Leakage-Resilient Authenticated Key Establishment Protocols

SeongHan Shin, Kazukuni Kobara, and Hideki Imai

Institute of Industrial Science, The University of Tokyo,
4-6-1 Komaba, Meguro-ku, Tokyo 153-8505, Japan
shinsh@imailab.iis.u-tokyo.ac.jp, {kobara,imai}@iis.u-tokyo.ac.jp
http://imailab-www.iis.u-tokyo.ac.jp/imailab.html

Abstract. Authenticated Key Establishment (AKE) protocols enable two entities, say a client (or a user) and a server, to share common session keys in an authentic way. In this paper, we review AKE protocols from a little bit different point of view, i.e. the relationship between information a client needs to possess (for authentication) and immunity to the respective leakage of stored secrets from a client side and a server side. Since the information leakage would be more conceivable than breaking down the underlying cryptosystems, it is desirable to enhance the immunity to the leakage. First and foremost, we categorize AKE protocols according to how much resilience against the leakage can be provided. Then, we propose new AKE protocols that have immunity to the leakage of stored secrets from a client and a server (or servers), respectively. And we extend our protocols to be possible for updating secret values registered in server(s) or password remembered by a client.

1 Introduction

1.1 Background

Authenticated Key Establishment (abbreviated by 'AKE') protocols, which include Authenticated Key Agreement (AKA) and Authenticated Key Transport (AKT), are designed for two entities, say a client and a server (in case of two-party protocols), to share common session keys in an authentic way over open networks where the session keys can be used for subsequent cryptographic algorithms (e.g., symmetric key cryptosystems and message authentication codes). Since AKE protocols are crucial cryptographic primitives, they have been widely used in various protocols, such as SSH (Secure SHell) [22], SSL/TLS (Secure Socket Layer/Transport Layer Security) [14, 23], and in many applications such as internet banking, electronic commerce, secure content download, secure remote access and so on. In the literature, there exist many efficient and secure AKE protocols (typical examples can be found in [18, 19]) in either the random oracle model or the standard model which consider an adversary that not only can eavesdrop the communication of the entities but also can actively modify, delete and insert messages sent between the entities of its own choice. For authentication, AKE protocols must require the involving entities to possess some information like stored secrets, passwords or public keys (or their fingerprints).

While the security of cryptosystems and protocols including AKE has been usually discussed with the assumption that stored secrets will never be revealed, we assume here the stored secrets may be leaked out. This can happen maybe due to a bug or a mis-configuration of the system. Formally,

Assumption 1 (Leakage) *Stored secrets may leak out due to accidents such as bugs or mis-configurations of the system. The source of the leakage, i.e. the bugs or the mis-configurations, will be fixed as soon as possible. But some clients continue to use the same personal information, such as passwords.*

Of course, once the bug or the mis-configuration is found, it will be patched or fixed as soon as possible and then the system will be rebuilt (if necessary). This is a common practice in Internet [10, 31]. Even though the patch or the system-rebuild may remove the risk of further leakage coming from the same bug or the mis-configuration, the leaked secrets may still be abusable to intrude the newly rebuilt system or the other systems, e.g. when a client registers the same password to different servers (see Section 1.4). Thus, we think it is very important to take into account the impact of the leakage (and the burden on the client). The idea of considering leakage of stored secrets is not a new one. Already, proactive schemes [17, 35], forward-secure schemes [1, 2, 5, 11], key-insulated systems [13, 24] and password-authenticated key exchange (PAKE) [3, 4, 6, 8, 15, 16, 19, 26–30, 32, 33, 36] assumed the similar situations.

Problem Setting. Let us think of secrets stored in devices in the model of proactive schemes, forward-secure schemes and key-insulated systems where the secrets should be updated or refreshed regularly in a predetermined time period or at a time when a client (or a server) notices the leakage of stored secrets. Specifically, proactive schemes [17, 35] improve threshold schemes by allowing multiple leakage of secrets, limiting only the number of simultaneous leakages. While forward-secure schemes [1, 2, 5, 11] evolve secrets at the end of each time period, key-insulated systems [13, 24] update secrets with update information coming from TRM (Tamper-Resistant Modules). All of them can minimize the impact of the leakage, but not completely prevent the damage. In addition, it takes some time from when stored secrets leaked out until the client (or the server) can realize the fact and then the secrets are updated by new ones. Within the term for realizing the fact or the time period for updating, an adversary who obtained the secrets can break its security in a limited time period. Bringing this problem into AKE protocols, which use *only stored secrets*, may end up with the same result as above. That's the reason why authentication totally depends on stored secrets so that leakage of the secrets is directly connected to impersonating the victimized entity. For the countermeasure, there exist AKE protocols using a password, without TRM. However, most of PAKE protocols requiring *only a password* on client's side can provide a solution against leakage of stored secrets from a client, not a server. Thus, it is desirable to provide immunity to the leakage of stored secrets from a client and a server, respectively. More detailed discussion will be provided through this section. From now on, we focus on AKE protocols using password where the password naturally takes a major role for authentication.

1.2 Classification of AKE Protocols using Password

In the literature, various AKE protocols have been proposed so far which could be divided by what is used to authenticate entities. Here, we classify them according to the types of information a client needs to possess.

At first, let us start by categorizing the *types of information* to be possessed by a client as follows. (i) Human-Memorable Secret (HMS): A secret, which is remembered and typed in by a client, such as a password. (ii) Stored Secret (SS): Secrets stored in a client's machine, in a memory card or somewhere that is not protected with perfect tamper-resistant modules. It may be merely secret values, a signing key of a digital signature scheme, a decryption key of a public key cryptosystem and/or a common key of a symmetric key cryptosystem. (iii) Public Information (PI): Public information, such as a verification key of a digital signature scheme, an encryption key of a public key cryptosystem or their fingerprints. While anyone can get the public information, its validity must be verified at their own responsibility.

Additionally, we assume the followings on the HMS and the SS.

Assumption 2 (Short but Secure to On-line Attacks) *The size of the human-memorable secret is short enough to memorize, but large enough to avoid on-line exhaustive search. This means the secret may be vulnerable to off-line exhaustive search.*

The on-line attack is a series of exhaustive search for a secret performed on-line where adversaries are willing to sieve out secret candidates one by one running an authentication protocol with the target entity (usually, server). In contrast, the off-line attack is performed off-line massively in parallel with recorded transcripts of a protocol. While on-line attacks are applicable to all of the protocols using password equally, they can be prevented by letting a server take appropriate intervals between invalid trials. But, we cannot avoid off-line attacks by such policies, mainly because the attacks are performed off-line and independently of the server. As a result, off-line attacks are critical to most of the protocols using human-memorable passwords.

Assumption 3 (No Perfect TRM) *TRM (Tamper-Resistant Modules) used to store the secrets are not perfectly free from bugs and mis-configurations.*

With the above types of information, we differentiate previous AKE protocols.[1] At first, we list up typical AKE protocols using HMS and explain how they work briefly. (We ignore protocols, which are vulnerable to off-line attacks as they are, such as CHAP [20], IPsec with pre-shared secret [21] and so on.)

[1] A more detailed description of previous AKE protocols will be given in [38].

SSL/TLS and SSH. We show two AKE protocols of SSL/TLS and SSH. (For formal description of the following protocols, refer to SSL/TLS [14, 23] and SSH [22].)

1. Password-based User Authentication over a Secure Channel: In this scheme, a client establishes a secure connection to a server, and then sends the client's password for authentication through the secure connection. The server verifies the given password in the same way as the usual password verification procedure. Note that the server needs to store (a hashed value of) the password.
2. Public-Key based User Authentication with a Password-Protected Secret-Key: A server verifies a client's secret key using a challenge-response protocol. In addition to that, the client stores the secret key in encrypted form with his password.

Password-Authenticated Key Exchange (PAKE) Protocols. PAKE protocols are designed for entities to share a fresh session key (to be secure against off-line attacks) by using *only a pre-shared human-memorable password*, which may be exhaustible with off-line attacks but not with on-line attacks.

A brief sketch of PAKE protocols, which only rely on a password, is given as follows. Both a client and a server share the same password in advance. For authentication and key exchange, they run a PAKE protocol using the shared password (or a hashed value of it). If their inputs coincide with each other, they can obtain the same value that is used to generate a session key for secure channels. Otherwise, they get distinct values which are hard to guess each other. Thus, no adversary can intrude in the middle of them or impersonate one entity to the other.

Up to now, a variety of studies on PAKE protocols [3, 4, 6, 8, 15, 16, 19, 26–30, 32] have appeared in the literature. In PAKE protocols, a client keeps in mind his password whereas the counterpart server should have its verification data that is used to verify the client's knowledge of the password. That means leakage of stored secrets (that is, verification data) from the server makes possible off-line dictionary attacks for an adversary.

Threshold-PAKE (T-PAKE) Protocols. In order to prevent the leakage of stored secrets from a server, [33, 36] proposed T-PAKE protocols where a client's password or verification data is not stored in a single server but rather shared among a set of servers using a secret sharing scheme. Since only a certain threshold of servers can reconstruct the client's password or verification data in the authentication phase, the leakage of stored secrets from any number of servers smaller than the threshold doesn't help an adversary to perform off-line attacks.

Table 1. Attack and security levels

Attack levels / Security levels	On-line attacks	Off-line attacks
Strongly secure*1 (◯)	Secure	Secure
Weakly secure*2 (△)	Secure	Insecure

*1: An AKE protocol using password is said to be strongly secure (denoted by ◯), if the protocol can be tolerant against both on-line and off-line attacks.
*2: An AKE protocol using password is said to be weakly secure (denoted by △), if the protocol can be tolerant against on-line but not off-line attacks.

1.3 Evaluation by Immunity to the Leakage

As mentioned in Section 1.1, we consider the situation that stored secrets from a client and a server may leak out due to a bug or a mis-configuration of the system. Before evaluating previous AKE protocols using password according to immunity to the leakage of stored secrets, we divide security levels into two cases with respect to whether an AKE protocol can maintain its security (with the client's password unknown to an adversary) against on-line and off-line attacks and then summarize them in Table 1.

With these security levels, we summarize comparative results in Table 2 about whether a client (or a server) can remain resistant against on-line and off-line attacks *even after* stored secrets from the client (or the server) are leaked out to an adversary, respectively. For simplicity, we evaluate immunity to the leakage of each class of AKE protocols presented in Section 1.2.

As shown in the table, PAKE protocols just require that a client keep in mind his password while the counterpart server should have its verification data associated with the password. Consequently, if stored secrets in the server are leaked out, an adversary who gets them can retrieve the original password through off-line attacks, simply by verifying password candidates one by one using the verification data. As a countermeasure to the leakage from server, [33, 36] provided a solution in which n ($n > 1$) servers share verification data (or verification function) using a secret sharing scheme and the threshold of servers participate in the protocol to authenticate a client. That is, the leakage of stored secrets from any number of servers smaller than the threshold does not make off-line attacks possible. However, the client's password can be retrieved if stored secrets from the threshold or more than the threshold of servers are leaked out. In a word, it is impossible for PAKE (T-PAKE) protocols using only HMS (and PI) to achieve strong security against the leakage from server(s).

Fact 1 (Impossibility of Strong Security in PAKE and T-PAKE) *PAKE (T-PAKE) protocols, requiring only HMS (and PI) as clients' possessions, cannot achieve the strong security against the leakage from server(s). For any such a protocol, an adversary can perfectly simulate the protocol using the leaked secrets from server(s) so that he/she can try the password candidates off-line in parallel.*

Table 2. Comparison of AKE protocols using password

Protocols	Client's possessions			Immunity to leakage	
	HMS	SS	PI	from Client	from Server
PAKE*1	√			○	△
Our Proposals	√	√		○	○
SSL/TLS*2, SSH*2, T-PAKE	√		√	○	△*4
SSL/TLS*3, SSH*3	√	√	√	△	○

*1: Most PAKE protocols, being secure against server compromise[a], which hold clients' verification data
*2: Key-establishment part of SSL/TLS and SSH in the password-based user authentication mode[b]
*3: Key-establishment part of SSL/TLS and SSH in the public-key based user authentication mode with a password-protected secret-key[c]
*4: T-PAKE protocols [33, 36] have the immunity up to its threshold of servers.

[a] Throughout this paper, we use the terminology of 'leakage' rather than 'compromise'.
[b] More specifically, password authentication after the server authentication in SSL/TLS or the password authentication in SSH.
[c] More specifically, mutual authentication in SSL/TLS, RSA authentication in SSH protocol version 1 or public key authentication in SSH protocol version 2.

SSL/TLS and SSH in the password-based user authentication mode make a server keep a hashed value of a client's password. As a matter of course, leakage of stored secrets from the server results in revealing the password through off-line attacks. In the SSL/TLS and SSH of the public-key based user authentication mode with a password-protected secret-key, leakage from a client can prevent an adversary, who is willing to get the client's password, from obtaining the password through only on-line attacks, but not off-line attacks.

As a consequence, Table 2 indicates that the existing AKE protocols using password are vulnerable to the leakage from either client or server. That means any of the AKE protocols (except our protocols) doesn't provide immunity to the leakage of stored secrets from *client and server*, respectively. Remind that AKE protocols, which use only stored secrets, can minimize the impact of the leakage by updating or refreshing the secrets, but not completely prevent the damage.

1.4 A Realistic, but Critical, Problem of AKE Protocols using Password

Are all the existing AKE protocols using password really secure *in the real world*? Instead of answering to the question, we take for an example a very compelling but critical situation in the real world.

Let us think of an ordinary client who would connect with several disparate servers, each requiring a password, over networks for internet banking, internet shopping, internet auction, ftp servers, electronic voting and so on. As of now, all of the AKE protocols *implicitly* have the assumption that the client registers *information-theoretically independent passwords* corresponding to different servers. Remember that password can be defined as human beings have something *memorable* usually in size of 6-8 characters (including numbers). Ironically, how many passwords can we remember? 10 or 20? Of course, it depends on the individual. Here, we have another assumption as follows:

Assumption 4 (One Memorized Secret) *A client remembers only one human-memorable secret, i.e. one password, even if he/she communicates with several different servers. That means the client use the same password to a distinct kind of servers, not sharing any secret information one another.*

Under the Assumption 4 in the multiple server scenario, we have to take into consideration the impact on other servers after the leakage of stored secrets from one server in the real world.

Definition 1 (Impact after the Leakage from One Server) *An AKE protocol using password, where there is no impact on other servers after the leakage of stored secrets from one server, is said to be desirable in the sense that an adversary, after obtaining verification data associated with a client's password from one server, cannot retrieve the password that makes possible to impersonate the client to other servers of the Assumption 4. That is, the password is completely protected against off-line attacks even if the adversary can get some verification data from servers.*

Of course, a client may change his password *instantly* to all servers at a time when he comes to know that stored secrets from one of the servers are revealed out. However, it triggers the burden on the client.

Motivation. The motivation of this paper is on how to design an AKE protocol that has immunity to the leakage of stored secrets from a client and servers, respectively, under the Assumption 4. That means the client need not change his password, *even if* stored secrets are leaked out from either the client or servers. However, we can easily deduce the following fact that there exists no AKE protocol, which is immune to the leakage from a client and servers *simultaneously.*

Fact 2 (Impossibility of Perfect Security) *Any AKE protocol cannot achieve the strong security against the leakage from both a client and servers simultaneously. If an adversary obtains stored secrets from both a client and servers at the same time, he/she can perfectly simulate the protocol using the leaked secrets. Thus the adversary can try the password candidates off-line in parallel.*

This fact motivates us to achieve the next highest goal, i.e. the strong security against the leakage from a client and servers, respectively. Notice that our protocol is *not* a kind of PAKE protocols, but a new one that requires *one password* and *secret values* on client's side.

1.5 Our Contributions

In this paper, we propose new AKE protocols that are immune to the leakage of stored secrets from *both a client and servers* respectively, as long as the leakages are not simultaneous, where the client keeps *one password* in his mind and stores *secret values* in devices. Specifically speaking, a client registers a partial secret value (which is not a share itself) of one password to a different kind of servers by means of a secret sharing scheme. The protocol of Section 2.2 is a generalized version in which the number of servers is fixed in advance whereas the second in Section 2.3 can be readily applied to the real world, simply because the latter considers synchronization between a client and one of the servers for registering a secret value. That means the client can compute a secret value with the same password at any time when needed to register to a necessary server without restricting the number of servers. In our protocols, an adversary obtaining stored secrets after the leakage from all of the servers (the client has been communicating with) cannot find out the password. Also, an adversary getting stored secrets after the leakage from the client cannot sieve out the password. More interestingly, the password remains *information-theoretically secure* even if the leakage of stored secrets from the client and servers happens, respectively.

In addition to that, our protocols have the following advantages: (1) the proposed protocols can be constructed with small modifications of the widely used Diffie-Hellman key exchange protocol [12]. (2) the proposed protocols have a formal validation of security in the standard model (instead of the random oracle model [9]) under the assumption that DDH (Decisional Diffie-Hellman) problem is hard and MACs are selectively unforgeable against partially chosen message attacks (which is a weaker notion than being existentially unforgeable against chosen message attacks).

Then, we extend our protocol of Section 2.2 to two protocols where one enables a client to update each of the secret values registered in different servers without changing his password (which might be remembered with considerable effort)[2] and the other enables a client to change his password with a new password while updating each of the secret values in different servers.

For better understanding, it may be helpful to state about what is different between our approach and T-PAKE protocols [33, 36]. The main difference is that [33, 36] cannot preserve its security (a client's password) against off-line attacks if stored secrets from the threshold or more than the threshold of servers would be revealed, whereas ours can maintain it *even after* stored secrets from all of the servers (a client is communicating with) would be revealed out. That's the reason why [33, 36] proposed T-PAKE protocols in order to protect a client's password from the leakage of stored secrets in a server where verification data (or function) associated with the password is distributed by a set of servers.

[2] This additional function is useful when we consider a situation where one administrator of servers resigns with secret values (associated with clients' passwords) in the server. However, recall that the frequent change of passwords rather increases the risk of password to be lost and cracked, simply because people tend to write it down on somewhere.

Contrarily, our protocols distribute secret values computed with the password by the client himself. Accordingly, if stored secrets from the threshold or more than the threshold of servers are leaked out in [33, 36], clients' passwords can be retrieved by an adversary which affects on different servers that don't share any secret information one another under the Assumption 4. Besides, both the communication and the computation complexity of [33, 36] are by far larger than ours. And, applications of [33, 36] are restricted since a certain threshold of servers must take part in the protocol for authentication.

1.6 Organization

This paper is organized as follows: In Section 2, we propose new AKE protocols that have immunity to the leakage of stored secrets from not only a client but also servers, respectively. Section 3 shows how our protocols can remain resistant against off-line attacks even after the leakage of stored secrets from the client and servers, respectively. Then, we extend the proposed protocols in Section 4.

2 Our Proposals: Leakage-Resilient AKE Protocols

2.1 Scenario

Here, we consider the following scenario that there are $n-1$[3] disparate kinds of servers communicating with a client, who wants to use *one password* and *secret values* to produce cryptographically secure (or, high entropy) session keys with different servers at any time.

Our protocols are defined over a finite cyclic group $\mathcal{G} = \langle g \rangle$ where $|\mathcal{G}| = q$ and q is a large prime (or, a positive integer divisible by a large prime). While $\mathcal{G}$ can be a group over an elliptic curve, we assume that $\mathcal{G}$ is a prime order subgroup over a finite field $\mathbb{F}_p$. That is, $\mathcal{G} = \{g^i \bmod p : 0 \leq i < q\}$ where p is a large prime number, q is a large prime divisor of $p-1$ and g is an integer such that $1 < g < p-1$, $g^q \equiv 1$ and $g^i \neq 1$ for $0 < i < q$. A generator of $\mathcal{G}$ is any element in $\mathcal{G}$ except 1. In the aftermath, all the subsequent arithmetic operations are performed in modulo p, unless otherwise stated. Both g and h are two generators of $\mathcal{G}$ so that its DLP (Discrete Logarithm Problem), i.e. calculating

$$a = \log_g h, \tag{1}$$

should be hard for each entity. Both g and h may be given as system parameters or chosen with an initialization phase between entities.

The protocols consist of the following four phases: an initialization phase, a secrecy amplification phase, a verification phase and a session-key generation

[3] In case of two-party protocols, n becomes 2. As our protocols also satisfy the two-party case, we set up n $(2 \leq n < q)$, at the same time, in order to consider the multiple server scenario of Assumption 4.

phase. In the initialization phase, a client registers each of the secret values computed by himself to different servers. Then, he stores the corresponding secret values in devices such as smart cards or computers and keeps only one password in mind. In the secrecy amplification phase, secrecy of a weak secret, i.e. a human-memorable password that may be vulnerable against off-line attacks, is amplified to a strong secret (we call it a keying material) that is secure even against off-line attacks. In the verification phase, both client and server can confirm whether or not they share the same keying material using a challenge-response protocol with the keying material as its key. In the session-key generation phase, a session key is generated using the keying material.

2.2 A Leakage-Resilient AKE Protocol

We describe a construction for a leakage-resilient AKE protocol which is illustrated in Fig. 1. The key idea behind our protocol is that a client can generate n shares of his password, where each of the $n-1$ shares is used for registering a secret value to the corresponding server and the remaining one share (not itself) is stored in his devices in the initialization phase, only by inputting the password (as a secret value) into (n, n)-threshold secret sharing scheme of [7, 37].

[Initialization] A client C, included in n entities, is willing to register each of the secret values generated by one password pw to the respective $n-1$ different server S_i $(1 \leq i \leq n-1)$. For simplicity, we assign the servers consecutive integer $1 \leq i \leq n-1$ where i can be regarded as each server's ID and n as the client's ID. First and foremost, the client picks a random polynomial $p(x)$ of degree $n-1$ with coefficients also randomly chosen in $(\mathbb{Z}/q\mathbb{Z})^*$:

$$p(x) = \sum_{j=0}^{n-1} \alpha_j \cdot x^j \bmod q \tag{2}$$

and sets $\alpha_0 = pw$[4] where pw is the client's password. After computing the respective shares $p(i)$ $(1 \leq i \leq n-1)$ with the above polynomial, he registers securely each of the secret values $h^{p(i)\cdot\lambda_i}$ to the corresponding server S_i $(1 \leq i \leq n-1)$ as follows:

$$S_i \leftarrow h^{p(i)\cdot\lambda_i}, \text{ where } \lambda_i = \prod_{k=1,k\neq i}^{n} \frac{k}{k-i} \bmod q \tag{3}$$

where $p(i)$ is a share of (n, n)-threshold secret sharing scheme and λ_i is a Lagrange coefficient. Note that share $p(n)$, which is for the client, is never registered to any server. Then, the client *just stores the corresponding secret values*

[4] Instead of pw, a hashed value of the password can be used. In either case where both have the same entropy, it doesn't affect on the security.

Client C		Server S_i $(1 \le i \le n-1)$
$r_1 \leftarrow_R (\mathbb{Z}/q\mathbb{Z})^*$		$r_2 \leftarrow_R (\mathbb{Z}/q\mathbb{Z})^*$
$y_1 \leftarrow g^{r_1} \cdot \underline{h_i} \cdot h^{-pw}$	$\xrightarrow{y_1}$	$y_2 \leftarrow g^{r_2} \cdot \underline{h^{p(i)\cdot\lambda_i}}$
	$\xleftarrow{y_2}$	
$km_c \leftarrow (y_2 \cdot h_i \cdot h^{-pw})^{r_1}$		$km_s \leftarrow (y_1 \cdot h^{p(i)\cdot\lambda_i})^{r_2}$
$v_1 \leftarrow \mathsf{MAC}_{km_c}(Tag_c\|\|y_1\|\|y_2)$	$\xrightarrow{v_1}$	$v_2 \leftarrow \mathsf{MAC}_{km_s}(Tag_s\|\|y_1\|\|y_2)$
	$\xleftarrow{v_2}$	
If $v_2 = \mathsf{MAC}_{km_c}(Tag_s\|\|y_1\|\|y_2)$,		If $v_1 = \mathsf{MAC}_{km_s}(Tag_c\|\|y_1\|\|y_2)$,
$sk_c \leftarrow \mathsf{MAC}_{km_c}(Tag_{sk}\|\|y_1\|\|y_2)$.		$sk_s \leftarrow \mathsf{MAC}_{km_s}(Tag_{sk}\|\|y_1\|\|y_2)$.

Fig. 1. A leakage-resilient AKE protocol. The underlined values represent stored secrets of client and server, respectively

h_i $(1 \le i \le n-1)$ to the servers S_i in devices, such as smart cards or computers, which may happen to leak the secrets h_i and *keeps his password pw in mind*.

$$h_i \leftarrow h^{\Sigma_{l=1,l\neq i}^{n} p(l)\cdot\lambda_l}. \tag{4}$$

Of course, all the other (intermediate) values should be deleted from the devices.

[**Secrecy Amplification**] When the client C wants to share a session key with one of the servers S_i $(1 \le i \le n-1)$, he chooses a random number $r_1 \leftarrow_R (\mathbb{Z}/q\mathbb{Z})^*$. Then, the client sends y_1 to server S_i, after calculating $y_1 \leftarrow g^{r_1} \cdot h_i \cdot h^{-pw}$ using the corresponding secret value h_i to the server and his password pw that is partially shared with the server. The server S_i also calculates $y_2 \leftarrow g^{r_2} \cdot h^{p(i)\cdot\lambda_i}$ with a random number $r_2 \leftarrow_R (\mathbb{Z}/q\mathbb{Z})^*$ and its secret value $h^{p(i)\cdot\lambda_i}$ (partial secret information about the password) registered by the client in the initialization phase, and then transmits it to the client. On both sides, the client's keying material becomes $km_c \leftarrow (y_2 \cdot h_i \cdot h^{-pw})^{r_1}$ and the server's one becomes $km_s \leftarrow (y_1 \cdot h^{p(i)\cdot\lambda_i})^{r_2}$.

Only if the client uses the right password pw and the corresponding secret value h_i to server S_i *and* the server S_i uses the right secret value $h^{p(i)\cdot\lambda_i}$, both of them can share the same keying material that is obtained by Lagrange interpolation:

$$km_c = \left(y_2 \cdot h_i \cdot h^{-pw}\right)^{r_1} = \left(g^{r_2} \cdot h^{p(i)\cdot\lambda_i} \cdot h^{\Sigma_{l=1,l\neq i}^{n} p(l)\cdot\lambda_l} \cdot h^{-pw}\right)^{r_1} = g^{r_2\cdot r_1}, \tag{5}$$

$$km_s = \left(y_1 \cdot h^{p(i)\cdot\lambda_i}\right)^{r_2} = \left(g^{r_1} \cdot h^{\Sigma_{l=1,l\neq i}^{n} p(l)\cdot\lambda_l} \cdot h^{-pw} \cdot h^{p(i)\cdot\lambda_i}\right)^{r_2} = g^{r_1\cdot r_2}. \tag{6}$$

Otherwise guessing the other's keying material is hard due to the DLP (see [38]). Also, adversaries cannot determine the correct password of the client through off-line attacks since they don't know the client's random number r_1 chosen at the time and the secret value h_i corresponding to sever S_i, both of which are required to narrow down the password pw.

This phase ends up with only one pass in parallel since both y_1 and y_2 can be calculated and sent independently (where $g^{r_1} \cdot h_i$ and y_2 are pre-computable). Additionally, the implementation cost of this phase is very low because it can be simply obtained from a small modification of widely used Diffie-Hellman key exchange protocol [12]. That's why $h^{-p(i)\cdot\lambda_i} = h_i \cdot h^{-pw}$.

[Verification] In this phase, a pair of entities can verify whether they share the same keying material or not with a challenge-response protocol using the keying material calculated in the secrecy amplification phase.

The client and the server calculate $v_1 \leftarrow \mathsf{MAC}_{km_c}(Tag_c||y_1||y_2)$ and $v_2 \leftarrow \mathsf{MAC}_{km_s}(Tag_s||y_1||y_2)$, respectively, using a MAC generation function $\mathsf{MAC}_k(\cdot)$ with the keying materials as its key k. Both Tag_c and Tag_s are pre-determined distinct values, e.g. $Tag_c = (ID_c||ID_s||00)$ and $Tag_s = (ID_c||ID_s||01)$ where ID_c and ID_s are IDs of the client and the server respectively. Then, they exchange v_1 and v_2 each other, before verifying $v_2 = \mathsf{MAC}_{km_c}(Tag_s||y_1||y_2)$ and $v_1 = \mathsf{MAC}_{km_s}(Tag_c||y_1||y_2)$ on both sides. If at least one of them does not hold, the corresponding entities wipe off all the temporal data including the keying materials, and then close the session. Otherwise they proceed to the session-key generation phase.

Adversaries can try off-line attacks for the keying material using $\{(Tag_c||y_1||y_2)$ and $v_1\}$ or $\{(Tag_s||y_1||y_2)$ and $v_2\}$. The success probability achieved within a polynomial time t can be negligible if a strong secret can be shared in the secrecy amplification phase and an appropriate MAC generation function, whose keys are unguessable, is used.

[Session-Key Generation] If the above verification phase succeeds in, the entities generate their session keys using the verified keying materials as follows:

$$sk_c \leftarrow \mathsf{MAC}_{km_c}(Tag_{sk}||y_1||y_2) \tag{7}$$

$$sk_s \leftarrow \mathsf{MAC}_{km_s}(Tag_{sk}||y_1||y_2) \tag{8}$$

where Tag_{sk} is a pre-determined distinct value from both Tag_c and Tag_s, e.g. $Tag_{sk} = (ID_c||ID_s||11)$. The generated session keys are used for their subsequent cryptographic algorithms.

The requirement for the MAC generation function in this phase and the previous phase is $\epsilon_{mac}(k_2, t, i)$ can be negligibly small for a practical security parameter k_2 and i (this is a polynomial of k_2). That's the reason why if adversaries cannot forge a MAC corresponding to $(Tag_{sk}||y_1||y_2)$ and km_c or km_s with significant probability, they cannot obtain any information of the session key. This requirement can be satisfied by using a universal one-way hash function [34] or by using a practical MAC generation function, such as HMAC-SHA-1 [25] (and even KeyedMD5), since any effective algorithms have not been known so far to make $\epsilon_{mac'}(k_2, t, i)$ non-negligible where $\epsilon_{mac'}(k_2, t, i)$ is larger than or equal to $\epsilon_{mac}(k_2, t, i)$.

2.3 A More Practical Leakage-Resilient AKE Protocol

The proposed protocol in Section 2.2 deployed a (n, n)-threshold secret sharing scheme, in order to generate $n-1$ secret values with each registered to $n-1$ servers respectively. That is, a client should determine the number of different servers in advance and register each of the secret values to the servers all at once. When it comes to the real world, it is desirable that a client be able to choose among a different kind of servers at his own will. Although a client in the protocol of Section 2.2 can choose $n-1$ different servers, we show how to apply the proposed protocol to the case where the client can compute a secret value (from one password) at any time when needed. This approach will lead the protocol of Section 2.2 to be more simpler in the initialization phase, just by replacing (n, n)-threshold secret sharing scheme with $(2, 2)$-threshold one.

[Initialization] A client C is willing to register a secret value generated by one password pw to one of different servers S_i where i can be regarded as each server's ID. Every time when needed to register a secret value to a server, the client picks a distinct random polynomial $p_i(x)$ (for the respective server S_i) of degree 1 with coefficient α_{i1} randomly chosen in $(\mathbb{Z}/q\mathbb{Z})^*$:

$$p_i(x) = \sum_{j=0}^{1} \alpha_{ij} \cdot x^j = \alpha_{i0} + \alpha_{i1} \cdot x \bmod q \tag{9}$$

and sets $\alpha_{i0} = pw$ where pw is the client's password. After computing a share $p_i(1)$ with the above polynomial, he registers securely a secret value $h^{p_i(1) \cdot \lambda_1}$ to one of different servers S_i as follows:

$$S_i \leftarrow h^{p_i(1) \cdot \lambda_1}, \text{ where } \lambda_1 = 2 \bmod q \tag{10}$$

where $p_i(1)$ is a share of $(2, 2)$-threshold secret sharing scheme for the server S_i and λ_1 is a Lagrange coefficient. Note that share $p_i(2)$ is for the client. Then, the client just stores *the corresponding secret value* h_i in devices and keeps *his password* pw in mind.

$$h_i \leftarrow h^{p_i(2) \cdot \lambda_2}, \text{ where } \lambda_2 = -1 \bmod q \ . \tag{11}$$

The rest phases of this protocol are as same as those of Section 2.2.

3 Security

This section shows the security of password in Section 2.2 against off-line attacks after the leakage of stored secrets from a client and servers, respectively. And the security of password in Section 2.3, which is a case of $n = 2$ in (n, n)-threshold secret sharing scheme, inherits from Section 3 straightforwardly. Moreover, the security against on-line and off-line attacks of the below adversary can be proven in the standard model as Theorem 2. (For formal security proof, refer to [38].)

In the security model of our protocol, we consider a far more powerful adversary who has ability to not only eavesdrop, modify and delete the messages exchanged by entities, but also to insert messages of its own choice. This adversarial power is modeled by giving the adversary oracle access to the instances of our protocol. In addition, the adversary is given access to a Leak oracle that simulates Assumption 1. That is, Leak oracle accepts an entity ID and then reveals the corresponding stored secrets. However, this oracle does not reveal stored secrets of its partner at the same time, because of Fact 2.

3.1 Security of Password against the Leakage

The primary goal of an adversary after obtaining stored secrets from a client and servers, respectively, is to perform off-line exhaustive search for the client's password that makes possible to impersonate the client to other servers under the Assumption 4.

Theorem 1 *The password in our protocol of Section 2.2 remains information-theoretically secure against off-line attacks after the leakage of stored secrets from the client C and $n-1$ servers S_i $(1 \leq i \leq n-1)$, respectively. Even if an adversary obtains stored secrets from the Leak oracle, she cannot retrieve the client's original password through off-line exhaustive search that is the best attack for the adversary.*

Proof. When an adversary gets secrets stored in devices from the client C and $n-1$ servers S_i $(1 \leq i \leq n-1)$ respectively, what she wants to know is the client's password pw or a value associated with the password

$$h^{pw} = h^{\Sigma_{m=1}^{n} p(m) \cdot \lambda_m}. \tag{12}$$

Only if the above value h^{pw} is computed, the adversary can narrow down the original password by checking possible password candidates with equation (12) one by one (through off-line exhaustive search). In order to simplify the proof, let us fix $n = 5$.

First, we think of the security of password against an adversary who obtains stored secrets h_i $(1 \leq i \leq 4)$ of the client C and is trying to deduce $h^{\Sigma_{m=1}^{5} p(m) \cdot \lambda_m}$ for the client's password pw. Below is the exponent part of h_i

$$\begin{bmatrix} \log_h h_1 \\ \log_h h_2 \\ \log_h h_3 \\ \log_h h_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p(1) \cdot \lambda_1 \\ p(2) \cdot \lambda_2 \\ p(3) \cdot \lambda_3 \\ p(4) \cdot \lambda_4 \\ p(5) \cdot \lambda_5 \end{bmatrix}. \tag{13}$$

Equation (13) means that the secrets h_i $(1 \le i \le 4)$ don't reveal any information about the password pw, simply because each row contains 4 shares (the number of shares needed for the client's password is more than that of h_i by one share) and each exponent part of h_i is linearly independent one another. That is the adversary cannot compute $h^{\Sigma_{m=1}^{5} p(m)\cdot\lambda_m}$ with stored secrets h_i.

Second, we think of the security of password against an adversary who obtains stored secrets $h^{p(i)\cdot\lambda_i}$ of all the servers S_i $(1 \le i \le 4)$ and is trying to deduce $h^{\Sigma_{m=1}^{5} p(m)\cdot\lambda_m}$ for the client's password pw. Below is the exponent part of $h^{p(i)\cdot\lambda_i}$

$$\begin{array}{l} \log_h S_1 \rightarrow \\ \log_h S_2 \rightarrow \\ \log_h S_3 \rightarrow \\ \log_h S_4 \rightarrow \end{array} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} p(1)\cdot\lambda_1 \\ p(2)\cdot\lambda_2 \\ p(3)\cdot\lambda_3 \\ p(4)\cdot\lambda_4 \\ p(5)\cdot\lambda_5 \end{bmatrix} . \tag{14}$$

Intuitively, the number of shares included in $h^{\Sigma_{m=1}^{5} p(m)\cdot\lambda_m}$ is one more than that of $h^{p(i)\cdot\lambda_i}$ $(1 \le i \le 4)$, since each row only contains one share of $(5, 5)$-threshold secret sharing scheme. Although the adversary gathers all of the secret values from servers S_i $(1 \le i \le 4)$, the number of shares is 4. That means the password is information-theoretically secure as a secret value of $(5, 5)$-threshold secret sharing scheme.

□

Theorem 2 (Indistinguishability of sk) *Suppose the following adversary $\mathcal{A}$, which accepts a challenge transcript (that may be obtained by eavesdropping a protocol, impersonating a partner or intruding in the middle of the target entities), and then asks q_{ex}, q_{se}, q_{re} and q_{le} queries to the Execute, Send, Reveal, Leak oracles respectively, and finally is given sk_x by $Test_{sk}$ oracle where sk_x is either the target session key or not with the probability of $1/2$. Then $\mathbf{Adv}_{\mathcal{A}}^{\mathtt{ind}_{sk}}$, the advantage of adversary $\mathcal{A}$ to distinguish whether sk_x is the target session key or not in a polynomial time t, is upper bounded by*

$$\begin{aligned} \mathbf{Adv}_{\mathcal{A}}^{\mathtt{ind}_{sk}} \le\ & \varepsilon_{mac}(k_2, t, q_{se} + 2q_{ex} + q_{re} + 2) + 2(q_{se} + q_{ex} + 1)\cdot\varepsilon_{ddh}(k_1, t) \\ & + \frac{2(q_{se}+1)}{N} + \frac{2(2q_{se} + q_{ex} + 1)}{|\mathcal{G}|} \end{aligned} \tag{15}$$

where both k_1 and k_2 are the security parameters.

4 Extensions

It is reasonable that a client has control of *each of the secret values* registered in a different kind of servers and of *password* kept in his mind, regularly or irregularly. Here, we provide two extended versions of Section 2.2, simply by using a proactive threshold scheme [35] in which there is a basic assumption that an adversary who gets stored secrets from a server cannot take the update information. One is for the secret-values update which enables a client to update each of the secret values stored in different servers without changing his password. And the other is for the password update which enables a client to change his password with a new one while updating each of the secret values in different servers. In the point of view of updating stored secrets, our approach is similar to those of key-insulated systems [13] and intrusion-resilient signatures [24]. However, the main difference is that we don't use TRM (Tamper-Resistant Modules) to produce update information, which can be computed by the client himself in our protocol. We omit two versions of Section 2.3, whose extensions can be readily shown in the same way of Section 4.

[**Secret-Values Update (for Proactive Security)**] When a client C, included in n entities, wants to update each of the secret values which has been registered to the respective $n-1$ different servers S_i $(1 \leq i \leq n-1)$ with new ones (to be generated by the same password pw), he picks another random polynomial $p'(x)$ of degree $n-1$ with coefficients randomly chosen in $(\mathbb{Z}/q\mathbb{Z})^*$:

$$p'(x) = \sum_{j=1}^{n-1} \beta_j \cdot x^j \bmod q \tag{16}$$

and sets $\beta_0 = 0$. After computing the respective shares $p'(i)$ $(1 \leq i \leq n-1)$ with the above polynomial, the client transmits securely each of the new secret values $h^{p'(i)\cdot\lambda_i}$ to the corresponding server S_i $(1 \leq i \leq n-1)$ as follows:

$$S_i \leftarrow h^{p'(i)\cdot\lambda_i}, \text{ where } \lambda_i = \prod_{k=1,k\neq i}^{n} \frac{k}{k-i} \bmod q \tag{17}$$

where $p'(i)$ is a new share of (n,n)-threshold secret sharing scheme and λ_i is a Lagrange coefficient. Consequently, each server S_i can produce an updated secret value $h^{(p(i)+p'(i))\cdot\lambda_i} = h^{p(i)\cdot\lambda_i} . h^{p'(i)\cdot\lambda_i}$ with multiplying the previous secret value $h^{p(i)\cdot\lambda_i}$ by a new one $h^{p'(i)\cdot\lambda_i}$. Note that share $p'(n)$, which is for the client, is never registered to any server. Then, the client also updates and stores *the corresponding secret values* ${h_i}' = h^{\Sigma_{l=1,l\neq i}^{n}(p(l)+p'(l))\cdot\lambda_l}$ $(1 \leq i \leq n-1)$ in devices and keeps *the same password* pw in mind.

$${h_i}' \leftarrow h_i \cdot h^{\Sigma_{l=1,l\neq i}^{n} p'(l)\cdot\lambda_l}. \tag{18}$$

Of course, the client doesn't need to update secret values stored in different servers S_i $(1 \leq i \leq n-1)$ *simultaneously*. That means he can update each of the secret values in servers at any time, only if the client chooses a different random polynomial every time.

[**Password Update**] If a client C wants to change his password pw with a new one pw_{new} while updating each of the secret values registered to the respective $n-1$ different servers S_i $(1 \leq i \leq n-1)$, he follows the above secret-values update in the same way except that the client picks another random polynomial $p''(x)$ of degree $n-1$ with coefficients randomly chosen in $(\mathbb{Z}/q\mathbb{Z})^*$:

$$p''(x) = \sum_{j=0}^{n-1} \gamma_j \cdot x^j \bmod q \tag{19}$$

and sets $\gamma_0 = -pw + pw_{new}$ where pw_{new} is the client's new password.

Acknowledgements

The authors would like to thank anonymous referees for useful comments.

References

1. M. Abdalla, S. Miner, and C. Namprempre. Forward-Secure Threshold Signature Schemes. In *Proc. of Topics in Cryptology (CT-RSA 2001)*, LNCS 2020, pages 441-456. Springer-Verlag, 2001.
2. R. Anderson. Two Remarks on Public Key Cryptology. *Technical Report*, No. 549, University of Cambridge, December 2002.
3. E. Bresson, O. Chevassut, and D. Pointcheval. Group Diffie-Hellman Key Exchange Secure against Dictionary Attacks. In *Proc. of ASIACRYPT 2002*, LNCS 2501, pages 497-514. Springer-Verlag, 2002.
4. S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-based Protocols Secure against Dictioinary Attacks. In *Proc. of IEEE Symposium on Security and Privacy*, pages 72-84, 1992.
5. M. Bellare and S. Miner. A Forward-Secure Digital Signature Scheme. In *Proc. of CRYPTO '99*, LNCS 1666, pages 431-448. Springer-Verlag, 1999.
6. V. Boyko, P. MacKenzie, and S. Patel. Provably Secure Password-Authenticated Key Exchange using Diffie-Hellman. In *Proc. of EUROCRYPT 2000*, LNCS 1807, pages 156-171. Springer-Verlag, 2000.
7. G. R. Blakley. Safeguarding Cryptographic Keys. In *Proc. of National Computer Conference 1979 (AFIPS)*, Vol. 48, pages 313-317, 1979.
8. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure against Dictionary Attacks. In *Proc. of EUROCRYPT 2000*, LNCS 1807, pages 139-155. Springer-Verlag, 2000.
9. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proc. of ACM CCS '93*, pages 62-73, 1993.
10. CERT Coordination Center, http://www.cert.org/.
11. R. Canetti, S. Halevi, and J. Katz. A Forward-Secure Public-Key Encryption Scheme. In *Proc. of EUROCRYPT 2003*, LNCS 2656, pages 255-271, 2003.
12. W. Diffie and M. Hellman. New Directions in Cryptography. In *IEEE Transactions on Information Theory*, Vol. IT-22(6), pages 644-654, 1976.
13. Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-Insulated Public Key Cryptosystems. In *Proc. of EUROCRYPT 2002*, LNCS 2332, pages 65-82. Springer-Verlag, 2002.
14. A. Frier, P. Karlton, and P. Kocher. The SSL 3.0 Protocol. Netscape Communications Corp., 1996, http://wp.netscape.com/eng/ssl3/.
15. O. Goldreich and Y. Lindell. Session-Key Generation using Human Passwords Only. In *Proc. of CRYPTO 2001*, LNCS 2139, pages 408-432, 2001.

16. R. Gennaro and Y. Lindell. A Framework for Password-based Authenticated Key Exchange. In *Proc. of EUROCRYPT 2003*, LNCS 2656, pages 524-543. Springer-Verlag, 2003, A full paper is available at http://eprint.iacr.org/2003/032.
17. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive Public Key and Signature Systems. In *Proc. of ACM CCS '96*, pages 100-110, April 1997.
18. IEEE Std 1363-2000. IEEE Standard Specifications for Public Key Cryptography. Main Document, pages 53-57, IEEE, August 29, 2000.
19. IEEE P1363.2. Standard Specifications for Password-based Public Key Cryptographic Techniques. Draft version 11, August 12, 2003.
20. IETF (Internet Engineering Task Force). Challenge Handshake Authentication Protocol. http://www.ietf.org/rfc/rfc1994.txt.
21. IETF (Internet Engineering Task Force). IP Security Protocol (ipsec) Charter. http://www.ietf.org/html.charters/ipsec-charter.html.
22. IETF (Internet Engineering Task Force). Secure Shell (secsh) Charter. http://www.ietf.org/html.charters/secsh-charter.html.
23. IETF (Internet Engineering Task Force). Transport Layer Security (tls) Charter. http://www.ietf.org/html.charters/tls-charter.html.
24. G. Itkis and L. Reyzin. SiBIR: Signer-Base Intrusion-Resilient Signatures. In *Proc. of CRYPTO 2002*, LNCS 2442, pages 499-514. Springer-Verlag, 2002.
25. H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. IETF RFC 2104, 1997, http://www.ietf.org/rfc/rfc2104.txt.
26. K. Kobara and H. Imai. Pretty-Simple Password-Authenticated Key-Exchange under Standard Assumptions. IACR ePrint Archieve, 2003, http://eprint.iacr.org/2003/038.
27. J. Katz, R. Ostrovsky, and M. Yung. Efficient Password-Authenticated Key Exchange using Human-Memorable Passwords. In *Proc. of EUROCRYPT 2001*, LNCS 2045, pages 475-494. Springer-Verlag, 2001.
28. T. Kwon. Authentication and Key Agreement via Memorable Password. In *Proc. of NDSS 2001 Symposium*, 2001.
29. P. MacKenzie. More Efficient Password-Authenticated Key Exchange. In *Proc. of Topics in Cryptology (CT-RSA 2001)*, LNCS 2020, pages 361-377, 2001.
30. P. MacKenzie. On the Security of the SPEKE Password-Authenticated Key Exchange Protocol. IACR ePrint Archieve, 2001, http://eprint.iacr.org/2001/057/.
31. Microsoft Corporation, http://www.microsoft.com/.
32. P. MacKenzie, S. Patel, and R. Swaminathan. Password-Authenticated Key Exchange Based on RSA. In *Proc. of ASIACRYPT 2000*, LNCS 1976, pages 599-613. Springer-Verlag, 2000.
33. P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold Password-Authenticated Key Exchange. In *Proc. of CRYPTO 2002*, LNCS 2442, pages 385-400. Springer-Verlag, 2002.
34. M. Naor and M. Yung. Universal One-Way Hash Functions and Their Cryptographic Applications. In *Proc. of STOC '98*, pages 33-43, 1998.
35. R. Ostrovsky and M. Yung. How to Withstand Mobile Virus Attacks. In *Proc. of 10th Annual ACM Symposium on Principles of Distributed Computing*, 1991.
36. M. D. Raimondo and R. Gennaro. Provably Secure Threshold Password-Authenticated Key Exchange. In *Proc. of EUROCRYPT 2003*, LNCS 2656, pages 507-523. Springer-Verlag, 2003.
37. A. Shamir. How to Share a Secret. In *Proc. of Communications of the ACM*, Vol. 22(11), pages 612-613, 1979.
38. A full version of this paper will appear in IACR ePrint Archieve.