1998 International Computer Symposium
Workshop on Cryptology and Information Security
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

A SYSTEMATIC METHOD FOR DESIGN OF KEY ESTABLISHMENT PROTOCOLS

Shu-Chung Ju,, and Sheng-De Wang

Department of Electrical Engineering,
National Taiwan University, Tainan, Taiwan, R.O.C.
Email: {earnest,sdwang}@hpc.ee.edu.tw

ABSTRACT

A key establishment protocol is to establish a session key among all of the communicating parties in a computer network, so that they can use the session key to secure the subsequent messages in the session. Most existing key establishment protocols assume a specific network environment and are not subject to a systematic design approach. In this paper, we analyzed the essentials of key establishment procedure, and proposed a systematic method to design key establishment protocols. The salient features of it include clarifying security achievements, capability of identifying potential efficiency, adaptability to diverse network environments and generalized authentication environments, and easy applicability.

1. INTRODUCTION

Before two or more parties in a network begin their sessions, they may exchange some messages to authenticate each other and establish a shared key to achieve the confidentiality and authentication of the subsequent communication messages. The key establishment protocols (KEP) serve for such propose, and the established key is named as session key (SK) because the key is used only for the session. Without a reliable key establishment protocol, the security requirements of sessions cannot be met at all. Therefore, the design and analysis of key establishment protocols are the very topics of the security research and many of them have been proposed.

There are four basic requirements of a KEP; they are authentication, confidentiality, confirmation, and freshness [1][2]. It is important to examine if a KEP satisfies these requirements and how the jobs are done. We claim that a complete KEP should achieve all these requirements [3].

1.1 The Implementation of a KEP

A KEP can be constructed by symmetric-key or asymmetric-key cryptosystems (e.g. [4][5][6]). But there are restrictions when the KEP is put into practical use. For example, such KEPs may consume too much computing power to be adopted in a computing-power- limited network. In addition, the use and export of cryptographic algorithms is restricted in many countries, so those KEP cannot be adopted widely [7][8].

On the other hand, using one-way hash functions as a basis for KEP is found to be feasible while it avoids the problems mentioned above. The mobile telephone system, GSM, also adopts KEP based on one-way hash functions [9]. However, it is more difficult to construct a KEP by one-way hash functions than by cryptosystems. For example, to keep messages confidential, one can encrypt them by cryptosystems. But you cannot do that by one-way hash functions. A special technique, called **one-time padding**, is needed to deal with it.

One important consideration of one-way hash functions is that the functions should have several properties, which is defined by Berson *et al.* [10] and revised by Boyd *et al.* [2]. A one-way function with these properties is defined as a secure keyed one-way hash function (SKOWHF). It is believed that conventional unkeyed hash functions, such as MD5 [11], SHA [12], Snefru [13] can be used to construct SKOWHF.

By either cryptosystems or SKOWHF, we can construct basic security building blocks to accomplish authentication or message confidentiality. And by these building blocks, we can construct a KEP. In this paper, our method for the design of KEP is not limited to any kinds of implementations of security building blocks.

1.2 Systematic Methods for Design of KEP

Taking network environment into consideration, a family of KEPs is needed to handle all the different situations. Some existing KEPs assume a specific communication paradigm, while KryptoKnight offers different KEPs scenarios for different network connectivity [7][14]. However, besides its flaws reported in [2], its assumption lacks of generality. It assumes that the communicating parties can contact each other directly and that the protocols run merely with one authentication server and two communicating parties. When the secret keys are hierarchically distributed over several authentication servers, or when more than two communicating parties want to establish a common session key among them, all its scenarios cannot be adopted. It does not take bandwidth or traffic load of the links between communicating parties into consideration, either. For efficiency and usability, limited data transmission speed of network links and limited processing power of network nodes should be also taken into consideration. Therefore, we need a systematic method for design of KEP in order to produce suitable KEP for different network environments.

A systematic approach to design and analysis of KEP can help to describe how the protocol works clearly, and to show how the protocol achieves requirements. Meanwhile, a KEP designed in a systematic way will be more analyzable and understandable. In [2], the authors identify two basic channels for KEP: an authentication channel and a confidentiality channel. Their analysis on several existing KEPs demonstrates that without a systematic design the KEPs are extremely prone to flaws. They also propose their methodology for systematic design of KEP. However, it shows a lack of generality and it is still too ambiguous to be put into practical use. Another issue of it is that it does not deal with key confirmation.

In [3], we claim that a systematic method has five requirements: clarified correctness, efficiency, adaptability to network environments, generality of authentication environments, and applicability. The method proposed in the paper is aimed to achieve these requirements.

2. BUILDING BLOCKS OF A KEP

2.1 Security Channels

Security channels are the most primitive security mechanisms by which we can construct KEP. We follow [2] and identify two building blocks of security channels: the confidentiality channel and the authentication channel. But our implementation of the security channels posses more generality.

Our implementation is based on a small mechanism, called **nonce**, a non-repeated random number. Any node in the network should generate its own nonce and keep the nonce non-repeated. For generality, we claim that a nonce should not be used solely and it should be used along with the ID number of the party that generates the nonce. Thus, the non-repetition property of nonce can be guaranteed all over the network.

Before we go further, we first introduce the notations in the paper to facilitate our discussion. We use f to denote a SKOWHF. The output of f is denoted by $f\,(Key,\ \{Data\})$, where the first input argument is a secret key and the second one is data, a bit string of any length. The capital letters such as A, B, C, and so on denote individual communicating parties, which may be computers or users; but we use the capital character S to denote an authentication server. *Kab* denotes the secret key shared by A and B in advance. *Na* denotes a nonce generated by A.

2.1.1 Confidentiality Channel

Two parties sharing a secret key between them can build a confidentiality channel to transfer their secret messages. Here we demonstrate two techniques for building a confidentiality channel by SKOWHF. The first is one-time padding technique, which generates a random bit string, called **one-time pad** to mask the secret. One has to generate the same random bit string to unmask the message in order to obtain the correct message. Our implementation for one-time padding technique is as follows:

A$\rightarrow$B: $Na,\ Pab \oplus m$ (A sends message to B),

where $Pab = f(Kab, \{Na, IDa\})$ (one-time pad),

and $\oplus$ denotes Boolean operation: exclusive-or.

The second technique is by transferring a nonce. Sender A generates a nonce *Na* and delivers it to receiver B. Then B can derive the confidential message *m* by computing:

$$m = f\,(Kab,\ \{Na,\ IDa\}\,)$$

Without knowledge of *Kab*, one cannot derive *m*. The idea of using $\{Na,\ IDa\}$ instead of $\{Na\}$ solely is to prevent the situation that when B sends a nonce of the same value as *Na*, then one can deduce that B delivers the same confidential message *m* to A. However, Sender A cannot control the value of *m* in advance. When the value of *m* can be randomly chosen, e.g. key, this technique can be used.

2.1.2 Authentication Channel

The authentication channel here is constructed by a challenge-response scenario. The initiator sends a nonce as a challenge to the responder, who then responses with a value that can demonstrate its identity. Our implementation is as follows:

B$\rightarrow$A : *Nb* (challenging nonce by B)

A$\rightarrow$B : $m, f(Kab,\ \{Nb,\ IDb,\ m\})$ (response by A)

If $f\,(Kab,\ \{Nb,\ IDb,\ m\})$ is correct, B can assure that *m* is actually and currently sent by A. The authentication channel can defend the communication from replay attacks. It should be noted that only B could verify A's identify, not could A. Besides, the message *m* is sent in plain text.

2.2 Building Blocks of a KEP

A complete KEP consists of three phases: **key generation**, **key distribution**, and **key confirmation**. In the first phase, the communicating parties agree on the method for the generation of the session key. In the second phase, the parties who have acquired the session key distribute it through security channels to the other parties who do not have the session key yet. When a party has received or derived the session key, it starts to deliver confirmation messages to other parties to accomplish the final phase, key confirmation.

There are two methods of session key generation: the **key generator** and the **key agreement**. In the former method, one party, called the **SK generator**, in the network is responsible to generate the session key. The session key of the later is computed by a one-way hash function; the input arguments to the function are a secret key and nonces contributed by the parties in the network. In either method, the parties who do not compute or generate the session key have to obtain the session key from other parties. To transfer a session key from one party to another, we can combine the confidentiality channel and the authentication channel to construct key transfer channels. We also give our implementation of them based on SKOWHF.

2.2.1 Key Transfer Channel

If B has acquired or derived the session key, A can obtain the key from B, directly or indirectly. If A and B share a secret key *Kab*, B can transfer a confidential session key to A directly, and A can authenticate B directly, too. A direct key transfer channel can be built between them. However, if they do not share a secret key, they have to build an indirect key transfer channel between them with the help of at least one authentication server, which shares secret keys with A and B respectively.

Our implementation of a direct key transfer channel based on SKOWHF is as follows:

$A \rightarrow B : Na$ (challenging nonce by A)

$B \rightarrow A : Nb, Pba \oplus SK, f(Kab, \{Na, IDa, SK\})$,

where $Pba = f(Kab, \{Nb, IDb\})$ (one-time pad)

Here the authentication channel of session key is defined as *SK* and $f(Kab, \{Na, IDa, SK\})$, and confidentiality channel is defined as *Nb* and $Pba \oplus SK$. By this protocol, B cannot authenticate A, but B is sure that no one except A can derive the *SK*; on the other hand, A can authenticate B and obtain an authenticated session key from B.

The following is our implementation of an indirect key transfer with one authentication server:

$A \rightarrow B : Na$ (challenging nonce by A)

$B \rightarrow S : Na, Nb, Pbs \oplus SK, f(Kbs, \{Na, IDa, SK\})$

$S \rightarrow A : Ns, Psa \oplus SK, f(Kas, \{Na, IDa, SK\})$,

where $Pbs = f(Kbs, \{Nb, IDb\})$, $Psa = f(Kas, \{Ns, IDs\})$

The security channels between A and B cannot be built directly, since they do not share a secret key. However, session key can be transferred through two confidentiality channels: one between B and S, and the other one between S and A. B is sure that no one except S can derive the *SK*, and S is sure that no one except A can derive the *SK*.

Now let us consider how A authenticates B and S. A gives its challenging nonce *Na* to B and S. Only when B responses correctly to *Na* does S send message to A. When A receives correct response from S, it then believes that B response correctly to S and the *SK* is truly sent by S. S neither authenticates A or B nor trusts the *SK* sent by B. And S has to be a trusted party because S can modify the *SK* or impersonate A or B.

2.2.2 Key Agreement

In key agreement of three-party KEP, the session key is computed by two nonces contributed by two of the parties, called **nonce contributors**. The idea of key agreement comes from the fact that nonce transfer can construct a confidentiality channel. When acquiring nonces, the parties with knowledge of the secret key, called **SK computing parties**, can derive the session key by computing the following equation, called **SK generation equation**:

$$SK = f(Key, \{N1, N2\}).$$

For example, when two communicating parties, A and B, share a secret key *Kab* with each other, then the session key is computed by:

$$SK = f(Kab, \{Na, Nb\}).$$

In order to compute the session key *SK*, A and B need to exchange each other's nonce. Therefore, the key agreement protocol for two parties is as follows:

$A \rightarrow B : Na$ (nonce generated by A)

$B \rightarrow A : Nb$ (nonce generated by B)

When an authentication server S is placed between A and B, the session key can be computed in two ways. Either the secret key Kas shared between A and S or the secret key *Kbs* shared between B and S can be the input of the SK generation equation. The parties who have the secret key can compute the session key. If it is *Kas*, A and S can compute the session key, and similarly if it is *Kbs*, B and S can. In addition, the two nonces can be contributed by A and S, B and S, or A and B. Therefore, there may be six possibilities of the session key generation equation.

2.2.3 Key Generator

An SK generator produces a session key for all communicating parties. If a party shares a secret key with the SK generator, it can obtain the session key through a direct key transfer channel. If it does not, at least one authentication server should intervene between them, so that an indirect key transfer channel can be constructed to deliver session keys from the SK generator. By either direct or indirect key transfer channels, all parties can obtain session keys from the SK generator.

The SK generators control the value of session key. The properties of session key such as freshness and unpredictability can be centrally managed and acquired. The communicating parties can choose a dependable SK generator to guarantee the quality of their session key. On the contrary, in key agreement the session key is produced by a one-way hash function with nonces randomly chosen; there is no central session key controlling. However, a possible solution is that a communicating party that found the session key is not appropriate can send an ABORT message to the others to abort the session key of the running KEP. Afterwards, they try to re-establish another session key until every party is satisfied with the session key.

On the other hand, the freshness and unpredictability of session keys in key agreement can be achieved easily, since the session key is generated by a one-way hash function. The session key is fresh because the input nonces and key are fresh, and it is unpredictable because the output of the function is unpredictable if the input is fresh. In the key generator approach, an SK generator can force the other parties to accept any session keys, so the freshness and unpredictability are determined solely by the SK generator. That makes the level of trustiness on the SK generators higher than other normal parties.

2.2.4 Key Confirmation

After the communicating parties have acquired session key either by receiving it from other parties or by computing the SK generation equation, they starts to confirm whether the others have the same session key as theirs. One party believes that the session key is distributed successfully to the party and that the party is not an imposter if the responses of the party for key confirmation are correct.

There are two primary methods to do key confirmation: the **self challenge-response** and exchanging **handshake numbers**. In the self challenge-response approach, each party uses its ID number as a self-challenge and uses the session key to compute a response to it. For two parties, they can exchange messages as follows:

$$A \rightarrow B : f(SK, \{IDa\})$$

$$B \rightarrow A : f(SK, \{IDb\})$$

Because the session key is newly generated and delivered confidentially, one cannot replay old self challenge-response messages or forges valid messages to mislead other parties.

The second method of key confirmation requires that the SK generation equation output not only session key but also two or more handshake numbers. When A receives proper handshake numbers from B, it then believes B has acquired the session key and vice versa; because one cannot have the handshake numbers without the session key. They can exchange such messages as follows:

$$A \rightarrow B : Ha$$

$$B \rightarrow A : Hb$$

Here *Ha* and *Hb* are the handshake numbers. An important requirement here is that one cannot derive the session key by the handshake numbers.

3. MESSAGE SEQUENCE RULES TO THREE-PARTY KEP

Message Sequence Rules (MSR) are the required patterns of message flow in a KEP depending on the method of the session key generation. A KEP can be constructed based on the building blocks defined above if the message flow of it obeys the rules. Before applying MSR, we have to transform the KEP into the **Message Sequence Diagram** (MSD), in which a MSR is denoted as a required path consisting of nodes and directional links.

3.1 Message Sequence Diagram (MSD)

Existing graphical representations of KEP often depict merely the list of the messages in the protocol. On the contrary, the MSD focuses on describing the precedence relationship of messages in the KEP and so it is easier to show the parallelism of the message flow.

Figure 1 presents the precedence relationship of the messages in a protocol. The arcs denote the messages. The node that has an arc pointing to it denotes the event of receiving the message, and the node that has an arc leaving it denotes the event of sending the message. The arcs also depict the transferring direction of messages. Therefore, an MSD cannot only present the precedence relationship of the messages, but also depict the receiving and sending of messages between the parties.

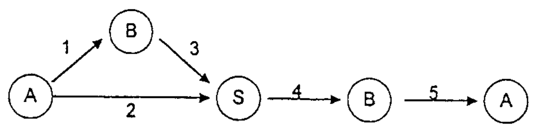

Figure 1. A message sequence diagram

3.1.1 Notation

We can follow along the nodes and the directional arcs to construct a path in the MSD. An MSR corresponds to a path in the MSD. A set of MSR specify what kind of paths an MSD should have. Another notation of MSR relates to key acquirement, which denotes whether the parties in the KEP have acquired the authenticated session key.

The simplest form of the path existing notation is A→B. It means that there is at least one path exists from A to B in the MSD. It should be noted that without an arc connecting A and B directly, a path from A to B may also exist. For example, in figure 1 along the arc 2 and 4 and the intermediate node S, the path A→B exists.

A→B→S denotes that there exists at least one path from A to S, while B is one of the intermediate nodes in the path.

{A, B}→S denotes that both A→S and B→S exists, and the path S→{A, B} denotes that both S→A and S→B exists. Using brace notation here is to simplify the representation of multiple existing paths.

3.1.2 Key Acquirement

When a party, say A, in a path has derived or received session key, its node in the MSD is denoted by (A). It is obvious that all nodes of A in the path after (A) should be (A) as well. How to decide if the party has acquired session key will be discussed in details later. And the following three sections are MSR for building blocks of three-party KEP.

3.2 MSR for Key Generator

In the key generator approach, the SK generator produces session keys. In two-party KEP, the party who does not generate session keys needs to obtain the session key from the other one in the KEP. For example, if and B is the SK generator, A has to obtain the session key from B through direct key transfer and the MSR is:

A → (B) → (A),

where because B generates session key, the first node of B in the path should have acquired the session key and every node of B in the path is denoted as (B). After A obtains response from B, it may also be able to acquire session key from B's message, so the final node of A in the path is denoted as (A).

In three-party KEP, the session key may be generated by the authentication server S or by the other communicating parties, A or B. If S generates the session key, it has to transfer the session key directly to A and B, so the MSR is:

A→ (S) →(A) AND B → (S) → (B), (KG 1)

where AND denotes the existence of both paths.

If A generates the session key, by the indirect key transfer channel, A can transfer it to B with the aid of S through the path:

B → (A) → (S) → (B) (KG 2.1)

Or, A can transfer the session key to S directly, and S in turn transfers it directly to B:

S → (A) → (S) AND B → (S) → B (KG 2.2)

Similarly, if B generates the session key, two possibilities exists:

A → (B) → (S) → (A) (KG 3.1)

S → (B) → (S) AND A → (S) → A (KG 3.2)

3.3 MSR for Key Agreement

In the key agreement approach, the session key is generated by a one-way hash function, whose input parameter of key part is a secret key shared between two parties in the KEP and that of data part is two nonces. The goal of MSR in key agreement is to distribute nonces to the SK computing parties, which have ability to compute session keys and to transfer them to the other parties, which are unable to compute the session keys. The MSR for the former is called **nonce transfer rules** and that for the later is called **key transfer rules**.

In two-party KEP, both of the computing parties can compute the session key after receiving nonce from the other one. So, the MSR is:

A → (B) AND B → (A)

In three-party KEP, the key part of the SK generation equation can be *Kas* or *Kbs*, and the nonce parameters can be two of the three nonces generated by three parties respectively. If the key part is *Kas*, the nonces are distributed to A and S, and B has to obtain the session key from either A indirectly or S directly. The MSR for *Kas* and different nonce contributors are listed as follows:

If *SK = f(Kas, {Na, Nb, IDb})* then

{A, B}→(S) AND B→(A) AND B →(S)→ (B) (KA 1.1)

If *SK = f(Kas, {Na, Ns, IDb})* then

A → (S) AND S → (A) AND B → (S) → (B) (KA 1.2)

If *SK = f(Kas, {Nb, Ns, IDb})* then

B→(S) AND {B, S}→(A) AND B→(S)→(B) (KA 1.3)

Now we give expatiation of (KA 1.1) to gain deeper understanding of how these MSR are derived. The first rule, {A, B} → (S) denotes that A and B have to sends their nonces *Na* and *Nb* respectively to S, and after that S can compute the session key. The second rule, B → (A) denotes that B sends *Nb* to A. The combination of these two rules is the nonce transfer rule. The nonce transfer rules of (KA 1.2) and (KA 1.3) are derived in a similar way. As for the third rule B → (S) → (B) of all MSR listed above, it is the rule that makes B acquire the session key, called the key transfer rule. It should be noted here that in this path S has derived the session key, that is, S has received all nonces. It is also possible for B to obtain the session key from A through indirect key transfer; its MSR is B → (A) → (S) → (B). It is evident that B → (S) → (B) is more efficient, so we choose direct key transfer from S to B in these rules.

However, (KA 1.3) should not be adopted. Because an intruder can replay old nonces to A, which then uses those nonces to compute a session key and accept it. Therefore, for A to make sure the freshness of session key, the input parameters of SK generation equation should contain nonce supplied by A. To be generalized, the communicating parties with the ability to compute session keys on his own should also be nonce contributors.

Similarly, if a session key is computed by *Kbs*, the MSR are listed as follows:

If *SK = f(Kbs, {Na, Nb, IDa}* then

{A, B}→(S) AND A→(B) AND A→(S)→(A) (KA 2.1)

If *SK = f(Kbs, {Nb, Ns, IDa})* then

B→(S) AND S→(B) AND A→(S)→(A) (KA 2.2)

3.4 MSR for Key Confirmation

After acquiring the session key, the communicating parties can either exchange self challenge-response messages or handshake numbers generated by the SK generation equation. They conform to the following MSR:

(A)→B AND (B)→A (KC 1)

Now let us consider how to decide when a party in the KEP may have acquired session keys. In the key generator approach the SK generator has the session key from the beginning of the protocol, and the other parties have the session key after obtaining the session key from the generator via direct or indirect key transfer. In the key agreement approach, the SK computing party has a session key after receiving the enough nonces, and the party that cannot compute session keys has to resort to direct or indirect key transfer from other party that has had a session key.

4. DESIGN OF THREE-PARTY KEPS

4.1 Design by a Session Key Generation Method

Given a session key generation method we can determine the corresponding MSR and building blocks. However, some links may not be available, so some messages should be sent with the aid of the intervening parties. To design a KEP, we have to transform the MSR and the building blocks into messages over the physical links. We will illustrate our ideas by an example of designing a KEP for mobile station depicted in figure 2, where A is the mobile station roaming to the visited domain administrated by station B, and the station S is the administrator of the home domain of A.

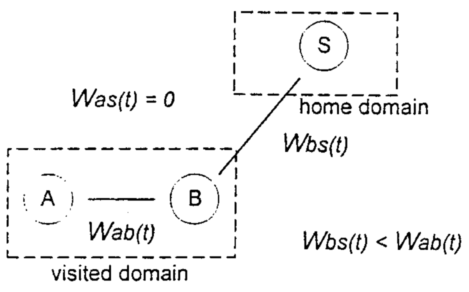

Figure 2. A mobile station roaming to a visited domain

If the session key generation method is key agreement in which the SK generation equation is $SK = f(Kas, \{Na, Ns, IDb\})$, the corresponding MSR is (KA 1.2). A and S need each other's nonce to compute the session key, so the nonce transfer rule and the messages are:

$$A \rightarrow (S) : Na$$

$$S \rightarrow (A) : Ns$$

Because B has to obtain the session key from S through the direct key transfer channel, the key transfer rule and the message are:

$$B \rightarrow (S) : Nb$$

$$(S) \rightarrow (B) : Ns', Psb \oplus SK, f(Kbs, \{SK, Nb, IDb\}),$$

where S should have required Na sent by A.

To have S obtain A's nonce, when A issues a register request to the administrator B in the visited domain, it also sends its nonce Na. Then B sends its own nonce Nb as well as Na to S. Now S can compute the session key and send it to B by the message Ns', $Psb \oplus SK$, $f(Kbs, \{SK, Nb, IDb\})$. Next, to have A obtain S's nonce, S has to send Ns to B, who then forwards that to A. The resulted MSD is depicted in figure 3, where the second phase, key distribution, have been completed.

$$\begin{tikzcd} A \arrow[r, "1"] & B \arrow[r, "2"] & (S) \arrow[r, "3"] & (B) \arrow[r, "4"] & (A) \end{tikzcd}$$

Figure 3. An MSD of mobile station roaming KEP by MSR without key confirmation

To complete the final phase, key confirmation, we choose the self challenge-response scenario. Therefore, the MSR and the messages are listed as follows:

$$(B) \rightarrow A : f(SK, \{IDb\})$$

$$(A) \rightarrow B : f(SK, \{IDa\}$$

The message, $f(SK, \{IDb\})$, can be put in the message 4 in figure 10(a). And after receiving Ns from the message 4, A sends the message, $f(SK, \{IDa\})$, to B. The resulted KEP is a variant of the KEP proposed by Li Gong in [5]. Its MSD is depicted in figure 4 and the messages in the protocol are listed as follows.

Message 1: Na

Message 2: Na, Nb

Message 3: $Ns, Ns', Psb \oplus SK, f(Kbs, \{SK, Nb, IDb\})$

Message 4: $Ns, Nb', f(SK, \{Nb', IDb\})$

Message 5: Na', f(SK, {Na', IDa}),

where $SK = f(Kas, \{Na, Ns, IDb\})$

$$\begin{tikzcd} A \arrow[r, "1"] & B \arrow[r, "2"] & (S) \arrow[r, "3"] & (B) \arrow[r, "4"] & (A) \arrow[r, "5"] & (B) \end{tikzcd}$$

Figure 4. The complete MSD of mobile station roaming KEP by MSR

4.2 Finding Alternatives to the KEP

If the session key generation method is not restricted, we can choose other method to gain more efficiency. Consider if $SK = f(Kas, \{Na, Nb, IDb\}$ and the MSR is (KA 1.1), so the required messages are listed as follows:

$$A \rightarrow S : Na$$

$$B \rightarrow \{S, (A)\} : Nb$$

$$(S) \rightarrow (B) : Ns, Psb \oplus SK, f(Kbs, \{SK, Nb, IDb\})$$

When B obtains Na from A, it sends Nb to A and $\{Na, Nb\}$ to S concurrently so that A and S can compute the SK simultaneously. Then B can obtain SK from S by the message Ns', $Psb \oplus SK$, $f(Kbs, \{SK, Nb, IDb\})$. The resulted MSD is depicted in figure 5. Here we choose the approach of exchanging handshake numbers to do key confirmation, where Ha and Hb denote the handshake numbers.

It can be found that A in figure 5 obtains the SK earlier than in figure 4 so that A can do key confirmation earlier. This KEP is one pass faster than that in figure 4. The messages in the protocol are listed as follows:

Message 1: Na

Message 2: *Na, Nb*

Message 2': *Nb*

Message 3: $Ns, Psb \oplus SK, f(Kbs, \{SK, Nb, IDb\})$

Message 3': *Ha*

Message 4: Hb,

where {SK, Ha, Hb} = f(Kas,{Na, Nb, IDb})

$$\begin{tikzcd}
A \arrow[r, "1"] & B \arrow[r, "2"] \arrow[dr, "2'"] & (S) \arrow[r, "3"] & (B) \arrow[r, "4"] & (A) \\
 & & (A) \arrow[ur, "3'"] & &
\end{tikzcd}$$

Figure 5. An alternative design of mobile station roaming KEP by MSR

5. CONCLUSIONS

In this paper we constructed the building blocks and the message sequence rules according to the analysis of KEPs, by which we transformed the procedure of a KEP into some well-defined modules. Thus, the process of designing a KEP becomes less involved with the details of the message sequence and the message contents. Given the environment of a KEP, one can easily find the proper message sequence by the message sequence rules and the message contents by the corresponding building blocks. Besides, it is also possible to find a more efficient KEP. Finally, because of the systematic design process, the KEPs designed by our method can also be analyzed and understood easily.

In a large network, it may be more efficient to have multiple authentication servers than to have one centralized authentication server. The existing protocols cannot apply to this kind of authentication environment. In [14], we have extended the message sequence rules for three-party KEPs to the generalized message sequence rules so that the KEPs in a generalized authentication environment can be produced systematically by our method, too.

To be more generalized, a communicating party may also provide the service of key establishment for other communicating parties, and the network environment may be changed dynamically. Therefore, the on-line generation of KEPs should be valuable. The possibilities that a computer program can generate a proper KEP given the network and authentication environment deserve further studying. In our method, the message sequence diagram and the message sequence rules can be easily represented in computer graph data structures. Generating message sequence of the protocol is to find if the required paths of the message sequence rules exist in the message sequence diagram. And after that, the messages of the protocol are decided according to the message sequence rules. Therefore, an on-line generator of KEPs based on our systematic design method is quite possible.

6. REFERENCES

[1] Rainer A. Rueppel and Paul C. van Oorschot, "Modern key agreement techniques", Computer Communications vol. 17, no. 7, pp. 458-465, July 1994.

[2] C. Boyd and A. Mathuria, "Systematic design of key establishment protocols based on one-way functions", IEE Proc.-Comput. Digit. Tech., vol. 144, no. 2, pp. 93-99, Mar. 1997.

[3] Shu-Chung Ju, "A systematic method for design of key establishment protocol," Master Thesis, Department of Electrical Engineering, National Taiwan University, 1998.

[4] Moore, J. H., "Protocol failures in cryptosystems", Proc. IEEE, 76, (5), pp. 594-602, 1988.

[5] Needham, R. M., and Schroeder, M. D., "Using Encryption for Authentication in Large Networks of Computers", Commun. ACM, vol. 21, pp. 993-999, Dec. 1978.

[6] Simmons, G. J., "Cryptanalysis and protocol failures", Communication ACM, 37, (11), pp.56-65, 1994..

[7] 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, vol. 3, no. 1, pp. 31-41, Feb. 1995.

[8] L. Gong, "Using one-way functions for authentication", ACM CCR, vol. 19, no. 5, pp. 8-11, Oct. 1989.

[9] M. Clayton, GSM Global System for Mobile Communication. Security Domain Pty Ltd. (CAN no: 003823461) (1991)

[10]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, California, May 1994.

[11]R. Rivest, "The MD5 Message Digest Algorithm", Internet RFC 1321, 1992.

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

[13]R. C. Merkle, "A Fast Software One-way Hash Function", Journal of Cryptology, vol.3, no. 1, pp. 43-58, 1990.

[14]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. Sel. Areas Commun., vol. 11, pp. 679-693, 1993.