A Secure and Efficient Protocol of Multiple Session Keys Generation

Chang-Kuo Yeh
Department of Information Management
National Taichung Institute of Technology
129 Sanmin Rd., Sec. 3, Taichung,
Taiwan , R.O.C.

yehlin@ntit.edu.tw

Abstract: - To negotiate a session key can benefit from the technique of the public-key cryptography such as key exchange and digital signature. This key cannot be used until the session is finished. In current model such as SSL, if the sender desires to re-establish another new session with the same receiver, the both sides, for security consideration, should repeat the same key exchange and digital signature processes to generate a new session key. Apparently, the two processes must lower the communication system efficiency. Therefore, a new session key generation protocol is proposed to overcome this demerit. In our new model, the key exchange and digital signature process only perform once, and the subsequent session keys can be computed in both sides without the two processes. Because the two processes can be eliminated, our protocol can promote more performance than the current-used model.

Key-Words: - Public-key cryptography, SSL, Authentication, Key Agreement

1 Introduction

The purpose of the session key generation is to assure communication security between sender and receiver. The sender deliver the message encrypted by the session key to the receiver and the receiver can decrypts it by the same session key, so the session key generation should be finished before communication.

To negotiate a session key can benefit from the technique of the public-key cryptography such as key exchange [1] and digital signature [2]. In most practical implementations, public-key cryptography is used to authentication and negotiation of session keys; those session keys are used with symmetric algorithms to secure message traffic. So if based on public-key cryptography, the session key generation process should include two phases; the first is key exchange process: two sides cooperate to negotiate a session key; the other is mutual authentication process: two sides authenticate each other by using the technique of digital signature to make sure the session key is not forged.

In current-used model such as SSL handshake protocol [15], while the session key is generated, it cannot be used until the session is finished. If the two parties desire to communicate each other again, the session key generation process should restart to generate a new session key [4,8]. This restart process wastes time and bandwidth and increases computational load for both sides, so many proposed protocols called Multiple-Key agreement protocols [3,6,7,9,10,11,12,13,14] are designed to generate several session keys in one round of session key generation process. For example, in [3], it can generate four shared secret keys in one round of the session key generation process. These keys can be used in four sessions for the same sender and receiver. The session key generation process restarts only after the four session keys are used, so the protocol can reduce overall computational load and increase the communicational efficiency.

Basically, our protocol inherits the same idea of the proposed protocols but further more efficient than them, because the new model can generate more secret session keys, theoretically no limitation, in one round of session key generation process without increasing too much computation load for both sides. Next section, SSL handshake protocol is described and the Multiple-Key agreement protocol [14] is described in Section 3. We describe our model in Section 4 and analyses our scheme to be secure in Section 5. In Section 6, we compare our scheme with current-used SSL model and the Multiple-Key agreement protocols to show the performance of our scheme is the best among them. Finally, the conclusions are given in Section 7.

2 SSL Handshake protocol

The SSL protocol [15] uses a combination of public-key and symmetric key encryption. Symmetric key encryption is much faster than public-key encryption, but public-key encryption provides better authentication techniques. An SSL session always begins with an exchange of messages called the *SSL handshake*. The handshake allows the server to authenticate itself to the client using public-key techniques, then allows the client and the server to cooperate in the creation of a symmetric key used for rapid encryption, decryption, and tamper detection during the session that follows. Optionally, the handshake also allows the client to authenticate itself to the server. The steps involved can be summarized as follows

1. The client sends the server the client's SSL version number, cipher settings, randomly generated data, and other information the server needs to communicate with the client using SSL.

2. The server sends the client the server's SSL version number, cipher settings, randomly generated data, and other information the client needs to communicate with the server over SSL. The server also sends its own certificate and, if the client is requesting a server resource that requires client authentication, requests the client's certificate.

3. The client uses some of the information sent by the server to authenticate the server. If the server cannot be authenticated, the user is warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server can be successfully authenticated, the client goes on to Step 4.

4. Using all data generated in the handshake so far, the client (with the cooperation of the server, depending on the cipher being used) creates the premaster secret for the session, encrypts it with the server's public key (obtained from the server's certificate, sent in Step 2), and sends the encrypted premaster secret to the server.

5. If the server has requested client authentication (an optional step in the handshake), the client also signs another piece of data that is unique to this handshake and known by both the client and server. In this case the client sends both the signed data and the client's own certificate to the server along with the encrypted premaster secret.

6. If the server has requested client authentication, the server attempts to authenticate the client. If the client cannot be authenticated, the session is terminated. If the client can be successfully authenticated, the server uses its private key to decrypt the premaster secret, then performs a series of steps (which the client also performs, starting from the same premaster secret) to generate the master secret.

7. Both the client and the server use the master secret to generate the *session key,* which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session and to verify its integrity-that is, to detect any changes in the data between the time it was sent and the time it is received over the SSL connection.

8. The client sends a message to the server informing it that future messages from the client will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the client portion of the handshake is finished.

9. The server sends a message to the client informing it that future messages from the server will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the server portion of the handshake is finished.

10. The SSL handshake is now complete, and the SSL session has begun. The client and the server use the session keys to encrypt and decrypt the data they send to each other and to validate its integrity.

It's important to note that both client and server authentication involves encrypting some piece of data with one key of a public-private key pair and decrypting it with the other key:

In the case of server authentication, the client encrypts the premaster secret with the server's public key. Only the corresponding private key can correctly decrypt the secret, so the client has some assurance that the identity associated with the public key is in fact the server with which the client is

connected. Otherwise, the server cannot decrypt the premaster secret and cannot generate the symmetric keys required for the session, and the session will be terminated.

In the case of client authentication, the client encrypts some random data with the client's private key-that is, it creates a digital signature. The public key in the client's certificate can correctly validate the digital signature only if the corresponding private key was used. Otherwise, the server cannot validate the digital signature and the session is terminated.

According the above description, one session key can generalized in one round of handshake protocol. If another session key is needed in both sides, the handshake protocol must be restarted. In other word, the digital signature and key exchange process must repeat to generate a new session key. Obviously, these restart processes will lower the whole performance

3 Review of Multiple-Key agreement protocol

Diffie-Hellman key exchange technique is applied to the Multiple-Key agreement protocol. So, the Diffie-Hellman key exchange technique should be described before introducing the Multiple-Key agreement protocol.

Diffie-Hellman key exchange, also called exponential key exchange, is a method of digital encryption that uses numbers raised to specific powers to produce decryption keys on the basis of components that are never directly transmitted, making the task of a would-be code breaker mathematically overwhelming.

To implement Diffie-Hellman, the two end users Alice and Bob, while communicating over a channel they know to be private, mutually agree on positive whole numbers p and q, such that p is a prime number and q is a generator of p. The generator q is a number that, when raised to positive whole-number powers less than p, never produces the same result for any two such whole numbers. The value of p may be large but the value of q is usually small.

Once Alice and Bob have agreed on p and q in private, they choose positive whole-number personal keys a and b, both less than the prime-number modulus p. Neither user divulges their personal key to anyone; ideally they memorize these numbers and do not write them down or store them anywhere. Next, Alice and Bob compute public keys a^* and b^* based on their personal keys according to the formulas

$$a^* = q^a \bmod p$$

and

$$b^* = q^b \bmod p$$

The two users can share their public keys a^* and b^* over a communications medium assumed to be insecure, such as the Internet or a corporate wide area network (WAN). From these public keys, a number x can be generated by either user on the basis of their own personal keys. Alice computes x using the formula

$$x = (b^*)^a \bmod p$$

Bob computes x using the formula

$$x = (a^*)^b \bmod p$$

The value of x turns out to be the same according to either of the above two formulas. However, the personal keys a and b, which are critical in the calculation of x, have not been transmitted over a public medium. Because it is a large and apparently random number, a potential hacker has almost no chance of correctly guessing x, even with the help of a powerful computer to conduct millions of trials. The two users can therefore, in theory, communicate privately over a public medium with an encryption method of their choice using the decryption key x.

The most serious limitation of Diffie-Hellman in its basic or "pure" form is the lack of authentication. Communications using Diffie-Hellman all by itself are vulnerable to man in the middle attacks. Ideally, Diffie-Hellman should be used in conjunction with a recognized authentication method such as digital signatures to verify the identities of the users over the public communications medium. Diffie-Hellman is well suited for use in data communication but is less often used for data stored or archived over long periods of time.

Inherited from Diffie-Hellman technique, the Multiple-Key agreement [14] is introduced.

The system authority publishes a large prime p and a primitive element g with order $p-1$ in $GF(p)$. We assume that A and B want to establish four secret keys in a protocol round. Long term public/private key pairs for A and B are (y_A, x_A) and (y_B, x_B), where

$$y_A = g^{x_A} \bmod p$$
$$y_B = g^{x_B} \bmod p .$$

"*mod p*" will be omitted in this section for easily described. We assume that long term public keys are exchanged via certificates, where Cert_A denotes *A's* public key certificate, containing a string of information that uniquely identifies A, her static public key y_A and a certifying authority CA's signature over this information. The protocol runs as follows:

1. A selects two random integers k_{A1} and K_{A2}, called short term secret keys, compute short term public keys

$$r_{A1} = y_B^{K_{A1}}$$
$$r_{A2} = y_B^{K_{A2}}$$

such that $0 < r_{A1}, r_{A2} < (p-1)/2$. Then A computes

$$r_{A1} = g^{k_{A1}+k_{A2}}$$

and generates its signature s_A on $\{r_{A1}, r_{A2}\}$ as follows:

$$s_A = x_A \cdot r_A - (r_{A1} + r_{A2}) \cdot (k_{A1} + k_{A2}) \bmod p-1.$$

Next, A sends the authenticated messages $\{r_{A1}, r_{A2}, s_A, Cert_A\}$ to B.

2. Similarly, B also chooses k_{B1} and K_{B2} and computes

$$r_{B1} = y_A^{k_{B1}},$$
$$r_{B2} = y_A^{k_{B2}},$$
$$r_B = y_B^{k_{B1}+k_{B2}}$$
$$s_B = x_B \cdot r_B - (r_{B1} + r_{B2}) \cdot (k_{B1} + k_{B2}) \bmod p-1.$$

Then B sends $\{r_{B1}, r_{B2}, s_B, Cert_B\}$ to A.

3. After receiving the message from B, A computes $r_B = (r_{B1} \cdot r_{B2})$ and verifies B's signature by checking

$$y_B^{r_B} = r_B^{(r_{B1}+r_{B2})} \cdot g^{s_B}.$$

If its verification holds, A computes four common secret keys as follows:

$$K_1 = r_{B1}^{x_A^{-1}k_{A1}} = g^{k_{A1}k_{B1}},$$
$$K_2 = r_{B1}^{x_A^{-1}k_{A2}} = g^{k_{A2}k_{B1}},$$
$$K_3 = r_{B2}^{x_A^{-1}k_{A1}} = g^{k_{A1}k_{B2}},$$
$$K_4 = r_{B2}^{x_A^{-1}k_{A2}} = g^{k_{A2}k_{B2}}.$$

4. Similarly, B verifies A's signature by checking the verification equation

$$y_A^{r_A} = r_A^{(r_{A1}+r_{A2})} \cdot g^{s_A}.$$

Finally, B also computes four common secret keys as follows.

$$K_1 = r_{A1}^{x_B^{-1}k_{B1}} = g^{k_{A1}k_{B1}},$$
$$K_2 = r_{A2}^{x_B^{-1}k_{B1}} = g^{k_{A2}k_{B1}},$$
$$K_3 = r_{A1}^{x_B^{-1}k_{B2}} = g^{k_{A1}k_{B2}},$$
$$K_4 = r_{A2}^{x_B^{-1}k_{B2}} = g^{k_{A2}k_{B2}}.$$

4 Our model

Our scheme includes four phases: Initialization, Authentication, The first session key generation and the subsequent session keys generation. The technique of the subsequent session keys generation phase is the key point of our scheme since it eliminates the key exchange process and has an effective mutual authentication process by using the technique of SAS protocol [5] instead of digital signature technique. And it carries as follows.

4.1 Initialization

Let *p* be a large prime number (1024 bits) and *g* be a generator for Z_p^*. *A* randomly selects x_A as the private key and calculates

$$y_A = g^{x_A} \bmod p$$

as the corresponding public key. *B* randomly selects x_B as the private key and calculates

$$y_B = g^{x_B} \bmod p$$

as the corresponding public key.

4.2 Authentication

Step 1

A sends y_A to *B* and *B* sends y_B to *A*. Upon receiving y_B from *B*, *A* generates two random number k_A and t_{A_1} and calculates

$$r_A = y_B^{k_A} \bmod p$$
$$m_{A_1} = h^2(t_{A_1})$$
$$s_A = x_A * m_{A_1} + r_A * k_A \bmod p\text{-}1,$$

where *h(.)* is a one way hash function and $h^2(t_{A_1})$ implies executes *h(.)* twice. Upon receiving y_A from *A*, *B* generates two random number k_B and t_{B_1} and calculates

$$r_B = y_A^{k_B} \bmod p$$
$$m_{B_1} = h^2(t_{B_1})$$
$$s_B = x_B * m_{B_1} + r_B * k_B \bmod p\text{-}1$$

Step 2

A sends r_A, s_A and m_{A_1} to *B* and *B* sends r_B, s_B and m_{B_1} to *A*. While receiving the messages from *B*, *A* verifies the messages by checking

$$g^{s_B} = y_B^{m_{B_1}} * [g^{k_B}]^{r_B} \bmod p$$

If the equation doesn't hold, reject the request; otherwise the request is valid that implies *B* is authenticated by *A* and only *A* can authenticate *B* since

$$g^{k_B} = (r_B)^{x_A^{-1}} \bmod p$$

and x_A^{-1} can only be calculated by *A*, where

$$x_A * x_A^{-1} \bmod p = 1.$$

On the other hand, while receiving the messages from *A*, *B* verifies the messages by checking

$$g^{s_A} = y_A^{m_{A_1}} * [g^{k_A}]^{r_A} \bmod p$$

If the equation doesn't hold, reject the request; otherwise the request is valid that implies *A* is authenticated by *B* and only *B* can authenticate *A* since

$$g^{k_A} = (r_A)^{x_B^{-1}} \bmod p$$

and x_B^{-1} can only be calculated by *B*, where

$$x_B * x_B^{-1} \bmod p = 1.$$

The authentication process is illustrated in Figure 1.

4.3 The first session key generation

g^{k_A}, g^{k_B} and $g^{k_A k_B}$ are shared by *A* and *B* after the authentication phase. A polynomial

$$f(x) = ax^2 + bx + c$$

can only be constructed by *A* and *B*, where

$$a = g^{k_A} \bmod p$$

$$b = g^{k_B} \bmod p$$

$$c = g^{k_A k_B} \bmod p .$$

The first session key is

$$C_1 = f(0) = c$$

The first session key generation process is illustrated as figure 2.

4.4 The subsequent session keys generation

This phase can be executed repeatedly to generate more session keys without executing the former phases such as Initialization, Authentication and First session key generation phase.

A randomly generates a random number t_{A_i} and calculates

$$h^2(t_{A_i}) = m_{A_i}$$

where *i* is an integer larger than *1*. *A* sends $[h(t_{A_{i-1}}), m_{A_i}]C_{i-1}$ to B, where $[h(t_{A_{i-1}}), m_{A_i}]C_{i-1}$ denotes message $h(t_{A_{i-1}})$ *and* m_{A_i} encrypted by key C_{i-1}, and *B* verifies the messages by checking whether

$$h(h(t_{A_{i-1}})) = m_{A_{i-1}}$$

If the equation doesn't hold, reject the request; otherwise the request is valid and the i_{th} session key

$$C_i = f(h(t_{A_{i-1}}))$$

is generated on both sides. Then, *B* updates $m_{A_{i-1}}$ to m_{A_i}.

On the other hand, *B* can also start the same process. *B* randomly generates t_{B_i} and calculates

$$h^2(t_{B_i}) = m_{B_i},$$

where *i* is an integer larger than *1*. *B* sends $[h(t_{B_{i-1}}), m_{B_i}]C_{i-1}$ to *A* and *A* verifies the messages by checking whether

$$h(h(t_{B_{i-1}})) = m_{B_{i-1}}$$

If the equation doesn't hold, reject the request; otherwise the request is valid and the i_{th} session key

$$C_i = f(h(t_{B_{i-1}}))$$

is generated on both sides. Then, *A* updates $m_{B_{i-1}}$ to m_{B_i}. The subsequent session keys generation can be organized as figure 3.

5 Security analysis

We prove our model is secure according to the each step of the whole process.

5.1 Initialization

Both parties generate their own private key and corresponding public key internally. No message is transmitted out; the opponent cannot learn any knowledge from this step.

5.2 Authentication Step 1

In this step, *A* and *B* exchange public keys each other in clear. It is difficult to calculate the private key only knowing the public key since the opponent is forced to take a discrete logarithm to determine the key, so it is useless to only intercept the public key for the attacker.

5.3 Authentication Step 2

A sends r_A, s_A and m_{A_1} to B where

$$r_A = y_B^{k_A} \bmod p\ ,\ m_{A_1} = h^2(t_{A_1})$$

$$s_A = x_A * m_{A_1} + r_A * k_A \ \bmod p\text{-}1.$$

k_A, t_{A_1} and x_A must be kept secret in this step. It is difficult to calculate k_A only knowing r_A and y_B for the opponent because it is the problem of discrete logarithm. It is hard to get $h(t_{A_1})$ and t_{A_1} from m_{A_1} since $h(.)$ is a one way hash function. In the equation

$$s_A = x_A * m_{A_1} + r_A * k_A \ \bmod p\text{-}1$$

s_A, m_{A_1} *and* r_A are public and x_A *and* k_A are kept secret. It is hard to get x_A *or* k_A from the equation

$$s_A = x_A * m_{A_1} + r_A * k_A \ \bmod p\text{-}1$$

since there are two variables in one equation.

On the other hand, B sends r_B, s_B and m_{B_1} to A. To get k_B, t_{B_1} and x_B are also difficult according the same analyses. After that, user A and B authenticate each other based on the equation

$$g^{s_B} = y_B^{m_{B_1}} * [g^{k_B}]^{r_B} \bmod p \text{ and}$$

$$g^{s_A} = y_A^{m_{A_1}} * [g^{k_A}]^{r_A} \bmod p\ .$$

If these equations hold that implies that A and B are the legal users and they keep the same secret values such as $g^{k_A} \bmod p, g^{k_B} \bmod p, g^{k_A k_B} \bmod p$.

5.4 The first session key generation

$$g^{k_B} = (r_B)^{x_A^{-1}} \bmod p$$

can only be derived by A since x_A^{-1} can only be calculated by A, where

$x_A * x_A^{-1} \bmod p = 1$

For the same reason,

$$g^{k_A} = (r_A)^{x_B^{-1}} \bmod p$$

can only be derived by B since x_B^{-1} can only be calculated by B, where

$$x_B * x_B^{-1} \bmod p = 1$$

In such a way, a polynomial

$$f(x) = ax^2 + bx + c$$

can only be constructed by A and B, where

$$a = g^{k_A} \bmod p$$

$$b = g^{k_B} \bmod p$$

$$c = g^{k_A k_B} \bmod p\ .$$

Very clearly, nobody can compute the first session key

$$C_1 = f(0) = c$$

except A and B.

5.5 The subsequent session keys generation

A generates the message $[h(t_{A_{i-1}}), m_{A_i}]C_{i-1}$ and sends it to B. B decrypts the message to get m_{A_i} which is equal to $h^2(t_{A_i})$, where t_{A_i} is a random number selected by A. It is very difficult to compute $h(t_{A_i})$ according to $h^2(t_{A_i})$, since $h(.)$ is a one way hash function which is relatively easy to compute, but significantly harder to reverse. If any attacker tries to replay this message to pass the authentication process, he cannot succeed since B will find out that the value $h(h(t_{A_{i-1}}))$ doesn't equal to m_{A_i} which is updated to $h^2(t_{A_i})$. If any attacker tries to forge this message to pass the authentication process, he cannot succeed since the message is encrypted by a session key C_i which is different from time to time and only known to A and B.

For security considerations, it is not reasonable to do this process all the time. Hence, a predefined constant n should be set to a reasonable constraint on the times to do the subsequent session keys generation process. In other words, after n times of the subsequent session keys generation process are used, the authentication process and the first session key generation process should restart again.

6 Performance evaluation

In this section, SSL protocol and the Multiple-Key agreement protocols will be compared with our scheme to show our scheme has the best efficiency among them.

Initialization and Authentication phases are needed in the three compared protocols. Therefore, the pubic key cryptography is needed in the SSL protocol, the Multiple-Key protocols and our new protocol. The pubic key cryptography is a very time-consumed computation. So, in the two phases, the computation load is heavy in the three compared protocols.

In SSL protocol, only one session key is generated after the two phases. Repeated authentication process is needed to generate another new session key while another new session is connected. In the Multiple-Key agreement protocols, four or eight session keys are generated after the two phases. It implies that executing one

initialization and authentication process supports four or eight sessions. The subsequent session key generation process must restarts the authentication process after the four or eight session keys are used, so the protocols can reduce overall computational load and increase the communicational efficiency compared with SSL protocol.

In our protocol, no session key is generated after the two phases. Even our protocol needs extra phase, called the first key generation phase, to generate the first session key after the two phases. It seems our protocol has the worst efficiency. As a matter of fact, the key point is that our new protocol has the subsequent session keys generation protocol, which is not found in SSL and Multiple-Key schemes. This phase can generate more session keys, theoretically no limitation, in one round of authentication process without increasing too much computation load for both sides. For example, in the subsequent session keys generation, only some simple computations such as hash computation and secret-key cryptography are needed. These computation loads is very light compared to the public-key cryptography.

In the SSL and Multiple-key schemes, they have to repeat the authentication process to perform the subsequent key generation process. For example, if 100 session keys are needed, it is necessary to repeat 99 times and 24 times authentication processes for SSL and Multiple-key schemes, respectively. Hence, the computation load is very heavy since pubic key cryptography is applied many times. Table 1 shows the comparison results between the three protocols. Obviously, we can find the new model can benefit the best performance although it has extra first key generation process.

7 Conclusion

Many Multiple-Key agreement protocols are designed to generate several session keys in one round of session key generation process. In these protocols, public-key cryptography is needed to initiate key exchange and mutual authentication process. However, when these session keys are used, the second round of session key generation process must start again. The restart process is vulnerable to reapply public-key cryptography for key exchange and mutual authentication. Obviously, the computation load is heavy. In our new model, only one round of session key generation process is necessary and the subsequent session keys can be computed in both sides without any key exchange process. In other words, the public-key cryptography only performs once. Because the public-key cryptography could be eliminated, our protocol can promote more performance than current-used models.

References:

[1] W. DIFFIE and M.E. HELLMAN, New Directions in Cryptography, IEEE Transactions on Information Theory, Vol. IT-22, No. 6, 1976, pp. 644-654.

[2] T. ELGAMAL, A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, IEEE Transactions on Information Theory, Vol. IT-31, No. 4, 1985, pp.469-47.

[3] L. Harn and H. Y. LIN, Authenticated Key Agreement Without Using One-way Hash functions, ELECTRONIC LETTERS 10th, Vol. 37 No. 10, May, 2001, pp. 620-630.

[4] J. H. Park and S. B. Lim, Key Distribution for Secure VSAT Satellite Communications, IEEE Transactions on Broadcasting, Vol. 44, No. 3, Sep, 1998.

[5] M. SANDIRIGAMA, A. SHIMIZU and M. T. NODA, Simple and Secure Password Authentication Protocol, IEICE Transactions on Communication, Vol. E83-B, No. 6, June, 2000, pp. 1363-1365.

[6] T. S. WU, W. H. HE, and C. L. HSU, Security of Authenticated Multiple-Key Agreement Protocols, ELECTRONIC LETTERS, Vol.35 No. 5, 1999, pp.391-392.

[7] S.M. YEN and M. JOYE, Improved Authenticated Multiple-Key Agreement Protocol, ELECTRONIC LETTERS, Vol. 34 No. 18, 1998, pp1738-1739.

[8] SSL Version 3.0, March 1996, http://wp.netscape.com/eng/ssl3/ssl-toc.html

[9] H. T. Yeh, H. M. Sun, Tzonelih Hwang, Improved Authenticated Multiple-Key Agreement Protocol, Computers and Mathematics with Applications, Vo. 46, Issue: 2-3, 2003, pp. 207-211.

[10] C. C. Yang and R. C. Wang, Cryptanalysis of improved authenticated multiple-key agreement protocol without using conventional one-way function, Applied Mathematics and Computation, Vol. 162, Issue: 1, March 4, 2005, pp. 211-214.

[11] Kyungah Shim and Sungsik Woo, Weakness in ID-based one round authenticated tripartite multiple-key agreement protocol with pairings, Vol. 166, Issue: 3, July 26, 2005, pp. 523-530.

[12] C. C. Chang, H. C. Tsai and Y. C. Chiu, A Simple and Robust Authenticated Multiple Key Agreement Scheme, Security Technology, 2008. SECTECH '08. International Conference on 13-15, Dec. 2008, pp. 214-218.
[13] C. C. Chang, H. C.n Tsai and S.Y. Lin, A token free password authentication scheme with multiple key agreements, Communications and Networking in China, 2008. ChinaCom 2008. Third International Conference on 25-27 Aug. 2008, pp. 1143-1150.
[14] Z. Shao, Security of robust generalized MQV key agreement protocol without using one-way hash functions, Computer Standards and Interfaces, Vol. 25 No. 5, 2003 pp. 431–436.
[15] Mozilla Developer Center, Introduction to SSL https://developer.mozilla.org/en/Introduction_to_SSL

Table 1 The performance comparisons between the three protocols if 100 session keys are needed

	SSL Scheme	Multiple-Key scheme	New scheme
Initialization	Public-key system	Public-key system	Public-key system
Authentication	Public-key system	Public-key system	Public-key system
First key generation	no	no	yes
Subsequent key generation	restart 99 times authentication processes	restart 24 times authentication processes	not to restart authentication process

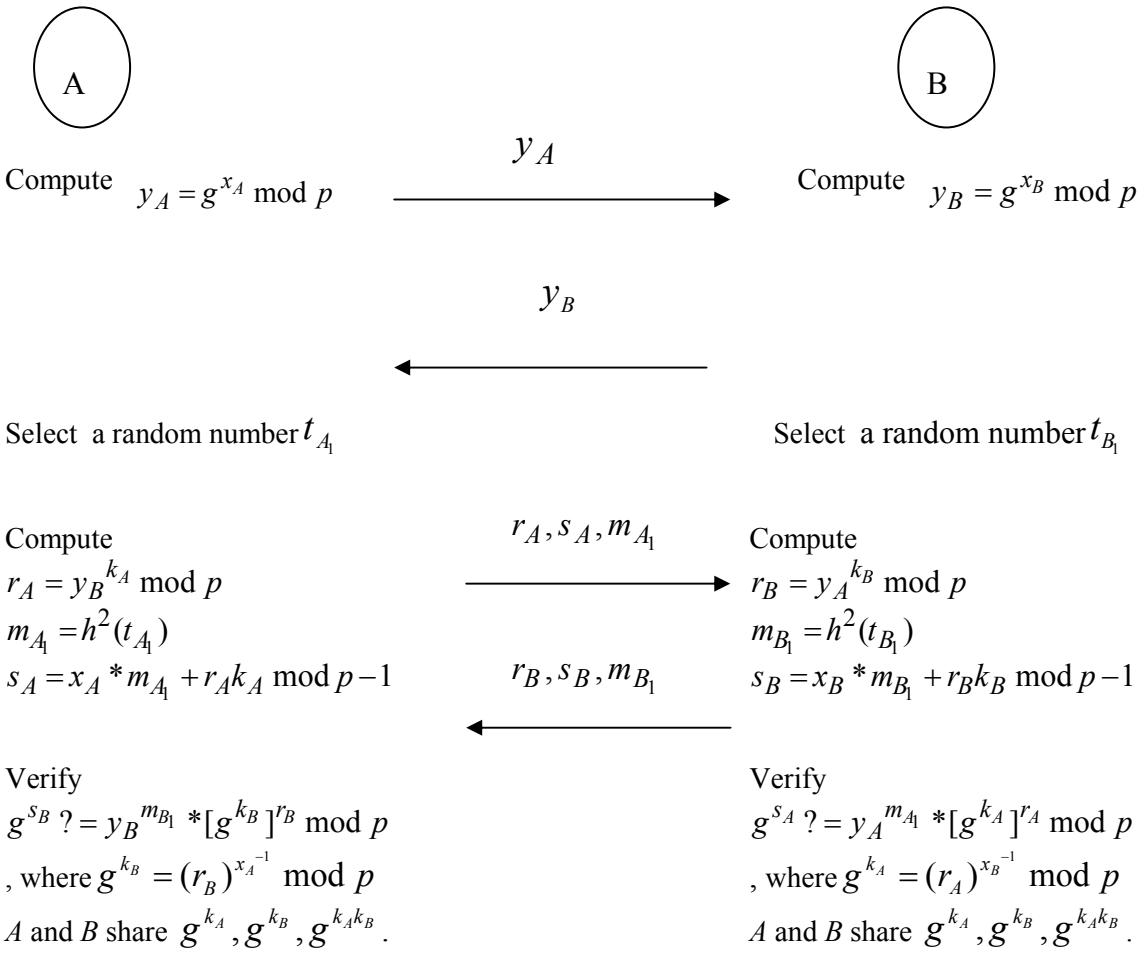

Fig. 1 The authentication phase

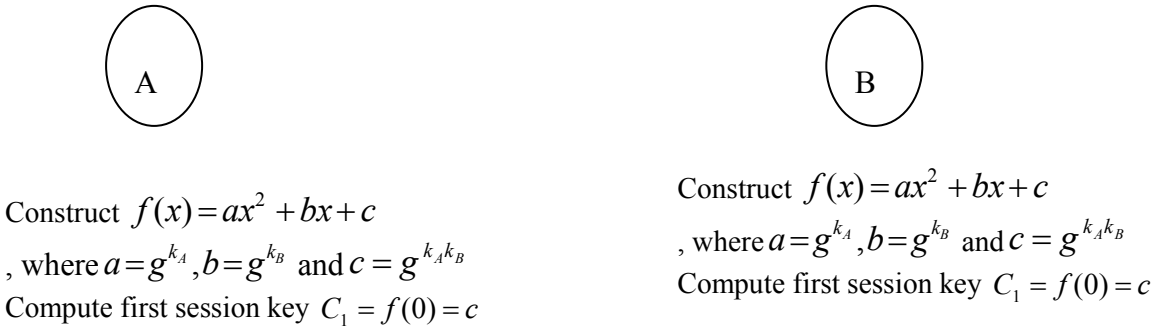

Fig. 2 The first session key generation

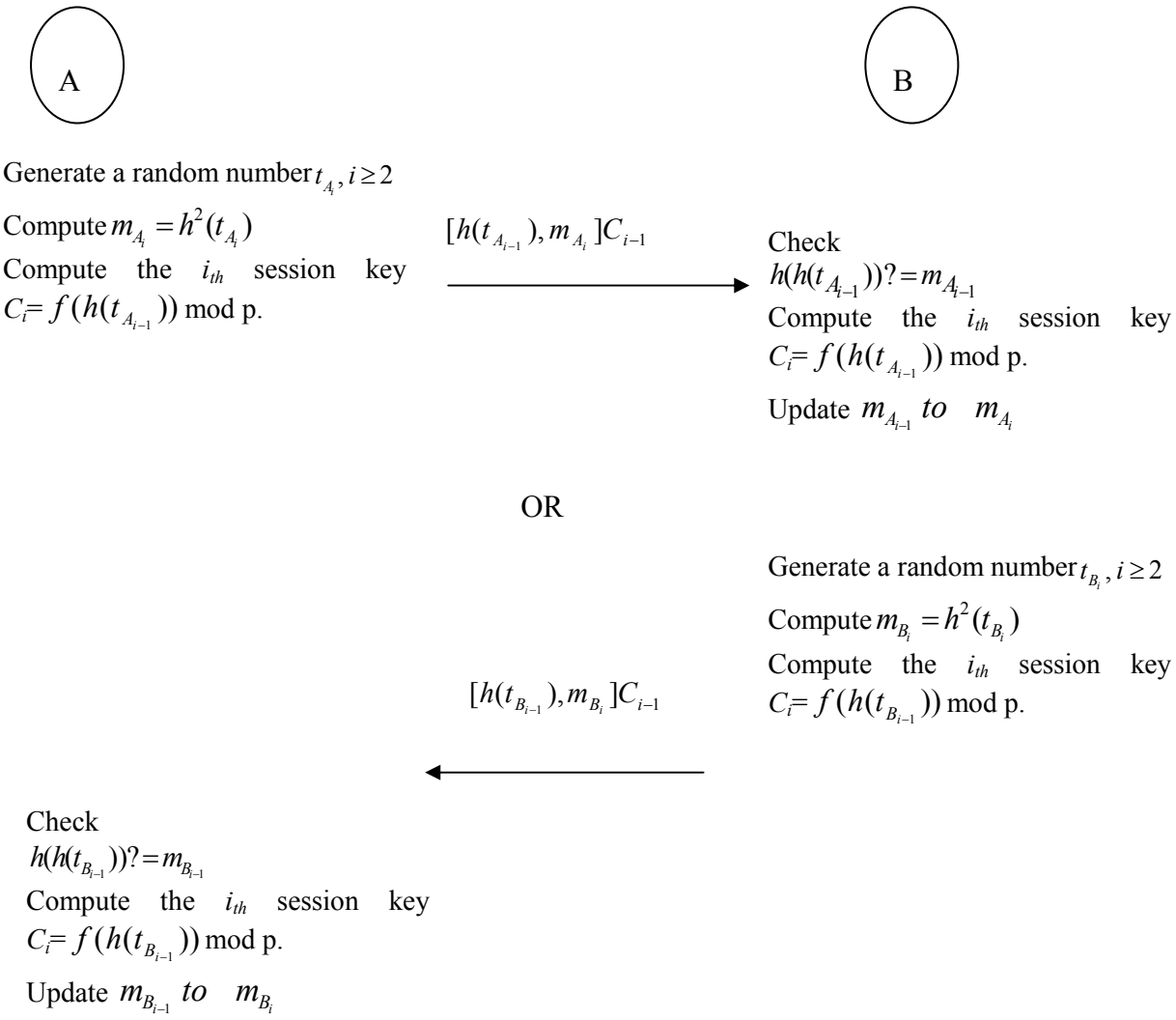

Fig. 3 The subsequent session keys generation