Generic Execution State Synchronization Framework for Authenticated Key Exchange Protocol

Zheng Yang
Ruhr University Bochum,Germany
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 through the use of stolen or compromised authentication secrets, in this paper, we propose a framework of execution state synchronization for authenticated key exchange protocol to either prevent IDI attack by detecting attempts thereof, or limit its consequences by detecting situations of previously unidentified IDI. In this framework, we introduce a generic protocol execution state representation mechanism, based on which we formalize the issue of execution synchronization by presenting a confirmation sub-protocol and corresponding synchronization rules. Our goal is to enhance the security by enduing with the capability of IDI detecting without modifications on the authenticated key exchange protocol, and withstand the interference of adversary. We also show a generic scheme to realize the proposed framework and some concrete applications.
Keywords: authenticated key exchange, impersonation detection,state synchronization

1 Introduction

Authenticated Key Exchange (AKE) protocols are foundation for building secure channel to protect communication over insecure networks. AKE protocol not only provide shared key for securing transmitted data, but also ensure the authenticity of the parties. The latter refers to some verification procedure which relies on authentication credential related to party's identity, and the commonly used authentication credential form including user password, private key of certification, fingerprint etc. There are diverse of reason for leaking party's authentication secrets, e.g., phishing or key logging, weak certificate, server intrusion etc. Since most of the AKE protocols always public known, a direct damage result of losing those secrets is the possibility of identity impersonation which might lead privacy loss, financial loss, and public discredit etc. Unfortunately, many users learn that their identity has been impersonated after some damage has been done. While the credential issuer 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, sometimes the consequences of identity impersonation would not be perceptible immediately if there is no evident trail to track. Hence one of the motivations of this paper is try to detect the impersonation on protocol level for saving time to remedy.[1] More specifically, besides the authentication of the parties while proceeding with the AKE protocol instance, it is necessary to authenticate the established communication channel either.

At present, commonly accepted methods for automatically on-line detecting are based on somehow comparing the recorded execution states of protocol. As for the existing pre-shared states

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

mechanism, which can be categorized as: (i) secret related, e.g. distinct (evolving) authentication key for each protocol instance, (ii) constant sequence, e.g., cumulative or degressive. Other methods can be deemed as the variant or combination of those two forms, in the following we denote those two schemes by abbreviation SR and CS respectively. The shortcoming of SR scheme is that the adversary is able to impersonate the victim after learning current long-term authentication secrets. Whereas the CS is predictable as long as the adversary learn the previous state at some point. Combining these two schemes is not much improvement, because that would inherit their defect either. Therefore in this paper, we seek new mechanism to represent execution states for protocol instance which is able to withstand the interference by adversary and enjoys the advantages of SR and CS, namely existing at least one state such that it is uniformly random distributed for each execution and not used as long-term authentication key and.

In general, the execution states are required to be updated according to protocol specified update timing, e.g., after the AKE protocol instances have been successfully established or terminated. Therefore the protocol instances always need extra confirmation steps to ensure update conditions satisfied. It is trivially to see that if the recent execution states between two parties are inconsistent then identity impersonation might have been occurred on either party. However, we cannot come to such conclusion arbitrarily, since there are numerous factors resulting in state non-synchronized. We believe the major reasons that result in execution state inconsistent include the following:

1. Identity impersonation attack. At some point, the adversary who obtains the long-term authentication key impersonates as the victim $\hat{A}$ to another honest party $\hat{B}$.
2. Interference of adversary. The passive adversary without the long-term secret is also able to disorder the execution state by intervening the communication between honest party $\hat{A}$ and $\hat{B}$, namely drop or delay the messages, etc.
3. Other situations. For instance, due to the network failure or system corruption, etc.

In particular, due to the interference of adversary, the inconsistent execution state seems inevitable. Provided that adversary intercept and drop (or substitute) the last protocol confirmation message assuming sent from party $\hat{A}$ to $\hat{B}$, then only $\hat{A}$ might update her state after she received the valid confirmation from $\hat{B}$, whereas $\hat{B}$ keeps the old states. For most of the AKE protocols e.g. [3][9] [15][10] [12], those protocols are prone to adapt two pass confirmation to provide mutual assurance on completing the matching sessions with peer and session key k. However, while we specify the update timing to be the point that the AKE process terminate in accepting, i.e., the session has been successfully established, then the two confirmation steps are not enough (particularly involving some execution states are required to be synchronized). Since the main purpose of recording execution state is for detecting the identity impersonation, and whether or not the long-term authentication key is exposed, it is necessary to distinguish between illegal state inconstant (e.g., caused by identity impersonation) and inevitable synchronizing fault (e.g., network failure or missing confirmation message etc.). The open question is how many steps are enough for synchronization.

We review a two pass update strategy between party $\hat{A}$ and $\hat{B}$ which could be: the $\hat{A}$ and $\hat{B}$ update their execution states upon receiving the valid confirmation messages from its peer. In this case, the adversary can adaptively drop the confirmation messages to determine which party could update. An alternative improvement could increase confirmation steps for both parties, but this won't change the result since the adversary can drop the confirmation messages either. Thus, besides the impersonation detection, another important issue is how to restore the normal sates. Once a party suspect if he has been impersonated, then he could check the communication records or transactions etc. to ensure no damage happened and re-synchronizing his execution state using

out-of-band mechanisms. However, this is not the most convenient way to solve the issue on restoring states or even any disputing.

In this paper we focus on general execution states synchronization problem for AKE protocol, which can be used as a universal identity impersonation detection solution, i.e. it is neither AKE protocol specific, nor restricted to applications. We strive to formalize the execution states model for AKE protocol which focuses on the aspects of states' representation, transition, and synchronization. In this framework, we particularly concern the requirements of identity impersonation detection and resilience of adversary interference. Besides those capabilities while the impersonation has already happened, we are also required to provide evidence to figure out which party has been impersonated, i.e., which party leaked the long-term authentication key.

Related work. In the academic literature, most focus on prevention of credential information exposure. The one-time password schemes, e.g., [11][7] are introduced to overcome a number of shortcomings of tradition static password, which typically make use of randomness can also be deemed as a kind of authentication key related execution state. Concerning perspective of protecting user's password, Shin et al. [15], dedicate to the immunity to the respective leakage of stored secrets from a client side and a server side. In their subsequent work [16], they also consider about dynamically update the authentication secret after establishing the session to provide its security against the leakage of stored secrets.

In order to limit the damage effect of key exposure in the public key infrastructure, the proposed schemes include threshold cryptosystems [2], proactive cryptosystems [8], proactive forward-secure schemes [1], key-insulated cryptosystems [6]. All of them are designed to decrease the odds that unauthorized public-key signatures be issued. Although, the schemes [8][1] involve the issue of secret synchronization, it is unable to detect the previous unauthorized using of the signature key.

As for the issue related to identity impersonation detection, Van Oorschot and Stubblebine [17] 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. online) transactions and loss of user location privacy (users are geographically tracked). In 2008, D. Nali and P.C. van Oorschot [13] propose a universal infrastructure and protocol so-called CROO (Capture Resilient Online One-time password scheme), to either prevent identity fraud (IDF) or identify cases of previous IDF in the environment of online transaction (i.e., between some client and server). This scheme highly depend on the *j-th* symmetric authentication key (as kind of execution state) which is derived from *j+1th* key. The initial key is generated by some trusted third party (TTP), and the TTP is also in charge of verifying the corresponding *j-th* on-time password and detecting the IDF. However, besides the defect of secret related execution state, the most important problem the CROO scheme never addresses is how to synchronize the state (i.e., the authentication key) between the client device and the TTP.

Contribution. We present a general framework on execution states synchronization for authenticated key exchange protocols. In which we introduce a generic execution state representation mechanism. Based on it, we formally model various circumstances while synchronizing protocol execution states between parties, particularly including the case of state non-synchronization caused by identity impersonation attack and situations of incomplete state synchronization process. In order to handle synchronization process of execution state, we propose a universal three pass confirmation protocol and rules concerning how those states update and impersonation be detected. More specifically, the AKE protocol realized our framework not only enjoys the capability of impersonation detection, but also is resilience of interference by adversary. Another interesting aspect of

such framework is that it provides means to revive execution states from adversary interference by protocol instances themselves. Furthermore, we introduce a generic scheme to realize the execution state synchronization framework based on the session key, and some concrete applications.

Notations and Terminology. We let κ denote the security parameter and 1^κ the string that consists of κ ones. The cryptographic primitives used in this paper include: one-way collision-resistant hash function,symmetric ciphers, message authentication code and pseudorandom functions. The details of their security definitions refer to the Appendix A.

Organization. The paper is structured as follows. In section 2, we first formalize the execution state of authenticated key exchange protocol, and present corresponding synchronization framework. We give example protocol realized the framework in section **??** to show its validity. At last, we conclude the paper in section 4.

2 Generic Execution State Synchronization Framework for AKE Protocol

2.1 Formalism of execution state synchronization

We call each instance of an AKE protocol run at a party a session. Technically, a session is an interactive subroutine executed inside a party. Each session is identified by the party that runs it, the parties with whom the session communicates and by a session identifier (sid). Different protocol instances might be either run concurrently between parties many times or never initialized. The protocol execution state denotes the execution context of protocol instances, which contains two aspects: (i) the status of protocol instances that have been executed between parties (incl., the established date-time, parties' identifier executed the session, and sid etc.), and (ii) the initial conditions for proceeding subsequent protocol instance.

We first classify the protocol execution state according to executing stage into three category $ES := (init, established, reset)$, where the $init$ stage indicates some protocol instance has not been initiated before, $established$ means the protocol instance has already been successfully performed between some parties at some time, and the $reset$ stage denotes that execution stage for corresponding instances will be reset to previous states for next run. In which the state values of $established$ stage are required to be uniformly random and not used as long-term authentication key. In the model description, we use macros $(INIT, RES)$ to denote the corresponding stage $init$ and $reset$ respectively (e.g., $INIT = \perp$, $RES = 1$), and est_i to denote the ith established stage which is required to be realized by specific protocol.

Definition 1 (Protocol instance execution state). *The execution state for each protocol instances is formed as tuple* $PIS := (lES, ID, Initiator, lET_{ID}, TET_{ID}, ID^*, Responder, lET_{ID^*}, TET_{ID^*})$, *where the implication of each elements described as below:*

1. *The* ID *and* ID^* *denote the identifier of the parties who own the* PIS.
2. *The* $\{Initiator, Responder\}$ *are the roles of corresponding parties when they execute some matching sessions. In the following, we denote the 'roles' by abbreviation* $\{I, R\}$ *respectively.*
3. lES *is the execution stage* ES *after the latest session executed between party* ID *and* ID^*.
4. lET_{ID}, lET_{ID^*} *denote the current established time of the protocol instance, i.e., established time of* lES *for party* ID *and* ID^*, *respectively.*
5. TET_{ID} *is the transformation of last successful established* lET_{ID} *and its corresponding execution stage* ES, *as well as* TET_{ID^*}.

We stress that the protocol instance execution state always describes the protocol execution state between parties (e.g., the protocol execution state between parties $\hat{A}$ and $\hat{B}$), whereas within a party the protocol execution state comprise the execution state of all recent protocol instances. Moreover, introducing the established time of protocol instance would be important criteria for setting up the synchronization rules. Especially, when there is some disputing, we need to figure out which party leak the authentication secrets according the established time. In addition, once successfully turning into *established* stage between parties, we further require the adversary who learns the authentication key is still unable to impersonate as the victim, unless he obtains current PIS at the same time. Since the ES would be disordered (e.g., one party is in *established* and the another is in *reset*) and the established time easy to be monitored, we utilize the TET_{ID} as a witness which is computed by last successful established lET_{ID} and its corresponding execution stage ES.[2]

We further assume that every party keep a state list SL to record corresponding PIS and the SL stored in some tamper-proof device (e.g., smart card). Then we are able to determine the execution state for a protocol instance s^{AB} (assuming $\hat{A}$ is initiator, and $\hat{B}$ is responder) executed between party $\hat{A}$ and $\hat{B}$ as: (i) If the protocol has been successfully executed before, then the tuple $PIS_{AB} := (lES, \hat{A}, I, IET_A, TET_A, \hat{B}, R, ITE_B, TET_B)$ must be recorded in SL, and if $lES = RES$, then the lES of s^{AB} is *reset* (ii) else if $lES = est_i$ (i.e., $lES \notin \{INIT, RES\}$), then the lES stands for ith successful session execution for s^{AB}; and (iii) if $lES = INIT$ or PIS_{AB} didn't record in SL then its ES is *init*.

After each protocol instance complete, the protocol execution state need to be update the previous states to recent one or shift to new execution stage. The protocol execution stage transition is informally depicted in figure 1. As illustrating in figure 1, the regular transition route is denoted

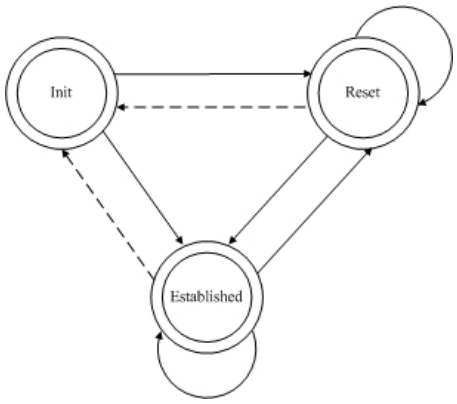

Fig. 1: Execution stage transition diagram

by the real line, whereas the dash line routes mean that transition caused by objective factors, e.g., PIS expired or agreed by both parties at some point etc. In normal situation, the states in PIS should be only transferred within the execution stage *established*, namely the lES of some PIS is shifted from ith established stage est_i to est_{i+1} after it has been initiated. However, if a protocol instance s terminates with missing confirmation messages then the execution stage of corresponding PIS might turn into *reset* stage, but his peer might keep in *established* stage. In order to deal

[2] The computation of TET_{ID} and TET_{ID^*} is given in the execution state synchronization rules.

with problem of state inconsistent, we formally give the definition of state synchronization to distinguishing different circumstances.

Definition 2 (Execution State synchronization).
Let $PIS_s := (lES_s, \hat{A}, I, lET_A, TET_A, \hat{B}, R, lET_B, TET_B)$ *be the current protocol instance execution state stored at party* $\hat{A}$ *and* $PIS_{s^*} := (lES_{s^*}, \hat{A}, I, lET_A^*, TET_A^*, \hat{B}, R, lET_B^*, TET_B^*)$ *be the corresponding matching protocol instance execution state stored at* $\hat{B}$*. Then the synchronization status of* PIS_s *and* PIS_{s^*} *should be the following:*

1. *Full state synchronization* (FSS)*. The* PIS_s *is said to be full synchronized with* PIS_{s^*} *if only if they share the same protocol execution state, namely every elements in* PIS_s *and* PIS_{s^*} *are equivalent.*
2. *Partial state synchronization* (PSS)*. We call* PIS_s *and* PIS_{s^*} *are partially synchronized, if* $lES_{s^*} = RES$ *and one of the following conditions hold:*
   (a) both $lET_A = lET_A^*$ *and* $TET_A = TET_A^*$*.*
   (b) both $lET_B = lET_B^*$ *and* $TET_B = TET_B^*$
3. *Non-synchronization* (NS)*. If the* PIS_s *and* PIS_{s^*} *don't satisfy either* FSS *or* PSS*, then they are non-synchronized.*

In the definition, we differentiate the legal state inconsistent (i.e., Partial state synchronization), from illegal cases (i.e., Non-synchronization). In which, the partial state synchronization model the situation of incomplete confirmation while session running which include the cause of adversary interference, and the non-synchronization case model the identity impersonation.

2.2 Framework for synchronizing protocol execution states

With respect to issue of normal execution state synchronization, we note that the status of matching PIS should be either $full\ state\ synchronization$ or *partial state synchronization*. Before reaching the conclusion that one party has been impersonated, we need to proceed with further confirmation to exclude interference of adversary or other exceptions.

We assume there are two honest parties $\hat{A}$ and $\hat{B}$ which are not corrupted. Let s be a AKE session held by an honest party $\hat{A}$ with some honest party $\hat{B}$, which possess the relevant protocol instance state $PIS_s := (lES_s, \hat{A}, I, lET_A, TET_A, \hat{B}, R, lET_B, TET_B)$. And Let s^* be the matching session of s, which is related to $PIS_{s^*} := (lES_{s^*}, \hat{A}, I, lET_A^*, TET_A^*, \hat{B}, R, lET_B^*, TET_B^*)$. Let ES_s, ES_{s^*} denote the current execution stage of s, s^* respectively, and ET_A, ET_B denote the current established time of s, s^* respectively. Let H be a collision-resistant hash function with output length l_h, and the ES is chosen from $\{0,1\}^{l_e}$ where l_e is the length of its value domain.

There is a deterministic algorithm in the framework: $Identify$. The algorithm $Identify : \{0,1\}^l \times \{0,1\}^l \rightarrow \{FSS, PSS, NS\}$ is the synchronization status identification algorithm (in which l is the length of the PIS), i.e., given two matching PIS_s and PIS_{s^*}, $Identify(PIS_s, PIS_{s^*})$ outputs their status in the set: full state synchronization, partial state synchronization or non-synchronization, as Definition 2.

Pre-synchronization. At the beginning of the key exchange procedure, the parties are required to negotiate the execution stage to process the protocol instances and determine the execution states about to confirm. In particular, while being in different stages, we specify the parties force to use *reset* stage to process the matching sessions.

One can specify that the *reset* execution stage only used to re-establish the protocol execution state, and the normal communications only work under *established* stage.

In addition, without of loss of generality, we only allow a PIS to be processed by one instance within a party at some point, namely the PIS need to be locked while corresponding instance being activated (e.g., with 'read-write' lock) and any other requests on current locked PIS have to wait until the previous operation committed or expired.[3]. That's ensure the PIS is exactly the one the parties intend to synchronize. Thus, we require the protocols which realize this synchronization framework are capable of handling the concurrent executions. The concurrent control is out of the scope of this paper and we focus on state synchronization confirmation and update.

Generic execution state confirmation protocol. In order to confirm the execution states, we rely on the assumptions that the matching sessions s, s^* share a session key k and a confirmation key k_m. Meanwhile, we stress that the confirmation messages are protected by the session key to be secure against passive adversary. Let $'||'$ be the operation of messages concatenation. The execution state confirmation protocol (ESCP) is depicted in figure 2 which can also be used to thwart unknown key-share attacks [4][5].

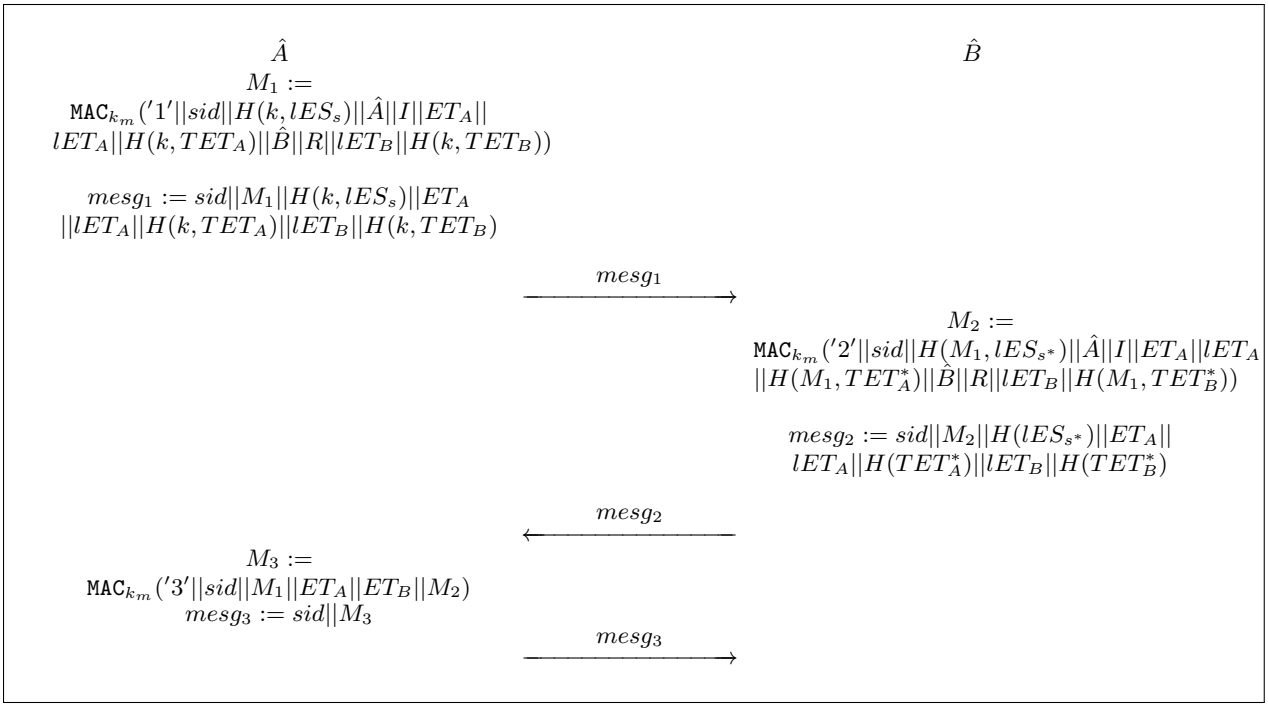

Fig. 2: Execution state confirmation protocol

.

We first note that using 'hash' version of original PIS in first confirmation, and the hashed value implies the pre-image in original PIS without negligible probability. Thus in the subsequent sections, we don't discriminate the two types of the PIS, i.e. the hashed version of PIS and its pre-image. The whole confirmation procedure are protected by the session key k to secure against the interception of passive adversary. A protocol instance s completed successfully, if only if it received the last intended message and all verifications are valid including the synchronization check (i.e.,

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

whether the status of current PIS_s and PIS_{s*} is either FSS or PSS in terms of Definition 2), then the protocol execution state could be full updated (i.e., all attributes in PIS will be renewed). Otherwise, the s is called incomplete execution and only partial of states might be updated. And we stress that only the responder will process the partial update because she might only receive the first valid confirmation message M_1 from the initiator without the last valid M_3, and then she doesn't know whether or not her confirmation message M_2 reached the initiator. Nevertheless, in this situation, the responder at least learns that the initiator has successfully established the session. On the other side, the initiator could either proceed with full PIS update if she has already received the valid M_2, or do nothing if the confirmation messages from responder are lost or invalid. In the following, we formally present the definition of the rules for sate synchronization.

Definition 3 (Execution state synchronization rules). *The transition rules of PIS for two-party protocol are described as following:*

1. *Initialize the PIS_s by retrieving from SL. This operation is done by both initiator and responder before the confirmation process start.*
2. *For the initiator $\hat{A}$:*
   (a) *If $\hat{A}$ receives the valid M_2, then $\hat{A}$ updates the PIS_s as following:*
      i. *If $Identify(PIS_s, PIS_{s^*}) = NS$, $\hat{A}$ abort with non- synchronization failure.*
      ii. *If $Identify(PIS_s, PIS_{s^*}) = FSS$, then $\hat{A}$ does the following:*
         A. *Set $lES_s = ES_s$.*
         B. *Set $lET_A = ET_A$ and $lET_B = ET_B$.*
         C. *Set $TET_A = H(ET_A||ES_s)$ and $TET_B = H(ET_B||ES_s)$.*
      iii. *If $Identify(PIS_s, PIS_{s^*}) = PSS$, then $\hat{A}$ does the following:*
         A. *Either set $lES_s = H(TET_A, ES_s)$ if $TET_A^* = TET_A$, or set $lES_s = H(TET_B, ES_s)$ if $TET_B^* = TET_B$.*
         B. *Set $lET_A = ET_A$ and $lET_B = ET_B$.*
         C. *Set $TET_A = H(ET_A||lES_s)$ and $TET_B = H(ET_B||lES_s)$.*
      iv. *Accept the session and store the updated PIS_s into SL.*
   (b) *Otherwise, $\hat{A}$ does nothing.*
3. *For the responder $\hat{B}$:*
   (a) *If $\hat{B}$ receives valid M_1, M_3, then $\hat{B}$ updates PIS_{s^*} as following:*
      i. *If $Identify(PIS_s, PIS_{s^*}) = NS$, $\hat{B}$ abort with non-synchronization failure.*
      ii. *If $Identify(PIS_s, PIS_{s^*}) = FSS$, then $\hat{B}$ does the following:*
         A. *Set $lES_{s^*} = ES_{s^*}$.*
         B. *Set $lET_B = ET_B$ and $lET_A = ET_A$.*
         C. *Set $TET_A^* = H(ET_A||ES_{s^*})$ and $TET_B^* = H(ET_B||ES_{s^*})$.*
      iii. *If $Identify(PIS_s, PIS_{s^*}) = PSS$, then $\hat{B}$ does the following:*
         A. *Either set $lES_{s^*} = H(TET_A, ES_{s^*})$ if $TET_A^* = TET_A$, or set $lES_{s^*} = H(TET_B, ES_{s^*})$ if $TET_B^* = TET_B$.*
         B. *Set $lET_A = ET_A$ and $lET_B = ET_B$.*
         C. *Set $TET_A^* = H(ET_A||lES_{s^*})$ and $TET_B^* = H(ET_B||lES_{s^*})$.*
   (b) *Else If $\hat{B}$ only receives the valid M_1, then $\hat{B}$ updates the PIS_{s^*} as following:*
      i. *If $Identify(PIS_s, PIS_{s^*}) = NS$, $\hat{B}$ abort with non-synchronization failure.*
      ii. *If $Identify(PIS_s, PIS_{s^*}) = FSS$, then $\hat{B}$ does the following:*
         A. *Set $lES_{s^*} = RES$.*
         B. *Set $lET_A = ET_A$ and $TET_A^* = H(ET_A||ES_{s^*})$.*
      iii. *If $Identify(PIS_s, PIS_{s^*}) = PSS$, then $\hat{B}$ does the following:*

A. *Set* $lES_{s^*} = RES$.
B. *Set* $lET_A = ET_A$
C. *Either set* $TET_A^* = H(ET_A||H(TET_A^*, ES_{s^*}))$ *if* $TET_A^* = TET_A$, *or set* $TET_A^* = H(ET_A||H(TET_B^*, ES_{s^*}))$ *if* $TET_B^* = TET_B$.

iv. Accept the session and store the updated PIS_{s^*} *into* SL.

(c) In other case $\hat{B}$ *does nothing.*

4. *Meanwhile, we require the whole* PIS *update procedure described above is atomic. In addition, the update steps need to ensure the established time of initiator is latest one. Moreover, If* PIS *is expired, then it will be set as* $PIS := (INIT, \perp, \hat{A}, I, \perp, \perp, \hat{B}, R, \perp, \perp)$ *or deleted from* SL. *And the parties can also negotiate together to roll back their* PIS *from established stage to either init or reset stage, based on successfully performing the protocol instance.*

Resistance of interference of adversary. Please note if the adversary is able to interfere with the communication between parties, and then there is unable to achieve full state synchronization. As describing in Definition 3, if the party $\hat{B}$ doesn't receive the last valid confirmation message, namely just receiving the valid M_1 and the $Identify(PIS_s, PIS_{s^*})$ satisfy either FSS or PSS, then only part of the PIS (i.e., the established time and its transformation of initiator) could be updated, in which case the protocol execution state of responder is set to be *reset*, namely the soundness property satisfied. To sum up, if the `MAC`scheme is secure as definition 6 then our proposed scheme is resist with the inference of adversary.

Impersonation detection. The PIS synchronization is the key criteria for detecting impersonation attack. So far we are able to online identify the identity impersonation attack undetected previously for honest party $\hat{A}$ and $\hat{B}$ when they proceed with some matching sessions.

Proposition 1. *When two honest parties* $\hat{A}$ *and* $\hat{B}$ *are executing two matching sessions with existing only benign passive adversary, and if the mutual authentication is valid but corresponding protocol execution states are non-synchronization after confirmation, then we could conclude that at least one of the parties has been impersonated to another.*

Proof. We require there exist only benign passive adversary who just act like wire between the two parties, due to the reasons: (i) if the passive adversary interfere with the communication then the confirmation protocol might not be able terminate with success, and(ii) if the active adversary who learned the long-term key and PIS of the responder then the impersonation detection would fail because the adversary could always impersonate as the responder (victim) to the session initiator.

Whereas, the validity of authentication procedure implies that the party who is processing the protocol instance possess corresponding long-term authentication key related to her identity. According to the execution synchronization rules if the instance aborted with non-synchronization error at step 2(a)i, 3(a)i, and 3(b)i, and then there must be some identity impersonation events has happened to either the honest parties, or even the ongoing instance is carried out by adversary.

Once the identity impersonation attack happened, we are require to figure out whose authentication credential is exposed, by comparing the established times recorded in PIS.[4]

[4] Perhaps the recorded information should include the 'location' of user while executing the protocol etc. which might help with figuring out the responsibility for the impersonation consequences, but that relies on specific application, AKE protocols and privacy concerns.

2.3 Security analysis of proposed framework

In order to define the security, we should treat three execution stages transition differently. In particular, there is no execution state at all for the *init* stage, thus we assume each PIS is initialized in secure way (namely, the stage shifted from *init* to *established* is secure and whole initialization procedure is isolated to the adversary). While within the *established* stage, we assume the protocol instance of Π generate its established stage value to be uniformly random. And the *reset* execution stage only used to re-establish the protocol execution state according to the definition 3, and the normal communications only work under *established* stage. Thus in the following we only discuss the security of proposed ESSF for established stage.

We assume there have n parties modeled as probabilistic polynomial time (in some security parameter) Turing machines (PPTs) and one special PPT called adversary. Each party has a long-term authentication key (asymmetric or symmetric) which is used to prove the identity of the party and can run a (polynomially-bounded) number of instances (sessions) implementing the considered AKE protocol Π. Basically, the adversary have full control over the network and make at most q_{se}, q_h queries on $Send$ and hash function H respectively, under security parameter λ. We allow the adversary interact with following types of queries:

- **EstablishSession**$(\hat{A}, I, \hat{B}, R, ID)$. The adversary may query this oracle to initiate a new session masquerading as party ID where ID provided by adversary which could be either $\hat{A}$ and $\hat{B}$. This query will respond with a session identifier *sid* and corresponding session key pair k, k_m. This query model the situation that the adversary learned the long-term authentication key of the party $\hat{A}$ at some point and try to impersonate as the victim to other party without knowing corresponding PIS. Note that the role of adversary is exactly as the same as the party ID's role in the session *sid*, e.g. if the $ID = \hat{A}$ then the role of adversary in the session is initiator either.
- **Send**$(\hat{A}, sid, m)$. $\mathcal{A}$ sends any message m to session *sid* held by $\hat{A}$.
- **Corrupt**$(\hat{A})$. The attacker learns the whole current internal session-specific states including the current PIS; reveal corresponding long-term secrets (such as private keys or master shared keys used across different sessions). Since then the corrupted party is considered completely controlled by the attacker from the time of corruption and the attacker can impersonate the party in all its actions departing arbitrarily from the protocol specifications. If a party is not corrupted it is said to be honest.

We define the security of ESSF from the perspective of secrecy and forgery resistance properties tied with the use of PIS in ESSF. The ESSF is said to satisfy secrecy property if there is no polynomial bounded adversary has non-negligible probability in winning the *Secrecy* experiment between a Challenger and the adversary under chosen-message attacks. In which experiment the following steps are performed:

1. The adversary $\mathcal{A}$ may issue polynomial number of queries as aforementioned in the security parameter, namely $\mathcal{A}$ makes queries: *EstablishSession*, *Send*, and *Corrupt*.
2. Meanwhile, the Challenger $\mathcal{C}$ take charge of generating the value of *established* stage (i.e. ES) for each session uniformly random, and then the corresponding parties update and store current PIS respectively in terms of the ESCP as Section 2.2 and synchronization rules as Definition 3.
3. At the end of the experiment, the $\mathcal{A}$ outputs a $PIS_t := (lES_t, \hat{A}, I, lET_A^t, TET_A^t, \hat{B}, R, lET_B^t, TET_B^t)$. We assume the current PIS identified by tuple $(\hat{A}, I, \hat{B}, R)$ stored at party $\hat{A}$ is PIS_a, and at party $\hat{B}$ is PIS_b. We say $\mathcal{A}$ wins the experiment if all the following conditions are hold:
   (a) The party $\hat{A}$ and $\hat{B}$ are not corrupted.

(b) Either $identify(PIS_t, PIS_a) \in \{FSS, PSS\}$, or $identify(PIS_t, PIS_b) \in \{FSS, PSS\}$.

We also define the $Forgery$ experiment between a Challenger and the adversary in the similar way. The ESSF is said to satisfy forgery resistance property under adaptively chosen-message attacks, if there is no polynomial bounded adversary has non-negligible probability in winning the *Fecrecy* experiment. In which experiment the following steps are performed:

1. The adversary $\mathcal{A}$ may issue polynomial number of queries as aforementioned in the security parameter, namely $\mathcal{A}$ makes queries: *EstablishSession*, *Send*, and *Corrupt*.
2. Meanwhile, the Challenger $\mathcal{C}$ take charge of generating the value of *established* stage (i.e. ES) for each session uniformly random, and then the corresponding parties update and store current PIS respectively in terms of the ESCP as Section 2.2 and synchronization rules as Definition 3.
3. At the end of the experiment, the $\mathcal{A}$ initializes a session via $(sid, k, k_m) =$ $EstablishSession(\hat{A}, I, \hat{B}, R, ID)$, where the party ID could be either $\hat{A}$ or $\hat{B}$ which are chosen by the adversary. And output a confirmation message m on her choice with a hashed test PIS^t included in m. We assume the current PIS identified by tuple $(\hat{A}, I, \hat{B}, R)$ stored at party $\hat{A}$ is PIS_a, and at party $\hat{B}$ is PIS_b. We say $\mathcal{A}$ wins the experiment if all the following conditions are hold:
   (a) The party $\hat{A}$ and $\hat{B}$ are not corrupted.
   (b) The partner of ID accepts the message m after performing the $Identify$ operation on the PIS^t carried in m, namely either $identify(PIS_t, PIS_b) \in FSS, PSS$ if $ID = \hat{A}$, or $identify(PIS_t, PIS_a) \in \{FSS, PSS\}$ if $ID = \hat{B}$.

In the subsequent, we formally analyze the secrecy and forgery resistance security properties respectively.

Theorem 1. *If the value of established execution stage for each session is chosen uniformly random, the hash function is one-way collision-resistant with respect to the definitions in Appendix A, then the ESSF provides secrecy.*

Poof: (sketch) We fix two honest party $\hat{A}$ and $\hat{B}$. In the *Secrecy* experiment, the goal of adversary is to learn at least one value of (lES, TET_A, TET_B) in current PIS_a or PIS_b. The adversary $\mathcal{A}$ might collect PIS related information by intercepting the confirmation messages between the two parties, or in particular performing $EstablishSession$ query as responder to obtain the confirmation message M_1. In order to win the *Secrecy* experiment, the adversary has two options: (i) randomly guess right the value of (lES, TET_A, TET_B); (ii) break the one-wayness security property of the hash function, namely $\mathcal{A}$ successfully reverse the hashed PIS_a or PIS_b carried in the first two messages of ESCP to corresponding pre-images. In which cases the probability of $\mathcal{A}$ in winning the experiment is bounded by a negligible fraction in the security parameter.

Theorem 2. *If the value of established execution stage for each session is chosen uniformly random, the hash function is one-way collision-resistant with respect to the definitions in Appendix A, then the ESSF is secure against forgery.*

Poof: (sketch) We examine the forgery resistance property in the extreme case that the adversary learns the long-term authentication key of a party and try to impersonate as the victim to another party without knowing corresponding PIS. Thus we prove the Theorem 2 by the following lemma.

Lemma 1. *If the value of established execution stage for each session is chosen uniformly random, the hash function is one-way collision-resistant, then the ESSF is resist with leakage of long-term secret.*

Proof. (sketch) Since the adversary $\mathcal{A}$ obtain the long-term key of $\hat{A}$ and $\hat{B}$, hence $\mathcal{A}$ is able to establish sessions to both parties respectively, to collect corresponding hashed PIS_a or PIS_b. However in order to make the party to accept the session, $\mathcal{A}$ is required to successfully perform the ESCP protocol, i.e. forge the confirmation message. To collect the useful information, the $\mathcal{A}$ could also just intercept the $mesg_2$ which contains the PIS_b. However in this case, $\mathcal{A}$ has to drop the $mesg_2$(that would lead PIS_b to be *reset* stage), otherwise $\hat{A}$ will update the PIS_a to new one. We particularly consider the situation that the $\mathcal{A}$ masquerade as $\hat{B}$, i.e. issue $(sid, k, k_m) = EstablishSession(\hat{A}, I, \hat{B}, R, \hat{B})$, and obtain the $mesg_1 := sid||M_1||H(k, lES_s)||ET_A||lET_A||H(k, TET_A)||lET_B||H(k, TET_B)$. Although $\mathcal{A}$ knows the session key k, she still need the information of lES_s, TET_A, TET_B to compute valid hashed PIS_b mainly including $H(M_1, lES_{s^*})$, $H(M_1, TET_A)$, $H(M_1, TET_B)$ and then the valid M_2. To achieve so, the adversary has the following options: (i) randomly guess right at least one value of (lES_s, TET_A, TET_B) or theirs corresponding hash value in session sid; (ii) break the one-wayness security property of the hash function, namely $\mathcal{A}$ successfully reverse the hashed PIS_a, i.e. $H(M_1, lES_{s^*}), H(M_1, TET_A), H(M_1, TET_B)$. In which cases the probability of $\mathcal{A}$ in winning the experiment is bounded by a negligible fraction in the security parameter.

3 Implementation of ESSF

While realizing the ESSF proposed in section 2, the most important issue is how to represent the j-th established stage value in particular AKE protocol. Furthermore, we require the protocol is able to deal with the circumstance of PSS, namely rollback to previous normal states. Those issues are left to specific AKE protocol.

Definition 4. *We say that an AKE protocol π securely realize ESSF, if each protocol instance of π generate the value of established execution stage to be uniform random, and shared only by unique-pairwise matching sessions.*

3.1 Abstracted *ES* generation scheme and applications

In practical, we are required to integrate the ESSF with an existent AKE protocol π (such as like SSL/TLS or IPSec/IKE) without any modification on π. Nevertheless, we only need the protocol π to provide an interface to access the transformation of its session key, e.g., $k' = H(k)$. Now we can give a generic ES generation scheme for AKE protocols based on the k'. The scheme for two parties' protocol proceeds as following:

1. $\hat{A}$ and $\hat{B}$ run the key exchange protocol with freshly generated parameters. Throughout this protocol run, assuming both parties compute transformation $k' = H(k)$, where k is the session key and H is a one-way hash function.
2. Compute a new encryption key $k_e = H_3(k', sid, "ENC")$, the MAC key $k_m = H_3(k', sid, "MAC")$, and the corresponding execution stage $ES_{j+1} = H_3(k', sid, ES'_j, "ES")$. Here we stress the ES'_j could be either the last successfully established stage value ES_j, or recorded transformation of ES_j (namely the TET_A or TET_B).

Concrete applications. The ESSF can be applied to wide application fields to design security solutions, for instance:

1. Build up two-factor authentication scheme. Where two-factor authentication is a scheme wherein two different factors are used in conjunction to authenticate. We can exploit the execution states as second authentication factor stored on some tamper-proof device (e.g., smart card). Then

the party can use its long-term private key to execute the AKE protocol as usual, and used the ESCP to do further authentication.
2. Enhance the password based authentication scheme. We can generate a high-entropy secret by modifying the password pw combining with the last established execution states, and the new password $pw' = H(pw, lES, ES)$. And then the client could send pw' instead of $H(pw)$ over TLS which is secure against 'off-line' guessing attack.

4 Closing Remark

We introduce a mechanism to characterize the execution states and propose a general framework for AKE protocols to mitigate the threat of leakage of long-term identity authentication key by providing identity impersonation detection capability with resistance of adversary interference. The impersonation detection feature can be very useful to help the user to reduce the damage caused by stolen long-term key. Additional, it requires no change to the AKE protocol when realizing the ESSF, since it is designed to simultaneously be used with multiple classes of AKE protocols. We have also discussed the proof that our scheme is secure under commonly used cryptographic assumptions. And We believe this is a viable solution to the problem of long-term credential theft.

References

1. Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In *CRYPTO*, pages 431–448, 1999.
2. 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.
3. Ran Canetti and Hugo Krawczyk. Security analysis of ike's signature-based key-exchange protocol. Cryptology ePrint Archive, Report 2002/120, 2002. `http://eprint.iacr.org/`.
4. Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock. Errors in computational complexity proofs for protocols. In Roy [14], pages 624–643.
5. Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock. Examining indistinguishability-based proof models for key establishment protocols. In Roy [14], pages 585–604.
6. 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.
7. 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.
8. 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.
9. Hugo Krawczyk. Sigma: The 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike-protocols. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 400–425. Springer, 2003.
10. Hugo Krawczyk. Hmqv: A high-performance secure diffie-hellman protocol. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer, 2005.
11. Leslie Lamport. Password authentication with insecure communication. *Commun. ACM*, 24(11):770–772, 1981.
12. Alfred Menezes and Berkant Ustaoglu. Comparing the pre- and post-specified peer models for key agreement. *IJACT*, 1(3):236–250, 2009.
13. D. Nali and Paul C. van Oorschot. Croo: A universal infrastructure and protocol to detect identity fraud. In *ESORICS*, pages 130–145, 2008.
14. Bimal K. Roy, editor. *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, volume 3788 of *Lecture Notes in Computer Science*. Springer, 2005.
15. SeongHan Shin, Kazukuni Kobara, and Hideki Imai. Leakage-resilient authenticated key establishment protocols. In *ASIACRYPT*, pages 155–172, 2003.
16. SeongHan Shin, Kazukuni Kobara, and Hideki Imai. Efficient and leakage-resilient authenticated key transport protocol based on rsa. In *ACNS*, pages 269–284, 2005.
17. 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 Preliminaries and Definitions

In this section, we recall the syntax and security definitions of the building blocks used in this paper.

A.1 One-way Collision-resistant Hash function

Let $H : \{h_k\}_{k \in K}$ be a family of keyed hash functions, where each h_k maps $\{0,1\}^d \to \{0,1\}^\kappa$ in which d is the length of domain D of pre-image. Let $H^{-1}(y) = \{x \in \{0,1\}^d : h_k(x) = y\}$ denote the pre-image set of y. A family H is one-way if it is computationally infeasible, given h_k and a range point $y = h_k(x)$, where x was chosen at random, to find a pre-image of y (whether x or some other).

Definition 5. *We say that H is collision resistant if the maximum success probability is ϵ over all PPT adversaries (i.e., probabilistic Turing machine running in time polynomial with security parameter κ) on finding a collision pair such that $H(m) = H(m')$, where ϵ is negligible and $(m, m') \in \{0,1\}^*$ with $m \neq m'$. We have*

$$\Pr\left[k \xleftarrow{\$} K : h_k(m) = h_k(m')\right] \leq \epsilon$$

A.2 Message Authentication Codes

A message authentication code is an algorithm $\mathtt{MAC}$. This algorithm implements a deterministic function $w = \mathtt{MAC}(K_m, m)$, 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 $w_i = \mathtt{MAC}(k_m, m_i)$. 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 6. *We say that $\mathtt{MAC}$ is a secure message authentication code, if*

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

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