\mathsf{id}_1}, vk^{sig}_{\mathsf{id}_1})$		$pk_{\mathsf{id}_2} = (pk^{nike}_{\mathsf{id}_2}, vk^{sig}_{\mathsf{id}_2})$
Protocol Execution:		
$er_{\mathsf{id}_1} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$		$er_{\mathsf{id}_2} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$
$(esk_{\mathsf{id}_1}, epk_{\mathsf{id}_1}) \leftarrow \mathsf{NIKE.Gen}(er_{\mathsf{id}_1})$		$(esk_{\mathsf{id}_2}, epk_{\mathsf{id}_2}) \leftarrow \mathsf{NIKE.Gen}(er_{\mathsf{id}_2})$
$\sigma_{\mathsf{id}_1} \xleftarrow{\$} \mathsf{SIG.Sign}(sk^{sig}_{\mathsf{id}_1}, epk_{\mathsf{id}_1})$		$\sigma_{\mathsf{id}_2} \xleftarrow{\$} \mathsf{SIG.Sign}(sk^{sig}_{\mathsf{id}_2}, epk_{\mathsf{id}_2})$
	$epk_{\mathsf{id}_1}, \sigma_{\mathsf{id}_1} \longrightarrow$	
	$\longleftarrow epk_{\mathsf{id}_2}, \sigma_{\mathsf{id}_2}$	
reject if $\mathsf{SIG.Vfy}(vk^{sig}_{\mathsf{id}_2}, \sigma_{\mathsf{id}_2}, epk_{\mathsf{id}_2}) \neq 1$		reject if $\mathsf{SIG.Vfy}(vk^{sig}_{\mathsf{id}_1}, \sigma_{\mathsf{id}_1}, epk_{\mathsf{id}_1}) \neq 1$
$T := \mathsf{sort}(epk_{\mathsf{id}_1}, epk_{\mathsf{id}_2})$		$T := \mathsf{sort}(epk_{\mathsf{id}_1}, epk_{\mathsf{id}_2})$
$k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_1}, pk^{nike}_{\mathsf{id}_2}), T)$		$k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_2}, pk^{nike}_{\mathsf{id}_1}), T)$
$k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_1}, epk_{\mathsf{id}_2}), T)$		$k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_2}, pk^{nike}_{\mathsf{id}_1}), T)$
$k^{epk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_1}, pk^{nike}_{\mathsf{id}_2}), T)$		$k^{epk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_2}, epk_{\mathsf{id}_1}), T)$
$k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_1}, epk_{\mathsf{id}_2}))$		$k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_2}, epk_{\mathsf{id}_1}))$
accept $K := k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$		accept $K := k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$

Fig. 1: The BJS Scheme

4.1. The PFS attack against BJS Scheme

In the following, we present an attack against the perfect forward secrecy of the BJS scheme. We just exploit the $\mathsf{Corrupt}(\mathsf{id}_i, pk^*_{\mathsf{id}_i})$ query which returns the original secret key of id_i. However, we stress that we do not need to change the public key of an honest party via the Corrupt query in the following PFS attack. The attack idea is to result in two oracles $\pi^s_{\mathsf{id}_1}$ and $\pi^t_{\mathsf{id}_2}$ generating the same keying material (i.e., the one computed by only ephemeral secrets) without breaking the eCK-PFS freshness.

Lemma 1. There exists some adversary $\mathcal{A}$ which $(t, 1)$-breaks the eCK-PFS security the BJS protocol.

The concrete PFS attack steps are presented as below:

1. The adversary $\mathcal{A}$ chooses two arbitrary target oracles $\pi^s_{\mathsf{id}_1}$ and $\pi^t_{\mathsf{id}_2}$. $\mathcal{A}$ first asks $\pi^s_{\mathsf{id}_1}$ to execute the protocol instance, and intercepts $(epk^s_{\mathsf{id}_1}, \sigma^s_{\mathsf{id}_1})$.
2. $\mathcal{A}$ queries $\mathsf{Corrupt}(\mathsf{id}_3, pk^*_{\mathsf{id}_3})$ to get the long-term secret key $sk_{\mathsf{id}_3} = (sk^{nike}_{\mathsf{id}_3}, sk^{sig}_{\mathsf{id}_3})$, where $pk^*_{\mathsf{id}_3}$ could be identical to its original one.
3. $\mathcal{A}$ asks $\pi^t_{\mathsf{id}_2}$ to execute the protocol instance, and intercepts $(epk^t_{\mathsf{id}_j}, \sigma^t_j)$.
4. $\mathcal{A}$ generates the signature $\sigma^*_{\mathsf{id}_3} := \mathsf{SIG.Sign}(sk^{sig}_{\mathsf{id}_3}, epk^s_{\mathsf{id}_1})$.
5. $\mathcal{A}$ sends $(epk^s_{\mathsf{id}_1}, \sigma^*_{\mathsf{id}_3})$ to $\pi^t_{\mathsf{id}_2}$, and forwards $(epk^t_{\mathsf{id}_2}, \sigma^t_{\mathsf{id}_2})$ to $\pi^s_{\mathsf{id}_1}$.
6. Note that, at this moment, the oracle $\pi^s_{\mathsf{id}_1}$ has an origin oracle $\pi^t_{\mathsf{id}_2}$. But the oracle $\pi^s_{\mathsf{id}_1}$ is not the origin-oracle of $\pi^t_{\mathsf{id}_2}$. Now, the adversary can ask $\mathsf{Corrupt}(\mathsf{id}_i, pk^*_{\mathsf{id}_1})$ and $\mathsf{Corrupt}(\mathsf{id}_2, pk^*_{\mathsf{id}_2})$ to obtain $sk_{\mathsf{id}_1} = (sk^{nike}_{\mathsf{id}_1}, sk^{sig}_{\mathsf{id}_1})$ and $sk_{\mathsf{id}_2} = (sk^{nike}_{\mathsf{id}_2}, sk^{sig}_{\mathsf{id}_2})$, where $pk^*_{\mathsf{id}_1}$ and $pk^*_{\mathsf{id}_2}$ could be the same as the original public keys.
7. Then $\mathcal{A}$ obtains the session key $K^t_{\mathsf{id}_2,\mathsf{id}_3}$ by querying $\mathsf{RevealKey}(\mathsf{id}_2, t)$, where $K^t_{\mathsf{id}_2,\mathsf{id}_3} = k^{pk_{\mathsf{id}_2}}_{pk_{\mathsf{id}_3}} \oplus k^{pk_{\mathsf{id}_2}}_{epk_{\mathsf{id}_1}} \oplus k^{epk_{\mathsf{id}_2}}_{pk_{\mathsf{id}_3}} \oplus k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$. Now the goal of the adversary is to get the keying material $k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$ from $K^t_{\mathsf{id}_2,\mathsf{id}_3}$. This is possible. Since $\mathcal{A}$ is able to compute $k^{pk_{\mathsf{id}_2}}_{pk_{\mathsf{id}_3}} \oplus k^{pk_{\mathsf{id}_2}}_{epk_{\mathsf{id}_1}} \oplus k^{epk_{\mathsf{id}_2}}_{pk_{\mathsf{id}_3}}$ using the secret keys $sk^{nike}_{\mathsf{id}_2}$ and $sk^{nike}_{\mathsf{id}_3}$. Then we have $k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} = K^t_{\mathsf{id}_2,\mathsf{id}_3} \oplus k^{pk_{\mathsf{id}_2}}_{pk_{\mathsf{id}_3}} \oplus k^{pk_{\mathsf{id}_2}}_{epk_{\mathsf{id}_1}} \oplus k^{epk_{\mathsf{id}_2}}_{pk_{\mathsf{id}_3}}$.
8. After obtaining $k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$, $\mathcal{A}$ is able to compute the real session key $K^s_{\mathsf{id}_1,\mathsf{id}_2}$ of $\pi^s_{\mathsf{id}_1}$, i.e., $K^s_{\mathsf{id}_1,\mathsf{id}_2} := k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$, where the key value $k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}}$ can be computed using the secret keys $sk^{nike}_{\mathsf{id}_1}$ and $sk^{nike}_{\mathsf{id}_2}$ received from Corrupt queries before.
9. Then $\mathcal{A}$ can ask $K^*_b \leftarrow \mathsf{Test}(\mathsf{id}_i, s)$ and wins the game by comparing K^*_b with $K^s_{\mathsf{id}_1,\mathsf{id}_2}$.

The above attack is valid in the eCK-PFS model [2], since the test oracle $\pi^s_{\mathsf{id}_1}$ is executed under the eCK-PFS rules. Note that $\mathcal{A}$ neither asked $\mathsf{RevealEphKey}(\mathsf{id}_1, s)$ nor $\mathsf{RevealEphKey}(\mathsf{id}_2, t)$. But the oracle $\pi^t_{\mathsf{id}_2}$, which holds the same keying material of the test oracle $\pi^s_{\mathsf{id}_1}$, has no matching session to $\pi^s_{\mathsf{id}_1}$. Therefore, the RevealKey query to $\pi^t_{\mathsf{id}_2}$ does not break the eCK-PFS freshness. Note also that the above attack against the BJS scheme holds in any AKE model wherein the PFS attacks and known session key attacks are formulated (e.g., [3]). As in our PFS attack, we do not make use of RevealEphKey query at all.

4.2. An Improvement Solution of the BJS Scheme

The main problem of the BJS scheme is that the protocol message transcript is not bound to the keying material generated involving only ephemeral public keys. Our remedy is straightforward that we put all exchanged messages and identities of session participants into the key derivation function, i.e. PRF, to generate all keying materials including $k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$. Furthermore, each message flow also consists of the identity of the message owner. This could first enable the protocol to be able to run under the post-specified peer setting [13] (i.e., without knowing the identity or public key of the communication partner). On the second, including the identities in the session key computation could resist with the unknown key share attacks [14]. Consider the case that the adversary replaces the public key pk_{id_3} with pk_{id_1}, i.e., $pk_{\mathsf{id}_3} = pk_{\mathsf{id}_1}$. Then the adversary masquerades as id_3 to execute a session interacting with the session $\pi^t_{\mathsf{id}_2}$ but using all messages from $\pi^s_{\mathsf{id}_1}$; and forwards all messages from $\pi^t_{\mathsf{id}_2}$ to $\pi^s_{\mathsf{id}_1}$. As a result, the party id_2 would mistakenly believe the session key is shared with id_3. But the party id_1 ends up believing that she shares a session key with id_2. In particular, the oracles $\pi^t_{\mathsf{id}_2}$ and $\pi^s_{\mathsf{id}_1}$ share the same session key, i.e., $K^s_{\mathsf{id}_1} = K^t_{\mathsf{id}_2}$ but $\mathsf{pid}^t_{\mathsf{id}_2} \neq (\mathsf{id}_1, pk_{\mathsf{id}_1})$. Readers can check this problem themselves.

id_1		id_2
Key Generation:		
$rs_{\mathsf{id}_1} \xleftarrow{\$} \mathcal{RS}_{\mathsf{SIG}}$		$rs_{\mathsf{id}_2} \xleftarrow{\$} \mathcal{RS}_{\mathsf{SIG}}$
$(sk^{sig}_{\mathsf{id}_1}, vk^{sig}_{\mathsf{id}_1}) \leftarrow \mathsf{SIG.Gen}(1^\lambda, rs_{\mathsf{id}_1})$		$(sk^{sig}_{\mathsf{id}_2}, vk^{sig}_{\mathsf{id}_2}) \leftarrow \mathsf{SIG.Gen}(1^\lambda, rs_{\mathsf{id}_2})$
$r_{\mathsf{id}_1} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$		$r_{\mathsf{id}_2} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$
$(sk^{nike}_{\mathsf{id}_1}, pk^{nike}_{\mathsf{id}_1}) \leftarrow \mathsf{NIKE.Gen}(r_{\mathsf{id}_1})$		$(sk^{nike}_{\mathsf{id}_2}, pk^{nike}_{\mathsf{id}_2}) \leftarrow \mathsf{NIKE.Gen}(r_{\mathsf{id}_2})$
$sk_{\mathsf{id}_1} = (sk^{nike}_{\mathsf{id}_1}, sk^{sig}_{\mathsf{id}_1})$		$sk_{\mathsf{id}_2} = (sk^{nike}_{\mathsf{id}_2}, sk^{sig}_{\mathsf{id}_2})$
$pk_{\mathsf{id}_1} = (pk^{nike}_{\mathsf{id}_1}, vk^{sig}_{\mathsf{id}_1})$		$pk_{\mathsf{id}_2} = (pk^{nike}_{\mathsf{id}_2}, vk^{sig}_{\mathsf{id}_2})$
Protocol Execution:		
$er_{\mathsf{id}_1} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$		$er_{\mathsf{id}_2} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$
$(esk_{\mathsf{id}_1}, epk_{\mathsf{id}_1}) \leftarrow \mathsf{NIKE.Gen}(er_{\mathsf{id}_1})$		$(esk_{\mathsf{id}_2}, epk_{\mathsf{id}_2}) \leftarrow \mathsf{NIKE.Gen}(er_{\mathsf{id}_2})$
$\sigma_{\mathsf{id}_1} \xleftarrow{\$} \mathsf{SIG.Sign}(sk^{sig}_{\mathsf{id}_1}, epk_{\mathsf{id}_1})$		$\sigma_{\mathsf{id}_2} \xleftarrow{\$} \mathsf{SIG.Sign}(sk^{sig}_{\mathsf{id}_2}, epk_{\mathsf{id}_2})$
	$\mathsf{id}_1, epk_{\mathsf{id}_1}, \sigma_{\mathsf{id}_1} \longrightarrow$	
	$\longleftarrow \mathsf{id}_2, epk_{\mathsf{id}_2}, \sigma_{\mathsf{id}_2}$	
reject if $\mathsf{SIG.Vfy}(vk^{sig}_{\mathsf{id}_2}, \sigma_{\mathsf{id}_2}, epk_{\mathsf{id}_2}) \neq 1$		reject if $\mathsf{SIG.Vfy}(vk^{sig}_{\mathsf{id}_1}, \sigma_{\mathsf{id}_1}, epk_{\mathsf{id}_1}) \neq 1$
$T := \mathsf{sort}(\mathsf{id}_1, pk_{\mathsf{id}_1}, epk_{\mathsf{id}_1}, \sigma_{\mathsf{id}_1}, \mathsf{id}_2, pk_{\mathsf{id}_2}, epk_{\mathsf{id}_2}, \sigma_{\mathsf{id}_2})$		$T := \mathsf{sort}(\mathsf{id}_1, pk_{\mathsf{id}_1}, epk_{\mathsf{id}_1}, \sigma_{\mathsf{id}_1}, \mathsf{id}_2, pk_{\mathsf{id}_2}, epk_{\mathsf{id}_2}, \sigma_{\mathsf{id}_2})$
$k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_1}, pk^{nike}_{\mathsf{id}_2}), T)$		$k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_2}, pk^{nike}_{\mathsf{id}_1}), T)$
$k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_1}, epk_{\mathsf{id}_2}), T)$		$k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_2}, pk^{nike}_{\mathsf{id}_1}), T)$
$k^{epk_{\mathsf{id}_1}}_{pk^{nike}_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_1}, pk_{\mathsf{id}_2}), T)$		$k^{epk_{\mathsf{id}_1}}_{pk^{nike}_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_2}, epk_{\mathsf{id}_1}), T)$
$k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_1}, epk_{\mathsf{id}_2}), T)$		$k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_2}, epk_{\mathsf{id}_1}), T)$
accept $K := k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$		accept $K := k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$

Fig. 2: An Improved Scheme

More specifically, our improved scheme is shown in Figure 2.

Theorem 1. Suppose that the pseudo-random function family PRF is $(d, t, \epsilon_{\mathsf{PRF}})$-secure, the non-interactive key exchange scheme NIKE is $(t, \epsilon_{\mathsf{NIKE}})$-secure, and the deterministic signature scheme SIG is $(d, t, \epsilon_{\mathsf{SIG}})$-secure, as defined in Section2. Then the proposed protocol is $(t', \epsilon_{\mathsf{AKE}})$-session-key-secure such that $t' \approx t$ and $\epsilon_{\mathsf{AKE}} \leq \epsilon_{\mathsf{NIKE}} + \ell \cdot \epsilon_{\mathsf{SIG}} + 4(d\ell)^2 \cdot (\epsilon_{\mathsf{NIKE}} + \epsilon_{\mathsf{PRF}})$.

Proof. Generally speaking, we are going to gradually reduce the security of our proposed scheme to that of the underlying building blocks. The proof is shown by a series of games. Let S_ξ be the event that the adversary wins the security experiment in Game ξ, and $\mathsf{ADV}_\xi := \Pr[\mathsf{S}_\xi] - 1/2$ denote the advantage of $\mathcal{A}$ in Game ξ.

Game 0. The first game is the real security experiment. Thus we have that

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

All queries in this game are simulated honestly in terms of the protocol specification. Specifically, each $\mathsf{Corrupt}(\mathsf{id}_i, pk^*)$ query returns the long-term key $sk_{\mathsf{id}_i} = (sk^{nike}_{\mathsf{id}_i}, sk^{sig}_{\mathsf{id}_i})$; and each $\mathsf{RevealEphKey}(\mathsf{id}_i, s)$ query returns the ephemeral secret key $esk^s_{\mathsf{id}_i}$ generated by the oracle $\pi^s_{\mathsf{id}_i}$. The protocol message returned by each $\mathsf{Send}(\mathsf{id}_i, s, \cdot)$ comprises of $(\mathsf{id}_i, epk^s_{\mathsf{id}_i}, \sigma^s_{\mathsf{id}_i})$.

Game 1. This game proceeds exactly as the previous game, but the challenger aborts if: the test oracle oracle $\pi^{s^*}_{\mathsf{id}_i}$ received a protocol message $m' = (\mathsf{id}_j, epk'_{\mathsf{id}_j}, \sigma'_{\mathsf{id}_j})$ such that $\mathsf{SIG.Vfy}(vk^{sig}_{\mathsf{id}_j}, \sigma'_{\mathsf{id}_j}, epk'_{\mathsf{id}_j}) = 1$, but the tuple $(epk'_{\mathsf{id}_j}, \sigma'_{\mathsf{id}_j})$ is not sent by any oracle of id_j. Note that the test oracle $\pi^{s^*}_{\mathsf{id}_i}$ must be fresh. This implies that the party id_j must be uncorrupted when the message m' is received by $\pi^{s^*}_{\mathsf{id}_i}$. The challenger could check the abort rule when the Test query is asked.

If the challenger aborts with overwhelming probability, then we could construct an efficient signature forger $\mathcal{F}$ to break the security of SIG by running the adversary $\mathcal{A}$ as a subroutine. To do this, $\mathcal{F}$ simulates as the AKE challenger for $\mathcal{A}$ as in Game 1 but with a few modifications. $\mathcal{F}$ first guesses the partner id_j of the test oracle at the beginning of the game. If $\mathcal{F}$ fails in this guess, it aborts the game. The probability of such correct guess is at least $1/\ell$. In the following, we assume that the guess of $\mathcal{F}$ is right. Note that $\mathcal{F}$ is given a challenge signature verification key vk^{sig^*} by the signature challenger. The signature verification key of id_j is assigned as $vk^{sig}_{\mathsf{id}_j} := vk^{sig^*}$. All other long-term key pairs in this game are generated by $\mathcal{F}$ honestly following the protocol specification. This implies that $\mathcal{F}$ knows all honest parties' long-term secret keys except for $sk^{sig}_{\mathsf{id}_j}$. Meanwhile, $\mathcal{F}$ can compute all protocol messages which are non-related to $sk^{sig}_{\mathsf{id}_j}$. As for each signature of id_j, $\mathcal{F}$ queries the signing oracle $\mathsf{SIG}(sk^{sig^*}, \cdot)$ with an input $epk^t_{\mathsf{id}_j}$ (which is generated by $\mathcal{F}$). So far, the simulation of $\mathcal{F}$ is perfect.

When the test oracle $\pi^{s^*}_{\mathsf{id}_i}$ receives a tuple $(epk'_{\mathsf{id}_j}, \sigma'_{\mathsf{id}_j})$ such that σ'_{id_j} is a valid signature for epk'_{id_j} with respect to $vk^{sig}_{\mathsf{id}_j}$ but $(epk'_{\mathsf{id}_j}, \sigma'_{\mathsf{id}_j})$ is not any tuple recorded in SList. Then, $\mathcal{F}$ could win in the signature security experiment by outputting $(epk'_{\mathsf{id}_j}, \sigma'_{\mathsf{id}_j})$. By applying the security of the signature scheme, we have that

$$\mathsf{ADV}_0 \leq \mathsf{ADV}_1 + \ell \cdot \epsilon_{\mathsf{SIG}}.$$

As a result, the test oracle $\pi^{s^*}_{\mathsf{id}_i}$ in this game always has an origin-oracle $\pi^{z^*}_{\mathsf{id}_j}$ before the party id_j is corrupted.

Game 2. The challenger proceeds as the previous game but it does the following two modifications:

- Generate all $\ell+\ell d$ NIKE key pairs $\{pk_i, sk_i\}_{i\in[\ell+\ell d]}$ (which may used later as either long-term or ephemeral key) at the beginning of the security experiment;
- Abort if: there are two distinct indexes $(i, j) \in [\ell + \ell d]$ such that $pk_i = pk_j$.

The first change is possible for an ORKE protocol. Since each oracle's protocol message (including ephemeral key) is independently generated. The second change could ensure that there is no collision among all NIKE key pairs during the subsequent AKE security experiment.

It is straightforward to see that if the above abort event occurs with non-negligible probability. Then we can construct an efficient algorithm $\mathcal{E}$ to break the NIKE scheme as follows. Instead of generating all NIKE keys on its own, $\mathcal{E}$ generates each NIKE public key pk_τ via asking a $\mathsf{RegH}(\tau)$ query. When there are two NIKE public keys, e.g., pk_i and pk_j, such that $i \neq j$ but $pk_i = pk_j$, $\mathcal{E}$ can break the NIKE security as follows:

- Ask a $\mathsf{Test}^{nike}(i, z)$ query with indexes such that $i \neq z \neq j$, to get a challenge NIKE shared key K^{nike^*};
- Ask a $\mathsf{EXT}(j)$ query to get the secret key sk_j which is identical to sk_i;
- If $\mathsf{ShKey}(sk_j, pk_z) = K^{nike^*}$ then return 1, otherwise return 0.

Note that all NIKE public keys generated in the above way have the same distribution as in the previous game. Due to the security of the NIKE scheme, the abort event would not occur with overwhelming probability. So that we have that

$$\mathsf{ADV}_1 \leq \mathsf{ADV}_2 + \epsilon_{\mathsf{NIKE}}.$$

In this game, all NIKE key pairs are uniquely generated. This also implies that the test oracle always has a unique origin-oracle, thanks to the uniqueness of each ephemeral NIKE key.

Game 3. Note that the adversary is not allowed to ask RevealKey query to the test oracle or its partner oracle (if it exists). But the RevealEphKey and Corrupt queries have various combinations in terms of the oracle freshness definition. Namely, we have the following disjointed freshness cases:

- **C1**: $\mathcal{A}$ does not query both $\mathsf{RevealEphKey}(\mathsf{id}_i, s^*)$ and $\mathsf{RevealEphKey}(\mathsf{id}_j, z^*)$.
- **C2** : $\mathcal{A}$ does not query both $\mathsf{Corrupt}(\mathsf{id}_i, pk^*)$ and $\mathsf{RevealEphKey}(\mathsf{id}_j, z^*)$.
- **C3** : $\mathcal{A}$ does not query both $\mathsf{Corrupt}(\mathsf{id}_i, pk^*)$ and $\mathsf{Corrupt}(\mathsf{id}_j, pk^*)$.
- **C4** : $\mathcal{A}$ does not query both $\mathsf{RevealEphKey}(\mathsf{id}_i, s^*)$ and $\mathsf{Corrupt}(\mathsf{id}_j, pk^*)$.

Note that only one of the above cases will occur in each simulation of the security experiment. Each freshness case is associated with at least two (either long-term or ephemeral) NIKE secret keys which are uncompromised by the adversary.

We change this game from the previous game by adding an another abort rule. Namely, the challenger aborts if it fails to guess (at the beginning of the game): (i) the test oracle and its origin-oracle, and (ii) one of the above freshness cases. Since there are 4 fresh cases and ℓ parties at all, and at most d oracles for each party, then the probability of a correct guess is at least $1/4(d\ell)^2$. Thus we have that

$$\mathsf{ADV}_2 \leq 4(d\ell)^2 \cdot \mathsf{ADV}_3.$$

Game 4. Note that each freshness case involves two unexposed NIKE secret keys. In this game, we replace the NIKE shared key which is computed based on those unexposed NIKE secret keys with a random key $\widetilde{K}^{nike}$. If there exists an adversary $\mathcal{A}$ distinguishing this game from the previous game, then we can make use of $\mathcal{A}$ to build an efficient algorithm $\mathcal{D}$ to break the security of NIKE. Again $\mathcal{D}$ will simulate the security experiment for $\mathcal{A}$. $\mathcal{D}$ first asks $\mathsf{RegH}(1)$ and $\mathsf{RegH}(2)$ queries to obtain two challenged NIKE public keys (pk_1^*, pk_2^*) (from the NIKE challenger). The test NIKE shared key K^{nike^*} is obtained by asking query $K^{nike^*} := \mathsf{Test}^{nike}(1,2)$. The goal of $\mathcal{D}$ is to distinguish whether or not K^{nike^*} is the real shared key with respect to (pk_1^*, pk_2^*). Due to the correct guess as in the previous game, $\mathcal{D}$ knows in advance which freshness case will occur during the game. So that $\mathcal{D}$ is able to embed these challenged NIKE public keys as follows:

- **C1** : $\mathcal{D}$ generates the ephemeral public keys $epk_{\mathsf{id}_i}^{s^*} := pk_1^*$ and $epk_{\mathsf{id}_j}^{z^*} := pk_2^*$. $\mathcal{D}$ substitutes K^{nike^*} for the value $\mathsf{ShKey}(esk_{\mathsf{id}_i}^{s^*}, epk_{\mathsf{id}_j}^{z^*})$ (c.f. $\mathsf{ShKey}(esk_{\mathsf{id}_j}^{z^*}, epk_{\mathsf{id}_i}^{s^*})$). As for the value $\mathsf{ShKey}(esk_{\mathsf{id}_j}^{z^*}, pk')$ where pk' is an arbitrary (either long-term or ephemeral) public key chosen by $\mathcal{A}$, it is computed as below:
  - $\mathcal{D}$ chooses an index $\tau \notin \mathsf{HID}$ and asks $\mathsf{RegC}^{nike}(\tau, pk')$.
  - $\mathcal{D}$ can compute $\mathsf{ShKey}(esk_{\mathsf{id}_j}^{z^*}, pk')$ by asking $\mathsf{RevealKey}^{nike}(2, \tau)$.
- **C2** : $\mathcal{D}$ generates the long-term key $pk_{\mathsf{id}_i}^{nike} := pk_1^*$ and the ephemeral public key $epk_{\mathsf{id}_j}^{z^*} := pk_2^*$. $\mathcal{D}$ substitutes K^{nike^*} for the value $\mathsf{ShKey}(sk_{\mathsf{id}_i}^{nike}, epk_{\mathsf{id}_j}^{z^*})$. With respect to the values $\mathsf{ShKey}(esk_{\mathsf{id}_j}^{z^*}, pk')$ and $\mathsf{ShKey}(sk_{\mathsf{id}_i}^{nike}, pk')$ where pk' is chosen by $\mathcal{A}$, they are computed analogously as in the case **C1** (i.e., exploiting the RegC^{nike} and $\mathsf{RevealKey}^{nike}$ queries).
- **C3** : $\mathcal{D}$ generates the long-term public keys $pk_{\mathsf{id}_i}^{nike} := pk_1^*$ and $pk_{\mathsf{id}_j}^{nike} := pk_2^*$. $\mathcal{D}$ substitutes K^{nike^*} for the value $\mathsf{ShKey}(sk_{\mathsf{id}_i}^{nike}, pk_{\mathsf{id}_j}^{nike})$. As for the values $\mathsf{ShKey}(sk_{\mathsf{id}_i}^{nike}, pk')$ and $\mathsf{ShKey}(sk_{\mathsf{id}_j}^{nike}, pk')$ where pk' is chosen by $\mathcal{A}$, they are computed analogously as in the case **C1**.
- **C4** : $\mathcal{D}$ generates the ephemeral public key $epk_{\mathsf{id}_i}^{s^*} := pk_1^*$ and the long-term public key $pk_{\mathsf{id}_j}^{nike} := pk_2^*$. $\mathcal{D}$ substitutes K^{nike^*} for the value $\mathsf{ShKey}(esk_{\mathsf{id}_i}^{s^*}, pk_{\mathsf{id}_j}^{nike})$. With respect to the value $\mathsf{ShKey}(sk_{\mathsf{id}_j}^{nike}, pk')$ where pk' is chosen by $\mathcal{A}$, it is computed analogously as in the case **C1**.

Due to the above changes, $\mathcal{D}$ can compute all NIKE shared keys involving the challenge NIKE public keys. All other (ephemeral or long-term or NIKE shared key) keys are simulated by $\mathcal{D}$ with secrets of her own choice. Meanwhile, $\mathcal{D}$ could answer the oracle queries as in the previous game but with the above modified values. If K^{nike^*} is a random key then the simulation of $\mathcal{D}$ is equivalent to this game. Otherwise, it is identical to the previous game. Applying the security of NIKE, we therefore obtain that

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

Game 5. In this game, we change the function $\mathsf{PRF}(\widetilde{K}^{nike}, T_{\mathsf{id}_i}^{s^*})$ of the test oracle (as well as its partner oracle) to be a truly random function $\mathsf{RF}(T_{\mathsf{id}_i}^{s^*})$. Recall that the seed $\widetilde{K}^{nike}$ is the NIKE shared key which is random and computed based on unexposed NIKE secret keys. Then any PPT adversary $\mathcal{A}$ distinguishing this game from the previous game can be used to build an efficient algorithm $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ to break the security of PRF.

The sub-algorithm $\mathcal{B}_1$ could first generate all values in $T^{s^*}_{\mathsf{id}_i} = \mathsf{sort}(\mathsf{id}_i, pk_{\mathsf{id}_i}, epk^{s^*}_{\mathsf{id}_i}, \sigma^{s^*}_{\mathsf{id}_i}, \mathsf{id}_j, pk_{\mathsf{id}_j}, epk^{z^*}_{\mathsf{id}_j}, \sigma^{z^*}_{\mathsf{id}_j})$ using the random long-term and ephemeral secret keys (i.e., $sk^{s^*}_{\mathsf{id}_i}$, $sk^{z^*}_{\mathsf{id}_j}$, $esk^{s^*}_{\mathsf{id}_i}$ and $esk^{z^*}_{\mathsf{id}_j}$) of her own choice. Note that all values in $T^{s^*}_{\mathsf{id}_i}$ can be pre-generated before running the security experiment. Because each oracle's protocol message can be generated independently. Then $\mathcal{B}_1$ sets $x^* = T^{s^*}_{\mathsf{id}_i}$ and $st = (sk^{s^*}_{\mathsf{id}_i}, sk^{z^*}_{\mathsf{id}_j}, esk^{s^*}_{\mathsf{id}_i}, esk^{z^*}_{\mathsf{id}_j}, T^{s^*}_{\mathsf{id}_i})$. Upon receiving the challenge message x^*, the PRF challenger returns the challenge value V^*. The job of $\mathcal{B}$ is to distinguish whether or not $V^* = \mathsf{PRF}(k^*, T^{s^*}_{\mathsf{id}_i})$. One could consider that $k^* = \widetilde{K}^{nike}$.

Next, the sub-algorithm $\mathcal{B}_2$ takes as input (st, V^*) and simulates the AKE game for $\mathcal{A}$. Basically, $\mathcal{B}_2$ simulates the game as before, except for the following modifications:

- The long-term public keys of id_i and id_j and the ephemeral public keys of $\pi^{s^*}_{\mathsf{id}_i}$ and $\pi^{z^*}_{\mathsf{id}_j}$ are assigned according to the corresponding values in st.
- The value $\mathsf{PRF}(\widetilde{K}^{nike}, T^{s^*}_{\mathsf{id}_i})$ which is supposed to be computed by the test oracle and its partner oracle is replaced with V^*.
- For the computation $\mathsf{PRF}(\widetilde{K}^{nike}, T_\iota)$ of any oracle such that $T_\iota \neq T^{s^*}_{\mathsf{id}_i}$, $\mathcal{B}_2$ computes its value by querying $\mathcal{FN}(T_\iota)$.

The other long-term keys and protocol messages will be generated using the secrets chosen by $\mathcal{B}_2$. Meanwhile, we have fact that only two partnered oracles would share the same protocol message transcript T. This is guaranteed by the uniqueness of identity and ephemeral public key. We consider the following situations concerning the protocol message of the test oracle. When the adversary replays the test oracle's message $m^{s^*}_{\mathsf{id}_i} = (\mathsf{id}_i, epk^{s^*}_{\mathsf{id}_i}, \sigma^{s^*}_{\mathsf{id}_i})$ to another oracle $\pi^t_{\mathsf{id}_u}$ which outputs a message $m^t_{\mathsf{id}_u}$. If $\pi^t_{\mathsf{id}_u}$ is not the origin-oracle of $\pi^{s^*}_{\mathsf{id}_i}$, then $m^t_{\mathsf{id}_u}$ must be distinct to $m^{z^*}_{\mathsf{id}_j}$ (i.e., the origin-oracle's message) – due to the uniqueness of the ephemeral key. In addition, any modification of $m^{s^*}_{\mathsf{id}_i}$ (e.g., replacing id_i with another identity id_v) will result in a protocol transcript T' which is distinct to $T^{s^*}_{\mathsf{id}_i}$. Similarly, the adversary cannot manipulate $m^{s^*}_{\mathsf{id}_i}$ to lead to non-partnered oracles having the same protocol transcript. This also implies that the value V^* would be only used by the test oracle and its partner oracle. For example, if both secrets $esk^{s^*}_{\mathsf{id}_i}$ and $esk^{z^*}_{\mathsf{id}_j}$ are not exposed. The value $\mathsf{PRF}(\mathsf{ShKey}(esk^{s^*}_{\mathsf{id}_i}, epk^{z^*}_{\mathsf{id}_j}), T^{s^*}_{\mathsf{id}_i})$ would be replace with V^*. If the origin-oracle $\pi^{z^*}_{\mathsf{id}_j}$ has no matching session to the test oracle, but should compute $\mathsf{ShKey}(esk^{s^*}_{\mathsf{id}_i}, epk^{z^*}_{\mathsf{id}_j})$. Then $\mathcal{B}_2$ would replace the value $\mathsf{PRF}(\mathsf{ShKey}(esk^{s^*}_{\mathsf{id}_i}, epk^{z^*}_{\mathsf{id}_j}), T^{z^*}_{\mathsf{id}_j})$ with the value returned by $\mathcal{FN}(T^{z^*}_{\mathsf{id}_j})$. Due to $T^{s^*}_{\mathsf{id}_j} \neq T^{z^*}_{\mathsf{id}_j}$, we must have that $\mathcal{FN}(T^{z^*}_{\mathsf{id}_j}) \neq V^*$. This is enough to thwart our PFS attack against the BJS scheme.

If $V^* = \mathsf{PRF}(k^*, T^{s^*}_{\mathsf{id}_i})$ then the simulation of $\mathcal{B}_2$ is identical to the previous game. Otherwise, it is identical to this game. The output bit b' of $\mathcal{A}$ is taken as the output of $\mathcal{B}_2$. By applying the security of PRF, we therefore have that

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

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

Sum up the probabilities from Game 0 to Game 5, we obtained the overall result of this theorem. The running time of $\mathcal{B}$, $\mathcal{D}$, $\mathcal{E}$ and $\mathcal{F}$ is equal to the running time of $\mathcal{A}$ plus the cost on simulating the security experiment for $\mathcal{A}$.

5. Conclusion

We conclude that the BJS scheme is not secure in the eCK-PFS model. A PFS attack has been shown, which enables us to falsify the security proof of the BJS scheme. An improved scheme has been proposed with a slight modification. We hope our results would help researchers to avoid similar problems while considering the security for one-round key exchange protocol.

Acknowledgment

This study was supported by National Natural Science Foundation of China (Grant No. 11647097), Research Project of Humanities and Social Sciences of Ministry of Education of China (Grant No. 16YJC870018), and Academy of Finland (Grant No. 303578).

6. References

[1] Whitfield Diffie, Paul C. Oorschot, and Michael J. Wiener.: 'Authentication and authenticated key exchanges'. *Des. Codes Cryptography*, 1992, 2 (2), pp. 107–125.

[2] Florian Bergsma, Tibor Jager, and Jörg Schwenk.: 'One-round key exchange with strong security: An efficient and generic construction in the standard model'. Proceedings of PKC 2015, Gaithersburg, MD, USA, March 30 - April 1, 2015, pp. 477–494.

[3] Simon Blake-Wilson, Don Johnson, and Alfred Menezes.: 'Key agreement protocols and their security analysis'. Proceedings of IMA International Conference on Crytography and Coding 1997, Cirencester, UK, December 17–19, 1997, pp. 30–45, 1997.

[4] Cas J. F. Cremers and Michele Feltz.: 'Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal'. Proceedings of ESORICS 2012: 17th European Symposium on Research in Computer Security,Pisa, Italy, September 2012, pp. 734–751.

[5] Hugo Krawczyk.: 'HMQV: A high-performance secure Diffie-Hellman protocol'. Proceedings of CRYPTO 2005, Santa Barbara, CA, USA, August 2005, pp. 546–566.

[6] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin.: 'Stronger security of authenticated key exchange'. Proceedings of ProvSec 2007, Wollongong, Australia, November 2007, pp. 1–16.

[7] Colin Boyd and Juanma González Nieto.: 'On forward secrecy in one-round key exchange'. Proceedings of IMA International Conference on Crytography and Coding 2011, Oxford, UK, December 12-15, 2011, pp. 451–468.

[8] Eduarda SV Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson.: 'Non-interactive key exchange. Proceedings of PKC 2013, Nara, Japan, February 26 – March 1, 2013, pp. 254–271.

[9] Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock.: 'Errors in computational complexity proofs for protocols'. Proceedings of ASIACRYPT 2005, Chennai, India, December 4-8, 2005, pp. 624–643.

[10] Wei-Chuen Yau, Raphael C-W Phan, Bok-Min Goi, and Swee-Huay Heng.: Cryptanalysis of a provably secure cross-realm client-to-client password-authenticated key agreement protocol of cans'09. Proceedings of CANS 2011, Sanya, China, December 10-12, 2011, pp. 172–184.

[11] Zheng Yang and Shuangqing Li.: 'On security analysis of an after-the-fact leakage resilient key exchange protocol'. *Inf. Process. Lett.*, 2016, 116, (1), pp. 33–40.

[12] Eduarda SV Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson.: 'Non-interactive key exchange'. Cryptology ePrint Archive, Report 2012/732, 2012. `http://eprint.iacr.org/`.

[13] Ran Canetti and Hugo Krawczyk.: 'Analysis of key-exchange protocols and their use for building secure channels'. Proceedings of EUROCRYPT 2001, Innsbruck, Austria, May 2001, pp. 453–474.

[14] Simon Blake-Wilson and Alfred Menezes.: 'Unknown key-share attacks on the station-to-station (sts) protocol'. Proceedings of PKC'99, Kamakura, Japan, March 1–3, 1999, pp 154–170.

[15] Zheng Yang.: 'Efficient eck-secure authenticated key exchange protocols in the standard model'. Proceedings of ICICS 2013, Beijing, China, 2013, pp. 185–193.

[16] Florian Bergsma, Tibor Jager, and Jörg Schwenk. (talk slides) one-round key exchange with strong security: An efficient and generic construction in the standard model, 2015.

Key Generation:

id_1	id_2
$rs_{\mathsf{id}_1} \xleftarrow{\$} \mathcal{RS}_{\mathsf{SIG}}$	$rs_{\mathsf{id}_2} \xleftarrow{\$} \mathcal{RS}_{\mathsf{SIG}}$
$(sk^{sig}_{\mathsf{id}_1}, vk^{sig}_{\mathsf{id}_1}) \leftarrow \mathsf{SIG.Gen}(1^\lambda, rs_{\mathsf{id}_1})$	$(sk^{sig}_{\mathsf{id}_2}, vk^{sig}_{\mathsf{id}_2}) \leftarrow \mathsf{SIG.Gen}(1^\lambda, rs_{\mathsf{id}_2})$
$r_{\mathsf{id}_1} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$	$r_{\mathsf{id}_2} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$
$(sk^{nike}_{\mathsf{id}_1}, pk^{nike}_{\mathsf{id}_1}) \leftarrow \mathsf{NIKE.Gen}(r_{\mathsf{id}_1})$	$(sk^{nike}_{\mathsf{id}_2}, pk^{nike}_{\mathsf{id}_2}) \leftarrow \mathsf{NIKE.Gen}(r_{\mathsf{id}_2})$
$sk_{\mathsf{id}_1} = (sk^{nike}_{\mathsf{id}_1}, sk^{sig}_{\mathsf{id}_1})$	$sk_{\mathsf{id}_2} = (sk^{nike}_{\mathsf{id}_2}, sk^{sig}_{\mathsf{id}_2})$
$pk_{\mathsf{id}_1} = (pk^{nike}_{\mathsf{id}_1}, vk^{sig}_{\mathsf{id}_1})$	$pk_{\mathsf{id}_2} = (pk^{nike}_{\mathsf{id}_2}, vk^{sig}_{\mathsf{id}_2})$

Protocol Execution:

id_1		id_2
$er_{\mathsf{id}_1} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$		$er_{\mathsf{id}_2} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$
$(esk_{\mathsf{id}_1}, epk_{\mathsf{id}_1}) \leftarrow \mathsf{NIKE.Gen}(er_{\mathsf{id}_1})$		$(esk_{\mathsf{id}_2}, epk_{\mathsf{id}_2}) \leftarrow \mathsf{NIKE.Gen}(er_{\mathsf{id}_2})$
$\sigma_{\mathsf{id}_1} \xleftarrow{\$} \mathsf{SIG.Sign}(sk^{sig}_{\mathsf{id}_1}, epk_{\mathsf{id}_1})$		$\sigma_{\mathsf{id}_2} \xleftarrow{\$} \mathsf{SIG.Sign}(sk^{sig}_{\mathsf{id}_2}, epk_{\mathsf{id}_2})$
	$epk_{\mathsf{id}_1}, \sigma_{\mathsf{id}_1} \rightarrow$	
	$\leftarrow epk_{\mathsf{id}_2}, \sigma_{\mathsf{id}_2}$	
reject if $\mathsf{SIG.Vfy}(vk^{sig}_{\mathsf{id}_2}, \sigma_{\mathsf{id}_2}, epk_{\mathsf{id}_2}) \neq 1$		reject if $\mathsf{SIG.Vfy}(vk^{sig}_{\mathsf{id}_1}, \sigma_{\mathsf{id}_1}, epk_{\mathsf{id}_1}) \neq 1$
$T := \mathsf{sort}(epk_{\mathsf{id}_1}, epk_{\mathsf{id}_2})$		$T := \mathsf{sort}(epk_{\mathsf{id}_1}, epk_{\mathsf{id}_2})$
$k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_1}, pk^{nike}_{\mathsf{id}_2}), T)$		$k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_2}, pk^{nike}_{\mathsf{id}_1}), T)$
$k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_1}, epk_{\mathsf{id}_2}), T)$		$k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_2}, pk^{nike}_{\mathsf{id}_1}), T)$
$k^{epk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_1}, pk^{nike}_{\mathsf{id}_2}), T)$		$k^{epk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_2}, epk_{\mathsf{id}_1}), T)$
$k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_1}, epk_{\mathsf{id}_2}))$		$k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_2}, epk_{\mathsf{id}_1}))$
accept $K := k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$		accept $K := k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$

Key Generation:

id_1	id_2
$rs_{\mathsf{id}_1} \xleftarrow{\$} \mathcal{RS}_{\mathsf{SIG}}$	$rs_{\mathsf{id}_2} \xleftarrow{\$} \mathcal{RS}_{\mathsf{SIG}}$
$(sk^{sig}_{\mathsf{id}_1}, vk^{sig}_{\mathsf{id}_1}) \leftarrow \mathsf{SIG.Gen}(1^\lambda, rs_{\mathsf{id}_1})$	$(sk^{sig}_{\mathsf{id}_2}, vk^{sig}_{\mathsf{id}_2}) \leftarrow \mathsf{SIG.Gen}(1^\lambda, rs_{\mathsf{id}_2})$
$r_{\mathsf{id}_1} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$	$r_{\mathsf{id}_2} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$
$(sk^{nike}_{\mathsf{id}_1}, pk^{nike}_{\mathsf{id}_1}) \leftarrow \mathsf{NIKE.Gen}(r_{\mathsf{id}_1})$	$(sk^{nike}_{\mathsf{id}_2}, pk^{nike}_{\mathsf{id}_2}) \leftarrow \mathsf{NIKE.Gen}(r_{\mathsf{id}_2})$
$sk_{\mathsf{id}_1} = (sk^{nike}_{\mathsf{id}_1}, sk^{sig}_{\mathsf{id}_1})$	$sk_{\mathsf{id}_2} = (sk^{nike}_{\mathsf{id}_2}, sk^{sig}_{\mathsf{id}_2})$
$pk_{\mathsf{id}_1} = (pk^{nike}_{\mathsf{id}_1}, vk^{sig}_{\mathsf{id}_1})$	$pk_{\mathsf{id}_2} = (pk^{nike}_{\mathsf{id}_2}, vk^{sig}_{\mathsf{id}_2})$

Protocol Execution:

id_1		id_2
$er_{\mathsf{id}_1} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$		$er_{\mathsf{id}_2} \xleftarrow{\$} \mathcal{R}_{\mathsf{NIKE}}$
$(esk_{\mathsf{id}_1}, epk_{\mathsf{id}_1}) \leftarrow \mathsf{NIKE.Gen}(er_{\mathsf{id}_1})$		$(esk_{\mathsf{id}_2}, epk_{\mathsf{id}_2}) \leftarrow \mathsf{NIKE.Gen}(er_{\mathsf{id}_2})$
$\sigma_{\mathsf{id}_1} \xleftarrow{\$} \mathsf{SIG.Sign}(sk^{sig}_{\mathsf{id}_1}, epk_{\mathsf{id}_1})$		$\sigma_{\mathsf{id}_2} \xleftarrow{\$} \mathsf{SIG.Sign}(sk^{sig}_{\mathsf{id}_2}, epk_{\mathsf{id}_2})$
	$\mathsf{id}_1, epk_{\mathsf{id}_1}, \sigma_{\mathsf{id}_1} \xrightarrow{\quad}$	
	$\xleftarrow{\quad} \mathsf{id}_2, epk_{\mathsf{id}_2}, \sigma_{\mathsf{id}_2}$	
reject if $\mathsf{SIG.Vfy}(vk^{sig}_{\mathsf{id}_2}, \sigma_{\mathsf{id}_2}, epk_{\mathsf{id}_2}) \neq 1$		reject if $\mathsf{SIG.Vfy}(vk^{sig}_{\mathsf{id}_1}, \sigma_{\mathsf{id}_1}, epk_{\mathsf{id}_1}) \neq 1$
$T := \mathsf{sort}(\mathsf{id}_1, pk_{\mathsf{id}_1}, epk_{\mathsf{id}_1}, \sigma_{\mathsf{id}_1}, \mathsf{id}_2, pk_{\mathsf{id}_2}, epk_{\mathsf{id}_2}, \sigma_{\mathsf{id}_2})$		$T := \mathsf{sort}(\mathsf{id}_1, pk_{\mathsf{id}_1}, epk_{\mathsf{id}_1}, \sigma_{\mathsf{id}_1}, \mathsf{id}_2, pk_{\mathsf{id}_2}, epk_{\mathsf{id}_2}, \sigma_{\mathsf{id}_2})$
$k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_1}, pk^{nike}_{\mathsf{id}_2}), T)$		$k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_2}, pk^{nike}_{\mathsf{id}_1}), T)$
$k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_1}, epk_{\mathsf{id}_2}), T)$		$k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_2}, pk^{nike}_{\mathsf{id}_1}), T)$
$k^{epk_{\mathsf{id}_1}}_{pk^{nike}_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_1}, pk_{\mathsf{id}_2}), T)$		$k^{epk_{\mathsf{id}_1}}_{pk^{nike}_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(sk^{nike}_{\mathsf{id}_2}, epk_{\mathsf{id}_1}), T)$
$k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_1}, epk_{\mathsf{id}_2}), T)$		$k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} := \mathsf{PRF}(\mathsf{ShKey}(esk_{\mathsf{id}_2}, epk_{\mathsf{id}_1}), T)$
accept $K := k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$		accept $K := k^{pk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{pk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{pk_{\mathsf{id}_2}} \oplus k^{epk_{\mathsf{id}_1}}_{epk_{\mathsf{id}_2}}$