Hierarchical Key Establishment Protocols Based on Secure Keyed One-Way Hash Functions

Wei-Chi Ku and Sheng-De Wang
Department of Electrical Engineering
National Taiwan University, Taipei 106, Taiwan
[wcku, sdwang]@star.ee.ntu.edu.tw

Abstract

As key establishment protocols (KEP) are usually the initial step for setting up a secure network-based service, they are very important in enabling the required security. Most of the existing KEPs make use of cryptographic algorithms, either secret key or public-key cryptography. However, it is also possible to employ secure keyed one-way hash functions (SKOWHF) in KEP designs. Though several works have been published in this area, they only focus on the small scale networks. In this paper, we first introduce the SKOWHF-based secure trunks and secure channels; then, two rules for KEP construction, KER1 and KER2, are described. Finally, a hierarchical KEP for large scale networks is proposed.

1. Introduction

Before conducting a secure session using secret-key cryptographic technique, a secret key should be agreed upon by the communicants. To assure strict security, the secret key should be renewed for each session, i.e., it should be a session key, so that compromising one key will not divulge the contents of other sessions. The protocol for establishing a session key between the communicants is referred to as a key establishment protocol (KEP). As KEP is usually the initial process for setting up the secure network-based services, they are crucial in enabling the required security.

To date, the majority of KEPs employ either secret-key cryptosystems or public key cryptosystems. However, it is also possible to use one-way hash functions rather than encryption algorithms to construct KEP. Much research has been published in this area [2]-[8]. Gong [2] presents the original idea of using a one-way hash function as the basic building block of a KEP. Later, Bull, Gong and Sollins [3] propose a KEP also based on one-way hash function. Another well-known instance is Kryptoknight [5][6], which is a family of KEPs developed by IBM. Compared with encryption algorithms, a one-way hash function is easier to be implemented for not having to provide the invertible property [2][5][6]. It also makes the object codes and the source codes of them exportable [2][3][5][6][7][8]. In particular, its computation is less complex [5][6]. Maybe it is the reason why Global System for Mobile Communications (GSM) [1] uses one-way functions in its security system. It has been recognized that the characterization of the one-way hash function specified in the above mentioned works is not appropriate for the way they adopt it in a keyed manner [7][8]. Berson, Gong and Lomas redefine the properties of the suitable one-way hash function for security use, labeled as secure keyed one-way hash function (SKOWHF). In addition, mix-up of confidentiality and authentication usually makes the design more difficult to be analyzed or to be implemented. To solve this problem, Boyd and Mathuria [8] propose a systematic method for KEP construction by sharply distinguishing between the confidentiality and authentication channels.

Most of the existing SKOWHF-base KEPs simply use one trusted key server (or key distribution center) for key establishment. However, since the number of users in the network increases extremely, it is inefficient for a single key server to handle the key establishment [9]. Hence a hierarchical KEP suitable for large scale networks should be contrived. In this paper, we first introduce the basic building blocks, including two secure trunks and three secure channels. Base on these blocks, two approaches to KEP construction are proposed. The first approach provides more flexibility for key generation while the second approach requires fewer computations. Finally, we introduce a hierarchical KEP for large scale and administratively heterogeneous networks, e.g. internet.

2. Basic building blocks

For convenience, the definition of SKOWHF, which

is proposed by Berson, Gong and Lomas [7] is repeated here. A function f is a SKOWHF if it satisfies:

- f maps key k and a bit string x to a output string of fixed length.
- Given k and x, it is easy to compute $f(k, \{x\})$.
- Given k and $f(k, \{x\})$, it is computationally infeasible to compute x.
- Given k, it is computationally infeasible to find two values x and y ($x \neq y$) such that $f(k, \{x\}) = f(k, \{y\})$.
- Given pairs x and $f(k, \{x\})$, it is computationally infeasible to compute k.
- Without knowledge of k, it is computationally infeasible to compute $f(k, \{x\})$ for any x.
- The mapping from $\{k, \{x\}\}$ to $f(k, \{x\})$ is randomly chosen in the sense that it should not be possible to predict any portion of $f(k, \{x\})$ [8].

Some researchers are convinced that an SKOWHF can be constructed by using existing unkeyed hash functions such as MD5 [10], SHA [11] and Snefru [12]. In the rest of this paper, f is used to referred to as an SKOWHF.

2.1. Secure trunks

For a sequence of Z nodes $X_1, X_2, \ldots, X_{Z-1}, X_Z$ (not necessary in geographical order), we assume that there exists a common key for each pair of adjacent nodes. That is, X_1 and X_2 have shared a common key CK_{12}, X_2 and X_3 have shared a common key CK_{23}, …, X_{Z-1} and X_Z have shared a common key $CK_{(Z-1)Z}$.

- *Confidentiality trunk*

A confidentiality trunk for m from X_i to X_{i+1} ($1 \leq i \leq z-1$) ensures that only the authorized X_{i+1} will be able to read m sent by X_i and is denoted by

$$X_i \ (c)\!\!\rightarrow X_{i+1}: m.$$

This confidentiality trunk can be realized by using f accompanying with the one-time padding technique in the way as follows:

$$X_i \rightarrow X_{i+1}: n_{Xi}, f(CK_{XiXi+1}, \{n_{Xi}\}) \oplus m$$

where n_{Xi} is the nonce issued by X_i, i.e., this cannot be repeatedly used by X_i with the same key as CK_{XiXi+1}. By padding with $f(CK_{XiXi+1}, \{n_{Xi}\})$, X_i can send m to X_{i+1} without hesitating that it will be compromised to others. Note that a confidentiality trunk doesn't have to ensure that the messages will certainly be obtained by the expected recipient.

- *Integrity trunk*

An integrity trunk for m from X_i to X_{i+1} ($1 \leq i \leq z-1$) ensures that X_{i+1} can check that the received m, which is sent by X_i, has never been tampered and is denoted by

$$X_i \ (i)\!\!\rightarrow X_{i+1}: m$$

And, this integrity trunk can be realized by using f as below.

$$X_i \rightarrow X_{i+1}: m, f(CK_{XiXi+1}, \{m\}).$$

From another view, X_{i+1} can check to see whether m is (or was) actually sent by X_i. Note that an integrity trunk doesn't have to ensure that the received information is newly sent by the claimed sender. In other words, X_{i+1} doesn't need to verify the freshness of that message.

2.2. Secure channels

Now, we employ the proposed two secure trunks to build the secure channels: confidentiality channel, integrity channel and authentication channel. Let X_1 and X_Z represents the end users of the secure channel, in addition, the intermediate nodes between them, i.e., $X_2, X_3, \ldots, X_{Z-1}$, are trustworthy.

- *Confidentiality channel*

The confidentiality channel for the message m from X_1 to X_z, denoted by

$$X_1 \ (c)\!\!\Rightarrow X_z: m,$$

can be constructed by orderly cascading the following secure trunks:

$$X_1 \ (c)\!\!\rightarrow X_2: m \text{ and } X_1 \ (i)\!\!\rightarrow X_2: \{m, ID_{Xz}\}$$
$$\ldots$$
$$X_{z-2} \ (c)\!\!\rightarrow X_{z-1}: m \text{ and } X_{z-2} \ (i)\!\!\rightarrow X_{z-1}: \{m, ID_{Xz}\}$$
$$X_{z-1} \ (c)\!\!\rightarrow X_z: m$$

where ID_{Xz} represents the identity of X_z. Since the intermediate nodes $X_2, X_3, \ldots,$ and X_{z-1} are assumed to be trustworthy, X_1 can believe that only the authorized user X_z will be able to read m.

- *Integrity channel*

The integrity channel for the message m from X_1 to X_z, which is denoted by

$$X_1 \ (i)\!\!\Rightarrow X_z: m,$$

can be constructed by orderly cascading the following secure trunks:

$$X_1 \ (i)\!\!\rightarrow X_2: m$$
$$X_2 \ (i)\!\!\rightarrow X_3: \{m, ID_{X1}\}$$
$$\ldots$$
$$X_{z-1} \ (i)\!\!\rightarrow X_z: \{m, ID_{X1}\}.$$

The integrity channel for m from X_1 to X_Z ensures that X_z can check that m is or has been sent by X_1. Moreover, each individual item contained in m cannot be modified alone unless the whole m is replaced with an old one ever sent by X_1.

- *Authentication Channel*

The authentication channel for the message m from X_1 to X_z, which is dented by

$$X_1 \ (a)\!\!\Rightarrow X_z: m,$$

can be constructed by directly using the integrity channel from X_1 to X_z:

$$X_1 \ (i)\!\!\Rightarrow X_z: \{m, n_{Xz}\},$$

where n_{Xz} is a nonce issued by X_z and is used as the *fresh-*

ness identifier for *m*. From this channel, X_z should be able to check the freshness of *m* and the sender of *m*. If verified, X_z can believe that *m* is newly sent by X_1.

Remark 1. The assumption of using trustworthy intermediate nodes between X_1 and X_z will be reasonably justified later in the proposed hierarchical KEP, in which these nodes are acted by key servers rather than common users.

3. Two approaches to KEP construction

In this section, two approaches to KEP construction are described. The major difference between these two approaches is the manner of generating a session key. In the first approach, the session key is generated by an independent random number generator. While in the second approach, the session key is generated by using an SKOWHF. Mutual confirmation can be achieved in both approaches so that one can check to see whether the peer end has obtained the right session key or not. In fact, Boyd and Mathuria omit such a function for simplification. However, for practical use, mutual confirmation should be considered in the KEP so that the resulting design can serve as a complete module for the adopted system. Notations *A* (the originator) and *B* represent the two users who are going to establish a session key.

3.1. The first approach

A session key *sk*, which is independently generated, can be established between *A* and *B* using the authentication and confidentiality protocols according to the following key establishment rule.

KER1:

(1) *A* sends n_A to *B*.

(2) *B* $\overset{(c)}{\Longrightarrow}$ *A*: {*sk*, *y*} and *B* $\overset{(a)}{\Longrightarrow}$ *A*: {*sk*, *y*}

(3) $A \rightarrow B$: *y*

Nonce *y* is issued by *B* and is used for confirming *A*'s knowledge of *sk*.

Theorem 1. According to KER1, A and B can secretly share sk.

Proof: Since *sk* is independently generated by *B*, it implies that *B* believes it is a fresh session key. From the first part of (2), {*sk*, *y*} can be sent to *A* confidentially. From the second part of (2), *A* can make sure that the retrieved {*sk*, *y*} is newly sent by *B*. If succeeds, *A* can also believe that *B* has obtained *sk*. From (3), *B* can check to see whether the received *y* is equal to the one previously issued, if so, he believes *A* has obtained the right *sk*. Consequently, *sk* is secretly shared by *A* and *B*. ■

3.2. The second approach

A session key can be generated by computing the session key generation formula (SKGF) rather than by a random number generator. With this approach, some SKOWHF computations can be mustered out.

KER2:

(1) *A* sends n_A to *B*.

(2) *Case I.* If *A* and *B* have shared a common key CK_{AB}, *B* computes the SKGF $\{sk, h_A, h_B\} = f(CK_{AB}, \{n_A, n_B\})$ and then sends $\{n_B, h_B\}$ to *A*. Next, *A* also computes the same formula as *B* does.

Case II. If *A* and *B* don't share a common key, *B* computes SKGF $\{sk, h_A, h_B\} = f(CK_{BW}, \{n_A, n_B, ID_A\})$ and then sends $\{ID_A, n_A, n_B, h_B\}$ to his adjacent node *W*. Next, *W* also computes $\{sk, h_A, h_B\}$ as *B* does. If the computed h_B equals the received one, *W* $\overset{(c)}{\Longrightarrow}$ *A*: $\{sk, h_A\}$ and *W* $\overset{(a)}{\Longrightarrow}$ *A*: $\{sk, h_A\}$.

(3) $A \rightarrow B$: h_A

Handshake numbers h_A and h_B are used by *A* and *B*, respectively, for mutual confirmation.

Theorem 2. Following KER2, A and B can secretly share sk.

Proof: In case I, CK_{AB} is only known to *A* and *B*, so no one else can obtain $\{sk, h_A, h_B\}$ using the SKGF. Furthermore, n_A and n_B are in the input of *f* and it implies that $\{sk, h_A, h_B\}$ is controlled by both *A* and *B* rather than by either one alone. On receiving n_B, *A* can compute the SKGF to derive h_B and then check to see whether it equals to the received one. If so, he believes that *B* has obtained the right session key. On receiving h_A in (3), *B* believes that *A* has obtained the right session key.

In case II, a confidentiality channel and an integrity channel for $\{sk, h_A, h_B\}$ from *A* to *W* can be constructed. If the n_A received by *W* is fresh, a confidentiality channel and an authentication channel for $\{sk, h_A\}$ from *W* to *A* is subsequently constructed. From the characteristic of the authentication channel, a confidentiality channel and an authentication channel for $\{sk, h_A\}$ from *B* to *A* is constructed. In the meanwhile, *A* believes that *B* has obtained the right session key. On receiving h_A in (3), *B* believes that *A* has obtained the right session key.

Hence *sk* can be secretly shared by *A* and *B* no matter in case I or II. ■

4. Hierarchical key establishment protocols

For practical use, we use the proposed key establishment rules to construct KEP for users within a large scale network such as Internet [9].

4.1. Hierarchical structure

Because the number of network users increases drasti-

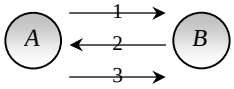

Figure 1. KEP steps of scenario 1.

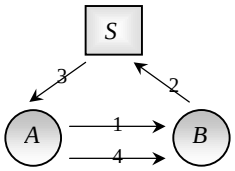

Figure 2. KEP steps of scenario 2.

cally, it is inefficient for a single key server (or key management center) to execute key establishment. To reduce the management complexity, the system can be organized hierarchically [9]. A number of users are under the control of a domain key server and a number of domain key servers are under the control of a cluster key server, and so on. It is assumed that each entity has shared a common key, or mater key, with his control server. To provide greater flexibility, entities of the same level are allowed to share a common key. It is economical when they frequently communicate. Let *A* and *B* denote the originator and the corresponding user of the protocol, respectively. MK_A, the master key of *A*, is used as the common key for *A* and his key server. Similarly, MK_B is the master key of *B* and is used as the common key for *B* and his key server.

4.2. Scenarios

In the following, we simultaneously apply KER1 and KER2 to construct hierarchical KEPs. Four scenarios are used to illustrate the way of establishing session keys for the most common spots in large scale networks.

Scenario 1. A and B have already shared a common key CK_{AB}. (*See also Figure 1.*)

Involved key servers: none

• According to KER1

In this situation, *A* and *B* can directly construct the needed confidentiality channel and authentication channel without the help of any key server. The KEP is illustrated as follows:

Step 1. $A \rightarrow B$: n_A

B generates the session key *sk* using an independent key generator, e.g., a random number generator. Then, *sk* combined with another nonce *y* is transmitted back to *A* over the secure channels where $f(CK_{AB},\{n_B\}) \oplus \{sk, y\}$ forms the confidentiality channel and $f(CK_{AB},\{sk, y, n_A\})$ forms the authentication channel.

Step 2. $B \rightarrow A$: n_B, $f(CK_{AB},\{n_B\})\oplus\{sk, y\}$, $f(CK_{AB},\{sk, y, n_A\})$

Once all the received messages have been verified, *A* believes that *sk* is newly sent by *B*. However, *B* doesn't know that whether *A* has obtained the correct session key or not. Accordingly, a further confirmation from *A* to *B* should be conducted as follows:

Step 3. $A \rightarrow B$: y

In the protocol, *A* needs to perform two SKOWHF computations, and so does *B*.

• According to KER2

Clearly, the rule specified in case I of KER2 can be directly used here.

Step 1. $A \rightarrow B$: n_A

On receiving n_A, *B* generates n_B and then computes the SKGF:

$$\{sk, h_A, h_B\} = f(CK_{AB}, \{n_A, n_B\}).$$

Next, *B* sends n_B and h_B back to *A*.

Step 2. $B \rightarrow A$: n_B, h_B

The handshake number h_B is used by *B* to inform his knowledge of *sk* to *A*. After retrieving n_B from the received message, *A* also computes the SKGF. Next, he compares the computed h_B with the second item of the received message, if equal, he believes that *B* has obtained the same session key as his own and then sends h_A to *B*.

Step 3. $A \rightarrow B$: h_A

where h_A is used by *A* to inform his knowledge of *sk* to *B*.

In the protocol, *A* needs to perform only one SKOWHF computation, and so does *B*.

Scenario 2. A and B don't share a common key but both are under the control of S. (*See also Figure 2.*)

Involved key servers: *S*

• According to KER1

Step 1. $A \rightarrow B$: n_A

Since *S* is the only intermediate server between *A* and *B*, there are two confidentiality trunks $B \xrightarrow{(c)} S$ and $S \xrightarrow{(c)} A$ and two integrity trunks $B \xrightarrow{(i)} S$ and $S \xrightarrow{(i)} A$ should be established first to construct the required confidentiality and authentication channels.

Step 2. $B \rightarrow S$: n_B, $f(MK_B, \{n_B\}) \oplus \{sk, y\}$, ID_A, n_A, $f(MK_B, \{sk, y, ID_A, n_A\})$

As MK_B is a common key between *B* and *S*, we can regard *B*'s identity ID_B as being implicitly contained in the use of MK_B, i.e. it could be omitted.

Step 3. $S \rightarrow A$: n_S,$f(MK_A,\{n_S\})\oplus\{sk, y\}$,$f(MK_A,\{sk,y,ID_B, n_A\})$

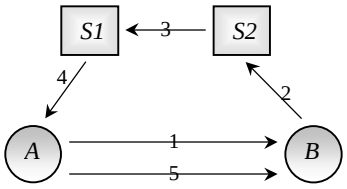

Figure 3. KEP steps of scenario 3.

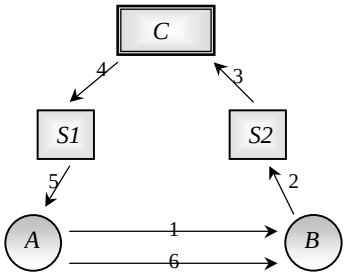

Figure 4. KEP steps of scenario 4.

Note that ID_B is not transmitted in clear mode in the flow. Obviously, *A* should know the identity of *B*.

Step 4. $A \rightarrow B$: y

In this protocol, *A* and *B* need to respectively perform two SKOWHF computations while *S* needs four.

• According to KER2

This could be regarded as the simplest instance of case II of KER2. And, we will go into a more detail about the protocol. Similarly, *sk* is generated by computing the SKGF: $\{sk, h_A, h_B\} = f(MK_B, \{n_A, n_B, ID_A\})$. Note that ID_A should be included in the input of *f*. The protocol can be briefly expressed as shown below.

Step 1. $A \rightarrow B$: n_A

Step 2. $B \rightarrow S$: ID_A, n_A, n_B, h_B

Step 3. $S \rightarrow A$: $n_S, f(MK_A, \{n_S\}) \oplus \{sk, h_A\}, f(MK_A, \{sk, h_A, ID_B, n_A\})$

Step 4. $A \rightarrow B$: h_A

Upon receiving this request, *B* generates n_B and then computes the SKGF to derive $\{sk, h_A\}$. It should be emphasized that ID_A should be included in the parameter for *f* so that *B* can be protected from being fooled into believing the impersonator of *A*. On receiving the request from *B* in flow 2, *S* also performs the SKGF to derive $\{sk, h_A\}$ as *B* does. Then, he compares the h_A retrieved from flow 2 with the computed one. Once unequal, *S* quits the protocol. Otherwise, *S* transmits $\{sk, h_A, h_B\}$ to *A* through a confidentiality channel and an authentication channel. Next, *A* can verify *sk* and recognize *B*'s knowledge of *sk* by checking the validity of the received messages. If correct, *A* sends h_A to *B*. Then, *B* can recognize *A*'s knowledge of *sk* by comparing the h_A retrieved from flow 4 and the one he holds.

We can see that *A* authenticates *S* as well as *B*, and *B* authenticates *S* as well as *A*. However, *S* authenticate neither *A* nor *B*. In total, *A* needs to perform two SKOWHF computations, *B* needs one SKOWHF computation and *S* needs three SKOWHF computations.

Scenario 3. A and B don't share a common key. But their key servers S1 and S2 have shared a common key CK_{S1S2}. (*See also Figure 3.*)

Involved key servers: *S2, S1*

• According to KER1

As *S2* and *S1* are the intermediate servers between *A* and *B*, there are three confidentiality trunks $B \xrightarrow{(c)} S2$, $S2 \xrightarrow{(c)} S1$ and $S1 \xrightarrow{(c)} A$ and three integrity trunks $B \xrightarrow{(i)} S2$, $S2 \xrightarrow{(i)} S1$ and $S1 \xrightarrow{(i)} A$ should be established first to construct the need confidentiality and authentication channels.

Step 1. $A \rightarrow B$: n_A

Step 2. $B \rightarrow S2$: $n_B, f(MK_B, \{n_B\}) \oplus \{sk, y\}, ID_A, n_A, f(MK_B, \{sk, y, ID_A, n_A\})$

Step 3. $S2 \rightarrow S1$: $n_{S2}, f(CK_{S1S2}, \{n_{S2}\}) \oplus \{sk, y\}, ID_A, ID_B, n_A, f(CK_{S1S2}, \{sk, y, ID_A, ID_B, n_A\})$

Step 4. $S1 \rightarrow A$: $n_{S1}, f(MK_A, \{n_{S1}\}) \oplus \{sk, y\}, f(MK_A, \{sk, y, ID_B, n_A\})$

Step 5. $A \rightarrow B$: y

• According to KER2

This protocol simply needs to establish two sets of secure trunks should be established, i.e., $S2 \xrightarrow{(c)} S1$ and $S1 \xrightarrow{(c)} A$, and $S2 \xrightarrow{(i)} S1$ and $S1 \xrightarrow{(i)} A$.

SKGF: $\{sk, h_A, h_B\} = f(MK_B, \{n_A, n_B, ID_A\})$

Step 1. $A \rightarrow B$: n_A

Step 2. $B \rightarrow S2$: ID_A, n_A, n_B, h_B

Step 3. $S2 \rightarrow S1$: $n_{S2}, f(CK_{S1S2}, \{n_{S2}\}) \oplus \{sk, h_A\}, ID_A, ID_B, n_A, f(CK_{S1S2}, \{sk, h_A, ID_A, ID_B, n_A\})$

Step 4. $S1 \rightarrow A$: $n_{S1}, f(MK_A, \{n_{S1}\}) \oplus \{sk, h_A\}, f(MK_A, \{sk, h_A, ID_B, n_A\})$

Step 5. $A \rightarrow B$: h_A

Scenario 4. Neither A and B nor their key servers S1 and S2 have shared a common key. However, S1 and S2 are under the control of C. (*See also Figure 4.*)

Involved key servers: *S2, C, S1*

• According to KER1

Step 1. $A \rightarrow B$: n_A

Step 2. $B \rightarrow S2$: $n_B, f(MK_B, \{n_B\}) \oplus \{sk, y\}, ID_A, n_A, f(MK_B, \{sk, y, ID_A, n_A\})$

Step 3. $S2 \rightarrow C$: n_{S2}, $f(MK_{S2}, \{n_{S2}\}) \oplus \{sk, y\}$, ID_A, ID_B, n_A, $f(MK_{S2}, \{sk, y, ID_A, ID_B, n_A\})$
Step 4. $C \rightarrow S1$: n_C, $f(MK_{S1}, \{n_C\}) \oplus \{sk, y\}$, ID_A, ID_B, n_A, $f(MK_{S1}, \{sk, y, ID_A, ID_B, n_A\})$
Step 5. $S1 \rightarrow A$: n_{S1}, $f(MK_A, \{n_{S1}\}) \oplus \{sk, y\}$, $f(MK_A, \{sk, y, ID_B, n_A\})$
Step 6. $A \rightarrow B$: y

- According to KER2

SKGF: $\{sk, h_A, h_B\} = f(MK_B, \{n_A, n_B, ID_A\})$
Step 1. $A \rightarrow B$: n_A
Step 2. $B \rightarrow S2$: ID_A, n_A, n_B, h_B
Step 3. $S2 \rightarrow C$: n_{S2}, $f(MK_{S2}, \{n_{S2}\}) \oplus \{sk, h_A\}$, ID_A, ID_B, n_A, $f(MK_{S2}, \{sk, h_A, ID_A, ID_B, n_A\})$
Step 4. $C \rightarrow S1$: n_C, $f(MK_{S1}, \{n_C\}) \oplus \{sk, h_A\}$, ID_A, ID_B, n_A, $f(MK_{S1}, \{sk, h_A, ID_A, ID_B, n_A\})$
Step 5. $S1 \rightarrow A$: n_{S1}, $f(MK_A, \{n_{S1}\}) \oplus \{sk, h_A\}$, $f(MK_A, \{sk, h_A, ID_B, n_A\})$
Step 6. $A \rightarrow B$: h_A

4.3. Analysis

The number of required SKOWHF computations by each constituent in all the four scenarios is depicted in Table 1.

Appr. / Scen.	The First Approach				The Second Approach			
	A	*B*	*Sever of B*	*other Server*	*A*	*B*	*Sever of B*	*other Server*
1	2	2	—	—	1	1	—	—
2	2	2	4	—	2	1	3	—
3	2	2	4	4	2	1	3	4
4	2	2	4	4	2	1	3	4

Table 1. Needed SKOWHF computations by each constituent in the four scenarios.

From the table, we know that the protocols of the second approach demands fewer computations. It may be an advantage of the second approach over the first approach especially when computation should be performed in a light-weight device such as a smart card. In contrast, the first approach provides more flexibility for session key selection because the session key can be chosen to have a specific structure. However, it is not obvious that the session key should have this feature.

5. Conclusion

A hierarchical KEP suitable for large scale networks is constructed based on SKOWHF. With hierarchical structure of key servers, the concept of using secure trunks to construct secure channels is practical and can also be applied to any system employing cryptographic techniques. It should be emphasized that no intermediate key server needs to authenticate the received messages. Instead, only the expected recipient (end user) should authenticate the messages. It is the reason why the authentication trunk is not discussed in this paper. Conscious reader may find that the sequence of protocol steps in each scenario forms a loop-around pattern. However, either proving its optimization or contriving another better design is a topic for our further investigation.

References

[1] R. Molva, D. Samfat, and G. Tsudik, "Authentication of Mobile Users," *IEEE Network*, 26–34, Mar./Apr. 1994.

[2] L. Gong, "Using one-way functions for Authentication," *ACM CCR*, 19(5): 8–11, Oct. 1989.

[3] J.A. Bull, L. Gong, and K.R. Sollins, "Towards security in an open systems federation," in Y. Deswarte, G. Eizenberg, and J.-J. Quisquater (Eds.): *Computer security — ESORICS 92', Lect. Notes Comput. Sci.*, 648: 3–20, 1992.

[4] A. Mathuria, "Addressing weaknesses of two cryptographic protocols of Bull, Gong and Sollins," *Elect. Lett.*, 31: 1543–1544, 1995.

[5] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung, "Systematic design of a family of attack-resistant authentication protocols," *IEEE J. Select. Areas Commun.*, 11: 679–693, June, 1993.

[6] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung, "The KryptoKnight Family of Light-Weight Protocols for Authentication and Key Distribution," *IEEE/ACM Trans. on Networking*, 3(1): 31-41, Feb. 1995.

[7] T.A. Berson, L. Gong, and T.M.A. Lomas, "Secure, keyed, and collisionful hash functions," Technical report SRI-CSL-94-08, Computer Science Laboratory, SRI International Menlo Park, CA., May 1994.

[8] C. Boyd, A.Mathuria, "Systematic design of key establishment protocols based on one-way functions," *IEE Proc.-Comput. Digit. Tech.*, 144(2): 93–99, Mar. 1997.

[9] T. Hwang and W.-C. Ku, "Reparable Key Distribution Protocols for Internet Environment," *IEEE Trans. on Commun.*, 43(5): 1947–1949, May 1995.

[10] R. Rivest, "The MD5 Message Digest Algorithm," Internet Draft, July 1991.

[11] National Institute of standards and Technology, "Secure Hash Standard," *NIST FIPS PUB* 180, U.S. Department of Commerce, Draft, 1993.

[12] R. C. Merkle, " A Fast Software One-Way Hash Function, " *Journal of Cryptology*, 3(1): 43–58, 1990.