2002

Master's thesis

A One-Time Password Authentication Method

1055124 Takasuke TSUJI

Advisor Akihiro SHIMIZU

January 31, 2003

Course of Information Systems Engineering,

Graduate School of Engineering,

Kochi University of Technology

Abstract

A One-Time Password Authentication Method

Takasuke TSUJI

Applications for transferring money or personal information are increasingly common on the Internet and in mobile communications. These applications require user authentication for confirming legal users. One-time password authentication methods change the verifier every time by sending the present verifier along with the next verifier. However, such methods risk attacks because those protocols use two verifiers every session. The SAS (Simple And Secure password authentication protocol) is a one-time password authentication method that uses a hash function five times, but it requires high overhead on low spec machines. In this paper, I propose a new method, SAS-2, which reduces hash function adaptation overhead by 40%. This method has a mutual authentication phase which maintains synchronous data communications in its authentication procedure. Moreover, SAS-2 can be applied to key-free systems.

key words cryptography, hash function, one-way function, password authentication, one-time password

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The Internet and mobile communications have been developing, and related applications for managing money or personal information are increasing in number. However, there is a risk that such private data can be wiretapped. Therefore, it is necessary to authenticate users. If a user sends the same password every session, an attacker can masquerade as the user, because the attacker can get the user's password via the Internet. So, the user requires one-time password authentication methods that change the verifier every time.

Those techniques are guaranteed secure by the use of a one-way function with which it is easy to compute $f(x)$ from x and difficult to compute x such that $y = f(x)$. Usually hash functions such as MD5 (Message Digest 5) [6], SHA-1 (Secure Hash Algorithm 1) [7], [8] or RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest 160) [9] are employed for this purpose. In the same manner, with a one-way function it is easy to compute $g(s,t)$ from s and t, but difficult to compute s such that $y = g(s,t)$ from y and t. Usually common-key cryptosystems such as DES (Data Encryption Standard) [1], FEAL (Fast Data Encryption Algorithm) [2], MISTY (Mitsubishi Electric's Information Security Technology) [3], Rijndael [4], [5] be AES (Advanced Encryption Standard) are employed for this purpose.

Public-key cryptosystems are one-way functions, too. The K1P method [10], [11]

and the OKGS method [12] are one-time password authentication methods. Those methods apply public-key cryptosystems and hash functions. However, public-key encryptions have high calculation costs in comparison with hash or common-key encryptions. In this thesis, I set forth one-time password authentication methods that apply hash functions or common-key cryptosystems.

1.2 Existing Studies

When a user logs in to the system, the user sends masking data to the server, and the server certifies the user using those masking data and the stored verifier. Then the user and the server use a one-time password authentication method apply a one-way function.

Lamport's method [19] is a one-time password authentication method, and uses a one-way function, but this method has two practical difficulties: high hash overhead and the requirement of resetting the verifier. The CINON (chained one-way data verification method) [20], [21] has solved those problems, but that method has another problem in that the user must memorize a random number. The PERM (Privacy Enhanced information Reading and writing Management method) [23] eliminates this problem by generating and sending a random number from the server. A. Shimizu *et al.* have detected a possible security flaw in both CINON and PERM methods: a kind of 'Man in the Middle' attack that modifies the authentication data by tapping the communication data in two consecutive sessions. To counter this, M. Sandirigama *et al.* propose the SAS (Simple And Secure) password authentication protocol [24], which eliminates the 'Man in the Middle' attack and reduces storage, processing, and transmission overhead. However, the method suffers from vulnerability to both replay

and denial of service attacks [25].

C. L. Lin *et al.* have argued that the OSPA (Optimal Strong-Password Authentication) [25] method and the revised SAS [26] method have solved those security problems, but C. -M. Chen *et al.* found a stolen-verifier problem on those methods [28]. This attacker has to steal the verifier from the server, but the attack will be unsuccessful if the server has strong management.

The authors have performed successful impersonation attacks on the OSPA method [29]. In this attack, the attacker intercepts the authentication data from the Internet and impersonates the user using the previous verifiers. Therefore, the only secure protocol with a one-time password method for changing verifiers infinitely is the revised SAS.

1.3 Statememt Problems

The revised SAS method uses a one-way function five times. This function has high overhead, because a one-way function apply hash functions or common-key cryptosystems. It's desirable to reduce the number of times this function is used in the revised SAS method, because authentication methods are now being applied to mobile communications and Internet protocols. The revised SAS and other methods are useless for low spec machines such as mobile phones.

It is possible that communications on the Internet can be interrupted and become asynchronous. In that case, an authentication method has to maintain the security of the system, but previous authentication methods have coped with those problems.

1.4 Objective

The Internet and mobile communications have been developing, and related applications for managing money or personal information have to authenticate the user. However, one-time password authentication methods use one-way functions extensively. So, I reduce the times a one-way function is used.

Moreover, a synchronous data communication procedure is possible for one-time password authentication methods. I realized mutual authentication using a one-time password method.

1.5 Study Scope and Contents

In this thesis, I propose a new method, Simple And Secure password authentication protocol, ver.2 (SAS-2) [27], applying a one-way function only three times by using two verifiers and another for masking. This reduces hash overhead by about 40% in comparison with the revised SAS. SAS-2 and other methods generate a random number, but this operation has high overhead such as the calculation of a one-way function. In consideration of that problem, I appended a procedure which can use a verifier instead of a random number. In addition, a synchronous data communication procedure is possible for one-time password authentication methods. The SAS-2 method solved this problem using a mutual authentication protocol.

The SAS-2 method has variations, and I have considered all variations of the SAS-2 and produced safe combinations. Moreover, I examined problems in the SAS-2 protocol and determined the most secure combinations remaining [30]. In addition I suggest here an application for key-free systems using the SAS-2 method. In addition, I have written a paper about this thesis, and have submitted it to a learned society.

In chapter 2, secure one-time password methods are explained. Those methods have some problems, which I discuss here. In chapter 3, SAS-2 protocol is explained. As well, I illustrate the SAS-2 protocol using a challenge response method. In chapter 4, security and performance analysis of SAS-2 is discussed. Security is proven by elimination of existing attacks. Performance evaluations are shown for some costs. Moreover, I discuss mutual authentication. The SAS-2 has algorithm variations. In the chapter 5, I show how to make such algorithm variations, and lay out preconditions for secure variations. Moreover, I discuss defective combinations of SAS-2, and show 7 strong variations. In chapter 6, applications apply SAS-2 are presented.

Chapter 2

Secure One-Time Password Authentication Protocols

Secure one-time password authentication methods are Lamport's method and the revised SAS method. In this chapter, those two procedures are explained.

2.1 Definitions and Notations

The following definitions and notations are used throughout in this chapter.

- User is a computer user who employs the protocol for authentication.
- Server is the server that authenticates users.
- ID is the user's identity.
- x and P are user's passwords.
- M is the maximum number of hash iterations (ranging from about 500 to 1000).
- i is an integer indicating the number of authentication sessions.
- N_i represents a random number corresponding to the ith authentication.
- F and E are one-way hash functions. For example, $F(x)$ means x is hashed once, E_i^n means x is hashed nth using the randome number for the ith verifier.
- $\oplus$ represents a bitwise XOR operation.

2.2 Lamport's Protocol

The mechanism of Lamport's method consists of two phases: the registration phase and the authentication phase. The registration process is performed only once, and the authentication procedure is executed every time the user logs in to the system. These two phases are described below.

2.2.1 Registration Phase of Lamport's Protocol

Figure 2.1 shows the initial registration phase of Lamport's method.

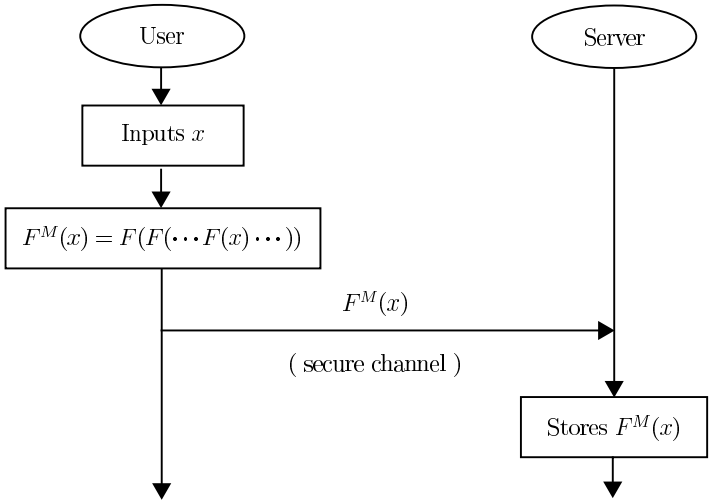

Fig. 2.1 Registration phase of Lamport's method

1. *The user inputs own password and calculates the verifier for the* 1*st authentication session.* The user inputs the user's password x and sets M when registering in the systems; then calculates $F^M(x)$ equals $F(F(\cdots F(x) \cdots))$ using the password x.

2. *The user sends the registration data to the server for subsequent authentication.* The user sends $F^M(x)$ to the server through a secure channel.

3. *The server stores the verifier for subsequent authentication.* The server stores $F^M(x)$ for subsequent authentication.

2.2.2 Authentication Phase of Lamport's Protocol

To log in, the user executes the ith authentication session of Lamport's protocol. Then the server stores the verifier $F^{M-i+1}(x)$. Figure 2.2 shows the ith authentication phase of the SAS-2 protocol.

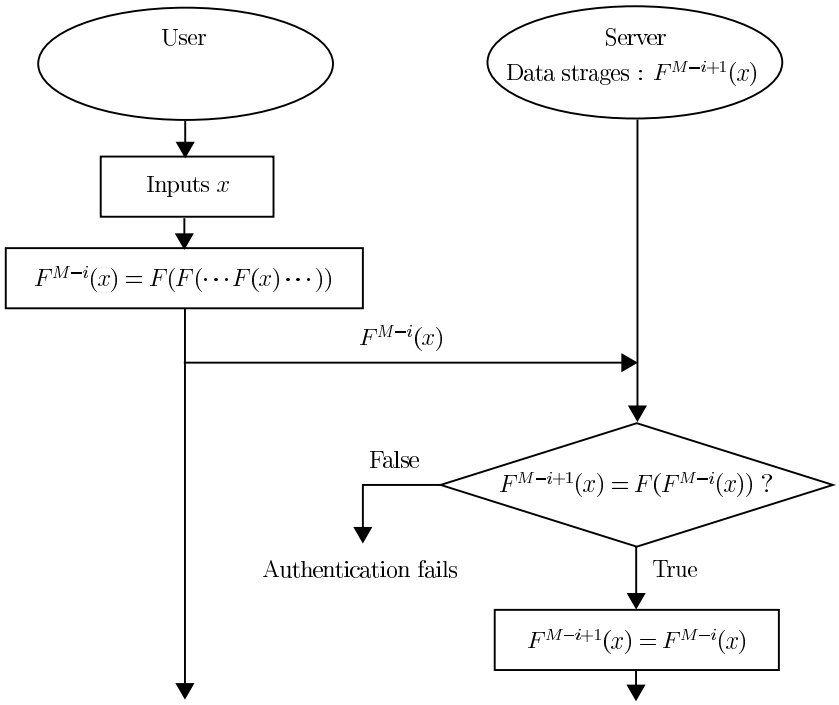

Fig. 2.2 The ith authentication phase of Lamport's method

1. *The user inputs own password and calculates authentication data for the ith authentication session.* The user inputs the user's password P when joining the system. Then the user calculates $F^{M-i}(x)$ equals $F(F(\cdots F(x)\cdots))$ using input data.

2. *The user sends the authentication data to be authenticated by the server.* The user sends $F^{M-i}(x)$ to the server through a common network such as the Internet.

3. *The server authenticates the user with the stored verifier and the transmission data.* The server compares the verifier $F^{M-i+1}(x)$ and the computed $F(F^{M-i}(x))$. Then if they don't match, the user is rejected and the server refuses the user. If they do match, the user is authenticated and the next process is executed.

4. *The server stores the verifier for the next authentication session.* The server stores $F^{M-i}(x)$ in place of $F^{M-i+1}(x)$ for the next authentication session.

2.2.3 Considerations of Lamport's protocol

Lamport's method is a simple procedure, but the user must use a one-way hash function many times in every authentication session. This technique is useless because the method costs a lot of loads and times. Moreover, the user has to register after the Mth authentication session. The registration step is troublesome to the user.

Those problems are solved by changing the verifier every session.

2.3 The Revised SAS Protocol

Lamport's method has two practical difficulties: high hash overhead and the requirement of resetting the verifier. Some one-time password authentication methods have solved such problems. However, most of those methods have security problems. The revised SAS is the only surviving one-time password authentication method which can change verifiers without limit.

The mechanism of the revised SAS method consists of two phases: the registration phase and the authentication phase. The registration process is performed only once, and the authentication procedure is executed every time the user logs in to the system. These two phases are described below.

2.3.1 Registration Phase of the Revised SAS Protocol

Figure 2.3 shows the initial registration phase of the revised SAS protocol.

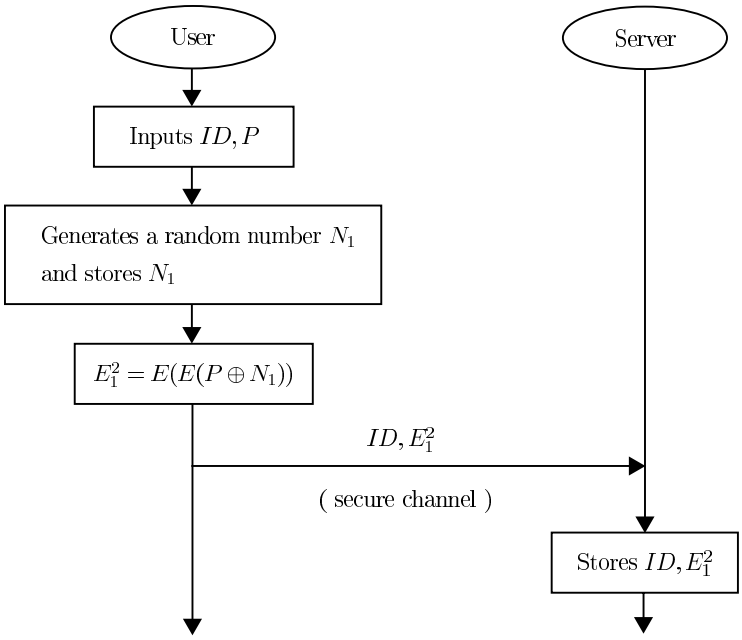

Fig. 2.3 Registration phase of the revised SAS

1. *The user inputs own personal data and calculates the verifier for the* 1*st authentication session.* The user inputs the user's identity ID and password P when registering in the systems; then generates and stores a random number N_1, and calculates E_1^2 equals $E(E(P \oplus N_1))$ using the random number and input data.

2. *The user sends the registration data to the server for subsequent authentication.* The user sends ID and E_1^2 to the server through a secure channel.

3. *The server stores the verifier for subsequent authentication.* The server stores E_1^2 with ID for subsequent authentication.

2.3.2 Authentication Phase of the Revised SAS Protocol

To log in, the user executes the ith authentication session of the revised SAS protocol. Then the user stores N_i, and the server stores the verifier E_i^2 with ID. Figure 2.4 shows the ith authentication phase of the revised SAS protocol.

1. *The user inputs own personal data and calculates two data for the ith authentication session.* The user inputs the user's identity ID and password P when joining the system. Then the user calculates E_i^1 equals $E(P \oplus N_i)$ and E_i^2 equals $E(E_i^1)$ using input data and the stored N_i. Next the user generates a random number N_{i+1} and stores N_{i+1}. Then the user calculates E_{i+1}^2 equals $E^2(P \oplus N_{i+1})$ and E_{i+1}^3 equals $E(E_{i+1}^2)$ using the random number N_{i+1} and input data, where E_{i+1}^2 is the next verifier. Next the user computes α equals $E_{i+1}^2 \oplus E_i^2$ and β equals $E_i^1 \oplus E_{i+1}^3$.

2. *The user sends the authentication data to be authenticated by the server.* The user sends ID, α, and β to the server through a common network such as the Internet.

3. *The server authenticates the user with the stored verifier and the transmission data.* The server retrieves E_{i+1}^2 from $\alpha \oplus E_i^2$ with the stored verifier E_i^2 and gets E_i^1 from $\beta \oplus E_{i+1}^3$ using $E(E_{i+1}^2)$. Next the server compares the computed $E(E_i^1)$

and the stored E_i^2. If they don't match, the attempt is rejected and the server refuses the user. If they do match, the user is authenticated and the next process is executed.

4. *The server stores the verifier for the next authentication session.* The server stores E_{i+1}^2 in place of E_i^2 for the next authentication session.

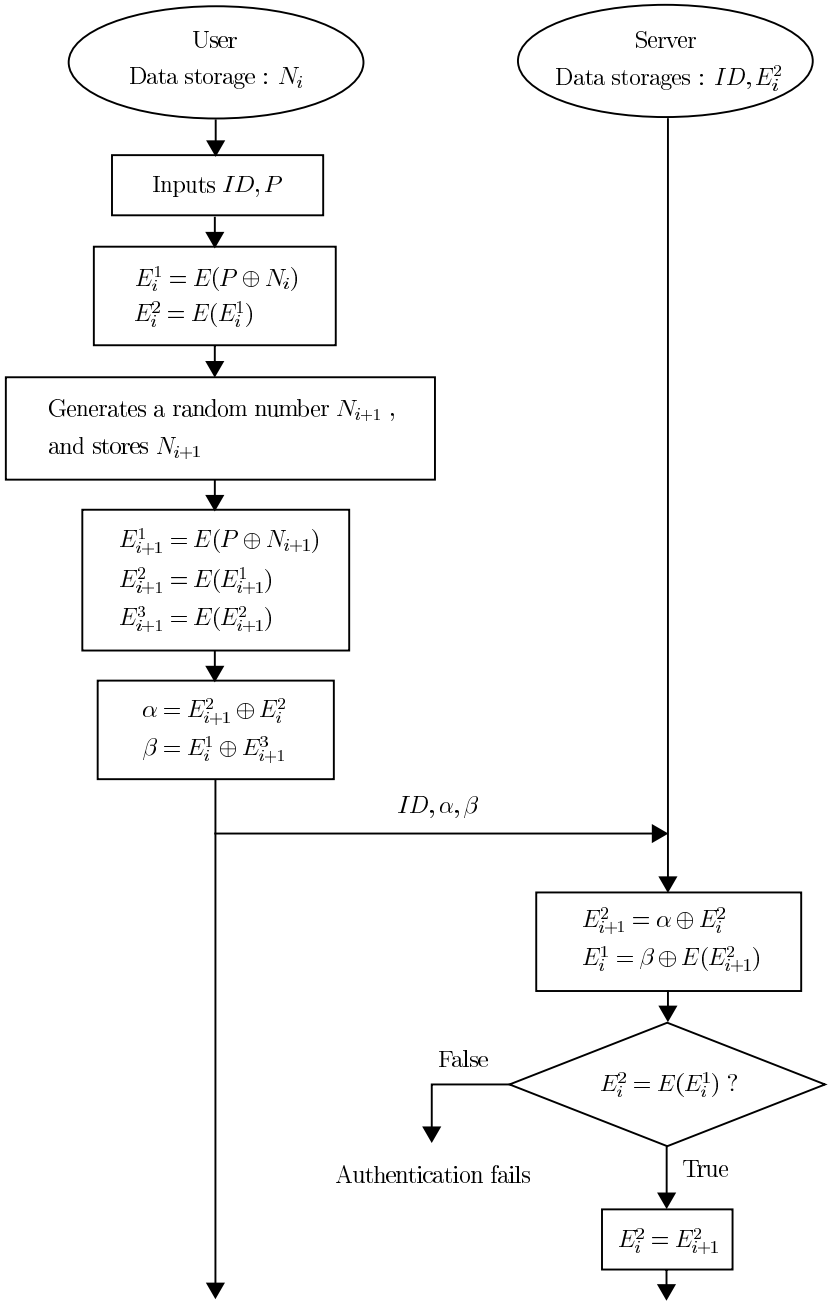

Fig. 2.4 The ith authentication phase of the revised SAS

2.3.3 Considerations of the Revised SAS protocol

The revised SAS method uses a one-way function five times. This function has high overhead, because a one-way function apply hash functions or common-key cryptosystems. It's desirable to reduce the number of times this function is used in the revised SAS method, because authentication methods are now being applied to mobile communications and Internet protocols. The revised SAS and other methods are useless for low spec machine such as mobile phones.

In the revised SAS method, a one-way hash function is used five times to generate the authentication data. The functions of each encryption are the following in the ith authentication session.

1. E_i^1 is used to authenticate the user on the ith authentication session.
2. E_i^2 is the verifier of the ith authentication session, and this verifier is used to remove the masking data on this authentication session.
3. E_{i+1}^1 is used to authenticate the user on the next authentication session.
4. E_{i+1}^2 is the verifier of the next authentication session.
5. E_{i+1}^3 is used for masking the authentication data.

When the server authenticates the user on the ith authentication session, the E_i^1 is necessary because the server verifies the legal user by the hashed E_i^1. The user has to make the E_{i+1}^2, because the E_{i+1}^2 is used on the next authentication session. Before the E_{i+1}^2 is calculated, the user must calculate the E_{i+1}^1. Therefore, the E_{i+1}^2 is needful, too. Moreover, the E_{i+1}^3 is necessary for masking the authentication data.

The E_i^2 is maybe needless, because the user can substitute the other data for the E_i^2. Moreover, the user can reduce the hash overhead by using epochal new method.

Chapter 3

New Protocol

The SAS-2 method can change verifiers every time and without limit. Moreover, SAS-2 applyies its function only three times by using two verifiers and another for masking. This reduces hash overhead by about 40% in comparison with the revised SAS. In addition, SAS-2 has a synchronous data communication procedure.

3.1 Definitions and Notations

The following definitions and notations are used throughout this thesis.

- User is a computer user who employs the protocol for authentication.
- Server is the server that authenticates users.
- ID is the user's identity.
- P is the user's password.
- X, F and H are one-way hash functions. For example, $H(x)$ means x is hashed once.
- i is an integer indicating the number of authentication sessions.
- N_i represents a random number corresponding to the ith authentication.
- $+$ represents the addition operation.
- $\oplus$ represents a bitwise XOR operation.

3.2 SAS-2 Protocol

The SAS-2 protocol consists of two phases: the registration phase and the authentication phase. The registration process is performed only once, and the authentication procedure is executed every time the user logs in to the system. These two phases are described below.

3.2.1 Registration Phase of the SAS-2 Protocol

Figure 3.1 shows the initial registration phase of the SAS-2 protocol.

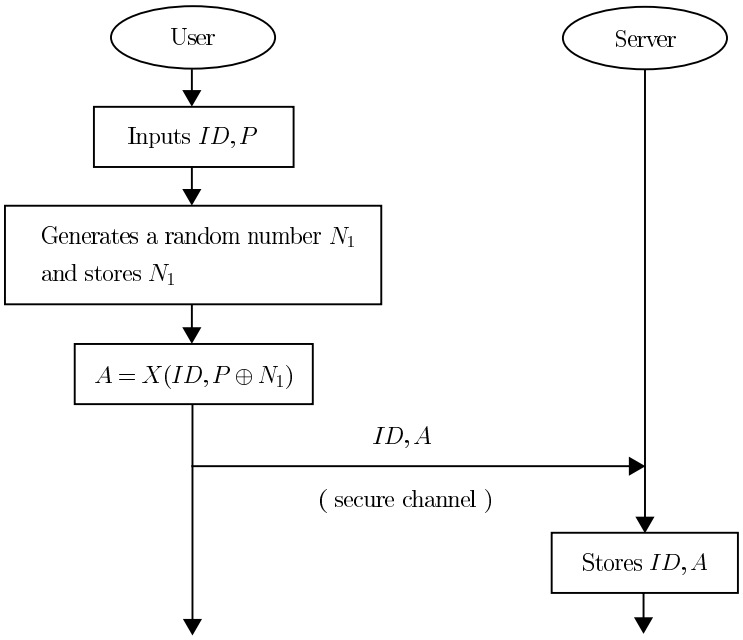

Fig. 3.1 Registration phase of the SAS-2

1. *The user inputs own personal data and calculates the verifier for the* 1*st authentication session.* The user inputs the user's identity ID and password P when registering in the systems; then generates and stores a random number N_1, and calculates A equals $X(ID, P \oplus N_1)$ using the random number and input data.

2. *The user sends the registration data to the server for subsequent authentication.* The user sends ID and A to the server through a secure channel.

3. *The server stores the verifier for subsequent authentication.* The server stores A with ID for subsequent authentication.

3.2.2 Authentication Phase of the SAS-2 Protocol

To log in, the user executes the ith authentication session of the SAS-2 protocol. Then the user is storing N_i, and the server stores the verifier A with ID. Figure 3.2 shows the ith authentication phase of the SAS-2 protocol.

1. *The user inputs own personal data and calculates two data for the ith authentication session.* The user inputs the user's identity ID and password P when joining the system. Then the user calculates A equals $X(ID, P \oplus N_i)$ using input data and the stored N_i. Next the user generates a random number N_{i+1} and stores N_{i+1}. At that time, the user can use A instead of N_{i+1}. Then the user calculates $C = X(ID, P \oplus N_{i+1})$ and $F(C) = F(ID, C)$ using the random number N_{i+1} and input data, where C is the next verifier. Next the user computes $\alpha = C \oplus (F(C) + A)$ and $\beta = F(C) \oplus A$. Then the user can substitute other combinations for α and β.

2. *The user sends the authentication data to be authenticated by the server.* The user sends ID, α, and β to the server through a common network such as the Internet.

3. *The server authenticates the user with the stored verifier A and the transmission data.* The server retrieves $F(C)$ from $\beta \oplus A$ with the stored verifier A and gets C from $\alpha \oplus (F(C) + A)$ using $F(C)$. Next the server compares $F(C)$ and the computed $F(ID, C)$. If they don't match, the attempt is rejected and the server refuses the user. If they do match, the user is authenticated and the next process is executed.

4. *The server stores the verifier for the next authentication session and calculates the data for a mutual authentication protocol.* The server stores C in place of A for the next authentication session, and calculates γ equals $H(ID, F(C))$.

5. *The server sends the authentication data to be authenticated by the user.* The server sends γ to the user through a common network.

6. *The user authenticates the server with the transmission data and the data which was used in the user's authentication.* The user calculates $H(ID, F(C))$ and compares $H(ID, F(C))$, which is transmission data γ. If they match, the server is authenticated. If they don't match, the user tries to log in again.

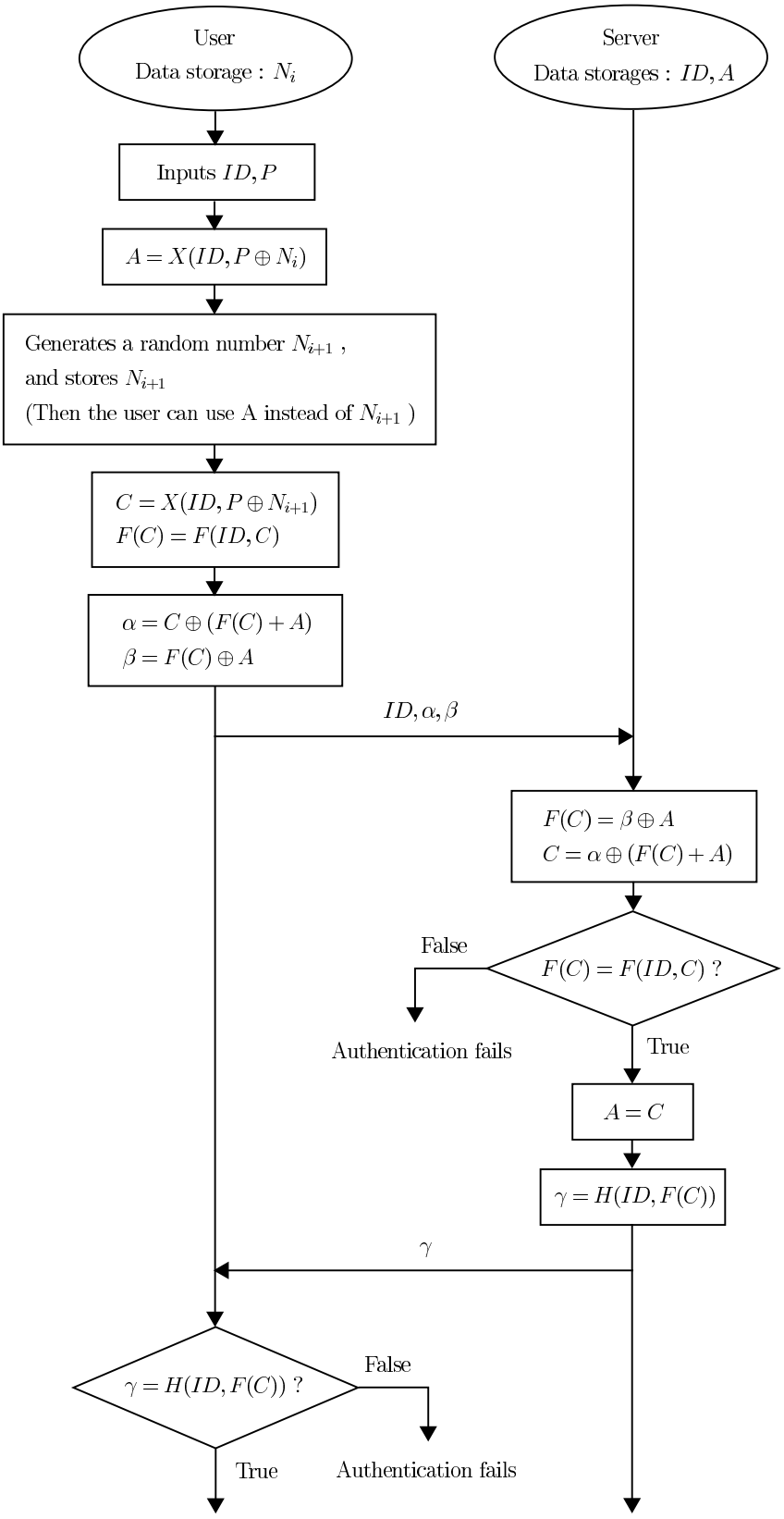

Fig. 3.2 The ith authentication phase of the SAS-2

3.3 SAS-2 Protocol Using a Challenge Response Method

If user can't store any data, the system can use the SAS-2 protocol using a challenge response method. In this procedure, the user need not store the random number, but transmission iterations are increased.

3.3.1 Registration Phase Using a Challenge Response Method

Figure 3.3 shows the initial registration phase of the SAS-2 protocol using a challenge response method.

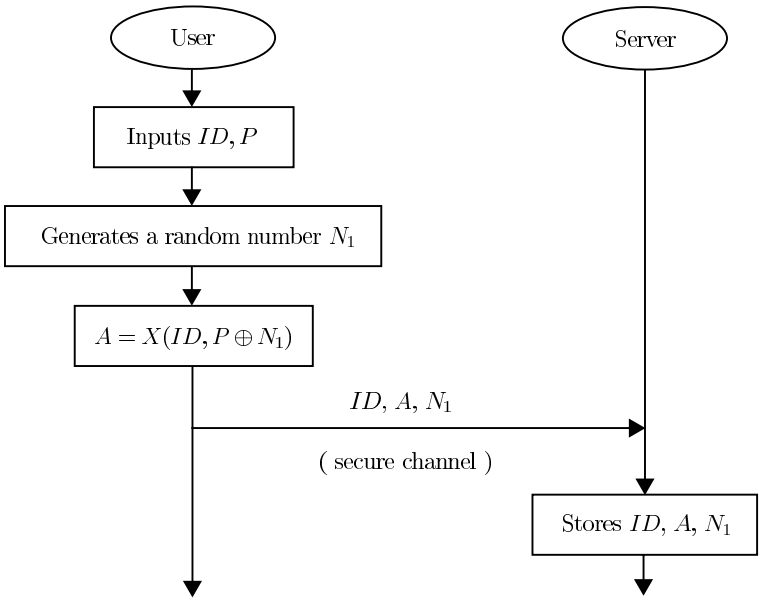

Fig. 3.3 Registration phase using a challenge response method

1. *The user inputs own personal data and calculates the verifier for the* 1*st authentication session.* The user inputs the user's identity ID and password P when registering in the systems; then generates a random number N_1, and calculates A

equals $X(ID, P \oplus N_1)$ using the random number and input data.

2. *The user sends the registration data to the server for subsequent authentication.* The user sends ID, A, and N_1 to the server through a secure channel.

3. *The server stores the verifier for subsequent authentication.* The server stores A and N_1 with ID for subsequent authentication.

3.3.2 Authentication Phase Using a Challenge Response Method

To log in, the user executes the ith authentication session of the SAS-2 protocol using a challenge response method. Then the user is storing N_i, and the server stores the verifier A with ID. Figure 3.4 shows the ith authentication phase of the SAS-2 protocol using a challenge response method.

1. *The user inputs own personal data and sends the service request to log in.* The user inputs the user's identity ID and password P when joining the system. Next the user sends the service request along with ID to the server.

2. *The server receives the seed to the user.* The server receives the random number N_i to the user.

3. *The user calculates two data for the ith authentication session.* The user calculates A equals $X(ID, P \oplus N_i)$ using input data and the stored N_i. Next the user generates a random number N_{i+1}. At that time, the user can use A instead of N_{i+1}. Then the user calculates $C = X(ID, P \oplus N_{i+1})$ and $F(C) = F(ID, C)$

using the random number N_{i+1} and input data, where C is the next verifier. Next the user computes $\alpha = C \oplus (F(C) + A)$ and $\beta = F(C) \oplus A$. Then the user can substitute other combinations for α and β.

4. *The user sends the authentication data to be authenticated by the server.* The user sends α, β, and N_{i+1} to the server through a common network such as the Internet.

5. *The server authenticates the user with the stored verifier A and the transmission data.* The server retrieves $F(C)$ from $\beta \oplus A$ with the stored verifier A and gets C from $\alpha \oplus (F(C) + A)$ using $F(C)$. Next the server compares $F(C)$ and the computed $F(ID, C)$. If they don't match, the attempt is rejected and the server refuses the user. If they do match, the user is authenticated and the next process is executed.

6. *The server stores the verifier for the next authentication session and calculates the data for a mutual authentication protocol.* The server stores C and N_{i+1} in place of A and N_i for the next authentication session, and calculates γ equals $H(ID, F(C))$.

7. *The server sends the authentication data to be authenticated by the user.* The server sends γ to the user through a common network.

8. *The user authenticates the server with the transmission data and the data which was used in the user's authentication.* The user calculates $H(ID, F(C))$ and compares $H(ID, F(C))$, which is transmission data γ. If they match, the server is authenticated. If they don't match, the user tries to log in again.

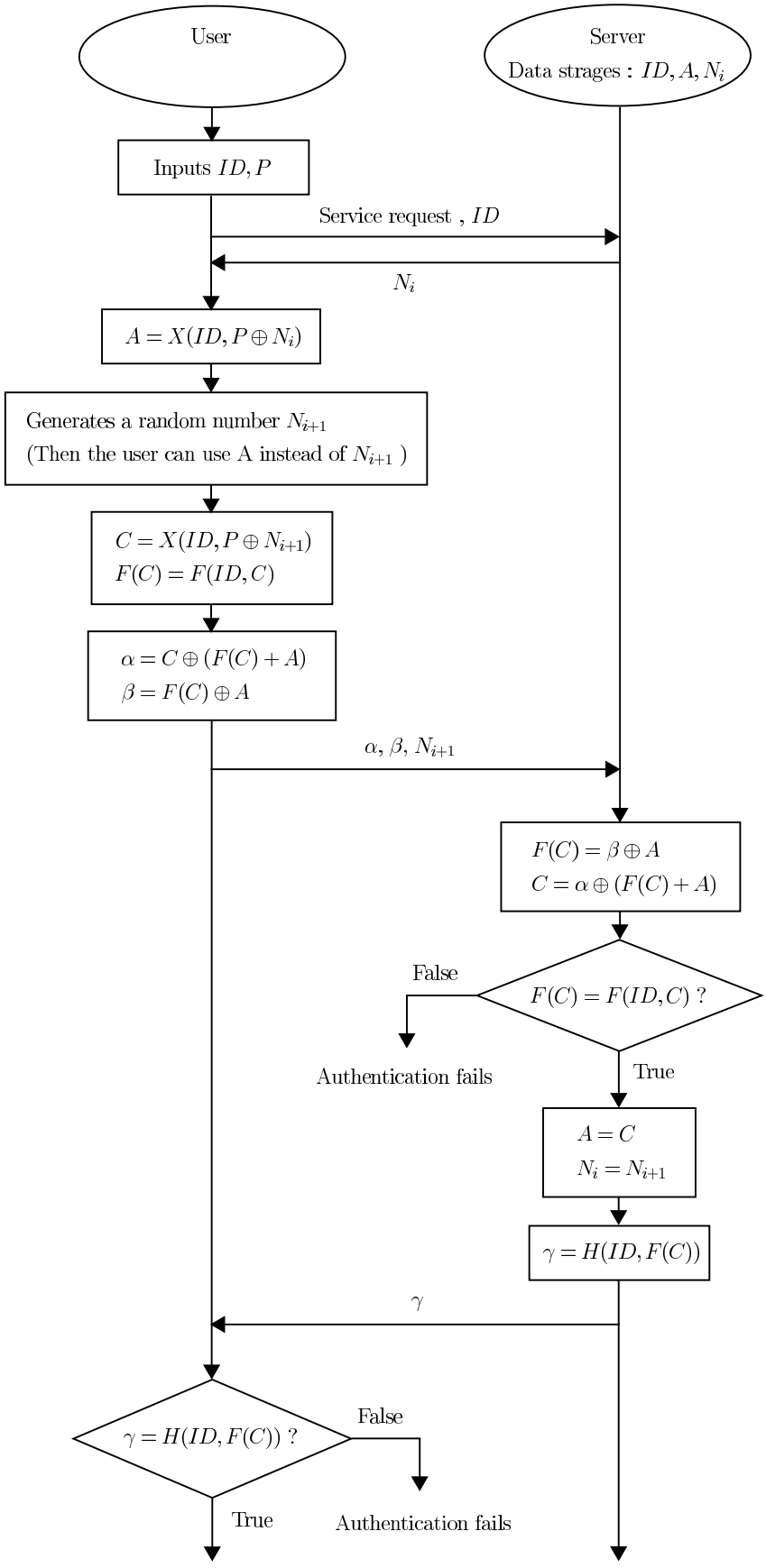

Fig. 3.4 The ith authentication phase using a challenge response method

3.4 Protocol Considerations

The SAS-2 protocol eliminates hash overhead by using a radical new method.

The theories of existing one-time password authentication methods are two kinds: procedures such as Lamport's method and techniques such as the SAS method. Figure 3.5 shows such a procedure:

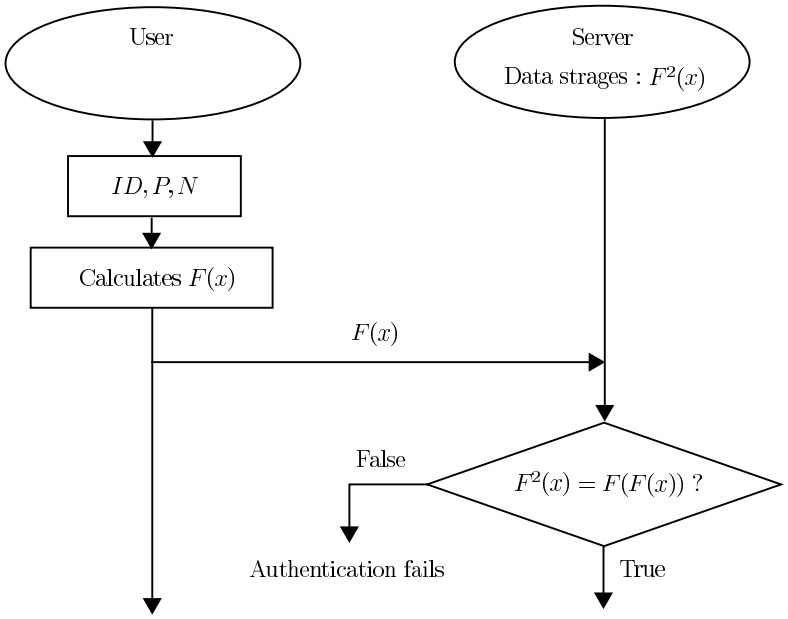

Fig. 3.5 An Lamport type procedure

1. The user calculates the authentication data $F(x)$ with the user's identity number ID, the user's password P, and a random number N.
2. The user send the authentication data $F(x)$ to the server.
3. The server compares the stored verifier $F^2(x)$ and the calculated $F(F(x))$.

This process is simple, but it can't change the verifier adinfinitum. This problem is solved by the user sending two verifiers at the same time. Figure 3.6 shows an SAS

type procedure which eliminates such problem ($U(s)$ using a masking function using $\oplus$ and $+$).

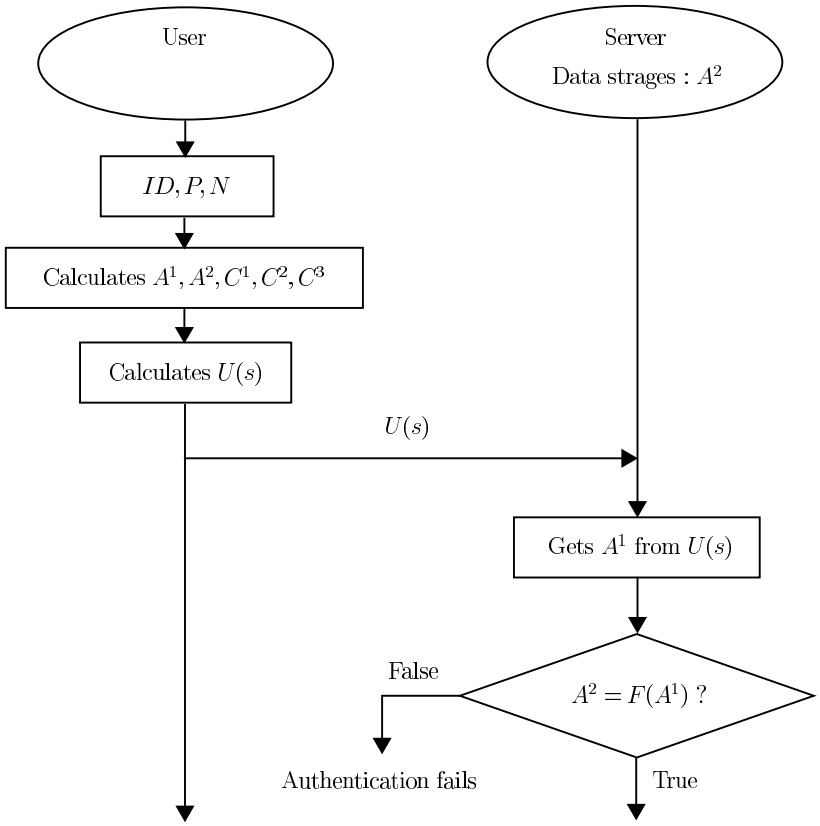

Fig. 3.6 An SAS type procedure

1. The user calculates the authentication data A^1, A^2, C^1, C^2, C^3 with the user's identity number ID, the user's password P, and a random number N.
2. The user calculates the masking data $U(s)$ with the authentication data.
3. The user send the masking data $U(s)$ to the server.
4. The server gets A^1 using the stored verifier A^2.
5. The server compares the stored verifier A^2 and the calculated $F(A^1)$.

This procedure can change the verifier adinfinitum, but uses a one-way function a lot on low spec machines. Figure 3.7 shows an SAS-2 type procedure which solves such cost as below.

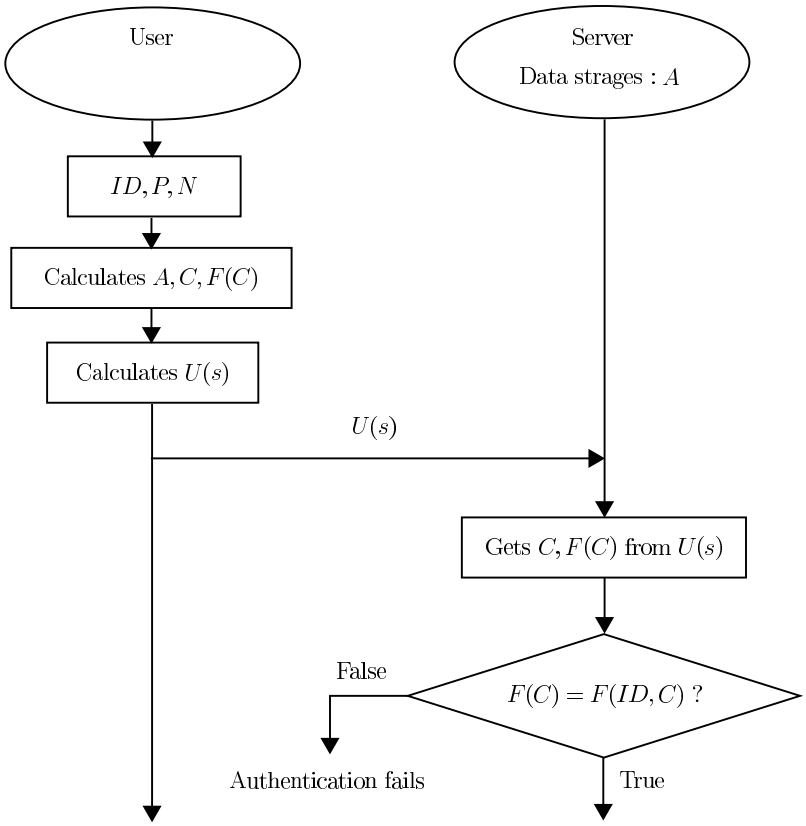

Fig. 3.7 An SAS-2 type procedure

1. The user calculates the authentication data $A, C, F(C)$ with the user's identity number ID, the user's password P, and a random number N.
2. The user calculates the masking data $U(s)$ with the authentication data.
3. The user send the masking data $U(s)$ to the server.
4. The server gets C and $F(C)$ using the stored verifier A.
5. The server compares the retrieved $F(C)$ and the calculated $F(ID, C)$.

In existing protocols, the user is authenticated when the server compares the stored verifier and the retrieved and hashed data using the authentication data and the verifier. In the new method proposed us here, the user is authenticated when the server compares the retrieved data and the retrieved and hashed data using the verifier and the authentication data. In this way, the number of hash iterations are reduced.

Moreover, the SAS-2 protocol can authenticate the server using the authentication data. If the system doesn't need such a function, the SAS-2 can remove it.

Chapter 4

Security and Performance Analysis

The SAS-2 method eliminates previous attacks, and the SAS-2 method is the most efficient of the one-time password methods.

4.1 Security Considerations

One-time password authentication methods are vulnerable to certain kinds of attacks [22] , [25] , [29]. The security of the SAS-2 method was evaluated in this study.

4.1.1 Attacks on One-Time Password Methods

Attacks on one-time password authentication methods are the replay attack, the forgery attack, the impersonation attack, and the denial of service attack.

Replay Attack

The replay attack replays communication data and changes the verifier. The attacker intercepts authentication data and replaces the next verifier after the authentication session. Then the attacker can impersonate the user or confuse the server.

It assumes that an attacker has intercepted the following two sets continuously from the Internet.

$$\alpha_i \leftarrow F(C) \oplus F(A) \ ,$$
$$\beta_i \leftarrow C \oplus A \ ,$$

and

$$\alpha_{i+1} \leftarrow F(E) \oplus F(C) \ ,$$
$$\beta_{i+1} \leftarrow E \oplus C \ .$$

Then the replay attack succeeds by replacing the authentication data with the following data.

$$\alpha'_{i+1} \leftarrow \alpha_i = F(A) \oplus F(C) \ ,$$
$$\beta'_{i+1} \leftarrow \beta_i = A \oplus C \ .$$

In this way, the attacker will be able to impersonate the user from the next authentication session.

Forgery Attack

When an attacker steals communication data from some continuous authentication sessions, the attacker can change the verifier using assorted authentication data. Then the attacker can impersonate the user or replace the verifier in the server.

If a user sends the following data,

$$\alpha_i \leftarrow F(A) \oplus A \ ,$$
$$\beta_i \leftarrow C \oplus A \ .$$

β is meaningless, because C isn't used in α. Then an attacker can intercept those data, set x to any value, and forge as below.

$$\alpha'_i \leftarrow \alpha_i = F(A) \oplus A \ ,$$
$$\beta'_i \leftarrow x \ .$$

Due to the sending of those data, the ith user's authentication is successful. Therefore, the attacker can't impersonate the user, but the user will be denied from the next authentication session.

Impersonation Attack

An attacker replaces the intended verifier using a replay attack or a forgery attack. Then the attacker can impersonate the user from the next authentication session onward.

For example, a user sends the following data using the verifiers A and C ($J(x, y)$ where J is a masking function using x and y),

$$\alpha_i \leftarrow J_\alpha(A,\ C) \ ,$$
$$\beta_i \leftarrow J_\beta(A,\ C) \ .$$

Then an attacker intercepts transmission data up to the ith authentication session. The attacker then computes the following data where ($K(x, y)$ is a masking function using x and y).

$$\alpha'_i \leftarrow K_\alpha(A,\ y) \ ,$$
$$\beta'_i \leftarrow K_\beta(A,\ y) \ .$$

Therefore, the attacker will be able to impersonate the user using the next verifier y on the next authentication session. In the same manner, the attacker will be able to impersonate from the $(i+2)$th authentication session.

Denial of Service Attack

The denial of service attack rejects all or specific users by means of an offensive action on the server or by means of a falsification of user's verifier. Then the attacker can inconvenience the user but cannot imitate the user.

When a user sends the following data using the verifiers A and C where ($J(x, y)$ is a masking function using x and y),

$$\alpha_i \leftarrow J_\alpha(A, C) \ ,$$
$$\beta_i \leftarrow J_\beta(A, C) \ .$$

Then an attacker can intercept transmission data until the ith authentication session, and compute the following data. However, the attacker can't know the next verifier z where ($L(x, y)$ is a masking function using x and y).

$$\alpha'_i \leftarrow L_\alpha(A, z) \ ,$$
$$\beta'_i \leftarrow L_\beta(A, z) \ .$$

Therefore, the attacker can't impersonate, but the user will be denied from the next authentication session.

4.1.2 Considerations on the SAS-2

When a user tries to be authenticated by the server on the $(i+1)$th authentication session, it is assumed that an attacker has intercepted transmission data before the $(i+1)$th authentication sessions. Then the user sends the following authentication data.

$\alpha_{i+1} \leftarrow E \oplus (F(E) + C)$,

$\beta_{i+1} \leftarrow F(E) \oplus C$,

and ID.

If the attacker takes the offensive, he has to send the following data.

$\alpha'_{i+1} \leftarrow x \oplus (F(x) + C)$,

$\beta'_{i+1} \leftarrow F(x) \oplus C$,

and ID.

However, the attacker can't make those combinations with the transmission data until the $(i+1)$th authentication session, because the combination such $(F(x) + C)$ isn't used for creating the authentication data. Therefore, the SAS-2 method is secure.

4.2 Performance Considerations

Secure one-time password authentication methods include the Lamport, the revised SAS, and the SAS-2. Those methods' performance are evaluated here.

Table 4.1 summarizes the performance of the Lamport, the revised SAS, and the SAS-2 in the ith authentication session. M is the maximum number of hash iterations (ranging from about 500 to 1000), h represents the hash value, V_i is the verifier in ith authentication session, and $L(x)$ represents the data length of x.

Table 4.1 Performance evaluations of the SAS-2 and other methods

Performances / Methods	Server		User		User → Server	
	Hash Iterations (times)	Data storages	Hash Iterations (times)	Data storages	Transmission Iterations	Transmission Bulk
Lamport	1	n	$M-n$	n	1	$L(h)$
Revised SAS	2	ID, V_i	5	N_i	1	$L(ID)+2L(h)$
SAS-2	1 (mutual 2)	ID, V_i	3 (mutual 4)	N_i	1	$L(ID)+2L(h)$

4.2.1 Calculation Costs

In Lamport's method, the user used hash functions extensively and had to register the verifier before it expired. Those problems were solved with the revised SAS, but there was a little more overhead. The SAS-2 method uses a hash function only three times in comparison with five times in the revised SAS method. This means that hash overhead is reduced by about 40%, and the server uses a hash function only once. Then

other performances are under the same or better conditions, data storage, transmission iterations, and transmission bulk.

4.2.2 Mutual Authentication

A synchronous data communication procedure is possible for one-time password authentication methods. The SAS-2 method solved this problem by using a mutual authentication protocol. When carrying out a mutual authentication in the SAS-2 method, the user uses a hash function four times and the server uses it twice.

Chapter 5

Algorithm Variations of the SAS-2

The base type of the SAS-2 is defined by the following two data.

$\alpha \leftarrow C \oplus (F(C) + A)$

$\beta \leftarrow F(C) \oplus A$

The SAS-2 method is capable of other variations without additional overhead. This section deals with SAS-2 variations.

5.1 Combination Variations

The SAS-2 method has some elements that are useful data, combination types, and operations. Those factors are considered here.

5.1.1 Useful Data and Necessary Data

In the base type, the user uses A, C, and $F(C)$. The user uses other data ID (or other arbitrary number), P, N_i and N_{i+1} in the ith authentication session of the SAS-2 system. As well the user can use $F(A)$ in place of $F(C)$. However, P, N_i , and N_{i+1}

shouldn't be adopted, because that allows a verifier to be created by an attacker.

In view of these facts, the useful data are ID (or other arbitrary number), A, C, and $F(C)$ (or $F(A)$). Even if the user uses those data, the calculation load doesn't improve.

It is necessary to adopt A (or $F(A)$) and C because A (or $F(A)$) is used to remove masking data in this authentication session and C is used to authenticate the user in the next authentication session.

5.1.2 Combination Type

In the base type, a one sum operation and two XOR operations are used. Consequently, the user can create a combination type as follows.

$\alpha \leftarrow s \oplus (t + u)$

$\beta \leftarrow v \oplus w$

We can use other types in place of those combinations, for example as follows.

$\alpha \leftarrow s \oplus (t + u)$

$\beta \leftarrow v \oplus (w + x)$

But these are defective in that not only are operations increased but also an attacker can crack or impersonate easily, because this method makes the same styles and works from a small amount of data.

5.1.3 Operators for the SAS-2

In the base type, a user uses only XOR and addition operations. I considered the usefulness of other operators: NOT, AND, OR, subtraction, multiplication, and division.

Division is useless, because the SAS-2 method uses 0, and 0 has no factors. Table 5.1 shows results of 1 bit calculation using 6 operators, provided that $1 + 1 = 0$ and $0 - 1 = 1$.

Table 5.1 Results of 1bit calculation

x y	XOR	AND	OR	addition	subtraction	multiplication
0 0	0	0	0	0	0	0
0 1	1	0	1	1	1	0
1 0	1	0	1	1	1	0
1 1	0	1	1	0	0	1
sum	2	1	3	2	2	1

Table 5.2 shows an evaluation of operator values. Hamming distance represents the distance of expectations from half. If Hamming distance is 0, the operators are useful. However, if it isn't 0, the operators are useless. In calculations with other numbers of bits, Hamming distances are similar.

If the AND operator is used, the results don't occur with equal frequency. As a result the AND creates weak combinations, which is an issue. In the same manner, the operators OR and multiplication are not feasible. Therefore, a user can use XOR, addition, and subtraction.

Table 5.2 Evaluation of operator values

	XOR	AND	OR	addition	subtraction	multiplication
expectation	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{4}$
Hamming distance	0	$\frac{1}{4}$	$\frac{1}{4}$	0	0	$\frac{1}{4}$

The NOT has no problem but it is useless because it increases calculation costs, and it isn't strong in the SAS-2 protocol. Its meaning is shown below.

The following relational expression is realized for NOT calculations.

$$A \oplus \{not(A \oplus B)\} = not\{A \oplus A \oplus B\} = not(B)$$

It is clear that those combinations are useless. In the same way, the following relational expression is realized.

$$A \oplus \{not(B + C)\} = not\{A \oplus (B + C)\}$$

This means that the following expressions have the same strength.

$$A \oplus \{not(B + C)\}$$

$$A \oplus (B + C)$$

Therefore, the NOT is useless on the SAS-2 protocol.

In the SAS-2 method, the user can use subtraction in place of addition, but subtraction complicates the operation. Therefore, it is preferable that the user uses XOR and addition operators.

5.2 Preconditions for Secure Variations

It is assumed that the user sends the authentication data as below.

$\alpha \leftarrow s \oplus (t + u)$

$\beta \leftarrow v \oplus w$

The following are preconditions for secure variations of the SAS-2.

- *One of A or $F(A)$ has to be used in s, v, or w.*

  The characteristics of the SAS-2 authentication protocol are the following two operations. One is taking out C or $F(C)$ with A or $F(A)$, and the other is confirmation that an expression using $F(C)$ (or C) is regular. A or $F(A)$ is necessary when the user retrieves C or $F(C)$ from the communication data, because only A is stored in the server, and the server can make $F(A)$ from A.

- *C or $F(C)$ have to be used in $\alpha(\beta)$.*

  If $\alpha(\beta)$ is made from ID, A, and $F(A)$, an attacker can change $\beta(\alpha)$ at will. Therefore, a denial of service attack can succeed. For example, if α and β are as follows,

  $\alpha \leftarrow F(A) \oplus (A + ID)$

  $\beta \leftarrow C \oplus F(A)$,

  the attacker can change β to anything, and the authentication will be successful, because there is only one C , and C isn't related to the authentication.

- *It is necessary that v be different from w.*

  If v equals w, then β equals 0, and the expression β is meaningless. Then the server can't authenticate the user using only α.

- *The user has to use A, C, or $F(C)$ (or $F(A)$) in s, v, and w.*

  If expressions adopt ID or other arbitrary numbers in a, v or w, an attacker could remove the masking data.

- *In α , t must not be identical with u.*

  If they are same, $t + u$ equals $2t$. Then an attack becomes simple.

  Now, let α and β be defined as follows.

  $$\alpha \leftarrow F(C) \oplus (A + A)$$
  $$\beta \leftarrow C \oplus A$$

  Then an attacker can compute 2β equals $2C \oplus 2A$, and calculate $\alpha' \leftarrow F(C) \oplus (C+C)$ with $\alpha \oplus 2\beta$. Next the attacker sends α' with $\beta' \leftarrow C \oplus C$ $(= 0)$. Therefore, he can impersonate the user from the next authentication session.

- *The user must use $F(C)$ or $F(A)$.*

  It is clear that an attack is easy if the user creates α and β without using $F(C)$ and $F(A)$ as follows.

  $$\alpha \leftarrow A \oplus (C + ID)$$
  $$\beta \leftarrow C \oplus A$$

  Then an attacker could compute $C \oplus (C + ID)$ from $\alpha \oplus \beta$, and send it with $C \oplus C$, and thus impersonate the user.

5.3 Secure Variations

Attacks were tried on all variations of the SAS-2, and 32 secure variations were found that met the preconditions. These patterns have the same calculation cost as the base type. The following combinations are 32 secure variations.

1. $\alpha \leftarrow C \oplus (F(C) + A)$, $\beta \leftarrow F(C) \oplus A$
2. $\alpha \leftarrow C \oplus (F(C) + ID)$, $\beta \leftarrow F(C) \oplus A$
3. $\alpha \leftarrow C \oplus (A + ID)$, $\beta \leftarrow F(C) \oplus A$
4. $\alpha \leftarrow A \oplus (F(C) + C)$, $\beta \leftarrow F(C) \oplus A$
5. $\alpha \leftarrow A \oplus (C + ID)$, $\beta \leftarrow F(C) \oplus A$
6. $\alpha \leftarrow A \oplus (C + A)$, $\beta \leftarrow F(C) \oplus A$
7. $\alpha \leftarrow F(C) \oplus (C + ID)$, $\beta \leftarrow F(C) \oplus A$
8. $\alpha \leftarrow F(C) \oplus (C + F(C))$, $\beta \leftarrow F(C) \oplus A$
9. $\alpha \leftarrow F(C) \oplus (C + A)$, $\beta \leftarrow F(C) \oplus A$
10. $\alpha \leftarrow A \oplus (F(C) + ID)$, $\beta \leftarrow F(C) \oplus C$
11. $\alpha \leftarrow A \oplus (F(C) + A)$, $\beta \leftarrow F(C) \oplus C$
12. $\alpha \leftarrow A \oplus (C + ID)$, $\beta \leftarrow F(C) \oplus C$
13. $\alpha \leftarrow A \oplus (C + A)$, $\beta \leftarrow F(C) \oplus C$
14. $\alpha \leftarrow F(A) \oplus (C + ID)$, $\beta \leftarrow F(A) \oplus C$
15. $\alpha \leftarrow F(A) \oplus (C + F(A))$, $\beta \leftarrow F(A) \oplus C$
16. $\alpha \leftarrow F(A) \oplus (C + A)$, $\beta \leftarrow F(A) \oplus C$
17. $\alpha \leftarrow C \oplus (C + ID)$, $\beta \leftarrow F(A) \oplus C$
18. $\alpha \leftarrow C \oplus (C + F(A))$, $\beta \leftarrow F(A) \oplus C$
19. $\alpha \leftarrow C \oplus (C + A)$, $\beta \leftarrow F(A) \oplus C$
20. $\alpha \leftarrow A \oplus (C + ID)$, $\beta \leftarrow F(A) \oplus C$

21. $\alpha \leftarrow A \oplus (C + F(A))$, $\beta \leftarrow F(A) \oplus C$
22. $\alpha \leftarrow A \oplus (C + A)$, $\beta \leftarrow F(A) \oplus C$
23. $\alpha \leftarrow F(C) \oplus (A + ID)$, $\beta \leftarrow C \oplus A$
24. $\alpha \leftarrow F(C) \oplus (A + F(C))$, $\beta \leftarrow C \oplus A$
25. $\alpha \leftarrow F(C) \oplus (A + C)$, $\beta \leftarrow C \oplus A$
26. $\alpha \leftarrow F(A) \oplus (C + ID)$, $\beta \leftarrow C \oplus A$
27. $\alpha \leftarrow F(A) \oplus (C + F(A))$, $\beta \leftarrow C \oplus A$
28. $\alpha \leftarrow F(A) \oplus (C + A)$, $\beta \leftarrow C \oplus A$
29. $\alpha \leftarrow C \oplus (A + F(C))$, $\beta \leftarrow C \oplus A$
30. $\alpha \leftarrow C \oplus (C + F(A))$, $\beta \leftarrow C \oplus A$
31. $\alpha \leftarrow A \oplus (A + F(C))$, $\beta \leftarrow C \oplus A$
32. $\alpha \leftarrow A \oplus (C + F(A))$, $\beta \leftarrow C \oplus A$

5.4 Defective Combinations on the SAS-2

One-time password authentication methods are vulnerable to some kinds of attacks. The SAS-2 method has another problem.

If the user sends the following combinations, the danger of attack is higher than with other strong combinations.

$\alpha \leftarrow F(C) \oplus (A + F(C))$

$\beta \leftarrow C \oplus A$

Then there is a high probability that α is the same as A. Figure 5.1 shows the probability of dangerous combinations.

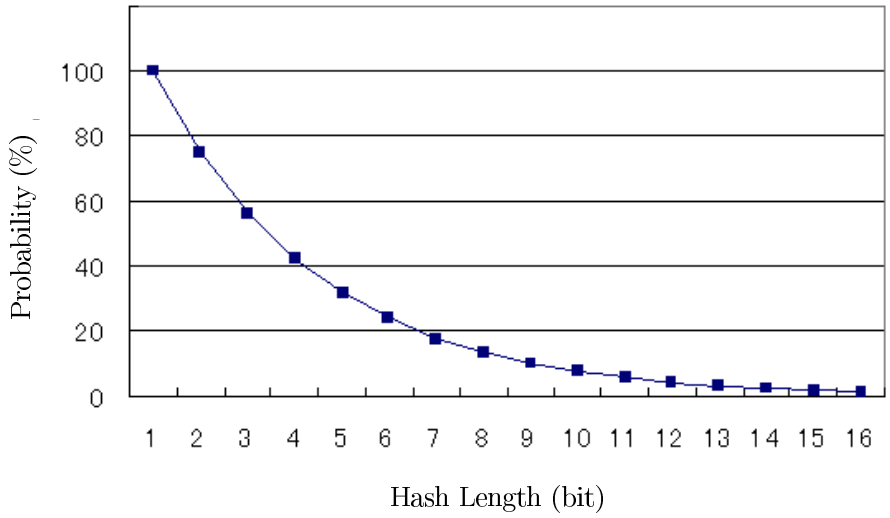

Fig. 5.1 Probability of dangerous combinations

The equation of this curve is $y_1 = \frac{400}{3} \times \left(\frac{3}{4}\right)^x$. If the hash length is enough long, the probability approaches 0 %.

Figure 5.2 shows the difference between defective combinations and strong combinations.

The equation of the secure combination curve is $y_2 = 100 \times \left(\frac{1}{2}\right)^x$. The probabilities shown are asymptotic for long bits, but the probabilities of defective combinations are high for small bits. This result means that defective combinations are weak because low bits can be discovered easily.

If an attacker computes A from α, the attacker can generate x freely and create the following data using A $(= \alpha)$.

$$\alpha' \leftarrow F(x) \oplus (A + F(x))$$

$$\beta' \leftarrow x \oplus A$$

When the attacker sends those data, the attacker can impersonate the user from the next authentication session using a substituted x.

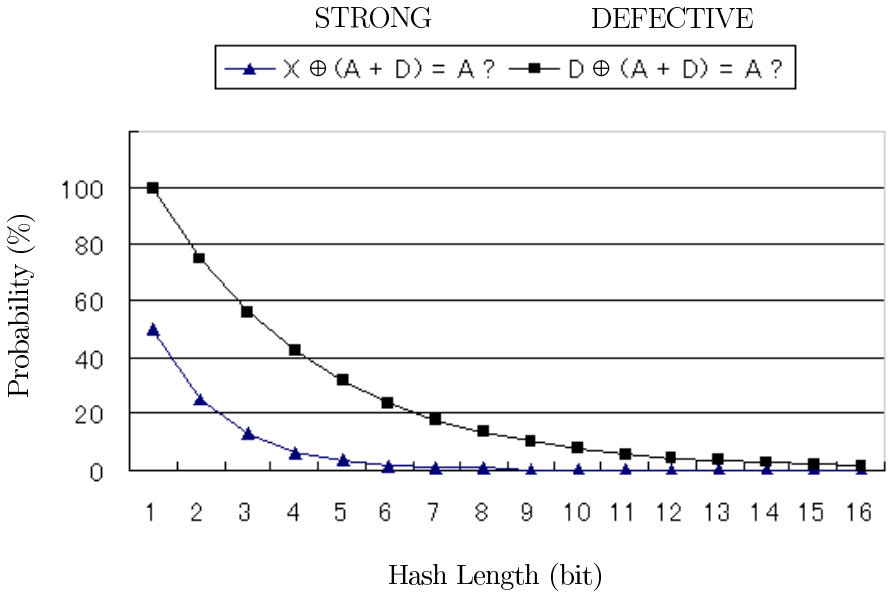

Fig. 5.2 Difference between defective combinations and strong combinations

Some other combinations are similarly dangerous. For example, the following combinations are defective.

$$\alpha \leftarrow A \oplus (C + A)$$

$$\beta \leftarrow F(C) \oplus A$$

or

$$\alpha \leftarrow A \oplus (F(C) + A)$$

$$\beta \leftarrow F(C) \oplus C$$

These two combinations use A in both neighborhoods of α. Then an attacker can extract C or $F(C)$ easily. Therefore, the attacker can impersonate the user.

5.5 Strong Variations

The following strong variations of the SAS-2 have no defective combinations. 7 combinations survived as strong variations among the 32 patterns.

1. $\alpha \leftarrow C \oplus (F(C) + A)$, $\beta \leftarrow F(C) \oplus A$
2. $\alpha \leftarrow C \oplus (F(C) + ID)$, $\beta \leftarrow F(C) \oplus A$
3. $\alpha \leftarrow C \oplus (A + ID)$, $\beta \leftarrow F(C) \oplus A$
4. $\alpha \leftarrow A \oplus (C + ID)$, $\beta \leftarrow F(C) \oplus A$
5. $\alpha \leftarrow F(C) \oplus (C + ID)$, $\beta \leftarrow F(C) \oplus A$
6. $\alpha \leftarrow A \oplus (F(C) + ID)$, $\beta \leftarrow F(C) \oplus C$
7. $\alpha \leftarrow A \oplus (C + ID)$, $\beta \leftarrow F(C) \oplus C$

5.6 Considerations of the SAS-2 Variations

The SAS-2 method has 32 secure variations and 7 strong variations among the 32 patterns. The 32 secure variations have defective combinations. However, those combinations are secure enough if the hash length is long bit. Therefore, the system can choose and use a combination from those 32 patterns.

Chapter 6

Applications

6.1 An Application for Key-Free Systems

Here I introduce key-free systems as one type of application system. These systems open and shut a key using the SAS-2 method, and this system is practicable not only in cars but also in cellular phones, PDAs, and ICs. Further more, in this system, one key device can deal with many open and shut devices. Moreover, it is easy to have a spare key.

6.1.1 The SAS-2 Protocol for Key-Free Systems

Key-free systems consist of two phases: the registration phase and the authentication phase. The registration process is performed only once, and the authentication procedure is executed every time the user opens or closes a lock such as a door. We describe these two phases below.

Registration Phase for Key-Free Systems

Figure 6.1 shows the initial registration phase of key-free systems. Then a lock has its own identity ID, the key's identity K, and secret key P.

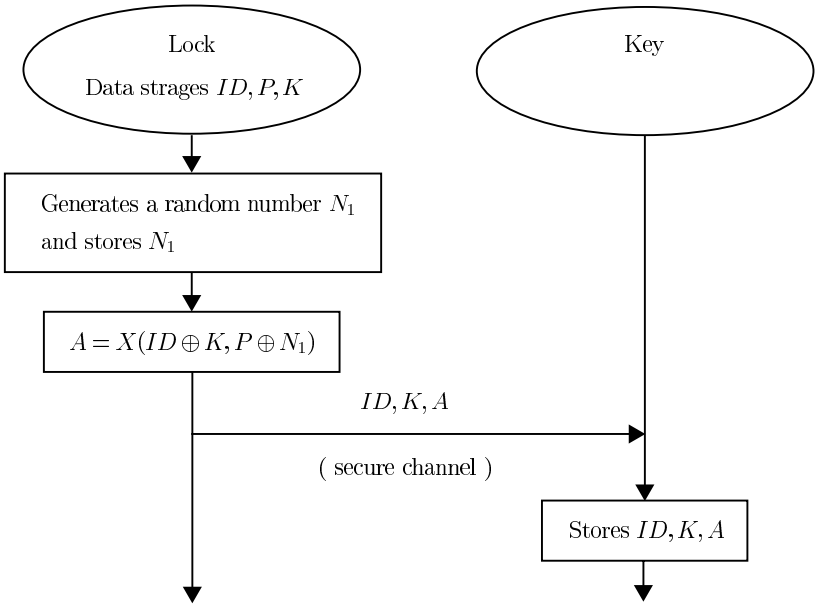

Fig. 6.1 Registration phase for key-free systems

1. *The lock calculates the verifier for the* 1*st authentication session.* The lock generates and stores a random number N_1, and calculates $A_1 = X(ID \oplus K, P \oplus N_1)$ using the random number and input data.
2. *The lock sends the registration data to the key for the subsequent authentication.* The lock sends ID, K, and A to the key through a secure channel.
3. *The key stores the verifier for subsequent authentication.* The key stores A with ID and K for subsequent authentication.

Authentication Phase for Key-Free Systems

When the user wants to open or close the lock, the following ith authentication protocol is executed on the lock and the key. Then the key has stored A with ID, and the apparatus has stored ID, K, P, and N_i. Figure 6.2 shows the ith authentication phase of key-free systems.

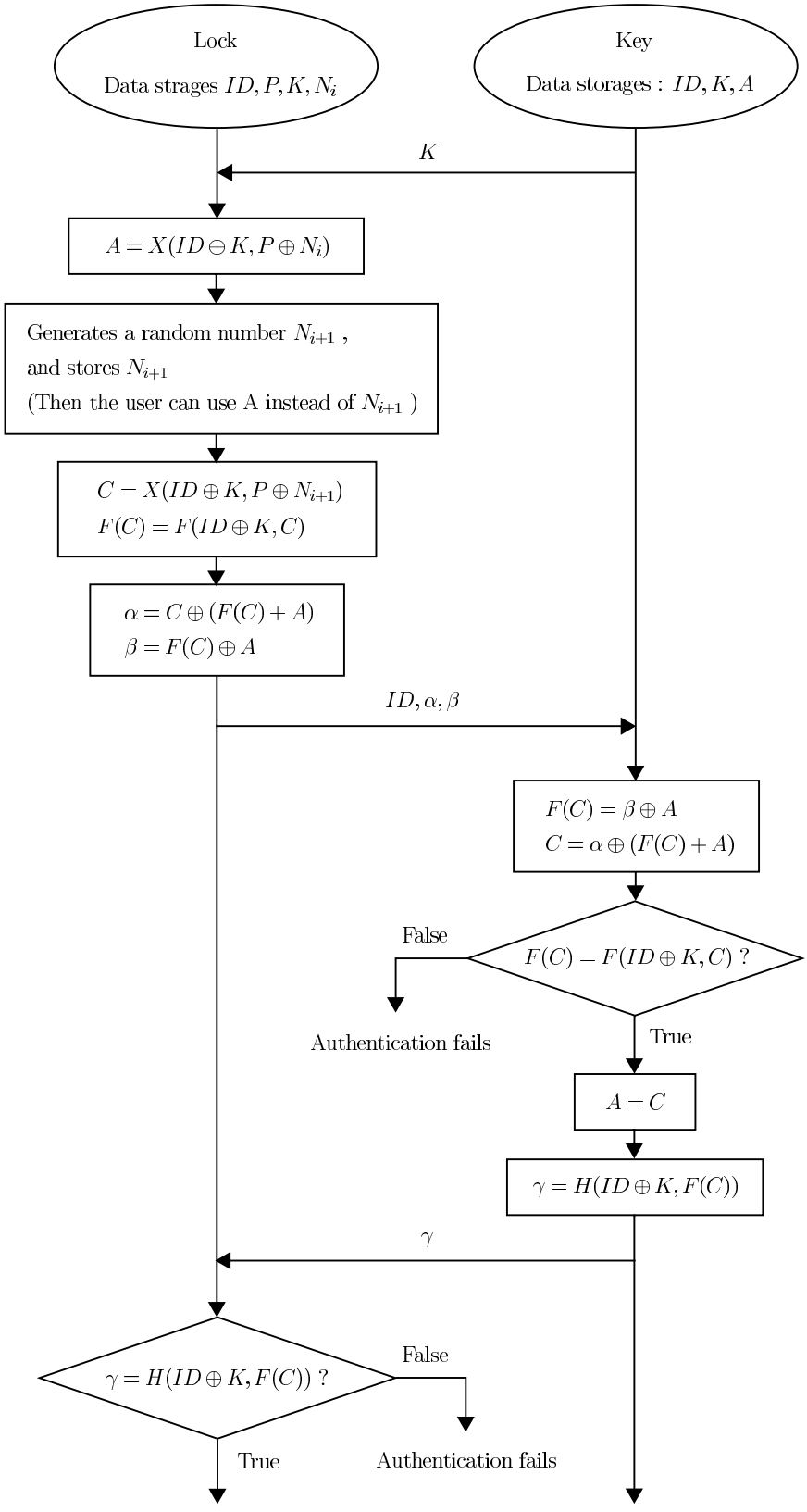

Fig. 6.2 The ith authentication phase for key-free systems

1. *The key sends the key's identity to open or close the lock.* The key sends the key's identity K to the lock.
2. *The lock calculates two data for the ith authentication session.* The lock calculates $A = X(ID\oplus, P \oplus N_i)$ using stored ID, P, K, and N_i. Then the lock generates and stores a random number N_{i+1}. At that time the lock can use A instead of N_{i+1}. Next the lock calculates $C = X(ID\oplus K, P\oplus N_{i+1})$ and $F(C) = F(ID\oplus K, C)$ using the random number and stored data. Then the lock computes $\alpha = C \oplus (F(C) + A)$ and $\beta = F(C) \oplus A$.
3. *The lock sends the authentication data to be authenticated by the key.* The lock sends ID, α and β to the key through a common network such as the Internet.
4. *The key authenticates the lock with the stored verifier A and the transmission data.* The key retrieves $F(C)$ from $\beta \oplus A$ with the stored verifier A, and gets C from $\alpha \oplus (F(C) + A)$ using $F(C)$. Next the key compares $F(C)$ and the computed $F(ID \oplus K, C)$. If they don't match, the attempt is rejected. If they do match, the next process is executed.
5. *The key stores the verifier for the next authentication session and calculates the data for a mutual authentication protocol.* The key stores C in place of A for the next authentication session, and calculates γ equals $F(ID \oplus K, F(C))$.
6. *The key sends the authentication data to be authenticated by the lock.* The key sends γ to the lock through a common network.
7. *The lock authenticates the key with the transmission data and the data which was used on the lock's authentication.* The lock calculates $F(ID \oplus K, F(C))$ and compares $F(ID \oplus K, F(C))$ which is transmission data γ. If they match, the key is authenticated and the lock opens or shuts. If they don't match, the user tries again.

6.1.2 Way to cut Calculation Cost

When this system is used on low spec machines such as key devices, it operates well at low cost. We suggest using XOR operations in place of hash functions.

For example, let a hash function be applied to data E, and let L denote the left half of E, and R denote the right half of E. Then the hash function F is calculated as follows.

$L \leftarrow L \oplus R$

$R \leftarrow L \oplus R$

Further let L_1 denote the left half of L, and L_2 denote the right half of L. In the same way let R_1 and R_2 be the half of R. Then the hash function H functions as follows.

$L_1 \leftarrow L_1 \oplus L_2$

$L_2 \leftarrow L_1 \oplus L_2$

$R_1 \leftarrow R_1 \oplus R_2$

$R_2 \leftarrow R_1 \oplus R_2$

If these operations are used, the quality of security drops, but costs are lowered considerably, because only XOR operations are managed on the SAS-2 method.

6.2 Other Applications

Lamport's method is applied to an S/KEY one-time password system in UNIX [16] , [17]. Consequently one-time password authentication methods can be regarded as practical techniques.

Some authentication systems using hash algorithms exist on the Internet: HMAC

(The Key-Hash Massage Authentication Code) [13], [14] and TLS (Transport Layer Security) [15]. Lamport's method maybe unfit for Internet protocols because Lamport's method has high hash overhead. Besides, an authentication procedure must be light because the systems on the Internet manage many users. The hash iterations in the SAS-2 method are the same in number as in HMAC. Therefore, SAS-2 can be used on Internet protocols.

The SAS method is applied to billing systems [18] for mobile phones. The SAS-2 method is useful for low spec machines such as the cellular phone because the SAS-2 method is light and such machines cannot bear heavy calculation costs.

The SAS-2 is a simple and limitless authentication method. Therefore, the SAS-2 is suitable for application in low spec machines and in Internet protocols.

Chapter 7

Conclusions and Discussions

In this thesis, I focused on hash overhead, and proposed a new method, SAS-2, which eliminates 40% overhead of hash function adaptation. The method has a mutual authentication phase which maintains synchronous data communications in an authentication procedure.

The SAS-2 method has many variations, and those algorithm variations are examined here. Then 7 strong patterns were identified. Moreover, the SAS-2 can be applied to low spec machines and Internet protocols.

One-time password authentications suffer from the stolen-verifier problem in which the attacker steals the user's verifier from the server and impersonates the user. The SAS-2 method suffers from this problem. Future research will examine new methods to eliminate such problems. It seems clear that this strong method will be applied to remote control systems.

One-time password authentication methods use two verifiers, one to authenticate the user, and the other after the authentication session. Those two verifiers have to be encrypted by a one-way function. This means that the user must use a one-way function twice. Therefore, a method using a one-way function twice is the minimum one-time password authentication method. I will research such minimum methods, as well. Such a fast method will be applicable to strong authentication servers.

Acknowledgement

I would like to express my deep gratitude to my thesis advisor, Professor Akihiro Shimizu, for his enthusiastic guidance, constructive criticism, and superior suggestions throughout the study. My grateful appreciation is extended to Professor Makoto Iwata and Professor Masanori Hamamura for their kindness and valuable advice. I would also like to thank Professor Takahiko Mendori for his instructions.

Grateful thanks are also extended to Professor Lawrence Hunter and Professor Mikio Kadota for their helpful English teaching towards the writing of this thesis.

Thanks a lot to Mr. Takashi Kamioka and Ms. Yoko Tanaka for their helpful in the analysis of proposed new methods. I would also like to thank the members of the Shimizu Laboratory for their precious opinions and support.

I would like to give special thanks to the faculty and staff of the Course of Information Systems Engineering during my studies at Kochi University of Technology.

Finally, deep thanks are extended to my father, mother, and brother for their endless love and emotional support.

References

[1] NBS, "Data Encryption Standard," FIPS-PUB-45, 1977.

[2] A. Shimizu, and S. Miyaguchi, "Fast data encipherment algorithm FEAL," IEICE Trans., vol.J70-D, no.7, pp.1413-1423, July 1987. (Japanese)

[3] M. Matsui, "New block encryption algorithm MISTY," Lecture Notes in Computer Science, FSE 1997, pp.54-68, 1997.

[4] J. Daemen and V. Rijmen, "The block cipher rijndael," Smart Card Research and Applications, LNCS 1820, J.-J. Quisquater and B. Schneier, Eds., Springer-Verlag, pp.288-296, 2000.

[5] J. Daemen and V. Rijmen, "Rijndael, the advanced encryption standard," Dr. Dobb's Journal, vol.26, no.3, pp.137-139, March 2001.

[6] R. Rivest, "The MD5 message-digest algorithm," Internet Request For Comments 1321, April 1992.

[7] National Institute of Standards and Technology, "Secure hash standard," FIPS Publication 180-1, April 1995.

[8] W. Stallings, "Secure hash algorithm," in Cryptography and Network Security: Principles and Practice Second Edition, pp.193-197, Prentice-Hall, 1999.

[9] H. Dobbertin, A. Bosselaers and B. Preneel, "RIPEMD-160: A strengthened version of RIPEMD," Fast Software Encryption, Third International Workshop, Lecture Notes in Computer Science 1039, Springer-Verlag, pp.71-82, 1996.

[10] T. Kwon, and J. Song, "Efficient key exchange and authentication protocols protecting weak secrets," IEICE Trans. Commun., vol.E81-A, no.1, pp.156-163, January 1998.

[11] T. Kwon, M. Kang, S. Jung, and J. Song, "An improvement of the password-based

authentication protocol (K1P) on Security against replay attacks," IEICE Trans. Commun., vol.E82-B, no.7, pp.991-997, July 1999.

[12] J.-Y. Park, D.-I. Lee, H.-H. Lee, and J.-G. Park, "One-time key generation system for agent data protection," IEICE Trans. Commun., vol.E85-D, no.3, pp.535-545, March 2002.

[13] H. Krawczyk, M. Bellare, and R. Coretti, "HMAC: Keyed-hashing for massage authentication," Request For Comments 2104, February 1997.

[14] H. Krawczyk, M. Bellare, and R. Coretti, "The key-hash massage authentication code(HMAC)," Federal Information Processing Standards Publication 198, March 2002.

[15] T.Dierks, and C. Allen, "The TLS protocol version 1.0," Request For Comments 2246, January 1999.

[16] N. Haller, "The S/KEY(TM) one-time password system," Proc. Internet Society Symposium on Network and Distributed System Security, pp.151-158, 1994.

[17] N. Haller, C. Metz, P. Nesser, and M. Straw, "A one-time password system," Request For Comments 2289, February 1998.

[18] M. Sandirigama, A. Shimizu, and M.T. Noda, "Simple and secure coin(SAS-Coin) - A practical micropayment system," IEICE Trans. Commun., vol. E83-A, no.12, pp.2679-2688, December 2000.

[19] L. Lamport, "Password authentication with insecure communication," Communications of the ACM, vol.24, no.11, pp.770-772, 1981.

[20] A. Shimizu, "A dynamic password authentication method by one-way function," IEICE Trans., vol.J73-D-I, no.7, pp.630-636, July 1990. (Japanese)

[21] A. Shimizu, "A dynamic password authentication method by one-way function," System and Computers in Japan, vol.22, no.7, 1991.

[22] N. Haller, and R. Atkinson, "On Internet Authentication," Request For Comments

1704, October 1994.

[23] A. Shimizu, T. Horioka, and H. Inagaki, "A password authentication method for contents communication on the internet," IEICE Trans. Commun., vol.E81-B, no.8, pp.1666-1673, August 1998.

[24] M. Sandirigama, A. Shimizu, and M.T. Noda, "Simple and secure password authentication protocol (SAS)," IEICE Trans. Commun., vol.E83-B, no.6, pp.1363-1365, June 2000.

[25] C. Lin, H. Sun, and T. Hwang, "Attacks and solutions on strong-password authentication," IEICE Trans. Commun., vol.E84-B, no.9, pp.2622-2627, September 2001.

[26] T. Kamioka, and A. Shimizu, "The examination of the security of SAS one-time password authentication," IEICE Technical Report, OFS2001-48, no.435, pp.53-58, 2001.

[27] T. Tsuji, T. Kamioka, and A. Shimizu, "Simple and secure password authentication protocol, ver.2 (SAS-2)," IEICE Technical Report, OIS2002-30, vol.102, no.314, pp.7-11, September 2002.

[28] C.-M. Chen and W.-C Ku, "Stolen-verifier attack on two new strong-password authentication protocols," IEICE Transactions on Communications, vol.E85-B, no.11, pp.2519–2521, November 2002.

[29] T. Tsuji, and A. Shimizu, "An impersonation attack on one-time password authentication protocol OSPA," IEICE Technical Report, ISEC2002-81, vol.102, no.436, pp.67-72, November 2002.

[30] T. Tsuji, and A. Shimizu, "Algorithm variations of SAS-2," IEICE Technical Report, IN2002-149, vol.102, no.498, pp.25-30, December 2002.