Special English Edition of Journal of KIISC, Vol. 18 No.6 (B), December 2008

Introduction to Leakage-Resilient Authenticated Key Exchange Protocols and Their Applications

(Invited paper)

Hideki Imai[1], *SeongHan Shin*[2], *and Kazukuni Kobara*[3]
Chuo University, Bunkyo-ku, Tokyo, Japan

Abstract

Secure channels, indispensable to many applications, can be established by using an authenticated key exchange (AKE) protocol where the involving parties authenticate one another and then share authenticated session keys over insecure networks. In this paper, we introduce a new type of AKE protocols that are especially designed to minimize the damages caused by leakages of stored secrets. Such protocols are called Leakage-Resilient AKE (LR-AKE) protocols, whose motivation, design principles, several constructions, security analysis and applications are explained in detail.

Key words : authentication, key exchange, leakage-resilience, DH, RSA

I. Introduction

After the appearance of public-key cryptography in Diffie and Hellman's seminal paper [1], many researchers have extensively studied the problem of how to establish secure channels among participating parties. In practice, secure channels are indispensable because the major security concerns in a variety of upper-layer applications (e.g., e-commerce, Internet banking, and web-mail services) are confidentiality and integrity of communications. Such secure channels can be achieved by an authenticated key exchange (so-called, AKE) protocol where the participating parties authenticate one another and then share "authenticated" session keys over insecure networks (e.g., Internet). That means, an AKE protocol is said to provide semantic security of session keys if an active attacker, who can fully control the communications (e.g., by eavesdropping/replaying the messages, modification attacks, and impersonation/man-in-the-middle attacks), does not get any information about the shared session keys. Note that the Diffie-Hellman protocol [1] is only secure against a passive attacker who can just eavesdrop the communications. In order to combine authentication with key exchange, the participating parties need to share some information in advance (we will discuss more details in the next subsection). On the other hand, there are several works on authentication protocols without the use of key exchange in the literature. As Bellare and Rogaway claimed in [2], authentication itself is rarely useful in the absence of an associated key distribution (i.e., key exchange) except the case that a physically secured communication channel is used.

Manuscript received September 30, 2008.
Manuscript revised November 6, 2008
1) Hideki Imai, Chuo University/AIST, h-imai@aist.go.jp.
2) SeongHan Shin, AIST/Chuo University, seonghan.shin@aist.go.jp
3) Kazukuni Kobara, AIST/Chuo University, kobara_conf@aist.go.jp

1. Classification of Authenticated Key Exchange

In this subsection, we classify the existing AKE protocols into four settings according to the information, shared among participating parties. For clarity, we deal with two-party case (client C and server S) from here on.

- **PKI** (Public-Key Infrastructure) setting: In this setting, any party holds a public key that is certified by Certification Authority (CA) in the form of digital signatures. The public key is shared and public to everyone whereas the corresponding private key is kept secret by its owner. The representative examples subject to this setting can be found in many standards (e.g., IETF [3,4], ISO/IEC [5], and IEEE [6]). A typical approach for AKE protocols is to append/incorporate digital signatures to the Diffie-Hellman key exchange protocol [7-10]. Before running such AKE protocols, client C (resp., server S) should check the validity of server S's public key (resp., client C's public key) via CRL (Certificate Revocation Lists) or OCSP (Online Certificate Status Protocol) or SCVP (Server-based Certificate Verification Protocol). If a party skips this validity check, he/she may communicate with an attacker who impersonates the intended party.
- **HEK** (High-Entropy Key) setting: In this setting, client C and server S share a high-entropy key in the initialization phase. Here, a high-entropy key means a

key that a computationally-bounded attacker cannot find out through exhaustive search. In the AKE protocols for this setting, client C and server S exchange random numbers (or Diffie-Hellman public values) along with MACs (Message Authentication Codes), generated by the shared key [11]. Alternatively, client C and server S exchange random numbers encrypted, using symmetric-key encryptions, with the shared key. Of course, each party should keep the key secret all the time.

- **LEK** (Low-Entropy Key) setting: In this setting, client C and server S share a low-entropy key in the initialization phase. Contrary to a high-entropy key, even a computationally-bounded attacker can find out a low-entropy key because it is chosen in the range of exhaustive search. Throughout this paper, we fix such low-entropy keys to passwords (e.g., 4-digit pin codes or 6-symbol alphanumerical passwords) human beings can remember. Of course, other types of information such as biometric information can be used for low-entropy keys (we will discuss that later). In fact, passwords have been widely used for authentication so far because it is very convenient to users. However, if a party uses his/her password in poorly-designed authentication (and AKE) protocols, it might reveal the password that is commonly chosen from a small size of dictionary. For example, let us think of a simple challenge-response authentication protocol: 1) server S sends a random number c (as "challenge") to client C, 2) client C replies r=H(pw,c) (as "response") where pw is the client's password and H is a one-way hash function and 3) on receiving r, server S can authenticate client C by checking whether r is correct or not. In the above example, even a passive attacker who eavesdrops c and r can find out the client's password pw by simply guessing a password pw' and testing if r is equal to H(pw',c). This attack is done off-line and performed for all password candidates. These kinds of attacks, called off-line dictionary attacks, should be avoided in the AKE protocols. The problem of how to design AKE protocols secure against active attacks and off-line dictionary attacks has been unclear until Bellovin and Merritt's work [12]. The main idea of [12] is that client C and server S mask/de-mask Diffie-Hellman public values or a randomly-generated public key with password. A list of secure AKE protocols for this setting can be found in [13,14] where client C remembers his/her password only and server S stores the password or its transformed value (called, password verification data). Another point one has to keep in mind is that security of session keys in the AKE protocols for this setting is weaker than that in the AKE ones for PKI and HEK settings. The reason is that, though off-line dictionary attacks are not possible, an attacker can test a series of password candidates by interacting with the legitimate party. These kinds of attacks, called on-line dictionary attacks, are inevitable in the AKE protocols for this setting. However, an appropriate countermeasure to on-line dictionary attacks can be taken (e.g., if there are a fixed number of failed trials on password, server S locks the client's account for some time interval).
- **HK** (Hybrid Key) setting: In this setting, client C and server S share a low-entropy key and the server's uncertified public key (and a high-entropy key) in the initialization phase. The corresponding private key is, of course, kept secret by sever S. The work for this setting started from [15] and further studied in [16-19] where client C remembers his/her password and stores server S's public key, and server S stores password verification data and the private key. The main construction of [15-19] is that client C sends his/her password encrypted, using public-key encryptions, with server S's public key and then sever S can authenticate client C with the private key and password verification data. Recently, Kolesnikov and Rackoff [20,21] have proposed an AKE protocol for this setting where client C remembers his/her password and stores server S's public key and a MAC key whereas server S stores password verification data, the private key and the same MAC key. The construction of [20,21] is similar to [15-19] except appending MACs to the message client C sent to server S. In addition, the underlying public-key encryptions in [20,21] should be IND-CCA2 (indistinguishability against adaptive chosen ciphertext attacks) secure [22].

As we showed above, there are already many secure AKE protocols for different settings in the literature. In the next section, we introduce a new type of AKE protocols that can provide resilience against leakage of stored secrets from client and/or server.

II. Leakage-Resilient AKE Protocols

1. Motivation

In cryptography, the security of cryptographic algorithms or protocols usually depends on the assumption that secret keys are secure. In other words, a private key and a high-entropy key must be securely stored while a low-entropy key is remembered by human beings. However, securing secret keys is quite difficult in the real world because an attacker has a wide range of ways for obtaining secret keys including cryptanalysis, mishandling of information, breaks-in into a computer system, insider attacks, side channel attacks and social engineering (e.g., phishing [23]).

According to [24,25], the first top three sources of financial losses are virus attacks, unauthorized access to information (insider abuse of networks), and theft of laptops and other mobile devices. From Table 1 in [25], one can see that virus attacks are on the decrease from 2004 to 2008 while the percentage of laptop theft has no significant change at that period. In particular, laptops and PCs are stolen in every 53 seconds in the United States [26]. These top three incidents account for 75% in total and can also be linked to the leakage of stored secrets in practice.

If we consider leakage of stored secrets in the existing AKE protocols, it may end up with the total breakdown. Such examples are [7-11] because authentication is based on the stored secrets (i.e., private keys and shared high-entropy keys) so that their leakages directly result in impersonation of the victimized party. One may wonder if the use of TRM (Tamper-Resistant Modules) can prevent leakage of stored secrets. Of course, it may be an option but it seems hard to materialize perfect TRM with low cost [27,28]. Note that there is no silver bullet for preventing stored secrets from leaking out. Therefore, our motivation is to design an AKE protocol that is resilient against leakage of stored secrets as much as possible. On the other hand, the problem of leakage of stored secrets (i.e., private keys) in public-key encryptions and digital signatures has been considered in the recent years. The idea is to limit the damage that can be caused by an attacker who obtains the secrets from time to time. A brief survey for these works can be found in [29,30].

2. Design Principles

Before introducing Leakage-Resilient AKE (so-called, LR-AKE) protocols [31,32] and their general construction, we explain the main design principles for LR-AKE protocols. Remember that the goal of LR-AKE protocols is to minimize the damage caused by leakages of stored secrets from client and/or server.

The first design principle is multi-factor authentication where a high-entropy key (i.e., a secret stored on devices) and a low-entropy key (i.e., a password remembered by human beings) are combined to be used for authentication. The rationale behind this design principle is that we regard human beings as another storage for a low-entropy key. However, human beings cannot remember a high-entropy key as well as many different low-entropy keys.

The second design principle is proactive security of password. The idea is similar to [33] in the sense that a password is distributed as shares between the involving parties and each share evolves from time to time without changing the password. However, the essential feature of LR-AKE protocols is that the update information for each share comes from a (temporary) secret, shared between the client and the server only if the two parties successfully authenticate each other.

The third design principle is to use masking techniques that can prevent an attacker from performing off-line dictionary attacks on the password even when the attacker obtains a stored secret of the client. This is important because the LR-AKE protocols we will introduce in the following subsections are designed for a situation where a client, who is communicating with many different kinds of servers, remembers only one password and TRM are not available.

3. DH-based LR-AKE Protocol

In this subsection, we introduce the Diffie-Hellman based LR-AKE protocol [31] whose security is proven in the standard model with the reduction to the DDH (Decisional Diffie-Hellman) problem [34]. In fact, we introduce a modified protocol of [31] where the stored secrets of client and server are updated within the protocol execution.

We first explain some notations to be used. Let p and q be two large primes such that $q|p-1$. Let G be a finite cyclic group of prime order q, and g and h are two generators of G. We also denote by $MAC(mk,msg)$ a message authentication code (e.g., HMAC-SHA-1 [35]) on a message msg with a key mk.

In the initialization phase of the DH-based LR-AKE protocol, client C chooses a random id $pcid_1$ from an appropriate range and a random element $s_1 \in Z_q^*$, and then registers a hashed id $hpcid_1 = H(pcid_1)$ and a verifier $v_1 \equiv h^{s1+pw} (\mathrm{mod}\, p)$ to server S securely where H is a secure hash function and pw is the client's password. At the end of the initialization phase, client C remembers only one password pw and stores another secret s_1 along with the id $pcid_1$ on his/her devices while server S stores the verifier v_1 and the hashed id $hpcid_1$. Note that the password pw is distributed into two shares s_1 and $s_1 + pw$ by using (2,2)-threshold secret sharing scheme [36]. The initialization is done only once.

In the j-th $(j \geq 1)$ execution of the DH-based LR-AKE protocol, client C and server S authenticate each other and generate a session key. W.l.o.g., at the start of the j-th protocol execution client C stores the id $pcid_j$

and the secret s_j, and server S stores the hashed id $hpcid_j$ and the verifier v_j.

1. First, client C chooses a random element $x \in Z_q^*$ and computes the Diffie-Hellman public value $X \equiv g^x \pmod p$. The public value X is masked with v_j as follows: $\overline{X} \equiv X \cdot v_j \pmod p$ where $v_j \equiv h^{s_j + pw} \pmod p$. The client sends $pcid_j$ and $\overline{X}$ to the server.
2. If a hashed value of $pcid_j$, received from the client, is not equal to $hpcid_j$, server S terminates the protocol execution. Otherwise, server S chooses a random element $y \in Z_q^*$ and computes its Diffie-Hellman public value $Y \equiv g^y \pmod p$. The public value Y is masked with v_j as follows: $\overline{Y} \equiv Y \cdot v_j \pmod p$. In addition, the server de-masks the value $\overline{X}$ and computes the Diffie-Hellman key as follows: $g^{xy} \equiv (\overline{X}/v_j)^y \pmod p$. Then, server S computes a MAC $V_S = MAC(mk, 10 \| pcid_j \| S \| \overline{X} \| \overline{Y})$ where mk is the Diffie-Hellman key. The server sends its id S, $\overline{Y}$ and V_S to the client.
3. On receiving S, $\overline{Y}$ and V_S from the server, client C de-masks the value $\overline{Y}$ and computes the Diffie-Hellman key as follows: $g^{xy} \equiv (\overline{X}/v_j)^x \pmod p$. If V_S is not equal to $MAC(mk, 10 \| pcid_j \| S \| \overline{X} \| \overline{Y})$, the client terminates the protocol execution. Otherwise, the client computes a MAC $V_C = MAC(mk, 01 \| pcid_j \| S \| \overline{X} \| \overline{Y})$ and generates a session key $SK = MAC(mk, 11 \| pcid_j \| S \| \overline{X} \| \overline{Y})$ where mk is the Diffie-Hellman key. The MAC value V_C is sent to the server.
4. On receiving V_C from the client, server S verifies the MAC value. If V_C is not equal to $MAC(mk, 01 \| pcid_j \| S \| \overline{X} \| \overline{Y})$, the server terminates the protocol execution. Otherwise, the server generates a session key SK as follows: $SK = MAC(mk, 11 \| pcid_j \| S \| \overline{X} \| \overline{Y})$.

At the end of the j-th protocol execution, client C updates the id $pcid_j$ to $pcid_{j+1} = H(g^{xy} \| pcid_j \| S \| \overline{X} \| \overline{Y})$ and the secret s_j to $s_{j+1} = s_j + MAC(mk, 00 \| pcid_j \| S \| \overline{X} \| \overline{Y}) \bmod q$ while server S updates the hashed id $hpcid_j$ to $hpcid_{j+1} = H^2(g^{xy} \| pcid_j \| S \| \overline{X} \| \overline{Y})$ and the verifier v_j to $v_{j+1} = v_j \cdot h^{us} \bmod p$ where $us = MAC(mk, 00 \| pcid_j \| S \| \overline{X} \| \overline{Y})$. Alternatively, the client can update the id $pcid_j$ as in the initialization phase: client C chooses a random id $pcid_{j+1}$ from an appropriate range and then registers a hashed id $hcid_{j+1} = H(pcid_{j+1})$ to server S securely with the session key SK. Finally, the client stores the id $pcid_{j+1}$ and the secret s_{j+1}, and the server stores the hashed id $hpcid_{j+1}$ and the verifier v_{j+1} which will be used for the next session.

As we explained in Section II.2, the DH-based LR-AKE protocol realizes the three design principles: 1) the client uses the password, remembered by his/her own, and the secret, stored on his/her devices, for authentication; 2) it provides proactive security of password because each share of the password is distributed to the client and the server, and is updated from an update secret, shared between them only if authentication succeeds, without changing the password; and 3) it uses the masking technique with another generator h for the Diffie-Hellman key exchange.

4. RSA-based LR-AKE Protocol

In this subsection, we introduce the RSA-based LR-AKE protocol [32] whose security is proven in the random oracle model [37] with the reduction to the RSA problem [38].

First and foremost, we explain the RSA function [38], one of the most famous public key cryptosystems. It has the following three algorithms:

- On input of the security parameter l, an RSA key generation algorithm outputs a pair of public/private keys $(PubK=(e,n), \Pr iK=(d,n))$ such that (1) p, q are distinct odd primes of the same length, (2) $n = pq$ where $2^{l-1} + 1 \le n \le 2^l$ and (3) $e, d \in Z_{\varphi(n)}^*$ are

integers satisfying $ed \equiv 1 \bmod \varphi(n)$ where $\varphi(\cdot)$ is Euler's phi function.

- Given the public key $PubK$ and a message $msg \in Z_n^*$, an encryption algorithm produces a ciphertext $c \equiv msg^e \bmod n$.
- Given the private key $\Pr iK$ and a ciphertext $c \in Z_n^*$, a decryption algorithm recovers the message $msg \equiv c^d \bmod n$.

Next, we explain some notations to be used. Let G be a full-domain hash (FDH) function that maps $\{0,1\}^*$ to $Z_n^* ¥ \{1\}$. Let H be a hash function whose input is arbitrary but output is in the range of $\{0,1\}^k$ where k is the security parameter for hash functions.

In the initialization phase of the RSA-based LR-AKE protocol, client C chooses a random id $pcid_1$ from an appropriate range and a random element $s_1 \in \{0,1\}^k$, and then registers a hashed id $hpcid_1 = H(pcid_1)$ and a verifier $v_1 = s_1 \oplus pw$ to server S securely where pw is the client's password. At the same time, server S sends its RSA public key $PubK = (e,n)$, generated from the above RSA key generation algorithm, securely to client C. At the end of the initialization phase, client C remembers only one password pw and stores another secret s_1, the RSA public key $PubK$ along with the id $pcid_1$ on his/her devices while server S stores the verifier v_1, its RSA private key $\Pr iK = (d,n)$ and the hashed id $hpcid_1$. Note that the password pw is distributed into two shares s_1 and v_1 by using (2,2)-threshold secret sharing scheme [36]. The initialization is done only once.

In the j-th $(j \geq 1)$ execution of the RSA-based LR-AKE protocol, client C and server S authenticate each other and generate a session key. W.l.o.g., at the start of the j-th protocol execution client C stores the id $pcid_j$, the secret s_j and the RSA public key $PubK$, and server S stores the hashed id $hpcid_j$, the verifier v_j and the RSA private key $\Pr iK$.

1. First, client C chooses a random element $x \in Z_n^*$ and encrypts x with the RSA public key $PubK$: $y \equiv x^e \bmod n$. The client also computes a FDH value $W = G(v_j)$, where $v_j = s_j \oplus pw$, and the ciphertext y is then masked with W as follows: $Z \equiv y \cdot W \pmod n$. The client sends the id $pcid_j$, and Z to the server.
2. If a hashed value of $pcid_j$, received from the client, is not equal to $hpcid_j$, server S terminates the protocol execution. Otherwise, the server computes a FDH value $W = G(v_j)$ with which it de-masks the value Z as follows: $y \equiv Z/W \pmod n$. Then, the server decrypts the resultant ciphertext y with the RSA private key $\Pr iK$ in order to obtain x: $x \equiv y^d \bmod n$. With x and other values, the server computes a hash value $V_S = H(1 \| pcid_j \| S \| Z \| v_j \| x)$ that is sent to the client with the id S.
3. On receiving V_S from the server, client C verifies the hash value. If V_S is not equal to $H(1 \| pcid_j \| S \| Z \| v_j \| x)$, the client terminates the protocol execution. Otherwise, the client computes a hash value $V_C = H(2 \| pcid_j \| S \| Z \| v_j \| x)$ and generates a session key SK as follows: $SK = H(3 \| pcid_j \| S \| Z \| v_j \| x)$. The hash value V_C is sent to the server.
4. On receiving V_C from the client, server S verifies the hash value. If V_C is not equal to $H(2 \| pcid_j \| S \| Z \| v_j \| x)$, the server terminates the protocol execution. Otherwise, the server generates a session key SK as follows: $SK = H(3 \| pcid_j \| S \| Z \| v_j \| x)$.

At the end of the j-th protocol execution, client C updates the id $pcid_j$ to $pcid_{j+1} = H(0 \| pcid_j \| S \| Z \| v_j \| x)$ and the secret s_j to $s_{j+1} = s_j \oplus H(4 \| pcid_j \| S \| Z \| v_j \| x)$ while server S updates the hashed id $hpcid_j$ to $hpcid_{j+1} = H^2(0 \| pcid_j \| S \| Z \| v_j \| x)$ and the verifier v_j to $v_{j+1} = v_j \oplus H(4 \| pcid_j \| S \| Z \| v_j \| x)$. Alternatively,

the client can update the id $pcid_j$ as in the initialization phase: client C chooses a random id $pcid_{j+1}$ from an appropriate range and then registers a hashed id $hpcid_{j+1} = H(pcid_{j+1})$ to server S securely with the session key SK. Similarly, the server also can update its RSA public key as in the initialization phase securely with the session key SK. Finally, the client stores the id $pcid_{j+1}$, the secret s_{j+1} and the RSA public key $PubK$, and the server stores the hashed id $hpcid_{j+1}$, the verifier v_{j+1} and the RSA private key $\mathrm{Pr}iK$ which will be used for the next session.

The RSA-based LR-AKE protocol realizes the same three design principles as sketched in the previous subsection with the only difference: it uses the masking technique with a FDH function G for the RSA ciphertext.

5. A General Construction for LR-AKE

Here, we propose a general construction for LR-AKE protocols by strengthening security of any underlying PAKE protocols (e.g., [13]) against leakage of stored secrets. Specifically,

1. In the initialization phase, client C chooses a random id $pcid_1$ and another secret s_1 from an appropriate range (which should be equal to or larger than the space of password), and then registers a hashed id $hpcid_1 = H(pcid_1)$ and a verifier v_1 to server S securely where H is a secure hash function and v_1 is a combined value from the secret s_1 and the client's password pw. At the end of the initialization phase, client C remembers only one password pw and stores the secret s_1 along with the id $pcid_1$ on his/her devices while server S stores the verifier v_1 and the hashed id $hpcid_1$.
2. In the j-th $(j \geq 1)$ protocol execution, client C and server S run the underlying PAKE protocol P using the id $pcid_j$ and the verifier v_j instead of the client's id and the password verification data (password pw or its transformed value), respectively. If the client and the server successfully authenticate each other, they end up with the shared master secret ms from which a session key is generated.
3. At the end of the j-th protocol execution, client C updates the id $pcid_j$ to $pcid_{j+1}$ and the secret s_j to s_{j+1} while server S updates the hashed id $hpcid_j$ to $hpcid_{j+1}$ and the verifier v_j to v_{j+1} where the $(j+1)$-th secrets (i.e., $pcid_{j+1}$, s_{j+1}, $hpcid_{j+1}$ and v_{j+1}) are updated with the j-th secrets (i.e., $pcid_j$, s_j, $hpcid_j$ and v_j), the master secret ms and other values. Finally, the client stores the id $pcid_{j+1}$ and the secret s_{j+1}, and the server stores the hashed id $hpcid_{j+1}$ and the verifier v_{j+1} which will be used for the next session.

The graphical description of the general LR-AKE construction from any PAKE protocol P is shown in Fig. 1.

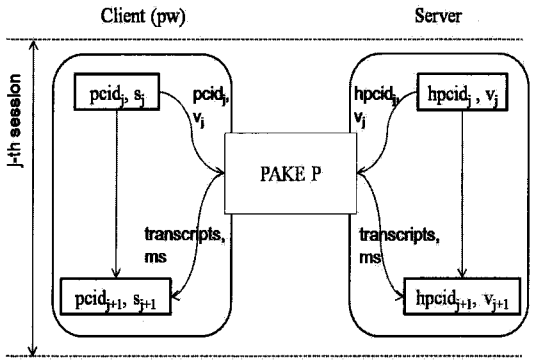

Fig. 1 A general LR-AKE construction from any PAKE protocol P

The basic idea of the above general LR-AKE construction is that any PAKE protocol P has its own masking technique so that we can construct an LR-AKE protocol by adding multi-factor authentication and proactive security of password to P. However, this approach does not necessarily result in an efficient LR-AKE protocol (see Section II.4). As for efficiency, the general LR-AKE construction does not require additional communications but, depending on the underlying PAKE protocol, it may need some extra computational costs for updating the server's stored secrets. The obvious, but challenging, topic for LR-AKE protocols is to find out a new masking technique.

III. Security of LR-AKE Protocols

1. Security against Leakage of Stored Secrets

In this subsection, we explain the security of the LR-AKE protocols against leakage of stored secrets from client and/or server. From here on, we consider semantic security of session keys and security of password under the following three cases:

- Case1: An attacker obtains the stored secrets from client C where the stored secrets are high-entropy keys or private keys corresponding to (certified) public keys for public-key encryptions or digital signatures.
- Case2: An attacker obtains the stored secrets from server S where the stored secrets are verifiers or password verification data, directly related to the client's authentication information, and/or high-entropy keys (except private keys).
- Case3: An attacker obtains the stored secrets from server S where the stored secrets are private keys corresponding to (certified) public keys for public-key encryptions or digital signatures.

In the above cases, we assume that the client/server's (certified) public keys are known to an attacker. When discussing semantic security of session keys in the followings, we consider an active attacker who can fully control the communications (e.g., by eavesdropping/replaying the messages, modification attacks, and impersonation/man-in-the-middle attacks). For other security properties (e.g., perfect forward secrecy and resistance to server-compromise impersonation attacks), we briefly explain each security notion with an attacker's somewhat-restricted capabilities.

1.1. Security against Case1

Here, we explain the security of session keys and the security of password against Case1 in the LR-AKE protocols. The Case1 means that an attacker obtains the client's id $pcid_j$ and another secret s_j of the j-th $(j \geq 1)$ protocol execution. Of course, Case1 can happen repeatedly over time.

The LR-AKE protocols provide semantic security of session keys against Case1 in the sense that, if Case1 happens at some specific sessions, the security of session keys depends on the number of on-line dictionary attacks on the password. On the other hand, if Case1 does not happen in the j-th $(j \geq 1)$ protocol execution, on-line dictionary attacks are not possible in the LR-AKE protocols since the previously-leaked secrets are useless due to proactive security of password. If an attacker wants to continue on-line dictionary attacks, the attacker should obtain the client's updated stored secret for every session (see Fig. 2).

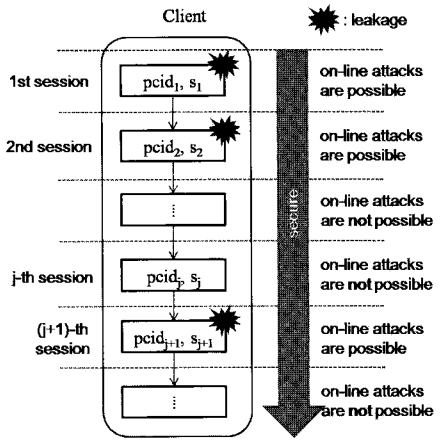

Fig. 2 Security of session keys against Case1

In addition, the LR-AKE protocols guarantee information-theoretic security of password against Case1 simply because an attacker cannot get any information about the password with only one share of (2,2)-threshold secret sharing scheme.

1.2. Security against Case3

Here, we explain the security of session keys against Case3 in the LR-AKE protocols where an attacker obtains the server's private key $\mathrm{Pr}iK$ (e.g., (d,n) in the RSA-based LR-AKE protocol of Section II.4). In Fig. 3, we consider the LR-AKE protocols where the server's private key $\mathrm{Pr}iK$ is fixed for all sessions.

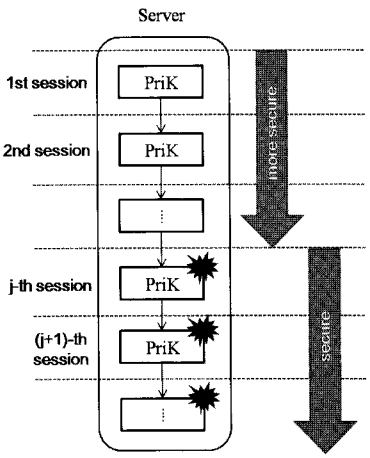

Fig. 3 Security of session keys against Case3

If Case3 does not happen, the LR-AKE protocols provide semantic security of session keys since an attacker cannot break the underlying public-key encryption and the high-entropy key based authentication (denoted by 'more secure' in Fig. 3). Even if Case3 happens in the j-th protocol execution, the security of session keys in the following sessions are guaranteed because the client and the server share the high-entropy key as in [11] (denoted by 'secure' in Fig. 3). In order to provide more security against Case3, the server can update its private key by sending a newly-generated public key securely to the client with the temporal session keys.

1.3. Security against Case2 (and Case1)

Here, we explain the security of password against Case2 (and Case1) in the LR-AKE protocols. The Case2 means that an attacker obtains the server's hashed id $hpcid_j$ and verifier v_j of the j-th $(j \geq 1)$ protocol execution. Clearly, the LR-AKE protocols provide information-theoretic security of password against Case2 by the same reason as we discussed in Section III.1.1.

Since the client remembers only one password in the LR-AKE protocols, we need to protect the password as much as possible. Due to the proactive security of password, we can allow multiple leakages of stored secrets from the client and the server. As the password is distributed with (2,2)-threshold secret sharing scheme, only the simultaneous leakages of stored secrets from both the client and the server in the same session lead to the exposure of the password (see Fig. 4).

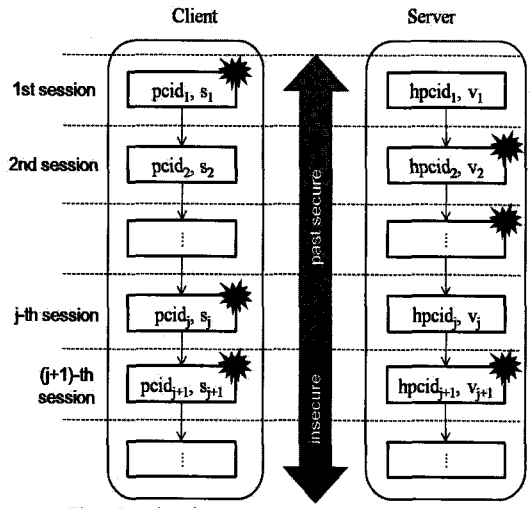

Fig. 4 Security of password against Case1 and Case2

1.4. Security against Case2

Here, we discuss the security of session keys against Case2 in the LR-AKE protocols. In fact, the LR-AKE protocols shown in Section II.3, II.4 and II.5 do not provide semantic security of session keys against Case2 because an attacker, who obtains the verifier v_j and intercepts the id $pcid_j$ of the j-th $(j \geq 1)$ protocol execution, can freely impersonate the client.

In the below, we propose a simple and generic method that can convert the LR-AKE protocols of Section II into ones with semantic security of session keys against Case2.

1. In the initialization phase of the LR-AKE protocols, client C additionally generates a pair of public/private keys $(PubK_C, \Pr iK_C)$ for public-key encryptions or digital signatures, and then registers additionally the public key $PubK_C$ to server S securely. At the end of the initialization phase of the LR-AKE protocols, client C and server S additionally stores the private key $\Pr iK_C$ and $PubK_C$, respectively.
2. In the j-th $(j \geq 1)$ protocol execution, client C and server S run the LR-AKE protocols as they are with the following additional procedures: 1) As a final message from the client to the server, client C computes a signature $Sig(\Pr iK_C, transcript)$, where $transcript$ is a collection of exchanged messages and $Sig(\Pr iK_C, msg)$ is a signature on a message msg with the signature key $\Pr ik_C$, and then sends $Sig(\Pr iK_C, transcript)$ to server S; and 2) The server successfully completes the protocol execution of the LR-AKE protocols only if the signature $Sig(\Pr iK_C, transcript)$ is valid (by verifying that with the verification key $PubK_C$).

The above converted LR-AKE protocols provide semantic security of session keys against Case2 since an attacker cannot generate a valid signature on the messages exchanged between the attacker and the server.

1.5. Security against Case2 and Case3

Any AKE protocols are not secure against Case2 and Case3 because an attacker has now enough information to impersonate the server to the client. However even in these cases, it is desirable to prevent the attacker from impersonating the client to the server. This security

property, called resistance to SCI (Server-Compromise Impersonation) attacks, provides another layer of security in the sense that sessions established by the server, without being actively controlled by the attacker, remain secure even if Case2 and Case3 happen. Here, we refer the above security notion to 'strong' resistance to SCI attacks. Also, we define 'weak' resistance to SCI attacks in the sense that sessions established by the server, without being actively controlled by the attacker, and the client, whose messages are not actively controlled by the attacker as well, remain secure even if Case2 and Case3 happen. In other words, the weak resistance to SCI attacks allows the attacker only to impersonate the client.

In the LR-AKE protocols, Case2 and Case3 means that an attacker obtains the server's hashed id $hpcid_j$ and verifier v_j (and its private key $\mathrm{Pr}iK$ if any) of the j-th $(j \geq 1)$ protocol execution. The LR-AKE protocols shown in Section II.3, II.4 and II.5 provide weak resistance to SCI attacks since the attacker cannot compute the pre-image of $hpcid_j$ where the pre-image $pcid_j$ should be sent out to the server by the attacker. On the other hand, these protocols do not provide strong resistance to SCI attacks: if the attacker intercepts messages from the legitimate client, the attacker knows $pcid_j$ that is used to impersonate the client to the server. Interestingly, the converted LR-AKE protocols by the method in Section III.1.4 are strongly resistant to SCI attacks since the attacker cannot generate a valid signature on messages, sent out to the server. Note that the general idea to prevent SCI attacks is to share the asymmetric-type keys between the client and the server (c.f., [39]).

1.6. Security against Case1, Case2 and Case3

In general, a key exchange protocol is said to have PFS (Perfect Forward Secrecy) if the previously-established session keys (and deleted from the memory) before the compromise of the long-term keys cannot be recovered. Here, we extend the security notion in the following way: A protocol has the extended PFS property if the leakages of all the secrets do not compromise the security of session keys established in the previous sessions and erased from the memory before the leakages occurred. The extended PFS property is concerned with limiting the effects of total leakages of all the secrets which can be possible with Case1, Case2 and Case3 in the LR-AKE protocols.

It is relatively clear that the DH-based LR-AKE protocol of Section II.3 provides the extended PFS property against Case1, Case2 and Case3 since the session keys are generated from the Diffie-Hellman key exchange. Actually, this is the same as shown in Fig. 4 in that the past session keys are secure. However, it becomes somewhat complicated in the RSA-based LR-AKE protocol of Section II.4. That is, the extended PFS property can be guaranteed against Case1, Case2 and Case3 in the RSA-based LR-AKE protocol only for the sessions without preceding leakage of stored secrets from the client and the server (e.g., 1 to $(j-1)$-th session in Fig. 5).

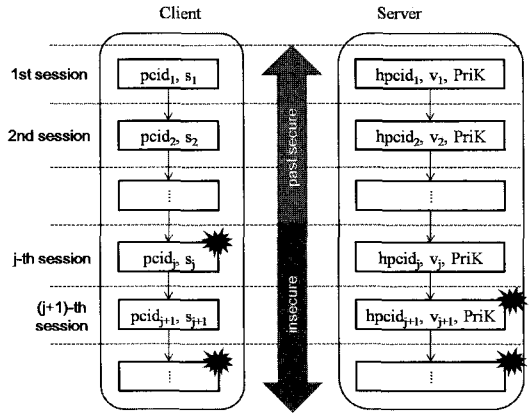

Fig. 5 Extended PFS property against Case1, Case2 and Case3 in the RSA-based LR-AKE protocol

2. Comparison

In this subsection, we compare the existing AKE protocols in terms of how much each protocol is secure against leakage of stored secrets from client or server. The brief comparative result is in Table 1 where ○ (resp., ×) indicates "secure" (resp., "insecure") with respect to security of session keys and security of password.

From Table 1, one can easily see that the LR-AKE protocols have the most leakage-resilience of stored secrets from the client and the server, respectively. Other security properties of the LR-AKE protocols against a combination of Case1, Case2 and Case3 are already discussed in Section III.1. For more detailed comparisons, refer to [31,32].

Table 1: Comparison of the existing AKE protocols in terms of security against leakage of stored secrets

Settings	Protocols	Security of session keys			Security of password	
		Case1	Case2	Case3	Case1	Case2
PKI	[7-10]	×	○	×	–	–
HEK	[11]	×	×	○	–	–
LEK	[12,13]	○	×	○	○	×
HK	[15-21]	○	×	×	○	×
	LR-AKE	○	○*1	○	○	○

*1: see Section III.1.4

IV. Applications

1. Credential Services

As one of the applications, the LR-AKE protocols can be extended for the use of credential services [40]. Let us consider a roaming client who accesses a network from different locations and wants to temporarily retrieve his/her private key corresponding to the certified public key for public-key encryptions or digital signatures. This roaming protocol can be supported by credential services where a credential server authenticates the client and then helps the client to recover his/her private key. The main problem in credential services is leakage of stored secrets (i.e., secrecy of private key with respect to the involving credential server). By proposing an extended LR-AKE protocol for credential services, we can provide a higher level of security against leakage of stored secrets as well as secrecy of private key w.r.t. to the credential server. Specifically, the client's private key is additionally distributed between the client and the server in the initialization phase and, after sharing an authenticated session key between them in the protocol execution, the server sends securely the partial private key to the client and then each share is updated with the shared master secret (i.e., proactive security of private key) like the stored secrets of the client and the server. For more details, refer to [40].

2. Wireless Networks Security

A challenging problem in wireless networks is to secure communications, without affecting mobility and quality-of-service, as well as to secure mobile devices of users. This problem can be solved by using the LR-AKE protocols [41,42]. In [41], a modified version of the RSA-based LR-AKE protocol and its cooperation with PKI are used for securing the handover procedure of host mobility protocol (i.e., Mobile Internet Protocol version 6 (MIPv6)). The similar approach is also used for dealing with AAA (Authentication, Authorization and Accounting) issues in the Network Mobility (NEMO) environment in [42]. Here, we stress that the RSA-based LR-AKE protocol [32] is very suitable for wireless networks since it is the most efficient among the existing AKE protocols based on RSA and password. In particular, if pre-computation is possible the client needs to compute only one modular multiplication (not modular exponentiation)! Refer to [32] for more detailed discussions.

3. Personal Networks Security

The concept of personal networks has been considered as representative for the next generation networks. However, the current security mechanism has a weakness because leakage of stored secrets from one node compromises the whole security of personal networks. In order to avoid this weakness, a modified LR-AKE protocol is used to deal with key management and security against leakage of stored secrets in the new security architecture for personal networks [43]. Specifically, the proposed security architecture involves two different types of communications: PN (Personal Networks) wide communication and communication between P-PANs (Private Personal Area Networks) of two different users. Note that the LR-AKE protocols are designed to provide maximum security against leakage of stored secrets, not perfect security.

4. Biometric-based Authentication

The LR-AKE protocols can be applied to biometric-based authentication simply by substituting password with biometric information. A concrete example is to use fuzzy extractors [44] where a client additionally stores public information about its biometric input and the randomness (extracted from the biometric input) is used as password in the LR-AKE protocols. Interestingly, this application directly provides privacy of the biometric template as the security of password in the LR-AKE protocols (c.f., [45]).

V. Summary

In this paper, we have introduced the Leakage-Resilient AKE (LR-AKE) protocols, whose primary goal is to minimize the damages caused by leakages of stored secrets, by explaining their motivation, design principles, several constructions, security analysis and applications. Note that security guarantees (e.g., data protection, prevention of identity theft and exposure of private information, and access control) of many applications with the use of cryptographic algorithms/protocols are also linked to the problem of how to protect the cryptographic keys.

As a final remark, we strongly claim that storing some secrets on client's devices is the very small cost for a higher level of security against leakage of stored secrets because the use of laptops and mobile devices (e.g., cell phones or PDAs) is already prevalent in the real world. According to [46], the number of cell phone subscribers is expected to reach at 4 billion people by the end of 2008.

References

[1] Diffie, W., and Hellman, M.: 'New directions in cryptography', IEEE Trans. Information Theory, 1976, IT-22, (6), pp. 644-654

[2] Bellare, M., and Rogaway, P.: 'Provably-secure session key distribution: the three party case', Proc. ACM Sym. Theory of Computing, 1995, pp. 57-66

[3] IETF: 'Transport layer security (tls)', http://www.ietf.org/html.charters/tls-charter.html

[4] IETF: 'Internet key exchange (IKEv2) protocol', 2004, http://tools.ietf.org/html/draft-ietf-ipsec-ikev2-17
[5] ISO/IEC 11770-3: 'Information technology – security techniques – key management – part 3: mechanisms using asymmetric techniques', 2008
[6] IEEE 1363-2000: 'Standard specifications for public key cryptography', http://grouper.ieee.org/groups/1363/P1363/index.html
[7] Shoup, V.: 'On formal models for secure key exchange', Theory of Cryptography Library, 1999
[8] Krawczyk, H.: 'SIGMA: the 'SIGn-and-Mac' approach to authenticated Diffie-Hellman and its use in the IKE protocols', Proc. CRYPTO 2003, 2003, pp. 400-425
[9] Menezes, A., Qu, M., and Vanstone, S.: 'Some new key agreement protocols providing mutual implicit authentication': Proc. Selected Areas in Cryptography, 1995
[10] Krawczyk, H.: 'HMQV: a high-performance secure Diffie-Hellman protocol', Proc. CRYPTO 2005, 2005, pp. 546-566
[11] Bellare, M., and Rogaway, P.: 'Entity authentication and key distribution', Proc. CRYPTO'93, 1993, pp. 232-249
[12] Bellovin, S. M., and Merritt, M.: 'Encrypted key exchange: password-based protocols secure against dictionary attacks', Proc. IEEE sym. Security and Privacy, 1992, pp. 72-84
[13] IEEE P1363.2: 'Standard specifications for password based public key cryptographic techniques', http://grouper.ieee.org/groups/1363/passwdPK/submissions.html
[14] http://jablon.org/passwordlinks.html
[15] Lomas, T., Gong, L., Saltzer, J., and Needham, R.: 'Reducing risks from poorly chosen keys', Proc. ACM Sym. Operating System Principles, 1989, pp. 14-18
[16] Gong, L., Lomas, T., Needham, R., and Saltzer, J.: 'Protecting poorly-chosen secrets from guessing attacks', IEEE J. Selected Areas in Communications, 1993, 11, (5), pp. 648-656
[17] Gong, L.: 'Optimal authentication protocols resistant to password guessing attacks', Proc. IEEE Computer Security Foundation Workshop, 1995, pp. 24-29
[18] Halevi, S., and Krawczyk, H.: 'Public-key cryptography and password protocols', ACM Trans. Information and System Security, 1999, 2, (3), pp. 230-268
[19] Boyarsky, M. K.: 'Public-key cryptography and password protocols: the multi-user case', Proc. ACM Conf. Computer and Communications Security, 1999, pp. 63-72
[20] Kolesnikov, V., and Rackoff, C.: 'Key exchange using passwords and long keys', Proc. TCC 2006, 2006, pp. 100-119
[21] Kolesnikov, V., and Rackoff, C.: 'Password mistyping in two-factor-authenticated key exchange', Proc. ICALP (2), 2008, pp. 702-714
[22] Rackoff, C., and Simon, D.: 'Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack', Proc. CRYPTO'91, 1992, pp.433-444
[23] Jakobsson, M., and Myers, S.: 'Phishing and counter-measures' (John Wiley and Sons Inc., 2006)
[24] Lawrence, A. G., Martin, P. L., William, L., and Robert R.: 'CSI/FBI computer crime and security survey', CSI, 2006, http://i.cmpnet.com/gocsi/db_area/pdfs/fbi/FBI2006.pdf
[25] Robert, R.: 'CSI computer crime & security survey', 2008, available at http://www.gocsi.com/
[26] http://www.user-groups.net/safenet/computer_theft.html
[27] Anderson, R., and Kuhn, M.: 'Tamper resistance – a cautionary note', Proc. USENIX Workshop on Electronic Commerce, 1996, pp. 1-11
[28] Anderson, R., and Kuhn, M.: 'Low cost attacks on tamper resistant devices', Proc. Security Protocols, 1997, pp. 125-136
[29] Franklin, M.: 'A survey of key evolving cryptosystems', Int. J. Security and Networks, 2006, 1, (1/2), pp. 46-53
[30] Itkis, G.: 'Forward security – adaptive cryptography: time evolution', Handbook of Information Security, 2006, 3, chapter 199 H. Bidgoli (Ed). Wiley Publishers
[31] Shin, S. H., Kobara, K., and Imai, H.: 'A simple leakage-resilient authenticated key establishment protocol, its extensions, and applications', IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences, 2005, E88-A, (3), pp. 736-754. A preliminary version appeared at ASIACRYPT 2003.
[32] Shin, S. H., Kobara, K., and Imai, H.: 'An efficient and leakage-resilient RSA-based authenticated key exchange protocol with tight security reduction', IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences, 2007, E90-A, (2), pp. 474-490
[33] Ostrovsky, R., and Yung, M.: 'How to withstand mobile virus attacks', Proc. ACM Sym. Principles of Distributed Computing, 1991, pp. 51-59
[34] Boneh, D.: 'The decisional Diffie-Hellman problem', Proc. ANTS-IV, 1998, pp. 48-63
[35] Krawczyk, H., Bellare, M., and Canetti, R.: 'HMAC: keyed-hashing for message authentication', IETF RFC 2104, 1997
[36] Shamir, A.: 'How to share a secret', Communications of the ACM, 1979, 22, (11), pp. 612-613
[37] Bellare, M., and Rogaway, P.: 'Random oracles are practical: a paradigm for designing efficient protocols', Proc. ACM CCS'93, 1993, pp. 62-73
[38] Rivest, R. L., Shamir, A., and Adelman, L.: 'A method for obtaining digital signature and public-key cryptosystems', Technical Memo LCS/TM82, 1977
[39] Gentry, C., MacKenzie, P., and Ramzan, Z.: 'A method for making password-based key exchange resilient to server compromise', Proc. CRYPTO 2006, 2006, pp. 142-159
[40] Shin, S. H., Kobara, K., and Imai, H.: 'A secure authenticated key exchange protocol for credential services', IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences, 2008, E91-A, (1), pp. 139-149
[41] Fathi, H., Shin, S. H., Kobara, K., Chakraborty, S., Imai, H., and Prasad, R.: 'Leakage-resilient security architecture for mobile IPv6 in wireless overlay networks', IEEE J. Selected Areas in Communications, 2005, 23, (11), pp. 2182-2193
[42] Fathi, H., Shin, S. H., Kobara, K., Chakraborty, S., Imai, H., and Prasad, R.: 'LR-AKE-based AAA for network mobility (NEMO) over wireless links', IEEE J. Selected Areas in Communications, 2006, 24, (9), pp. 1725-1737
[43] Shin, S. H., Fathi, H., Kobara, K., Prasad, N. R., and Imai, H.: 'A new security architecture for personal networks and its performance evaluation', IEICE Trans. Communications, 2008, E91-B, (7), pp. 2255-2264
[44] Dodis, Y., Reyzin, L., and Smith, A.: 'Fuzzy extractors: how to generate strong keys from biometrics and other noisy data', Proc. EUROCRYPT 2004, 2004, pp. 523-540