# Authenticated Key Exchange with Synchronized States

Zheng Yang
Zheng.Yang@rub.de

Ruhr University Bochum,
D-44801 Bochum, Germany

**Abstract.** Nowadays, most of sensitive applications over insecure network are protected by some authenticated secure channel which is highly relies on specific authenticated key exchange (AKE) protocol. Nevertheless, the leakage of authentication credential used in AKE protocol somehow result in unauthorized exploitation of credential information via identity impersonation (IDI) attack. To address the problem of IDI, we introduce a new dynamic authentication factor for AKE protocols, i.e., the secret execution states, to either prevent IDI attack by detecting attempts thereof, or limit its consequences by on-line detecting situations of previously unidentified IDI. In this paper, we model the security for authenticated key exchange with synchronized states (AKESS) based on Bellare-Rogaway model, and we particularly formalize the IDI and IDI detection. We propose a generic execution states synchronization framework for AKE, in which we utilize the session key to generate the secret execution states on both sides, and present a new AKESS protocol which is provably secure in the standard model. Our goal is to enhance the security of existing authenticated key exchange with long-lived key (AKELL) protocols by equipping them with the capabilities of both IDI prevention and detection without modifications on those protocols.

**Keywords: authenticated key exchange, impersonation detection, state synchronization, security model**

## 1 Introduction

Authenticated Key Exchange (AKE) protocols are foundation for building secure channel to protect communication over insecure networks. Meantime the parties' identity authentication always relies on some credentials (e.g., password, private key of certificate, fingerprint), which are the primary objectives of attacker. It is trivially to see that the compromised authentication secrets are easy to be exploited by the adversary to launch identity impersonation (IDI) attack as the victim which might result in privacy loss, financial loss, and public discredit etc. Unfortunately, many users learn that their identity had been impersonated after some damage has been done. Whereas the credential issuers always encourage user to armed with the knowledge of how to protect themselves and take action to monitor their accounts periodically on a regular basis. Although that is quite necessary to the users, the consequences of IDI attack might not be perceptible immediately if there is no evident trail to track. Hence one of the motivations of this paper is to design new dynamic authentication factor which is not only to prevent IDI attack by detecting attempts thereof, but also to limit its consequences by on-line detecting events of previously unidentified IDI on protocol level (i.e. automatically detect while executing the protocol).[1]

Consider a commonly used authentication and IDI detection scenario for two honest people in real word, besides proving identity via public identification attributes (e.g., name, ID-number and photo), each parties might require its peer to wisely present some 'secret sign' agreed during last conversation that they involved together. Provide that one party failed to give the correct response,

[1] We prefer the term "identity impersonation" over "identity theft" (IDT) or "identity fraud"(IDF). Although the term "identity impersonation" might be consequence of identity theft, or even of form of fraud, the IDT and IDF encompass wider scope of crimes concerning different identities which are not our focus.

then it might be either impersonated before or an impostor. This scheme is simple and effective, which can be easily applied to cyberspace for automatically on-line detecting previous IDI, by comparing some kind of shared secret states shared between parties. However, there are a number of realistic problems that we need pay attention to, such as bootstrapping of the secret states, resilience of adversary's spy and interference, and synchronizing the secret states, etc. Meanwhile no matter what kind of secret states are equipped, the most important issue should be addressed first is that how to synchronize those secret states (i.e., set-up update strategy and timing), e.g., after the authentication protocol instances have been successfully established or terminated. Therefore the protocol instances always need extra confirmation steps to ensure the update conditions are satisfied. If the most recent shared execution states between two parties are inconsistent then some IDI events might have been occurred for either party. However, we cannot come to such conclusion arbitrarily, since there are numerous factors resulting in state inconsistent. We believe the major reasons that result in state out of synchronization between two parties, include the following:

1. Identity impersonation attack. At some point, the adversary who obtains the long-lived authentication key impersonates the victim $A$ to another honest party $B$.
2. Interference of adversary. The passive adversary without the long-lived secret is also able to disorder the execution state by intervening the communication between honest party $A$ and $B$, e.g. drop the messages.
3. Other situations. For instances, due to the network failure or system crash, etc.

In particular, due to the interference of adversary, the state inconsistent of two parties seems inevitable. Since a party $A$ who sends the last message flow can't know whether or not its last message was received by its partner, so when $A$ update its execution secrets states, it cannot know (due to the above situations) whether or not its partner will do the same update. This asymmetry is an inherent aspect of state synchronization protocols with a fixed number of moves, giving a certain information benefit to the party who refrains from receiving the last confirmation message. Thus, it is necessary to distinguish between illegal state inconstant (e.g., caused by identity impersonation attack) and inevitable synchronization failure (e.g., network fault or missing confirmation message etc.). Besides the impersonation attack detection, another important issue is how to recover the execution states from such legal (unavoidable) states inconsistent to normal ones (i.e. re-synchronizing). Furthermore, we note that the existing pre-shared states mechanism are not withstand the interference of adversary, which can be categorized as: (i) authentication credential based (e.g. evolving authentication key for each protocol instance), (ii) sequential constants. Since the information carried by those two scenarios is so limited, and the protocol would be in-executable once those execution states come to out of synchronization due to synchronization failures as aforementioned.

In this paper, we focus on generic execution states synchronization problem for AKE, which cover the aspects of states' representation, transition, synchronization and fault-tolerance. We also strive to formalize the security of authenticated key exchange with synchronized states (AKESS) and the validity of identity impersonation detection. Instead of designing a new infrastructure, we provide a universal IDI detection solution for existing authenticated key exchange with long-lived key (AKELL) protocols, i.e. it is neither AKELL protocol specified, nor restricted to applications. From a practical standpoint, the synchronized secret execution states can be easily obtained from any AKELL protocols without modification, and don't rely on specific identification credentials issued to parties. But the bootstrap problem of secret states is not our focus. Instead we assume that the parties are able to agree upon the initial secret states (e.g. use out-of-band mechanisms), and we focus on secret states' generation and synchronization during the execution of each protocol instances and the on-line IDI detection.

### 1.1 Related Work

As for the prevention of static password's exposure, the one-time password schemes, e.g., [13,10] are introduced, which typically make use of randomness that can also be seen as a kind of authentication key related execution state. In [16] Shin et al. dedicate to the immunity to the respective leakage of stored authentication secrets (both low-entropy password and long-lived key) from a client side and a server side. In their subsequent work [17], although they introduce a dynamically update solution for authentication secret after establishing the session to provide security against the leakage of stored secrets, they overlook the synchronization failures as mentioned above.

In order to limit the damage consequences of long-lived key's exposure in the public key infrastructure, there are also many proposals including threshold cryptosystems [6], proactive cryptosystems [11], proactive forward-secure schemes [1], key-insulated cryptosystems [9], etc. Although, the schemes proposed in [11,1] involve the issue of secret synchronization (i.e., time-evolved update strategy), they are unable to detect the previous unauthorized using of the key. It is also questionable if the renewed key is still 'fresh' (unknown) to adversary, once the adversary compromised some old keys at some time (e.g., the secret factors of the $RSA$ composite $n$ are also compromised).

With respect to the problem related to identity impersonation detection, Van Oorschot and Stubblebine [19] propose an identity theft detection scheme, whereby users' identity claims are corroborated with trusted claims of these users' location. This scheme has limitations including restriction to on-site (vs. on-line) transactions and loss of user location privacy (i.e., users are geographically tracked). In 2008, D. Nali and P.C. van Oorschot [15] propose a universal infrastructure and protocol so-called CROO (Capture Resilient Online One-time Password) scheme, to either prevent identity fraud (IDF) or identify the cases of previous IDF in the environment of on-line transaction (i.e., between some client and server). This scheme highly depends on the sequential one-time symmetric authentication keys (also severed as kind of execution state) used to generate the one-time password, in which the *j-th* key is derived from *j+1th* key with some key derivation function. Where the initial authentication key is issued by some trusted third party (TTP) which is also in charge of verifying the corresponding *j-th* one-time password and detecting the IDF. However, the CROO scheme never addresses that how to synchronize the authentication keys between the client device and the credential issuer, thus it is incapable of fault-tolerant in practice.

Other works, such as detecting double spending behavior in e-cash system [5]. However the tracing of double-spender is designed for off-line application environment, thus such detection has latency and is not efficient. By contrast, we focus on the issues of on-line mutual authentication with synchronized states and IDI detection.

### 1.2 Contribution

In order to achieve the goals of both IDI attack prevention and on-line detection of previously unidentified IDI, in this paper, we first formalize the execution states of AKE as a dynamic authentication factor. On the second we model the security of authenticated key exchange with synchronized states (AKESS) based on the BR model, which enables us to prove the security of authentication involving both long-lived key and synchronized states. In the security model, we also explicitly formalize the issues on IDI and IDI detection that make us be able to evaluate the validity of the IDI detection capability.

Moreover, we propose a generic framework for synchronizing execution states of AKE, which include the execution secret states generation scheme (i.e., exploiting the session key generated by AKELL protocol), an AKESS protocol and corresponding states synchronization rules. The framework allows for a modular design of new AKESS protocols, using exiting protocols (e.g. TLS,

IPSec IKE), in which the most important case is mutual authentication with synchronized states (MASS) built on top of AKELL channel. This is possible since we require the session key generated by AKELL protocol is secure against passive adversary. Thus the proposed AKESS protocol is able to provide both IDI prevention and detection capability since to launch a IDI attack the adversary need to learn the long-lived key to execute the AKELL protocol and previously established secret states (which are dynamically updated in each protocol instances) to run the MASS protocol for further authentication and IDI detection.

**Notations and Terminology.** We let $\kappa$ denote the security parameter and $1^\kappa$ the string that consists of $\kappa$ ones. The cryptographic primitives used in this paper include: pseudo-random function and message authentication code. The details of their security definitions are presented in the Appendix A.

## 2 Formalism of Execution States Synchronization for AKE

### 2.1 Execution States of AKE

We call each instances of an AKE protocol run at a party a session denoted as $s$ for simple. Technically, a session is an interactive subroutine executed inside a party. At the beginning, we first classify the stage of states synchronization into three categories: $init, established$ and $reset$ , where

- the *init* stage denotes that the secret states need to be initiated between two parties.
- the *established* stage denotes some secret execution states have already been agreed upon at some time,
- the *reset* stage denotes the secret states need to be re-established at next run.

In the subsequent description, we use macros $(INIT, RES)$ to denote the corresponding stages *init* and *reset* respectively (e.g., $INIT := 0$, $RES := 1$). Let $I = I[\kappa]$ and $S = S[\kappa]$ be polynomials in the security parameter $\kappa$, where $I$ be a set of identities which defines the parties who can participate in the AKE protocol, and $S$ be set of sessions. We assume each session maintains its own internal states which comprise of two parts: (i) the current protocol execution states, and (ii) the recorded knowledge of previous protocol execution states supposedly shared by the session participants.

**Definition 1 (Current Protocol Execution States).** *The current execution states for a session $s \in [S]$ executed by party $A$ with its peer $B$ are formed as a tuple $CPES := (ES_s, T_s, \Lambda_s)$, where the implication of each elements are described below:*

1. *The variable $ES_s$ stores the current secret execution states of session $s$.*
2. *The variable $T_s$ stores an ordered list of all messages received and sent by session $s$.*
3. *The variable $\Lambda_s \in \{$accept-ll, accept-session, reject-session, reset, $\perp\}$ records the current execution status, in which*
   - *the accept-ll denotes the acceptance of the authentication related to long-lived key.*
   - *the accept-session denotes the acceptance of current session with valid states authentication.*
   - *the reject-session denotes the current session aborts with rejection. This event might happen at any point during the session execution.*
   - *the reset denotes the incomplete states authentication process without rejection.*
   - *the symbol $\perp$ denotes the empty string and no state changes.*

Whereas, the knowledge of previous protocol execution states are used for authenticating current session which are defined as following.

**Definition 2 (Protocol Execution States Record).** *The recorded execution states for parties* $(A, B) \in I$*, are formed as a tuple* $PESR := (RS_{A,B}, A, lST, lET_A, SI_A, B, lET_B, SI_B)$*, where the former party* $A$ *in the list is always initiator and* $B$ *is the responder in sessions executing between the two parties (i.e., the* $PESR$ *are identified via ordered parities' identifier pair). The implications of each element are described as below:*

1. *The variable* $RS_{A,B}$ *stores either a reset sign, or the recorded secret execution state used for authentication in the established execution stage.*
2. *The variable* $lST$ *stores the latest starting time of states synchronizing.*
3. *The variable* $SI_A, SI_B$ *are secret indicators for recording the latest successful synchronization process from party* $A$ *or* $B$ *respectively.*
4. *The variable* $lET_A, lET_B$ *store the establish time of latest recorded (i.e. without rejection) secret execution states* $SI_A$ *and* $SI_B$ *at party* $A$ *and* $B$*, respectively.*

Please note that each protocol execution states record ($PESR$) always describes the state between two parties, since we here only consider the most recent execution states shared by the corresponding parties. We further assume that every party keep a state list $SL$ to record each $PESR$. Without of loss of generality, we only allow a $PESR$ to be processed by one instance within a party at some point, namely the $PESR$ need to be locked while corresponding instance being activated (e.g., with 'read-write' lock) and any other requests on current locked $PESR$ have to wait until the previous operation is committed or expired.[2] It's ensure that the $PESR$ is exactly the one which the parties intend to synchronize, thus in the following we write $PESR^{s,A}_{A,B}$ to denote that it is locked by session $s$ at party $A$. Since the execution stage of each $PESR$ might be disrupted (e.g., one party is in *established* and the other in *reset* stage), the $SI_A$ is used as a witness to re-establish the secret execution.

After each protocol instances completed, the corresponding $PESR$ need to be updated from the previous secret states to recent ones in terms of $CPES$. The protocol execution stage transition of $PESR$ is informally depicted in figure 1.

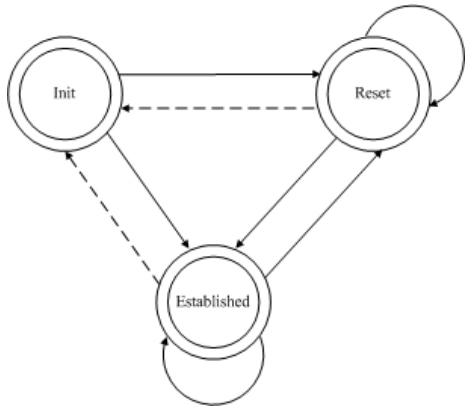


**Fig. 1:** Execution stage transition diagram

As illustrating in figure 1, the regular transition route is denoted by the real line, whereas the dash line routes mean that transition caused by some objective factors, e.g., $PESR$ expired or

[2] Note that, the $SL$ itself could work as a table in a database, hence the $PESR$ as a item in $SL$ should be protected by some concurrency control mechanism

agreed by both parties at some point, etc. In normal situation, the states in $PESR$ should be only transferred within the *established* stage after it has been initiated. However, if a session terminates with incomplete execution (e.g., missing confirmation messages) then the execution stage of its $PESR$ might turn into *reset*, but its peer might still keep in *established* stage. In order to deal with different cases of state inconsistent, we formally give the definition of state synchronization for two parties.

**Definition 3 (Execution State Synchronization).**
*Let* $PESR_{A,B} := (RS_{A,B}, A, lST, lET_A, SI_A, B, lET_B, SI_B)$ *be the latest protocol execution state recorded at party A. Let* $PESR^*_{A,B} := (RS^*_{A,B}, A, lST^*, lET^*_A, SI^*_A, B, lET^*_B, SI^*_B)$ *be the corresponding matching protocol execution states record at B. Then the synchronization status of* $PESR_{A,B}$ *and* $PESR^*_{A,B}$ *can be the following:*

1. *Complete In Synchronization (**CIS**). The* $PESR_{A,B}$ *is said to be fully synchronized with* $PESR^*_{A,B}$ *if only if every elements in* $PESR_{A,B}$ *and* $PESR^*_{A,B}$ *are pair-wise equivalent.*
2. *Partial In Synchronization (**PIS**). We call* $PESR$ *and* $PESR^*$ *are partially synchronized, if* $RS^*_{A,B} = RES$ *and* $lST \geq lST^*$ *and one of the following conditions holds:*
   *(a)* $lET_A = lET^*_A$ *and* $SI_A = SI^*_A$.
   *(b)* $lET_B = lET^*_B$ *and* $SI_B = SI^*_B$
3. *Out Of Synchronization (**OOS**). We say that* $PESR_{A,B}$ *and* $PESR^*_{A,B}$ *are out of synchronization, if the synchronization status of* $PESR_{A,B}$ *and* $PESR^*_{A,B}$ *satisfies neither* $CIS$ *nor* $PIS$.

Please note that the $lST$ is a very important criteria for identifying the $PSS$ event, since the *reset* execution stage of responder might result from the adversary's impersonation behavior in which case the $lST$ is smaller than $lST^*$.

### 2.2 Security Model for Authenticated Key Exchange with Synchronized States

**Passive Security of AKELL.** A (two-party) AKELL protocol is a protocol executed among two parties $A$ and $B$. At the end of the protocol, both parties output the same uniformly random session key $K_0 \in \{0,1\}^{l_k}$ where $l_k$ is the length of key space of the protocol in terms of the security parameter.

**Definition 4.** *We say that a AKELL protocol capture passive key exchange security if for all polynomial-time adversary holds that* $|\Pr[b = b'] - 1/2| \leq \epsilon$ *for some negligible function* $\epsilon$ *in the following experiment:*

1. *A challenger* $\mathcal{C}$ *generates the public parameters* $\delta$ *of the protocol (e.g. a generator describing a group etc.), and a set of long-term keys for each party.*
2. *The adversary receives* $\delta$ *and the parties' long-term key of as input. The adversary can run protocol instances interacting with the challenger by sending messages on her choice. Meanwhile, the adversary may query the challenger to obtain session key of some protocol instance run by the challenger itself. Then, the challenger runs a protocol instance, and obtains the transcript* $T$ *of all messages exchanged during the protocol and a key* $K_0$. *The challenger returns* $(T, K_0)$.
3. *At some point, the adversary outputs a test symbol* $\top$. *Given* $\top$, *the challenger runs a new protocol instance itself, obtaining the transcript* $T$ *and key* $K_0$, *samples* $K_1$ *uniformly at random from key space, and tosses a fair coin* $b \in \{0,1\}$. *Then it returns* $(T, K_b)$ *to the adversary.*
4. *The adversary may continue making session establishing queries (as step 2) to the challenger.*

*5. Finally, adversary $\mathcal{A}$ outputs a bit $b'$.*

*We say that the adversary wins the game, if $b = b'$.*

The above passive security definition of AKELL protocols is easy to achieve (e.g., a signature based Diffie-Hellman protocol under DDH assumption), and is the basic correctness and security requirement for more complex key exchange with entities' authentication using synchronized states.

**Execution Environment for Active Adversary.** In order to define the secure authenticated key exchange with synchronized States (AKESS), the security model is required to model the capabilities of active adversaries (especially for the adversary which is able to learn long-lived key of a party). We must describe the attacks against which AKESS protocol should be secure, and which outcome we expect if we run the protocol with the defined adversary. Following the line of research for AKE security [4,7,12,14,8] initiated by Bellare and Rogaway [2], we model the adversary by providing an execution environment, which emulates the real-world capabilities of an active adversary.

Informally speaking, the adversary has full control over the communication network, thus may forward, interleave, or drop any message sent by the participants, or inject new messages on her own choice. In our model, the adversary $\mathcal{A}$ is a probabilistic polynomial time (PPT) Turning Machine which is equipped with a collection of oracles $\pi_{i,j}^{s,i}$, where $i, j \in [I]$ and $s \in [S]$. An oracle $\pi_{i,j}^{s,i}$ models an party $ID_i$ running session $s$ with party $ID_j$ in which party $ID_i$ is the initiator and $ID_j$ is responder. All oracles formed as $\pi_{i,\cdot}^{\cdot,i}$ share the same long-lived secrets of party $ID_i$. We assume each $\pi_{i,j}^{s,i}$ will be automatically expired within the scheduled time $t_e$ if meantime there is no message is received, and the next process $\pi_{i,j}^{s+1,i}$ can be activated only if the $\pi_{i,j}^{s,i}$ has been terminated. Assume further, each process outputs the session key when its execution status $\Lambda =$ *accept-ll*. An adversary $\mathcal{A}$ is able to ask following types of queries:

- **Send**$(\pi_{i,j}^{s,i}, m)$. The adversary can issue this query to send any message $m$ of her choice to oracle $\pi_{i,j}^{s,i}$. The oracle will respond with the messages according to the protocol specification. If $m$ is empty and there is no process at party $ID_i$ locking $PESR_{i,j}^{s,i}$, then this query will active a new process $\pi_{i,j}^{s,i}$ with locking the $PESR_{i,j}^{s,i}$ and respond with the first message of the protocol.

- **Long-termKeyReveal**$(ID_i)$. The adversary can issue this query to learn the long-term key of specified party.
- **SessionKeyReveal**$(\pi_{i,j}^{s,i})$. The adversary may learn the encryption key $k_e$ computed in process $\pi_{i,j}^{s,i}$ via this query. The adversary If process $\pi_{i,j}^{s,i}$ has terminated in $\Lambda_s \in \{$*accept-session*$, reset\}$, the black-box responds with the key in $\pi_{i,j}^{s,i}$, otherwise some failure symbol $\perp$ is returned.
- **Test**$(\pi_{i,j}^{s,i})$: This query is only allowed be asked once throughout the experiment. If $\pi_{i,j}^{s,i}$ terminates in execution status $\Lambda_s =$ *accept-session*, then this query tosses a fair coin $b \in \{0, 1\}$, otherwise it returns a failure symbol $\perp$. If $b = 0$, the real encryption key $k_e$ is returned and a random key $k$ otherwise.

**Secure Authenticated Key Exchange with Synchronized States.** Informally speaking, an authentication protocol using introduced $PESR$, is a protocol run between two processes $\pi_{i,j}^{s,i}$ and $\pi_{i,j}^{t,j}$ of entites $i$ and $ID_j$, where the two processes mutually prove knowledge of every items in corresponding $PESR$ and both output execution status either "accept-session", "reset" or "reject-session" at the end of execution. The major goal of mutual authentication with synchronized states

(MASS) is to mitigate the damage consequences due to the leakage of identity related long-lived credentials. Comparing to those static long-lived key, the secret execution states serve as dynamical authentication factor that record the party's activity engaged in most recent established session, for both purposes of IDI prevention and IDI detection.

We define security of an MASS protocol following the idea of matching sessions (conversations), as introduced by Bellare and Rogaway [2]. However in contrast to BR model, we don't require any process terminates in acceptance, due to the interference of the adversary. In the sequel, let $T_{s,i}$ denote the transcript of all messages sent and received by process $\pi_{i,j}^{s,i}$, and let $|T|$ denote the number of its messages recorded. We say that two message list $T_{s,i}$ and $T_{t,j}$ are matched, written $T_{s,i} \cong T_{t,j}$, if the first $t = min\{|T_{s,i}|, |T_{t,j}|\}$ messages of the two lists are pairwise equal. And we let $T_{s,i} \equiv T_{t,j}$ denote the two message list are complete equivalent, namely $|T_{s,i}| = |T_{t,j}|$ and every messages in the two lists are pairwise equal. Please notice a special case of the *matched* case that two message lists can be said to be matched if one of them is empty, i.e. $|T| = 0$.

**Definition 5 (Matching Sessions).** *We say two processes $\pi_{i,j}^{s,i}$ and $\pi_{i,j}^{t,j}$ have matching sessions if one of the following conditions is satisfied:*

- *$\pi_{i,j}^{t,j}$ updated by receiving a message, and it holds that $T_{s,i} \equiv T_{t,j}$.*
- *$\pi_{i,j}^{s,i}$ updated by sending a message, and it holds that $T_{s,i} \cong T_{t,j}$.*

**Definition 6 (Secure Mutual Authentication with Synchronized States).** *We say that an mutual authentication protocol with synchronized states is secure, if for all probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$, interacting with the black-box $\mathcal{O}(\Pi)$ as described above in the execution environment, it holds that: each process $\pi_{i,j}^{s,i}$ of $\mathcal{O}(\Pi)$ terminates in execution status $\Lambda_s \in \{accept\text{-}session, reset\}$ only if there exists a process $\pi_{i,j}^{t,j}$ such that $\pi_{i,j}^{s,i}$ and $\pi_{i,j}^{t,j}$ have matching sessions, except with negligible probability $\epsilon = \epsilon(\kappa)$ in the security parameter.*

**Definition 7 (Secure Authenticated Key Exchange with Synchronized States).** *We say that a authenticated key-exchange with synchronized states protocol $\Pi$ is secure if*

1. *$\Pi$ is a secure mutual authentication with synchronized states protocol according to Definition 6.*
2. *For all polynomial-time adversary $\mathcal{A}$ holds that $|\Pr[b = \mathcal{A}^{\mathcal{O}(\Pi)}(1^\kappa)] - 1/2| \leq \epsilon$ for some negligible function $\epsilon$ under the following conditions.*
   *(a) $\mathcal{A}$ issued test query on a process $\pi_{i,j}^{s,i}$ without failure.*
   *(b) $\mathcal{A}$ doesn't issue $SessionKeyReveal(\pi_{i,j}^{s,i})$ query, or $SessionKeyReveal(\pi_{i,j}^{t,j})$ query if $\pi_{i,j}^{t,j}$ and $\pi_{i,j}^{s,i}$ have matching sessions.*

**Identity Impersonation and Detection.** We here formally give the definition of IDI based on the definition of matching sessions. We formalize IDI by extending the execution environment by one more type of query, which may be asked by the adversary.

- **PESR-Reveal**$(\pi_{i,j}^{s,i})$. The adversary may issue this query to learn the $PESR$ which is used for authenticating the session $s$. If the $PESR_{i,j}^{s,i}$ is not locked, then the black-box responds with the $PESR_{i,j}^{s,i}$. Note that, this query model the corresponding $PESR$ was leaked from either party $ID_i$ or $ID_j$.

**Definition 8 (Identity Impersonation).** *If a process $\pi_{i,j}^{s,i}$ terminates in execution status $\Lambda_s \in \{accept\text{-}session, reset\}$ but there is no process $\pi_{i,j}^{t,j}$ such that $\pi_{i,j}^{s,i}$ and $\pi_{i,j}^{t,j}$ have matching sessions, then we say the party $ID_j$ is impersonated.*

In order to evaluate the validity of the IDI detection capability, we require some additional definitions on freshness of each oracle, and we here focus on the corruption of secret relationship (i.e., $PESR$) between parties.

**Definition 9.** *We say an oracle $\pi_{i,j}^{s,i}$ which terminates in $\Lambda_s = \{accept\text{-}session, reset\}$ is fresh (unopened) if that adversary didn't make both queries on Long-termKeyReveal($ID_j$) and PESR-Reveal ($\pi_{i,j}^{s,i}$), otherwise from this point we call all oracles formed as $\pi_{i,j}^{\cdot,i}$ (including $\pi_{i,j}^{s,i}$) and $\pi_{i,j}^{\cdot,j}$ for executing subsequent sessions are opened (corrupted).*

**Definition 10.** *We say a mutual authentication with synchronized states (MASS) protocol $\Pi$ provides valid identity impersonation detection capability within time $t_{idi}$, if*

1. *$\Pi$ is secure MASS protocol in the sense of Definition 6,*
2. *If for all polynomial-time adversary only has negligible advantage in winning the following experiment:*

   *(a) A challenger $\mathcal{C}$ generates the public parameters $\delta$ of the protocol.*
   *(b) The adversary $\mathcal{A}$ receives $\delta$ as input, and may query the challenger interacting with the black-box $\mathcal{O}(\Pi)$ as described above in the execution environment.*
   *(c) Eventually, the adversary $\mathcal{A}$ terminates after time $t_{idi}$. Meanwhile, $\mathcal{A}$ might impersonate arbitrary parties and run protocol instances on behalf of corresponding opened oracles.*
   *(d) $\mathcal{C}$ runs (a new round) matching sessions between opened oracles and corresponding partners, with existing passive adversary $\mathcal{B}$ who is only allowed to have access to the Send query.*
   *(e) At the end of the experiment, we say the $\mathcal{A}$ or $\mathcal{B}$ wins if there exists some opened oracles terminates in $\Lambda \in \{accept\text{-}session, reset\}$.*

In the experiment, the time $t_{idi}$ models the longest time interval of a party executing two protocol instances with specific partner in real world. As to the IDI detection step 2d of the experiment, we require there exists only passive adversary since the active adversary who corrupted the oracles is able to prevent being detected via further IDI attacks.

## 3 Generic Execution States Synchronization Framework for AKE

The generic execution states synchronization framework (ESSF) contains three components: the execution secret (ES) state generation, the authenticated key exchange with synchronized states (AKESS) protocol and corresponding states synchronization rules. Let us first describe the following building blocks for our ESSF:

- An authenticated key exchange with (only) long-lived key (AKELL), which is only required to be secure in the sense of Definition 4.
- Three pseudo-random functions $F_1$ and $F_2$ with output length $l_k$ and $l_e$ respectively.
- A message authentication code scheme `MAC`.
- A deterministic algorithm: $Identify : \{0,1\}^{l_r} \times \{0,1\}^{l_r} \rightarrow \{CIS, PIS, OOS\}$. This algorithm is a $PESR$ synchronization status identification algorithm (where the $l_r$ is the length of the $PESR$).

### 3.1 A Generic *ES* Generation Scheme for AKE

An important property that we want of a protocol with some synchronized secret execution states that the compromise of one $PESR$ should have minimal consequences overall. For example, its revelation should not allow one to subvert subsequent authentication, nor should it leak information about other (as yet uncompromised) secret execution states. In practical, we are also required to integrate the proposed ESSF with an existing AKELL protocol $\pi$ (such as SSL/TLS or IPSec/IKE) without any modification on $\pi$. To capture those requirement, we only need the protocol $\pi$ to provide an interface to access the image of its session key and communication transcript $T$, e.g., $k_0 := F(k, 1^\kappa)$ where $F$ is a pseudo-random function. Now we can give a generic $ES$ generation scheme for AKE protocols based on the $k_0$. The $ES$ generation scheme for a session is described as following:

1. Run the AKELL protocol, and both parties compute session key image $k_0$.
2. Compute a new encryption key $k_e := F_1(k_0, "ENC")$, the MAC key $k_m := F_1(k_0, "MAC")$, and the corresponding execution secret state $ES := F_1(k_0, "ES")$.

### 3.2 An Authenticated Key Exchange with Synchronized States Protocol

*Preliminary.* Let $'||'$ be the operation of messages concatenation. Let $s$ be a AKE session held by an honest party $A$ with some honest party $B$, which has the corresponding latest protocol execution states record $PESR^{s,A}_{A,B} := (RS_{A,B}, A, lST, lET_A, SI_A, B, lET_B, SI_B)$. Let $t$ held by $B$ be the matching session of $s$, which has $PESR^{t,B}_{A,B} := (RS^*_{A,B}, A, lST^*, lET^*_A, SI^*_A, B, lET^*_B, SI^*_B)$. Let $(ES_s, ES_t)$ denote the current established execution secret states selected by $s$ and $t$ respectively, and $(ET_A, ET_B)$ denote the current established time of $s$ and $t$ respectively. Let $ST$ be the starting time of current authentication with synchronized states which is equal to $ET_A$. Now let us show how to compile an AKELL protocol into an authenticated key exchange with synchronized states (AKESS) protocol. The AKESS protocol between two parties $A$ and $B$ proceed as follows (also informally depicted in figure 2).

**Definition 11 (AKESS protocol).**

1. *$A$ and $B$ execute the AKELL protocol instances $s$ and $t$ respectively, which are supposed to be matching. At the end of the (AKELL) protocol run, both sessions output the image of the session key $k_0$, record a transcript $T_s$ and $T_t$ and execution status $\Lambda_t$ = accept-ll. The key $k_0$ is used to derive the new encryption key, MAC key and the ES as described in section 3.1.*
2. *$A$ retrieve and locks the latest $PESR$ identified by $(A, B)$ from $SL$, and does the following:*
   (a) *Set transcript record $T^A_1 := T_s||A||ST||lST||ET_A||lET_A||B||lET_B$.*
   (b) *Compute $c1_s := F_2(k_m \oplus RS_{A,B}, T^A_1)$, $c2_s := F_2(k_m \oplus SI_A, T^A_1)$ and $c3_s := F_2(k_m \oplus SI_B, T^A_1)$.*
   (c) *Compute $M_1 := \mathtt{MAC}_{k_m}('1'||T^A_1||c1_s||c2_s||c3_s)$.*
   (d) *Prepare messages $mesg_1 := M_1||ET_A||lET_A||lET_B||c1_s||c2_s||c3_s$, and send it to $B$*
   (e) *Set a temp $PESR$ $p_s := (c1_s, A, lST, lET_A, c2_s, B, lET_B, c3_s)$.*
3. *Upon receiving $mesg^B_1$, $B$ retrieve and locks the latest $PESR$ identified by $(A, B)$ from $SL$ and does the following:*
   (a) *Set transcript record $T^B_1 := T_t||A||lST^*||ET_A||lET_A||B||lET_B$ obtained from $mesg^B_1$.*
   (b) *Compute $M'_1 = \mathtt{MAC}_{k_m}('1'||T^B_1||c1_s||c2_s||c3_s)$, and reject the session $t$ if $M'_1 \neq M_1$.*
   (c) *Compute $c1_t := F_2(k_m \oplus RS^*_{A,B}, T^B_1)$ if $RS^*_{A,B} \neq RES$, otherwise set $c1_t := RES$.*
   (d) *Compute $c2_t := F_2(k_m \oplus SI^*_A, T^B_1)$ and $c3_t := F_2(k_m \oplus SI^*_B, T^B_1)$.*
   (e) *Set a temp $PESR$ $p_t := (c1_t, A, lST^*, lET^*_A, c2_t, B, lET^*_B, c3_t)$.*

(f) *Reject if* $Identify(p_s, p_t) \notin \{CIS, PIS\}$.
(g) *Set transcript record* $T_2^B = M_1||A||lST^*||ET_A||lET_A^*||B||ET_B||lET_B^*$.
(h) *Compute* $c1_t := F_2(k_m \oplus RS_{A,B}^*, T_2^B)$ *if* $RS_{A,B}^* \neq RES$, *otherwise set* $c1_t := RES$.
(i) *Compute* $c2_t := F_2(k_m \oplus SI_A^*, T_2^B)$ *and* $c3_t := F_2(k_m \oplus SI_B^*, T_2^B)$.
(j) *Compute* $M_2 := \mathtt{MAC}_{k_m}('2'||T_2^B||c1_t||c2_t||c3_t)$.
(k) *Set* $\Lambda_t = reset$.
(l) *Prepare messages* $mesg_2 := M_2||lST^*||lET_A^*||ET_B||lET_B^*||c1_t||c2_t||c3_t$, *and send it to A*

4. *Upon receiving* $mesg_2^A$, *A does the following:*
(a) *Set transcript record* $T_2^A = M_1||A||ET_A||lET_A^*||B||lET_B^*$.
(b) *Compute* $M_2' := \mathtt{MAC}_{k_m}('2'||T_2^A||c1_t||c2_t||c3_t)$, *and reject if* $M_2' \neq M_2$.
(c) *Compute* $c1_s := F_2(k_m \oplus RS_{A,B}, T_2^A)$ *if* $c1_t \neq RES$, *otherwise set* $c1_s := RES$.
(d) *Compute* $c2_s := F_2(k_m \oplus SI_A, T_2^A)$, $c3_s := F_2(k_m \oplus SI_B, T_2^A)$, *and update* $p_s$.
(e) *Reject if* $Identify(p_s, p_t) \notin \{CIS, PIS\}$.
(f) *Set transcript record* $T_3^A := M_1||ET_A||ET_B||M_2$.
(g) *Compute* $M_3 := \mathtt{MAC}_{k_m}('3'||T_3^A)$, *and send* $mesg_3 = M_3$ *to hatB*.
(h) *Accept the session with* $\Lambda_s := accept\text{-}session$.

5. *Upon receiving* $mesg_3^B$, *B does the following:*
(a) *Set transcript record* $T_3^B := M_1||ET_A||ET_B||M_2$.
(b) *Compute* $M_3' := \mathtt{MAC}_{k_m}('3'||T_3^B)$.
(c) *If* $M_3' \neq M_3$ *then abort the session with* $\Lambda_t := reset$, *otherwise accept the session with* $\Lambda_t := accept\text{-}session$.

6. *In additional, the verification steps need to ensure the established time of session s and t are valid.*

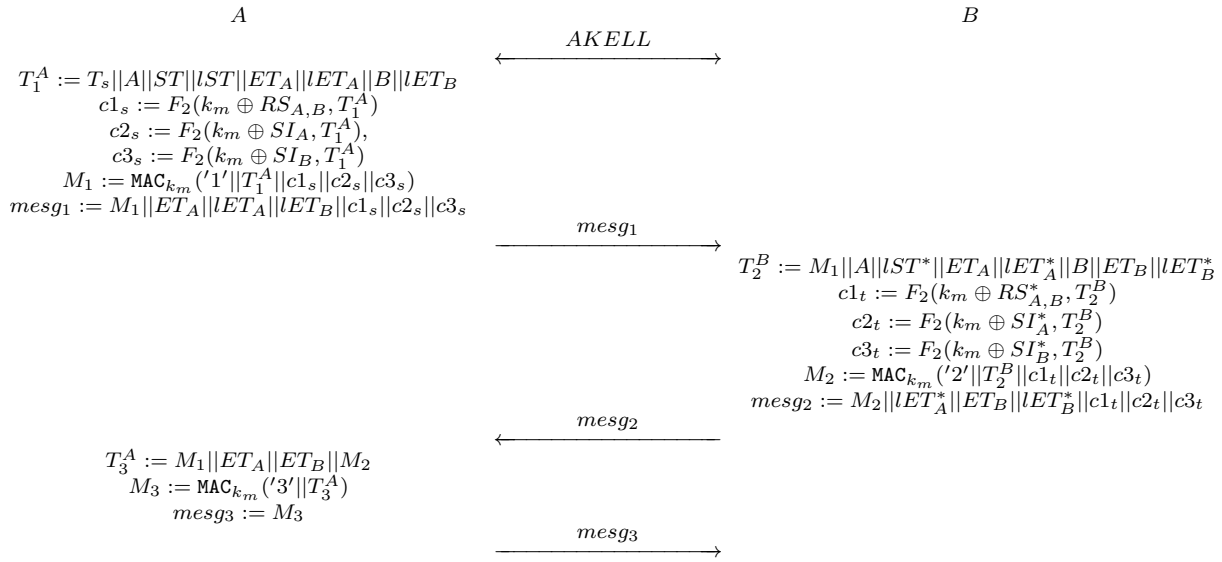


**Fig. 2:** AKESS protocol: an authenticated key exchange with synchronized states protocol

Please note that a session $s$ terminates in acceptance, if only if it received the last intended message and all verifications (incl. the synchronized states) are valid, then the protocol execution

state record should be fully updated (i.e., all items of $PESR$), otherwise the $s$ is called incomplete execution and only partial states of $PESR$ might be updated. In the following, we formally present the definition of the rules for synchronizing and recording the secret states.

**Definition 12 (Execution States Synchronization Rules).** *The states synchronization rules of $PESR$ for a two-party protocol between $A$ and $B$ are described as following:*

1. *For a session $s$ of initiator $A$:*
   (a) *If $\Lambda_s =$ accept-session, then $A$ updates the $PESR^{s,A}_{A,B}$ as following:*
      i. *If $Identify(PESR^{s,A}_{A,B}, PESR^{t,B}_{A,B}) = CIS$, then $A$ does the following:*
         A. *Set $RS_{A,B} := F_2(ES_s \oplus RS_{A,B}, 1^\kappa)$, $lST := ST$, $lET_A := ET_A$ and $lET_B := ET_B$.*
         B. *Set $SI_A := F_2(RS_{A,B}, ET_A)$ and $SI_B := F_2(RS_{A,B}, ET_B)$.*
      ii. *If $Identify(p_s, p_t) = PIS$, then $A$ does the following:*
         A. *If $SI^*_A := SI_A$ set $RS_{A,B} := F_2(SI_A \oplus ES_s, 1^\kappa)$, otherwise set $RS_{A,B} := F_2(SI_B \oplus ES_s, 1^\kappa)$.*
         B. *Set $lST := ST$, $lET_A := ET_A$ and $lET_B := ET_B$.*
         C. *Set $SI_A := F_2(RS_{A,B}, ET_A)$ and $SI_B := F_2(RS_{A,B}, ET_B)$.*
   (b) *Otherwise, $A$ only set $lST := ST$.*
   (c) *Store the updated $PESR^{s,A}_{A,B}$ into $SL$ and unlock it.*
2. *For the responder $B$ in $t$ which is matching to $s$:*
   (a) *If $\Lambda_t =$ accept-session, then $B$ updates $PESR_t$ as following:*
      i. *If $Identify(p_s, p_t) = CIS$, then $B$ does the following:*
         A. *Set $RS^*_{A,B} := F_2(ES_t \oplus RS^*_{A,B}, 1^\kappa)$, $lST^* := ST$, $lET^*_B := ET_B$ and $lET^*_A := ET_A$.*
         B. *Set $SI^*_A := F_2(RS^*_{A,B}, ET_A)$ and $SI^*_B := F_2(RS^*_{A,B}, ET_B)$.*
      ii. *If $Identify(p_s, p_t) = PIS$, then $B$ does the following:*
         A. *If $SI^*_A := SI_A$ set $RS^*_{A,B} := F_2(ES_t \oplus SI^*_A, 1^\kappa)$, otherwise set $RS^*_{A,B} := F_2(ES_t \oplus SI^*_B, 1^\kappa)$.*
         B. *Set $lST^* = ST$, $lET^*_A := ET_A$ and $lET^*_B := ET_B$.*
         C. *Set $SI^*_A := F_2(RS^*_{A,B}, ET_A)$ and $SI^*_B := F_2(RS^*_{A,B}, ET_B)$.*
   (b) *Else If $\Lambda_t = reset$, then $B$ updates the $PESR^{t,B}_{A,B}$ as following:*
      i. *If $Identify(p_s, p_t) = CIS$, then $B$ sets $lST^* := ST$, $lET^*_A := ET_A$ and $SI^*_A := F_2(F_2(ES_t \oplus RS^*_{A,B}, 1^\kappa), ET_A)$.*
      ii. *Set $RS^*_{A,B} := RES$.*
      iii. *If $Identify(p_s, p_t) = PIS$, then $B$ does the following:*
         A. *Set $lST^* := ST$ and $lET^*_A := ET_A$*
         B. *If $SI^*_A := SI_A$ set $SI^*_A := F_2(F_2(SI^*_A \oplus ES_t, 1^\kappa), ET_A)$, otherwise set $SI^*_A := F_2(F_2(SI^*_B \oplus ES_t, 1^\kappa), ET_A)$.*
   (c) *In other case $B$ does nothing.*
   (d) *Store the updated $PESR^{t,B}_{A,B}$ into $SL$ and unlock it.*
3. *Meanwhile, we require the whole $PESR$ update procedure described above is atomic. The $p_s$ and $p_t$ are temp images of $PESR^{s,A}_{A,B}$ and $PESR^{t,B}_{A,B}$ used by AKESS protocol as Definition 11.*

## 4 Security Analysis

**Theorem 1.** *If the AKELL protocol satisfy passive KE security as Definition 4, and the pseudo-random functions ($F_1$, $F_2$) and the message authentication code (`MAC`) are secure with respect to the definitions in Appendix A, then the proposed AKESS protocol is a secure in the sense of Definition 7.*

We prove the above theorem by two lemmas. Lemma 1 states that the AKE protocol meets property 1) of Definition 7, Lemma 2 states that it meets property 2) of Definition 7.

**Lemma 1.** *If the AKELL protocol satisfy passive KE security as Definition 4, and the pseudo-random function and the message authentication code are secure with respect to the definitions in Appendix A, then the proposed AKESS protocol meets Property 1 of Definition 7.*

*Proof.* The proof proceeds in a *sequence of games*, following [3,18]. The first game is the real security experiment. By assumption there exists an adversary $\mathcal{A}$ that breaks the security of the above protocol. We then describe several intermediate games that step-wisely modify the original game. Next we show that in the final security game the adversary has only negligible advantage in breaking the security of the protocol. Finally we prove that (under the stated security assumptions) no adversary can distinguish Game $i+1$ from Game $i$. Let $G_i$ be the event that $\mathcal{A}$ wins in Game $i$. In the following let $negl(\kappa)$ be some (unspecified) negligible function in the security parameter $\kappa$.
**Game 0.** This is the original security game, in which the simulator init $I[\kappa] \times (I[\kappa]-1)$ uniformly random $PESR$s. Assume an adversary $\mathcal{A}$ breaking Property 1 of Definition 7 with probability $\epsilon$. In the sequel we will show that $\epsilon$ is negligible for any algorithm $\mathcal{A}$.

**Game 1.** This game proceeds exactly like the previous game, except that the simulator chooses a uniformly random key $\tilde{k_0}$ for those processes which have matching sessions when their $\Lambda = \{accept-ll\}$, to derive $k_e, k_m$ and $ES$ as $k_e := F_1(\tilde{k_0}, "ENC")$, $k_m := F_1(\tilde{k_0}, "MAC")$ and $ES := F_1(\tilde{k_0}, "ES")$. We abort if one of the following conditions holds:

- if $B$ terminates a process in execution status $\Lambda \in \{reset\}$, and $T_1^A \neq T_1^B$.
- if $A$ or $B$ terminates a process in execution status $\Lambda \in \{accept\text{-}session\}$, and either $T_1^A \neq T_1^B$, $T_2^A \neq T_2^B$ or $T_3^A \neq T_3^B$.

We claim that

$$|\Pr[G_1] - \Pr[G_0]| \leq negl(\kappa)$$

by the passive KE security of AKELL protocol. This implies that the adversary forwards all messages during key exchange without altering anything. We can thus use an adversary distinguishing Game 1 from Game 0 to break the KE security of AKELL protocol against passive adversaries.

**Game 2.** This game proceeds exactly like the previous game, except that the simulator chooses a uniformly random keys $\tilde{k_e}, \tilde{k_m}$ and $\tilde{ES}$ for those processes which have matching sessions when their $\Lambda = \{accept-ll\}$. The secrets $\tilde{k_e}, \tilde{k_m}$ and $\tilde{ES}$ are used to compute the corresponding authentication messages $c1_s, c2_s, c3_s$ and MACs. We claim that

$$|\Pr[G_2] - \Pr[G_1]| \leq negl(\kappa)$$

by the security of the pseudo-random function $F_1$. In the proof we exploit that we have exchanged the real key computed in AKELL with a random key $\tilde{k_0}$ in Game 1.

**Game 3.** This game proceeds exactly like the previous game, except that the simulator replace the function $F_2(\cdot,\cdot)$ with three random functions $RF_1, RF_2$ *and* $RF_3$ (used to compute the temp $PESR$ for authentication). It is feasible because the seeds of $F_2$ (i.e., $RS \oplus ES$ and $SI \oplus ES$) are computed by (uniformly random) secrets from $CPES_s$ and $PESR_s$ which are not both compromised by adversary. With respect to the opened processes, we compute the $F_2$ honestly. Therefore

an algorithm distinguishing *Game 3* from *Game 2* can be used to construct an algorithm breaking the security of the pseudo-random function $F_2$. We claim that

$$|\Pr[G_3] - \Pr[G_2]| \leq negl(\kappa)$$

**Game 4.** In this game, the protocol's authentication messages in the first two moves of protocol (e.g., $c1_s, c2_s$ and $c3_s$) are computed by evaluating three truly random function due to *Game 3*, to show that any adversary has only a negligible probability of making an oracle accept without a partner oracle having matching session. Thus, this game proceeds exactly like the previous game, except that the challenger now raises event $Abort_{ns}$ and aborts if an process $\pi_{i,j}^{s,i}$ terminates in $\Lambda_s \in \{accept\text{-}session, reset\}$, but there is no process $\pi_{j,i}^{t,j}$ having matching session to $\pi_{i,j}^{s,i}$. We claim that

$$|\Pr[G_4] - \Pr[G_3]| \leq negl(\kappa)$$

by the security of the pseudo-random function $F_2$. The truly random function $RF_1, RF_2$ *and* $RF_3$ which are only accessible to the partner oracles sharing $CPES$, and the full transcript containing all previous messages is used to compute the authentication messages. If there is no process having matching session to $\pi_{i,j}^{s,i}$ , the adversary receives no information about $RF_x^s(T_s)$(resp. $RF_x^t(T_t)$) where $x \in \{1, 2, 3\}$.

**Lemma 2.** *If the AKELL protocol satisfy passive KE security as Definition 4, and the pseudo-random functions* $(F_1, F_2)$*, and the message authentication code are secure with respect to the definitions in Appendix A, then the proposed AKESS protocol meets Property 2 of Definition 7.*

*Proof.* We consider the following two disjoint cases on a process $\pi_{i,j}^{s,i}$ which terminates in execution status $\Lambda \in \{accept\text{-}session, reset\}$ (which cover the whole):

1. There is no process $\pi_{i,j}^{t,j}$, such that $\pi_{i,j}^{s,i}$ and $\pi_{i,j}^{t,j}$ have matching sessions. In this case, although the adversary learns the session key of $\pi_{i,j}^{s,i}$ (or $\pi_{i,j}^{t,j}$), in order to make $\pi_{i,j}^{s,i}$ (or $\pi_{i,j}^{t,j}$) 'accept' or 'reset', she still need to break the security of pseudo-random function $F_2$ as above Game 3 to Game 4 in the proof of Lemma 1.
2. There exists a process $\pi_{i,j}^{t,j}$, such that $\pi_{i,j}^{s,i}$ and $\pi_{i,j}^{t,j}$ have matching sessions. The adversary needs to break the passive KE security as above Game 1 to Game 2 in the proof of Lemma 1.

We proceed in a sequence of games which is very similar to the sequence of games.

**Game 0.** This is the original security game. We assume an adversary $\mathcal{A}$ breaking Property 2 of Definition 7 with probability $1/2+\epsilon$. In the sequel we will show that $\epsilon$ is negligible for any algorithm $\mathcal{A}$ in security parameter.

**Game 1.** In this game, we make the same modifications as in Games 1 to 4 in the proof of Lemma 1. With the same arguments as before, we have

$$|\Pr[G_1] - \Pr[G_0]| \leq negl(\kappa).$$

Collect probabilities from Game 0 to 1 we obtain that Game 1 is indistinguishable from Game 0 (except for some negligible probability), which proves indistinguishability of real from random keys. Thus, the protocol meets Property 2 of Definition 7.

**Theorem 2.** *If the AKELL protocol satisfy passive KE security as Definition 4, and the pseudo-random functions* $(F_1, F_2)$ *and the message authentication code (*`MAC`*) are secure with respect to the definitions in Appendix A, then the proposed AKESS protocol provides valid identity impersonation detection capability in the sense of Definition 10.*

*Proof.* With respect to the passive adversary $\mathcal{B}$, its advantage of winning the IDI detection (IDID) experiment is negligible due to the security of MASS protocol since $\mathcal{B}$ has to forge the authentication messages. We evaluate the advantage of active adversary $\mathcal{A}$ in the IDID experiment. In order to impersonate a party, an active adversary has to corrupt corresponding oracle (i.e. learn both the $PESR$ and $CPES$). Assume the $PESR$ for an opened oracle $\pi_{i,j}^{s,i}$ is $PESR_{i,j}^{s,i}$ and for its partner oracle $\pi_{i,j}^{t,j}$ is $PESR_{i,j}^{t,j}$. Let $q_{es}$ be the number of adversary adaptively selecting the $ES$ for a session $s$ within session expiration time $t_e$. Hence there are at most $q_{in} = t_{idi}/t_e$ such time blocks, i.e. adversary runs at least $q_{in}$ opened processes. In order not to be detected, $\mathcal{A}$ has to result in the subsequent $PESR_{i,j}^{s,i}$ and $PESR_{i,j}^{t,j}$ such that $Identify(PESR_{i,j}^{s,i}, PESR_{i,j}^{t,j}) \in \{CIS, PIS\}$ after launching identity impersonation attack, and we denote this target event as $TE$. Let the event $TE_j$ denotes that the adversary wins in $j$-th opened process.

Please first recall that the definitions of $CIS$ or $PIS$, besides the secret states $RS$ and $SI$s, two parties have to synchronize the last synchronization time $lST$ and established time $lET$s. Therefore, the adversary has to launch IDI attacks on both parties at the same time who supposedly shared the same $PESR_{i,j}$. In each opened sessions, both the adversary and the simulator contribute to the randomness used to generate the session keys and the $ES$, which are independent to the secret states in other sessions. Thus the advantage of $\mathcal{A}$ winning the IDID experiment in $j$-th opened session is bounded by birthday paradox: $\Pr[TE_j] \leq \frac{q_{es}^2}{2^{l_e-1}}$. We have the overall advantage of $\mathcal{A}$ for time range $t_{idi}$ in terms of the binomial distribution

$$Adv_{IDID,AKESS}^{\mathcal{A}}(t_{idi}) \leq \frac{q_{in}q_{es}^2}{2^{l_e-1}}.$$

The result shows that the advantage of adversary is proportional to $q_{es}$ and $q_{in}$ which are related to the $t_e$ and $t_{idi}$ respectively. Therefore from the protocol perspective to reduce the odd of IDI detection failure could decrease expiration time $t_e$, and at the same time encourage users to execute the AKESS protocol as often as possible, namely decreasing the $t_{idi}$.

## 5 Closing Remark

We have formally modeled the authenticated key exchange with synchronized states and proposed a generic framework to synchronize the execution states of AKE. Although we have analyzed the security and validity of proposed framework, the obligations of parties should be specified before it was put into practical application. For example, once the IDI was detected, who should take responsibility for consequences of IDI? Add digital signatures on the proofs of $PESR$ (e.g., the $c1_t$, $c2_t$ and $c3_t$) in the AKESS protocol could be a possible solution to help figure out which parties' long-term key are exposed. Whereas the secret states also should be stored honestly, namely a party should not arbitrarily modify the stored secret states by himself other than correctly executing the AKESS protocol. Moreover, we encourage further exploration in the applications of synchronized states, e.g. enhance the security of password based authentication schemes to be secure against off-line guessing attack, or phishing attacks etc.

## References


1. Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In *CRYPTO*, pages 431–448, 1999.
2. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993.
3. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, 2006.

4. Simon Blake-Wilson, Don Johnson, and Alfred Menezes. Key agreement protocols and their security analysis. In *IMA Int. Conf.*, pages 30–45, 1997.
5. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321. Springer, 2005.
6. Ran Canetti and Shafi Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In *EUROCRYPT'99: Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, pages 90–106, Berlin, Heidelberg, 1999. Springer-Verlag.
7. Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2001.
8. Cas J. F. Cremers. Session-state reveal is stronger than ephemeral key reveal: Attacking the naxos authenticated key exchange protocol. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS*, volume 5536 of *Lecture Notes in Computer Science*, pages 20–33, 2009.
9. Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Key-insulated public key cryptosystems. In Lars R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 65–82. Springer, 2002.
10. Neil Haller. The s/key one-time password system. In *In Proceedings of the Internet Society Symposium on Network and Distributed Systems*, pages 151–157, 1994.
11. Amir Herzberg, Markus Jakobsson, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive public key and signature systems. In *ACM Conference on Computer and Communications Security*, pages 100–110, 1997.
12. Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2007.
13. Leslie Lamport. Password authentication with insecure communication. *Commun. ACM*, 24(11):770–772, 1981.
14. Alfred Menezes and Berkant Ustaoglu. Comparing the pre- and post-specified peer models for key agreement. *IJACT*, 1(3):236–250, 2009.
15. D. Nali and Paul C. van Oorschot. Croo: A universal infrastructure and protocol to detect identity fraud. In *ESORICS*, pages 130–145, 2008.
16. SeongHan Shin, Kazukuni Kobara, and Hideki Imai. Leakage-resilient authenticated key establishment protocols. In *ASIACRYPT*, pages 155–172, 2003.
17. SeongHan Shin, Kazukuni Kobara, and Hideki Imai. Efficient and leakage-resilient authenticated key transport protocol based on rsa. In *ACNS*, pages 269–284, 2005.
18. Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/`.
19. Paul C. van Oorschot and Stuart G. Stubblebine. Countering identity theft through digital uniqueness, location cross-checking, and funneling. In *Financial Cryptography*, pages 31–43, 2005.

# A Security Definitions

## A.1 Pseudo-Random Functions

A *pseudo-random function* is an algorithm $\texttt{PRF}$. This algorithm implements a deterministic function $z = \texttt{PRF}(k, x)$, taking as input a key $k \in \{0,1\}^\kappa$ and some bit string $x$, and returning a string $z \in \{0,1\}^\kappa$. Consider the following security experiment played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger samples $k \xleftarrow{\$} \{0,1\}^\kappa$ uniformly random.
2. The adversary may query arbitrary values $x_i$ to the challenger. The challenger replies each query with $z_i = \texttt{PRF}(k, x_i)$. Here $i$ is an index, ranging between $1 \leq i \leq q$ for some polynomial $q = q(\cdot)$. Queries can be made adaptively.
3. Eventually, the adversary outputs value $x$ and a special symbol $\perp$. The challenger sets $z_0 = \texttt{PRF}(k, x)$ and samples $z_1 \xleftarrow{\$} \{0,1\}$ uniformly random. Then it tosses a coin $b \xleftarrow{\$} \{0,1\}$, and returns $z_b$ to the adversary.
4. Finally, the adversary outputs a guess $b' \in \{0,1\}$.

**Definition 13.** *We say that* PRF *is a secure pseudo-random function, if*

$$\Pr\left[b = b'\right] \leq \frac{1}{2} + \epsilon$$

*for all probabilistic polynomial-time (in $\kappa$) adversaries $\mathcal{A}$, where $\epsilon$ is some negligible function in the security parameter.*

### A.2 Message Authentication Codes

A message authentication code is an algorithm MAC. This algorithm implements a deterministic function $w = \texttt{MAC}(K_m, m)$, taking as input a key $k_m \in \{0,1\}^\kappa$ and a message $m$, and returning a string $w$. Consider the following security experiment played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger samples $k_m \xleftarrow{\$} \{0,1\}^\kappa$ uniformly random.
2. The adversary may query arbitrary messages $m_i$ to the challenger. The challenger replies each query with $w_i = \texttt{MAC}(k_m, m_i)$. Here $i$ is an index, ranging between $1 \leq i \leq q$ for some polynomial $q = q(\cdot)$. Queries can be made adaptively.
3. Eventually, the adversary outputs a pair $(m, w)$.

**Definition 14.** *We say that* MAC *is a secure message authentication code, if*

$$\Pr\left[(m,w) \xleftarrow{\$} \mathcal{A}^{\mathcal{C}}(1^\kappa) : w = \texttt{MAC}(k_m, m) \text{ and } m \notin \{m_1, \ldots, m_q\}\right] \leq \epsilon$$

*for all probabilistic polynomial-time (in $\kappa$) adversaries $\mathcal{A}$, where $\epsilon$ is some negligible function in the security parameter.*