Secure Authentication Schemes Using One-Way Funcions

Takasuke Tsuji

A dissertation submitted to
Kochi University of Technology
in partial fulfillment of the requirements
for the degree of

Doctor of Engineering

(Engineering Course)
Department of Engineering
Graduate School of Engineering
Kochi University of Technology
Kochi, Japan

March 2006

Abstract

Authentication Schemes Using One-Way Functions

Takasuke Tsuji

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. This can be accomplished by the use of a one-way function.

A hash function is one of functions which generate a lossy value. The S/Key system is a famous authentication system using hash functions. This system authenticates the user using different password every time and keeps off replaying the communication data. However, S/Key has two practical difficulties: high hash overhead and the requirement of resetting the verifier. Some schemes are proposed, which solves problems such as practical difficulties of the S/Key. OSPA (Optimal Strong Password Authentication protocol) is one of solutions is propsed in 2001.

In this paper, I point out security problems of existing schemes included the above schemes, and those solution is proposed. Stolen-verifier attack, in which the attacker steals user's verifier from the server, is strong attack. This paper discuss the stolen-verifier attacks and those solutions are proposed. Moreover, some application systems of those solutions are presented.

key words authentication, password, one-way function, hash function, security

Contents

Chapter 1

Introduction

Internet and mobile communication applications for managing personal information have been developing. Those application systems need password authentications to protect users' right from attacker. Authentication schemes apply one-way functions for protecting password leakages. One-way functions can generate a random hash value from a password and cannot get the password from the generated hash value. Public key cryptosystems, common key cryptosystems, and hash functions are one-way functions, and many authentication schemes are proposed using each function. If the security of the cryptosystem applied an authentication system is failed, the authentication system is depended a particular cryptographic algorithm must be built not only a cryptosystem but also an authentication system again. Services using authentication schemes diversify, their specifications are different. Thus, a general-purpose authentication scheme, which isn't depended a particular cryptographic algorithm, is desired. In this research, I consider one-way functions to be functions which generate a lossy value from a value and propose authentication schemes using such functions. It is easy to compute $f(x)$ from x and difficult to compute x such that $y = f(x)$ with one-way functions. Usually hash functions such as MD5 [4], SHA-1 [10], [17] or RIPEMD-160 [11] are employed.

A hash function is one of functions which generate a lossy value. Lamport's scheme [1] is a famous authentication system using hash functions. This scheme authenticates the user using different password every time and keeps off replaying the communication data. However, Lamport's scheme has two practical difficulties: high hash overhead and

the requirement of resetting the verifier. The CINON (chained one-way data verification method) [2], [3] 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) [16] eliminates this problem. However, both CINON and PERM methods suffer from vulnerability to a kind of 'Man in the Middle' attack. To counter this, M. Sandirigama *et al.* have proposed the SAS (Simple And Secure) password authentication protocol [19], which prevents such attacks and reduces storage, processing, and transmission overhead.

C.L. Lin *et al.* have detected replay and denial of service attacks on the SAS method and have argued for the OSPA (Optimal Strong-Password Authentication) [21] method. The revised SAS [22] method solves those security problems, but those methods suffer from vulnerability to a stolen-verifier problem [25], [28], and the authors have performed successful impersonation attacks on the OSPA method [26].

In this paper, secure authentication schemes using one-way functions are discussed. In the chapter 2, attacks using communication data are discussed, and their solution is proposed. In the chapter 3, stolen-verifier attacks are discussed, and their solutions are proposed. In the chapter 4, applications using proposal authentication schemes are proposed. In the chapter 5, this paper is concluded.

Chapter 2

Attacks Using Communication Data and Their Solutions

One-time password authentication schemes suffer from attacks using communication data. In this chapter, attacks using communication data are discussed, and their solution is proposed.

2.1 Timing Attack and Combination Attacks on Existing Schemes

In the authentication session, Server and user communicate authentication data. Then the attacker can intercept and forge those data. Some authentication schemes use counter value. Server and user confirm authentication session number using the counter value. However, the attacker can forge the counter value and she/he can impersonate the user. Authentication schemes without the counter create authentication data using secret value. The same secret value is used to create authentication data in sequential authentication sessions, and the attacker can combine those authentication data and can create passable authenticaiton data. Here, Timing attack and combination attacks are discussed.

2.1.1 Cryptanalysis on S/Key Authentication System

Some authentication schemes apply public-key cryptosystems. Recently, user authentications are applied to mobile phones and smart cards, which have low processing speed. Then it is impractical for applying public-key cryptosystems, which involve complicated computations. L. Lamport has proposed a one-time password authentication scheme [1], which requires a different password in each transaction. This scheme uses a conventional hash function, which is easier to compute than pubic-key cryptographies. In 1994, The S/Key one-time password authentication system [5], which is based on the Lamport's scheme, has been proposed.

One-time password authentication schemes using a challenge response mechanism are used if the user's machine cannot rewrite the authentication data. In those schemes, the server sends a counter and seed values for challenge. However, those data are sent through the Internet, and an attacker can intercept them and commit an offense using those modified data. Here, I discuss a security problem of the S/Key scheme [43].

S/Key Scheme

The S/key scheme has three stages: the registration stage, the login stage, and the authentication stage. The registration stage is performed only once, and the login stage and the authentication stage are executed every time the user logs in to the system. These three stages are described below.

Definitions and Notations

The following definitions and notations are used throughout this subsection.

- U refers to the computer user who employs the protocol for authentication.
- S refers to the server that authenticates users.
- K is the user's secret key/password.
- H is a conventional hash function. For example, $H(x)$ means x is hashed once.
- N is the maximum allowable number of login attempts.
- $C(= N - t)$ is an initial counter value.
- t is an integer indicating the number of authentication sessions.
- $SEED, D$ represent random numbers.
- $\oplus$ represents a bit-wise exclusive-or (XOR) operation.
- p_t denotes $H^{N-t}(K \oplus SEED)$, where $N - t$ is the number of hash iterations. For example, $H^3(K \oplus SEED)$ denotes $H(H(H(K \oplus SEED)))$.
- The expression '$s \rightarrow t$: u' represents the fact that s sends u to t.

Registration Stage

In the registration stage, S/Key manages as follows.

1. $U \leftarrow S$: N, $SEED$.
2. $U \rightarrow S$: p_0.

Initially, S selects a number N, and generates a seed value $SEED$. Then S sends those data to U. After receiving them, U inputs her/his secret K and then calculates $p_0 = H^N(K \oplus SEED)$. Next U sends the calculated data to S.

Login Stage

To login, U executes as follows. In the tth login, S is storing p_{t-1} with the counter $C(= N - t)$.

1. $U \leftarrow S$: C, $SEED$.
2. $U \rightarrow S$: p_t.

For the tth login, S sends C with $SEED$ to U as a challenge. After receiving them, U inputs K and calculates $p_t = H^C(K \oplus SEED)$. Then U sends p_t as a response.

Authentication Stage

When S receives U's response, p_t is hashed once and is compared with the stored p_{t-1}. If they match, U is authenticated. Then S updates p_t with C.

Timing Attack on the S/Key Scheme

The S/Key scheme uses counter values. In this scheme, an attacker caneasily change the counter value by intercepting the authentication communication and stealing the user's verifier. If the attacker changes the counter, she/he can get the next verifier from the user's response. Then the attacker can impersonate the user using the verifier in the next authentication session. Below, timing attack on the S/Key scheme is explained.

First, an attacker A disturbs the following communication on the ith login.

$$U \leftarrow S : N - i, SEED.$$

Then A replaces those data with $N-(i+1)$, $SEED$ and sends them to U. After receiving them, U calculates $p_{i+1} = H^{N-(i+1)}(K \oplus SEED)$, and then sends the calculated data to S.

$$U \rightarrow S : p_{i+1}.$$

Then A disturbs and substitutes its data for $H(p_{i+1})$ and sends $H(p_{i+1})$ to S. After receiving it, the received data is hashed and compared with the stored p_{i-1}. Then they match and U is authenticated. Next S stores the received data $H(p_{i+1})$ in place of p_{i-1}.

In the $(i+1)$th login, A can impersonate U as follows.

$$A \leftarrow S : N-(i+1), SEED.$$
$$A \rightarrow S : p_{i+1}.$$

First, S sends $N-(i+1), SEED$ to A. Then A returns p_{i+1}. When S receives it, the system calculates $H(p_{i+1})$ and compares it to the stored $H(p_{i+1})$. They are the same so A is authenticated. Therefore, A can impersonate U.

2.1.2 Cryptanalysis on OSPA

The Lamport password authentication method [1] 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) [2], [3] has solved those problems, but it has a problem in that the user must remember a random number. The PERM (Privacy Enhanced information Reading and writing Management method) [16] eliminates this problem by generating and sending a random number from the server. The authors 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. In response, the authors propose the SAS (Simple And Secure) [19] password authentication protocol that eliminates the 'Man in the Middle' attack and reduces storage, processing, and transmission overhead. However, the method is vulnerable to both replay and denial of service attacks [21]. C. L. Lin *et al.* have argued that the Optimal Strong-Password Authentication (OSPA) [21] has solved those security problems.

C. -M. Chen *et al.* have devised a stolen-verifier attack on the SAS and the OSPA methods [25]. In such an attack, the adversary has stolen the verifier from the server. However, this problem is solved if the server provides strong management. Here, I devise an impersonation attack on the OSPA method without an active attack on the server

[26], [33].

The OSPA Protocol

The OSPA 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. I describe these two phases below.

Definitions and Notations

The following definitions and notations are used throughout this subsection.

- The user refers to the user of a computer who uses the protocol for authentication.
- The server refers to the server that authenticates users.
- A is the user's identity.
- P is the user's password.
- h is a one-way hash function. For example, $h(x)$ means x is hashed once.
- n is an integer indicating the number of authentication sessions.
- $\oplus$ represents bitwise XOR operation.

Registration Phase

Figure 2.1 shows the initial registration phase of the OSPA protocol.

1. The user inputs the user's identity A and password P when the user registers in the system; then the user calculates the first verifier $h^2(P \oplus 1)$ with password P and initial value $n = 1$.

2. The user sends A and $h^2(P \oplus 1)$ to the server through a secure channel.
3. The server stores $h^2(P \oplus 1)$ with A for later authentication.

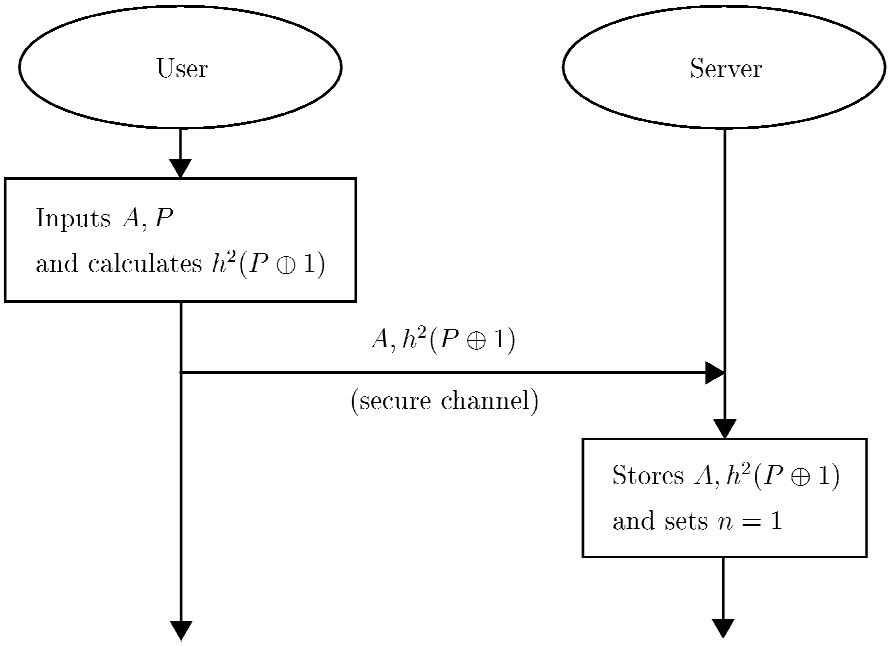

Fig. 2.1 Registration phase of the OSPA.

Authentication Phase

To login, the user executes the nth authentication protocol. Then the server stores n and $h^2(P \oplus 1)$ with A. Figure 2.2 shows the ith authentication session of the OSPA protocol.

1. The user inputs the user's identity A when joining the system.
2. The user issues a login service request with A for accessing to the system.
3. The server responds to the user with the ith sequential number n to the user to authenticate the user.
4. The user inputs his password P, then calculates $h(P \oplus n)$, $h^2(P \oplus n)$, $h^2(P \oplus (n+1))$, and $h^3(P \oplus (n+1))$ using password and response data from the server. Next the

server sets c_1 equals $h(P \oplus n) \oplus h^2(P \oplus n)$, c_2 equals $h^2(P \oplus (n+1)) \oplus h(P \oplus n)$, and c_3 equals $h^3(P \oplus (n+1))$.

5. The user sends c_1, c_2, and c_3 to the server through a common network such as the Internet.
6. The server first checks whether $c_1 \neq c_2$. If it holds, the server retrieves $h(P \oplus n)$ from $c_1 \oplus h^2(P \oplus n)$ with the verifier and gets $h^2(P \oplus (n+1))$ from $c_2 \oplus h(P \oplus n)$. Then the server passes the authentication only if $c_1 \neq c_2$, $h(h(P \oplus n))$ equals the verifier, and $h(h^2(P \oplus (n+1)))$ equals c_3. If the authentication is successful, the next process is executed.
7. The server stores $h^2(P \oplus (n+1))$ with A and sets $n = n+1$ for the next authentication session.

2.1 Timing Attack and Combination Attacks on Existing Schemes

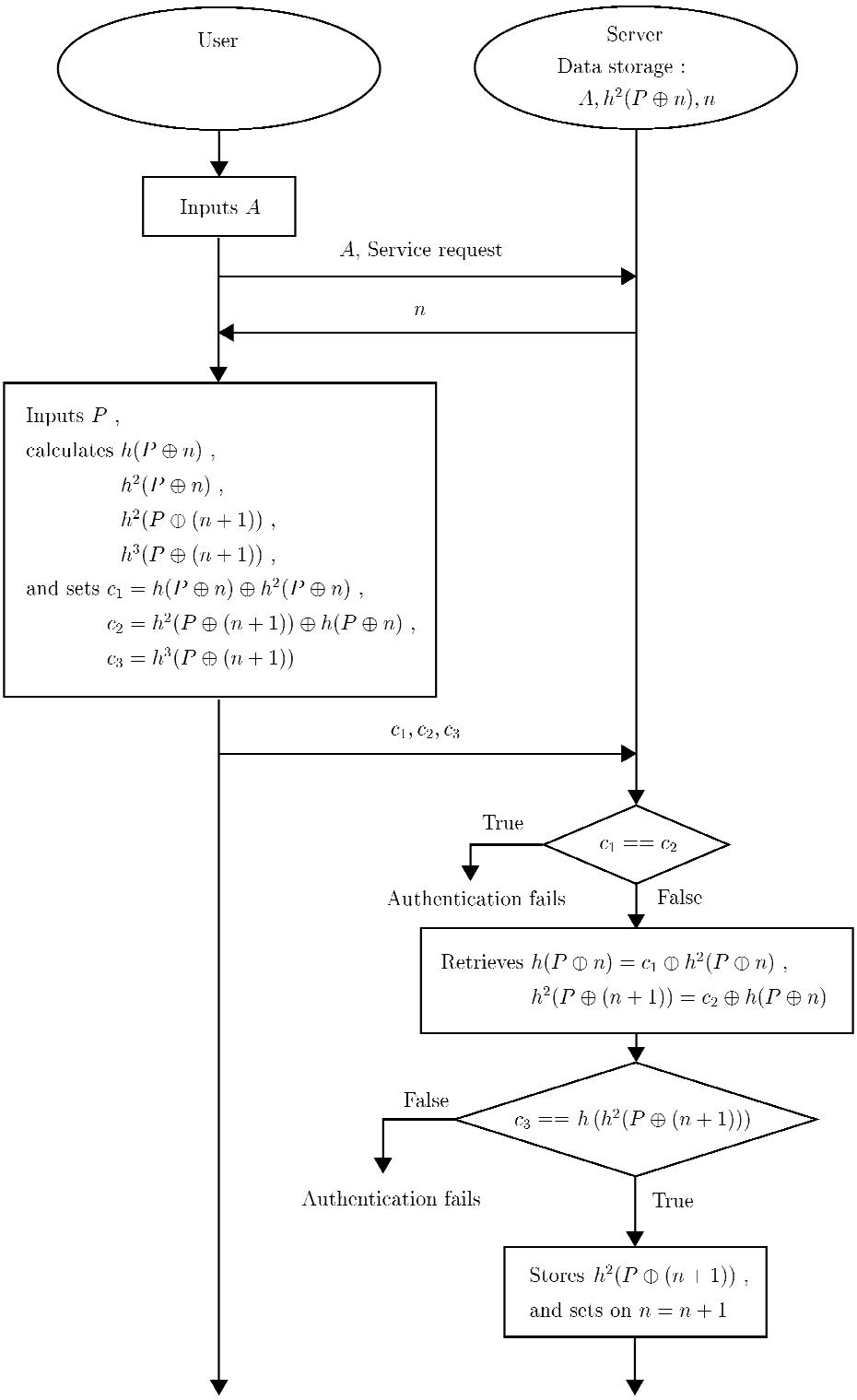

Fig. 2.2 Authentication phase of the OSPA.

Impersonation Attacks on the OSPA

When the server authenticates the user using the OSPA system, the server receives data consisting of two verifiers. Then the attacker impersonates a regular user with that authentication data.

In an attack on the OSPA, the attacker replays and forges authentication data with communication data from the previous authentication session.

Impersonation Attack with Two Previous Verifiers

When the user tries to be authenticated by the server on the $(n+1)$th authentication session, it is assumed that an attacker has intercepted transmission data from the $(n-1)$th to the $(n+1)$th authentication sessions. Those data are as follows:

The $(n-1)$th authentication data

$$\begin{aligned} c_{(n-1)1} &= h(P \oplus (n-1)) \oplus h^2(P \oplus (n-1)) \ , \\ c_{(n-1)2} &= h^2(P \oplus n) \oplus h(P \oplus (n-1)) \ , \\ c_{(n-1)3} &= h^3(P \oplus n) \ , \end{aligned}$$

the ith authentication data

$$\begin{aligned} c_{(n)1} &= h(P \oplus n) \oplus h^2(P \oplus n) \ , \\ c_{(n)2} &= h^2(P \oplus (n+1)) \oplus h(P \oplus n) \ , \\ c_{(n)3} &= h^3(P \oplus (n+1)) \ , \end{aligned}$$

and the $(n+1)$th authentication data

$$\begin{aligned} c_{(n+1)1} &= h(P \oplus (n+1)) \oplus h^2(P \oplus (n+1)) \ , \\ c_{(n+1)2} &= h^2(P \oplus (n+2)) \oplus h(P \oplus (n+1)) \ , \\ c_{(n+1)3} &= h^3(P \oplus (n+2)) \ . \end{aligned}$$

When the attacker impersonates the user, he first computes $h^2(P \oplus n) \oplus h(P \oplus (n+1))$ from $c_{(n)1} \oplus c_{(n)2} \oplus c_{(n+1)1}$ and substitutes $c_{(n-1)3}$ for $c'_{(n+1)3}$. Next the attacker sends

the following data to the server for the $(n+1)$th authentication session.

$$\begin{aligned} c_{(n+1)1} &= h(P \oplus (n+1)) \oplus h^2(P \oplus (n+1)) \ , \\ c'_{(n+1)2} &= h^2(P \oplus n) \oplus h(P \oplus (n+1)) \ , \\ c'_{(n+1)3} &= h^3(P \oplus n) \end{aligned}$$

After receiving the authentication data, the server first checks whether $c_{(n+1)1} \neq c'_{(n+1)2}$. The data is accepted because $h^2(P \oplus (n+1))$ is different from $h^2(P \oplus n)$. Next the server retrieves $h(P \oplus (n+1))$ from $c_{(n+1)1} \oplus h^2(P \oplus (n+1))$, and gets $h^2(P \oplus n)$ from $c'_{(n+1)2} \oplus h(P \oplus (n+1))$. Then the server compares $h\,(h(P \oplus (n+1)))$ with the stored verifier $h^2(P \oplus (n+1))$, and compares $h(h^2(P \oplus n))$ with the received $c'_{(n+1)3}$. These are the same so the user is authenticated. Then the server stores $h^2(P \oplus n)$ and sets $n = n+1$ for the next authentication.

In the future, if the attacker wants to login, he alternately sends the following two sets. The attacker uses the first set in the $(n+2k)$th authentication session (k is a natural number)

$$\begin{aligned} c_1 &= h(P \oplus n) \oplus h^2(P \oplus n) \ , \\ c_2 &= h^2(P \oplus (n+1)) \oplus h(P \oplus n) \ , \\ c_3 &= h^3(P \oplus (n+1)) \ , \end{aligned}$$

and the attacker uses the second set in the $(n+2k+1)$th authentication session

$$\begin{aligned} c_1 &= h(P \oplus (n+1)) \oplus h^2(P \oplus (n+1)) \ , \\ c_2 &= h^2(P \oplus n) \oplus h(P \oplus (n+1)) \ , \\ c_3 &= h^3(P \oplus n) \ , \end{aligned}$$

then the verifier in the server is changed to $h^2(P \oplus (n+1))$ and $h^2(P \oplus n)$ in turns. In this way, the attacker can impersonate the regular user.

2.1 Timing Attack and Combination Attacks on Existing Schemes

Impersonation Attack Using Many Previous Verifiers

In an impersonation attack with two previous verifiers, the invasion can be detedted if the server has stored two verifiers. As a result, the attacker has to change the next verifier intentionally.

The attacker can make $c_{(n+1)1}$, $c'_{(n+1)2}$, $c'_{(n+1)3}$ using transmission data from the $(n-1)$th to the $(n+1)$th authentication sessions, where

$$\begin{aligned} c_{(n+1)1} &= h(P \oplus (n+1)) \oplus h^2(P \oplus (n+1)) \ , \\ c'_{(n+1)2} &= h^2(P \oplus n) \oplus h(P \oplus (n+1)) \ , \\ c'_{(n+1)3} &= h^3(P \oplus n) \ . \end{aligned}$$

In the same manner, the attacker can generate the following data using communication data from the $(n-2)$th to the $(n+1)$th authentication sessions, and changes the verifier form $h^2(P \oplus (n+1))$ to $h^2(P \oplus (n-1))$. Then the attacker computes $c''_{(n+1)2}$ from $c_{(n-1)1} \oplus c_{(n-1)2} \oplus c_{(n)1} \oplus c_{(n)2} \oplus c_{(n+1)1}$, and substitutes $c_{(n-2)3}$ for $c''_{(n+1)3}$.

$$\begin{aligned} c_{(n+1)1} &= h(P \oplus (n+1)) \oplus h^2(P \oplus (n+1)) \ , \\ c''_{(n+1)2} &= h^2(P \oplus (n-1)) \oplus h(P \oplus (n+1)) \ , \\ c''_{(n+1)3} &= h^3(P \oplus (n-1)) \end{aligned}$$

The attacker can generate the following data in a similar manner, and then change the verifier form $h^2(P \oplus (n+1))$ to $h^2(P \oplus s)$ (s is an integer, $1 < s < n+1$). Then the attacker computes $c'''_{(n+1)2}$ from $c_{(s)1} \oplus c_{(s)2} \oplus c_{(s+1)1} \oplus c_{(s+1)2} \oplus \cdots \oplus c_{(n-1)1} \oplus c_{(n-1)2} \oplus c_{(n)1} \oplus c_{(n)2} \oplus c_{(n+1)1}$, and substitutes $c_{(s-1)3}$ for $c'''_{(n+1)3}$.

$$\begin{aligned} c_{(n+1)1} &= h(P \oplus (n+1)) \oplus h^2(P \oplus (n+1)) \ , \\ c'''_{(n+1)2} &= h^2(P \oplus s) \oplus h(P \oplus (n+1)) \ , \\ c'''_{(n+1)3} &= h^3(P \oplus s) \end{aligned}$$

Therefore, the attacker can change the verifier using previous verifiers.

2.1 Timing Attack and Combination Attacks on Existing Schemes

Impersonation Attack with Various Verifiers

In the above attacks, the attacker can change the verifier to one of the previous verifiers. Then it is assumed that the attacker changes the verifier used in the sth authentication session. Next the attacker logs in to the system, and changes the verifier. However, the new verifier is limited to $h^2(P \oplus t)$ (t is an integer, $1 < t < s$ or $t = s+1$). This means that the attacker changes the verifier to the next one ($h^2(P \oplus (s+1))$) if s is small.

For example, an attacker could change verifiers as follows:

$$\cdots,\ h^2(P \oplus 7),\ h^2(P \oplus 4),\ h^2(P \oplus 2),\ h^2(P \oplus 3),\ \cdots$$

Then the attacker changes verifiers sequentially, like $h^2(P \oplus 2),\ h^2(P \oplus 3)$. In that case, the crime can easily be detected because the verifiers are changed sequentially. Then the server memorizes only two sequential verifiers.

This problem is solved then: the attacker doesn't change the verifier sequentially. For example, when the attacker changes the verifier from $h^2(P \oplus s)$ to $h^2(P \oplus (s+2))$, he generate the following transmission data.

The attacker uses the sth authentication data

$$\begin{aligned} c_{(s)1} &= h(P \oplus s) \oplus h^2(P \oplus s)\ , \\ c_{(s)2} &= h^2(P \oplus (s+1)) \oplus h(P \oplus s)\ , \\ c_{(s)3} &= h^3(P \oplus (s+1))\ , \end{aligned}$$

and the $(s+1)$th authentication data

$$\begin{aligned} c_{(s+1)1} &= h(P \oplus (s+1)) \oplus h^2(P \oplus (s+1))\ , \\ c_{(s+1)2} &= h^2(P \oplus (s+2)) \oplus h(P \oplus (s+1))\ , \\ c_{(s+1)3} &= h^3(P \oplus (s+2))\ . \end{aligned}$$

The attacker computes c_2 from $c_{(s)2} \oplus c_{(s+1)1} \oplus c_{(s+1)2}$, and substitutes $c_{(s+1)3}$ for

c_3. Then he sends the following data and changes the verifier to $h^2(P \oplus (s+2))$.

$$c_1 = h(P \oplus s) \oplus h^2(P \oplus s) \ ,$$
$$c_2 = h^2(P \oplus (s+2)) \oplus h(P \oplus s) \ ,$$
$$c_3 = h^3(P \oplus (s+2))$$

Similarly, the attacker computes c_2 from $c_{(s)2} \oplus c_{(s+1)1} \oplus c_{(s+1)2} \oplus \cdots \oplus c_{(u-1)1} \oplus c_{(u-1)2}$, and substitutes $c_{(u-1)3}$ for c_3. Then the attacker can change the verifier to $h^2(P \oplus u)$ using from the sth to the $(u-1)$th authentication data (u is an integer, $s < u < n+2$) as below.

$$c_1 = h(P \oplus s) \oplus h^2(P \oplus s) \ ,$$
$$c_2 = h^2(P \oplus u) \oplus h(P \oplus s) \ ,$$
$$c_3 = h^3(P \oplus u)$$

Therefore, the attacker can change the next verifier to a various verifier.

Hybrid Impersonation Attack

The attacker can change the verifier in various ways using a combination of the three techniques explained in the above attacks.

However, the crime can be detected if the server has stored all verifiers, because the attacker changes to the verifier used before, but storing all the verifiers isn't practical. Further, the user can't use the same verifier if server stores and checks all verifiers. Such a mechanism can't ensure security if the user has used the system for long time.

2.1.3 Cryptanalysis on Lin-Chang Authentication Scheme

Many authentication schemes have been proposed, and the Lamport's scheme [1] and the S/Key [5], [6] are representation of one-time password authentication schemes appiled hash functions. For mobile commerce, T.-C. Yeh *et al.* have proposed an authentication scheme using smart cards [24], which solves some kinds of attacks. Though, T.-C. Yeh *et al.*'s scheme does not satisfy the low computation requirement. In 2004, M.-H. Lin and C.-C. Chang have proposed a one-time password authentication scheme [39], in which the user entrusts almost hash operations to the server. However, the Lin-Chang scheme is vulnerable an attack with the present communication data.

Lin-Chang Scheme

Lin-Chang scheme is a one-time password authentication protocol. In this subsection, this scheme is illustrated.

Definitions and Notations

The following definitions and notations are used throughout this subsection.

- U refers the user.
- S refers to the server that authenticates users.
- $||$ denotes concatenation.
- $\oplus$ represents a XOR operation.
- h is a cryptographic hash function. $h(x)$ means x is hashed once, and $h^2(x)$ means x is hashed twice.
- K is a secret key is generated by U.
- $SEED$ is an unique secret and is a large random number generated by S.

- SK is a session key, equals $D||T$, where D is a random number and T is a timestamp.
- IK is an initial key, equals $K \oplus SEED$.
- N is the number of logins.
- p is an initial password, is calculared as $p_i = h^{N-i}(IK)$
- The expression '$s \rightarrow t$: u' represents the fact that s sends u to t.

Registration Stage

For access to the system, the user receives an unique secret, the number of logins, and initial passwords from the server and stores those data in her/his smart card. The registration process is below.

1. S issues a smart card containing $SEED$ to U.
2. S generates SK and computes $SEED \oplus SK$.
3. $U \leftarrow S$: $SEED \oplus SK$.
4. U extracts SK by XORing the received data and the stored $SEED$ and generates K. Next, U computs IK, and stores it. Then, U decides on N, and computes $IK \oplus SK$, $N \oplus SK$.
5. $U \rightarrow S$: $IK \oplus SK$, $N \oplus SK$.
6. S extracts IK and N from the received data and computes p_0, p_1, and p_2. Then, S stores p_0 with N.
7. $U \leftarrow S$: $p_0 \oplus SK$, $p_1 \oplus SK, p_2 \oplus SK$.
8. U obtains p_0, p_1, and p_2 from the received data. Then, U computes the hashed N times of IK and compares with p_0. If the two values are equivalent, U can be sure of the authenticity of S and stores p_0, p_1, p_2, and N in her/his smart card.

Login and Authentication Stage

To log in, the user executes the tth login and authentication stage. The login and authentication process is below.

1. S generates SK and computes $SEED \oplus SK$. Next, S computes p_{t-1} and $p_{t-1} \oplus SK$.
2. $U \leftarrow S$: $p_{t-1} \oplus SK$, $SEED \oplus SK$.
3. U extracts SK by XORing the received $p_{t-1} \oplus SK$ and the stored p_{t-1}. Then, U checks T of the session key SK. If the timestamp is valid, U obtains $SEED$ from the received data. If the obtained data is the same as the stored $SEED$ in her/his smart card, U can be certain of S's identity and computes $p_t \oplus SK$ with the stored p_t.
4. $U \rightarrow S$: $p_t \oplus SK$.
5. S extracts p_t and computes the hash value of p_t. If the hashed value and p_{t-1} are equivalent, S can be sure of U's identity. Next, S updates N with $N - t$ and computes p_{t+1} using p_0 of U. S performs the XOR operation on p_{t+1} and SK
6. $U \leftarrow S$: $p_{t+1} \oplus SK$.
7. U obtains p_{t+1} from the received data and stores it in her/his smart card for the next login.

Attack with the Present Communication Data

In the tth authentication stage, the attacker intercepts communication data: $c_{t,2} = p_t \oplus SK_t$ and $c_{t,3} = p_{t+1} \oplus SK_t$, where SK_t is the tth session key. Then, the attacker computes $w_t = p_t \oplus p_{t+1}$ by XORing $c_{t,2}$ and $c_{t,3}$.

After U's session, the attacker try to logs into S. Then, S sends $c_{t+1,1a} = p_t \oplus SK_{t+1}$ and $c_{t+1,1b} = SEED \oplus SK_{t+1}$ to the attacker. After receiving, the attacker computes $p_{t+1} \oplus SK_{t+1} = w_t \oplus c_{t+1,1a}$. Next, the attacker sends the computed data. S receives

it and extracts p_{t+1} with the $(t+1)$th session key SK_{t+1}. Then, the authentication is succeed because the p_t and the hash value of the extracted p_{t+1} are the same. Thus, the attacker can impersonate U.

2.2 Authentication Scheme against Combination Attacks

The Lamport's password authentication scheme [1] uses a one-way function but this scheme has two practical difficulties: high hash overhead and the requirement of resetting the verifier. The CINON (chained one-way data verification scheme) [2], [3] has solved those problems but the scheme has another problem; the user must memorize a random number. The PERM (Privacy Enhanced information Reading and writing Management scheme) [16] eliminates this problem by generating and sending a random number from the server. M. Sandirigama *et al.* have detected a possible security flaw in both CINON and PERM schemes: 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, the SAS (Simple And Secure) password authentication protocol, which eliminates the 'Man in the Middle' attack and reduces storage, processing, and transmission overhead is proposed. However, the scheme suffers from vulnerability to both replay and denial of service attacks [21]. C. L. Lin *et al.* have argued into the OSPA (Optimal Strong-Password Authentication) [21] scheme, and the revised SAS [22] scheme have solved those security problems. The authors have been successful impersonation attacks on the OSPA scheme [26], and the only secure protocol with a one-time password scheme for changing verifiers infinitely is the revised SAS.

The revised SAS scheme uses a one-way function five times. This function has high overhead. It's desirable to reduce the number of this function, which is used in the SAS

scheme, because authentication schemes are now being applied to mobile communications and Internet protocols. In this section, I propose a new scheme, SAS-2(Simple And Secure password authentication protocol, ver.2), applying one-way function only three times by using two verifiers and another for masking. This reduces hash overhead is by about 40% in comparison with the revised SAS. SAS-2 and other schemes generate a random number, but this operation has high overhead such as the calculation of one-way function. In consideration of this problem, we appended the 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 schemes, and the SAS-2 scheme solved this problem using a mutual authentication protocol.

The SAS-2 scheme has variations, and I have considered all variations of the SAS-2 and have produced safe combinations. Moreover, I have examined problems in the SAS-2 protocol and have determined the most secure combinations remaining.

2.2.1 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.

Definitions and Notations

The following definitions and notations are used throughout this section.

- User is the computer user who employs the protocol for authentication.
- Server is the server that authenticates users.
- ID is the user's identity.

- S 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.

Registration Phase

Figure 2.3 shows the initial registration phase of the SAS-2 protocol.

1. The user inputs the user's identity ID and password S when registering in the systems; then the user generates and stores a random number N_1, and calculates $A = X(ID, S \oplus N_1)$ using the generated N_1 and input data.

2. The user sends ID and A to the server through a secure channel.

3. The server stores A with ID for subsequent authentication.

Authentication Phase

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 the user's identity ID and password S when she/he joins the system. Then the user calculates $A = X(ID, S \oplus N_i)$ using input data and

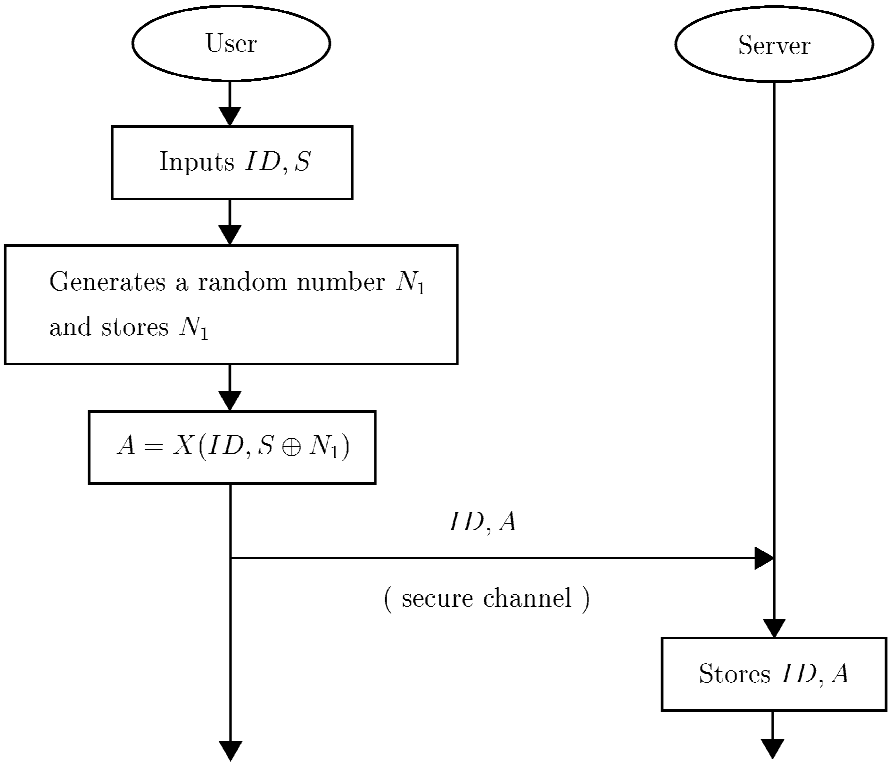

Fig. 2.3 Registration phase of the SAS-2.

the stored N_i. Next the user generates a random number N_{i+1} and stores it. At that time, the user can use A instead of N_{i+1}. Then the user calculates $C = X(ID, S \oplus N_{i+1})$ and $F(C) = F(ID, C)$ using the generated 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 ID, α, and β to the server through a common network such as the Internet.

3. 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 do not match, the user is rejected, and the server refuses the user. If they match, the user is authenticated, and the next process is executed.

4. 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 γ to the user through a common network.

6. The user calculates $H(ID, F(C))$ and compares the received γ and $H(ID, F(C))$. If they match, the server is authenticated. If they don't match, the user tries again to log in.

2.2 Authentication Scheme against Combination Attacks

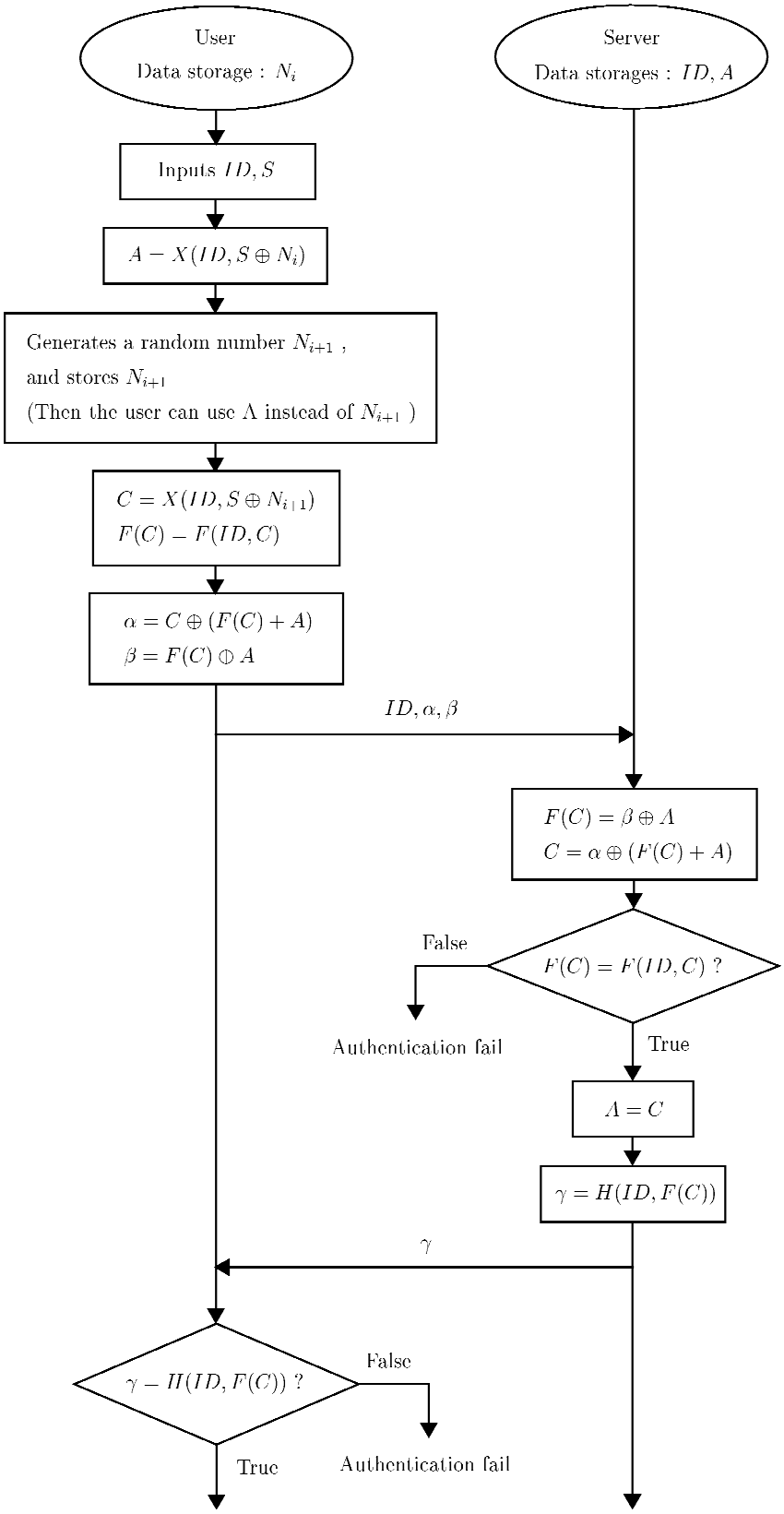

Fig. 2.4 Authentication phase of the SAS-2.

2.2.2 Security and Performance Analysis

The SAS-2 scheme eliminates attacks, and the SAS-2 scheme is lightest of the one-time password schemes.

Security Considerations

One-time password authentication schemes are vulnerable to some kinds of attacks. We have evaluated the security on the SAS-2 scheme.

Attacks on One-time Password Schemes

Attacks on one-time password authentication schemes 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.

Forgery Attack

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

2.2 Authentication Scheme against Combination Attacks

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.

Denial of Service Attack

The denial of service attack rejects all or specific users by means of an offensive action to the server or by redacting the verifier of a particular user. Then the attacker can disturb the user but the attacker cannot imitate the user.

Considerations on the SAS-2

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

$$\alpha \leftarrow E \oplus (F(E) + C),$$
$$\beta \leftarrow F(E) \oplus C,$$
$$\text{and } ID.$$

When the attacker takes the offensive, she/he has to send the following data.

$$\alpha' \leftarrow x \oplus (F(x) + C),$$
$$\beta' \leftarrow F(x) \oplus C,$$
$$\text{and } ID.$$

However, the attacker cannot make those combinations with the transmission data until the $(i+1)$th authentication session. Therefore, the SAS-2 scheme is secure.

Performance Considerations

In the Lamport's scheme, the user uses hash functions extensively and has to register the verifier when before it expires. The revised SAS solves those problems but the revised SAS has a little more overhead. The SAS-2 scheme uses a hash function only three times in comparison with five times in the revised SAS scheme. It means that hash overhead is reduced by about 40%, and the server uses a hash function only once. Then other performances are the same or better condition, data storage, transmission iterations, and transmission bulk. When carrying out a mutual authentication in the SAS-2 scheme, the user uses a hash function four times and the server uses it twice.

Table 2.1 summarizes the performance of 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.

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	$L(ID)+2L(h)$

Table 2.1 Performance evaluations of the SAS-2 and other schemes.

The S/Key password authentication scheme [5] is famous system of challenge-

response based authentication schemes. In challenge-response based authentication schemes, the system sends a seed with a counter to the user for the challenge. Then the user calculates one-time password using the user's password and the received challenge. However, the S/Key is unsecured because the attacker can create the one-time password using the challenge, which is intercepted from the Internet [14].

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

Combination Variations

The SAS-2 scheme has some elements: use data, combination types, and operations. We consider those factors here.

Useful Data and Necessary Data

In the base type, the user uses A, C, and $F(C)$. The user can use other data ID (or other arbitrary number), S, 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, S, N_i , and N_{i+1} shouldn't be adopted because it allows to create a verifier 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 quantity does not 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.

Combination Type

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

$$\alpha \leftarrow s \oplus (t + u),$$
$$\beta \leftarrow v \oplus w.$$

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

$$\alpha \leftarrow s \oplus (t + u),$$
$$\beta \leftarrow v \oplus (w + x).$$

However, these are defective that not only operations are increased but also an attacker can crack or impersonate easily because those types make same styles and use the same data from few useful data.

Operators for the SAS-2

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

If the AND operator is used, the results do not occur with equal frequency. According as the AND creates weak combinations, this operator is useless. In the same manner, the operators OR, multiplication, and division are not feasible. NOT has not such problem but it is useless because it increases calculation costs and does not make the SAS-2 protocol strong. The user can use subtraction in place of addition but sub-

traction complicates the operation. Therefore, it is preferable that the user uses XOR and addition operators.

Preconditions for Secure Variations

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

$$\alpha \leftarrow s \oplus (t + u),$$
$$\beta \leftarrow v \oplus w.$$

There following are preconditions for performable and 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 another is confirmation that an expression using $F(C)$ (or C) is regular. A or $F(A)$ is necessary when the user outs 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 $F(C)$ has to be used in α or β.*

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

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

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

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

  If v equals w, then β equals 0, and an expression β is meaningless. Then the server

cannot 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 s, v or w, an attacker could remove the masking data.

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

  If t and u 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 computes 2β equals $2C \oplus 2A$, and calculates $\alpha' \leftarrow F(C) \oplus (C + C)$ with $\alpha \oplus 2\beta$. Next the attacker sends α' with $\beta' \leftarrow C \oplus C$ $(= 0)$. Thus, she/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 computes $C \oplus (C + ID)$ from $\alpha \oplus \beta$, and sends it with $C \oplus C$. Then the attacker can impersonate the user.

Secure Variations

We tried for attacking on all variations of the SAS-2 and found 32 secure variations, which meet the preconditions. They have the same calculation cost as the base type. These combinations are as follows.

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

Defective Combinations on the SAS-2

One-time password schemes are vulnerable to some kinds of attacks, and the SAS-2 scheme 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 it is a high probability that α is the same as A. Figure 2.5 shows the difference between defective combinations and strong combinations. The probabilities are shown 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 are found out easily.

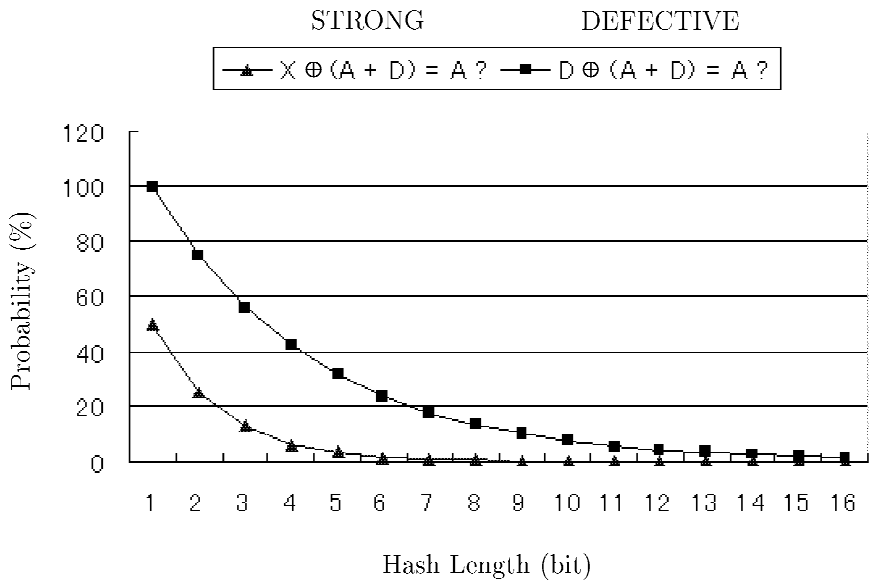

Fig. 2.5 Difference between defective combinations and strong combinations.

If an attacker can compute 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 the substituted x.

2.2.4 Strong Variations

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

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

5. $\alpha \leftarrow F(C) \oplus (C + ID), \quad \beta \leftarrow F(C) \oplus A.$
6. $\alpha \leftarrow A \oplus (F(C) + ID), \quad \beta \leftarrow F(C) \oplus C.$
7. $\alpha \leftarrow A \oplus (C + ID), \qquad \beta \leftarrow F(C) \oplus C.$

If hash length is small, these strong variations are safe than the secure variations. In one-time password authentication schemes, the hash length must be 64, 128, or more over bits for the security. Then the secure combinations perform same security as the strong combinations.

2.3 Conclusion

In this chapter, attacks using communication data and their solutions are discussed. The SAS-2 [42] solves the above attacks using complicated authentication data.

The proposed scheme is very simple authentication scheme and can be use to mobile communications. This scheme become more quickly if the user calculate authentication data during communicate with the server [29]. The SAS-2 has a synchronous problem, but the synchronous problem can be solved using the past verifier [49].

Chapter 3

Stolen-Verifier Attacks and Their Solutions

One-time password authentication schemes suffer from stolen-verifier attacks, in which the attacker steals user's verifier from the server's database. In this chapter, stolen-verifier attacks are discussed, and their solutions are proposed.

3.1 Stolen-Verifier Attacks on Existing Schemes

Internet communication systems are diversified and people can access to the servers of those systems. Those servers have users' information on their database. Those information stored in database can be stolen, and attacker can impersonate user by using the stolen data. Here, stolen-verifier attacks on the existing authentication schemes are discussed.

3.1.1 Cryptanalysis on SAS-2

In the revised SAS method, the user uses a one-way function five times. However, this function has high overhead, because a one-way function applies 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. Accordingly, the authors have proposed the SAS-2 (SAS

ver.2) [23], [27] which reduces such overhead. However, the SAS-2 suffers from stolen-verifier attack. Here, the attack is explained.

The 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 U logs into the system. These two phases are described below.

Definitions and Notations

The following definitions and notations are used throughout this subsection.

- U refers the computer user who employs the protocol for authentication.
- S refers to the server that authenticates users.
- ID is the user's identity.
- P is the user's password.
- h is one-way hash function. 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 exclusive OR (XOR) operation.
- $||$ denotes concatenation.
- VS_i denotes $h(ID||P \oplus N_i)$.
- The expression '$s \Rightarrow t$: u' represents the fact that s sends u to t through a private and authenticated channel. For example, s delivers to t directly or encrypts and sends u with certification.

- The expression '$s \rightarrow t$: u' represents the fact that s sends u to t through a common channel such as the Internet.

Registration Phase

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

1. U inputs the ID and P. Then U generates and stores N_1, and calculates $VS_1 = h(ID||P \oplus N_1)$.
2. $U \Rightarrow S$: ID and VS_1.
3. S stores VS_1 with ID for subsequent authentication.

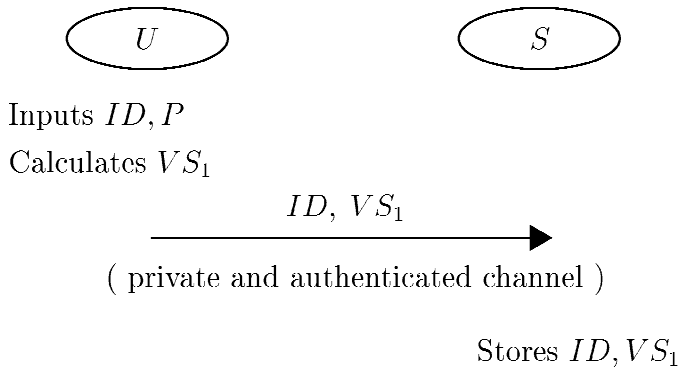

Fig. 3.1 Registration phase of the SAS-2

Authentication Phase

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

1. U inputs ID and P. Then U generates N_{i+1}, and calculates VS_i, VS_{i+1}, and

$h(ID||VS_{i+1})$. Next U computes $\alpha = VS_{i+1} \oplus \{h(ID||VS_{i+1}) + VS_i\}$ and $\beta = h(ID||VS_{i+1}) \oplus VS_i$.

2. $U \rightarrow S$: ID, α, and β.
3. S retrieves $h(ID||VS_{i+1})$ from $\beta \oplus VS_i$ with the stored verifier VS_i and gets VS_{i+1} from $\alpha \oplus \{h(ID||VS_{i+1}) + VS_i\}$. Next S compares $h(ID||VS_{i+1})$ and the calculated $h(ID||VS_{i+1})$. If they match, U is authenticated, and S stores VS_{i+1} in place of VS_i and calculates $\gamma = h(ID||h(ID||VS_{i+1}))$.
4. $S \rightarrow U$: γ.
5. U compares the received data γ and the calculated data $h(ID||h(ID||VS_{i+1}))$. If they match, S is authenticated by U.

Stolen-Verifier Attack on SAS-2

SAS-2 is vulnerable to a stolen-verifier problem. If an attacker A steals the U's verifier, A can impersonate U as shown below.

A disturbs U's communication and intercepts the following data in the ith authentication session.

$$\alpha_i = VS_{i+1} \oplus \{h(ID||VS_{i+1}) + VS_i\} ,$$
$$\beta_i = h(ID||VS_{i+1}) \oplus VS_i ,$$
$$\text{and } ID.$$

Then A steals VS_i from S, extracts $h(ID||VS_{i+1})$ by $\beta_i \oplus VS_i$, and executes VS_{i+1} by $\alpha_i \oplus \{h(ID||VS_{i+1}) + VS_i\}$. Next A computes and sends the following data with y which is A's secret key.

$$\alpha'_i = VS'_{i+1} \oplus \{h(ID||VS'_{i+1}) + VS_i\} ,$$
$$\beta'_i = h(ID||VS'_{i+1}) \oplus VS_i ,$$
$$\text{and } ID,$$

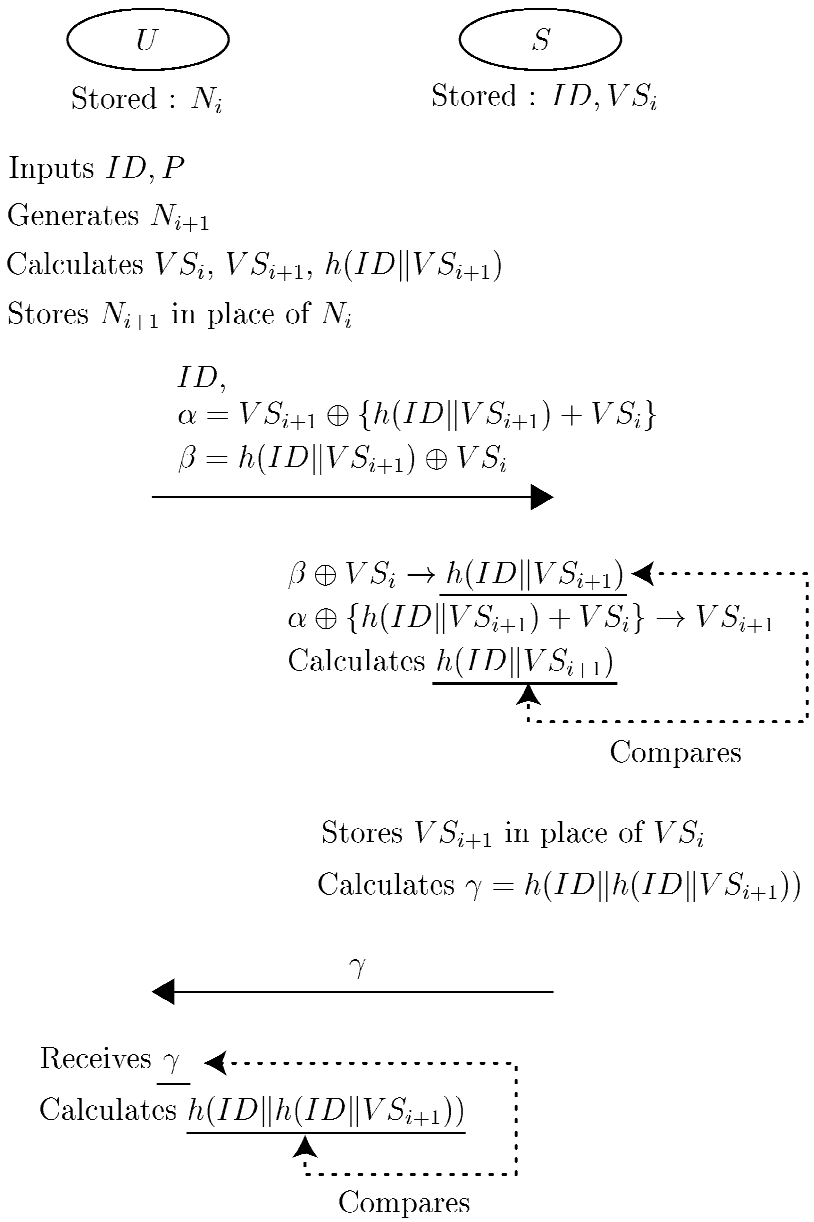

Fig. 3.2 The ith authentication phase of the SAS-2

where

$$VS'_{i+1} = h(ID||y \oplus N'_{i+1}) \ .$$

At that time, S authenticates U, changes the next verifier to VS'_{i+1}, and receives $\gamma = h(ID||VS'_{i+1})$ to U. Then A disturbs S's communication, and replaces $h(ID||VS'_{i+1})$ with $h(ID||VS_{i+1})$. Thus a mutual authentication can succeed, too. After that success,

A can impersonate U from the $(i+1)$th authentication session onwards. For example, A sends the following data using $VS'_{i+2} = h(ID||y \oplus N'_{i+2})$ in the $(i+1)$th authentication session.

$$\alpha_{i+1} = VS'_{i+2} \oplus \{h(ID||VS'_{i+2}) + VS'_{i+1}\} \ ,$$
$$\beta_{i+1} = h(ID||VS'_{i+2}) \oplus VS'_{i+1} \ ,$$
$$\text{and } ID.$$

Similarly, A can send data such as the above to be authenticated from the $(i+2)$th authentication session onwards. Therefore, A can impersonate the user.

3.1.2 Cryptanalysis on ROSI

Many authentication schemes such as the SAS [19] and the OSPA [21] suffers from stolen-verifier attacks [25], [28]. To counter this, H.Y. Chien and J.K. Jan have proposed the ROSI (RObust and SImple authentication protocol) [28], which eliminates the stolen-verifier problem. In a stolen-verifier problem, an attacker steals the user's verifier from the server. In the same manner, the attacker can steal other data from the user's personal information. The ROSI can be impersonation attack using its data.

The ROSI Protocol

The ROSI has two phases: the registration phase and the authentication phase. The registration phase is performed only once, and the authentication phase is executed every time the user logs in to the system. These two phases are described below.

Definitions and Notations

The following definitions and notations are used throughout this subsection.

- U refers the computer user who employs the protocol for authentication.
- S refers to the server that authenticates users.
- ID is the user's identity.
- P is the user's password.
- x is the server's strong secret key.
- h is one-way hash function. 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.
- $\oplus$ represents a bitwise exclusive OR (XOR) operation.
- $||$ denotes concatenation.
- VR_i^j denotes $h^j(P||N_i)$, where j is the number of hashing times. For example, $h^3(P||N_i)$ denotes $h(h(h(P||N_i)))$.
- The expression '$s \Rightarrow t$: u' represents the fact that s sends u to t through a private and authenticated channel. For example, s delivers to t directly or encrypts and sends u with certification.
- The expression '$s \rightarrow t$: u' represents the fact that s sends u to t through a common channel such as the Internet.

Registration Phase

Figure 3.3 shows the initial registration phase of ROSI.

1. U inputs ID, P, and N_1.
2. $U \Rightarrow S$: ID, P and N_1.
3. S calculates VR_1^2 and stores it with ID and x.
4. S also writes the data $R = h(x||ID) \oplus P$ and VR_1^1 into the U's smart card.
5. S issues the card to the user.

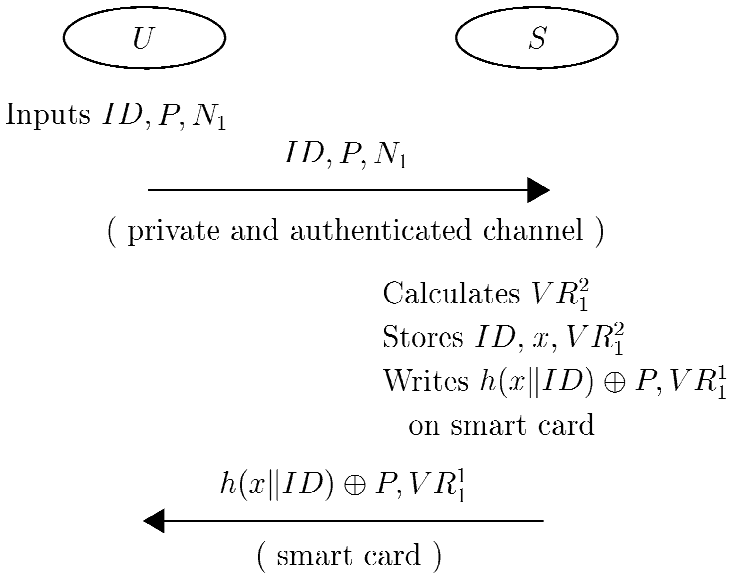

Fig. 3.3 Registration phase of the ROSI

Authentication Phase

To log in, U executes the ith authentication session of the ROSI protocol. Then U is storing R and VR_i^1 in her/his smart card, and S is storing x and VR_i^2 with ID. Figure 3.4 shows the ith authentication phase of the ROSI.

1. U inputs ID and P into her/his smart card. The card extracts $h(x||ID)$ by $R \oplus P$, selects N_{i+1} and computes $\alpha = h(h(x||ID) \oplus VR_i^2) \oplus VR_{i+1}^2$ and $\beta = VR_{i+1}^3 \oplus VR_i^1$ by using N_{i+1}, P, and VR_i^1.
2. $U \rightarrow S$: ID, α, and β.
3. S computes $h(x||ID)$, and then it computes $\alpha' = h(h(x||ID) \oplus VR_i^2)$, where VR_i^2 is the stored verifier for U. S extracts VR_{i+1}^2 by $\alpha' \oplus \alpha$ and then extracts VR_i^1 by $\beta \oplus h(VR_{i+1}^2)$. S checks whether $h(VR_i^1)$ equals the stored VR_{i+1}^2. If so, it accepts this authentication request, S stores VR_{i+1}^2 for the next authentication session, and the next process is executed.

4. $S \rightarrow U$: $VR_i^2 \oplus VR_{i+1}^3$.
5. U verifies the received data. If the verification succeeds, U authenticates S and replaces the stored VR_i^1 with VR_{i+1}^1.

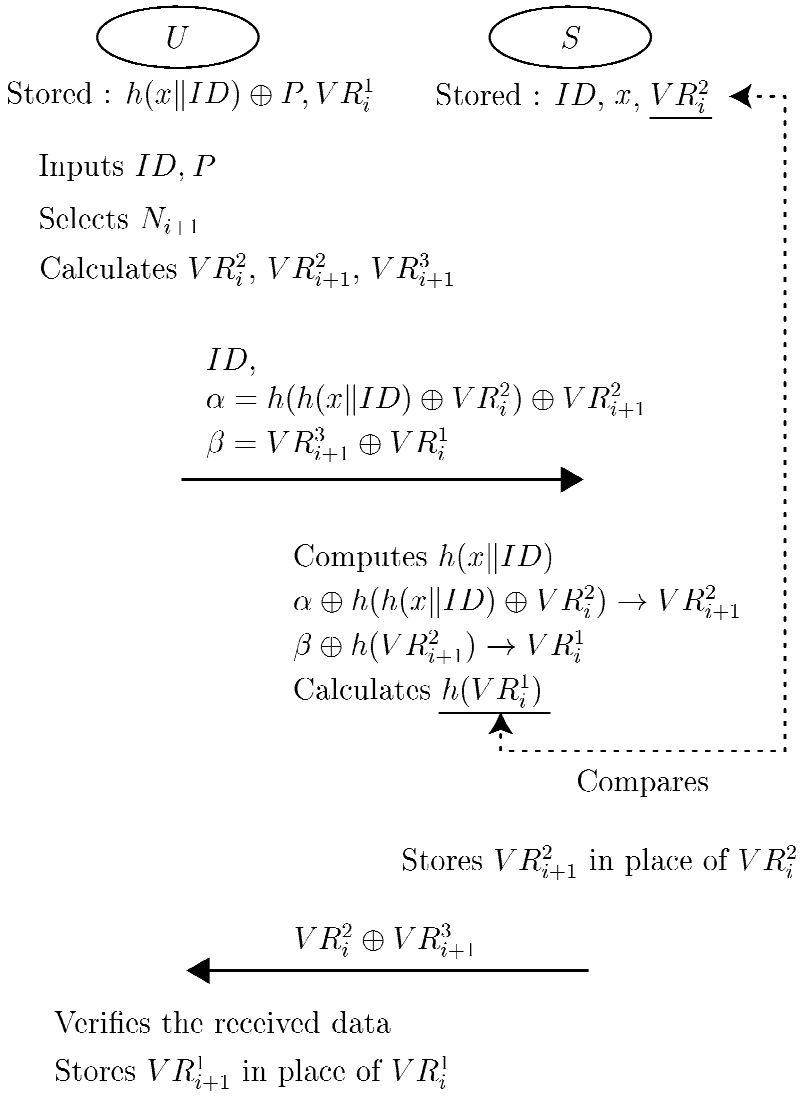

Fig. 3.4 The ith authentication phase of the ROSI

Theft Attack on the ROSI

ROSI is also vulnerable to theft attacks. In ROSI, the S's secret key x is strongly protected, but x is stolen by an attacker A because S usually connects to the Internet and A can crack and invade S. If A can steal x, she/he can impersonate U as follows.

A disturbs U's communication and intercepts the following data in the ith authentication session.

$$\alpha_i = h(h(x||ID) \oplus VR_i^2) \oplus VR_{i+1}^2 \ ,$$
$$\beta_i = VR_{i+1}^3 \oplus VR_i^1 \ ,$$
$$\text{and } ID.$$

Then A calculates $h(h(x||ID) \oplus VR_i^2)$ using stolen x and VR_i^2, extracts VR_{i+1}^2 by $\alpha_i \oplus \{h(h(x||ID) \oplus VR_i^2)\}$, and extracts VR_i^1 by $\beta_i \oplus h(VR_{i+1}^2)$. Next A calculates and substitutes the following data for y, which is the A's secret key.

$$\alpha_i' = h(h(x||ID) \oplus VR_i^2) \oplus (VR_{i+1}^2)' \ ,$$
$$\beta_i' = (VR_{i+1}^3)' \oplus VR_i^1 \ ,$$
$$\text{and } ID,$$

where

$$(VR_{i+1}^2)' = h(h(y||N_{i+1}')) \ ,$$
$$(VR_{i+1}^3)' = h((VR_{i+1}^2)') \ .$$

At that time, S authenticates U, changes the next verifier to $(VR_{i+1}^2)'$, and receives $(VR_{i+1}^3)' \oplus VR_i^2$ to U. Then A disturbs S's communication, and replaces $(VR_{i+1}^3)' \oplus VR_i^2$ with $VR_{i+1}^3 \oplus VR_i^2$. Thus a mutual authentication can succeed, too. After that successful attack, A can impersonate U from the $(i+1)$th authentication session. For example, A sends the following data in the $(i+1)$th authentication session.

$$\alpha_{i+1} = h(h(x||ID) \oplus (VR_{i+1}^2)') \oplus (VR_{i+2}^2)' \ ,$$
$$\beta_{i+1} = (VR_{i+2}^3)' \oplus (VR_{i+1}^1)' \ ,$$
$$\text{and } ID,$$

where

$$(VR^1_{i+1})' = h(y||N'_{i+1}) \ ,$$
$$(VR^2_{i+2})' = h(h(y||N'_{i+2})) \ ,$$
$$(VR^3_{i+2})' = h((VR^2_{i+2})') \ .$$

Similarly, A can send data such as the above to be authenticated from the $(i+2)$th authentication session onwards. Therefore A can impersonate the user.

3.1.3 Cryptanalysis on Yeh-Shen-Hwang's Scheme

Some authentication schemes apply public-key cryptosystems. Recently, user authentications are applied to mobile phones and smart cards, which have low processing speed. Then it is impractical for applying public-key cryptosystems, which involve complicated computations. L. Lamport has proposed a one-time password authentication scheme [1], which requires a different password in each transaction. This scheme uses a conventional hash function, which is easier to compute than pubic-key cryptographies.

The S/Key one-time password authentication system [5], which is based on the Lamport's scheme, has been proposed. However, the S/Key system has two problems. First, the system has a security weakness because the system decreases a counter by one on every login and uses a unique seed value. Second, the user cannot identify the server. For those counters, C. J. Mitchell and L. Chen have proposed two possible solutions: storing a seed value in the user and verifying the server's challenge by using the server's public-key [14]. As well, the S/Key system suffers from vulnerability to off-line dictionary attack. To counter this, S.-M. Yen and K.-H. Liao have proposed a scheme [15] which stores the seed value in a shared tamper resistant device, such as a smart card.

T.-C. Yeh *et al.* have pointed out security problems of the S/Key scheme and have proposed a scheme [24], which overcomes those flaws mentioned above and can

be applied to smart cards. I refer to this variant of the S/Key scheme Yeh-Shen-Hwang's scheme in this thesis. However, one-time password authentication schemes suffer from vulnerability to certain kinds of attacks: the replay attack [6], the denial of service attack [21], the forgery attack [26], and the stolen-verifier attack [25], [28]. Here, I discuss security problems of Yeh-Shen-Hwang's scheme using forgery attacks and stolen-verifier attacks [36], [43].

Yeh-Shen-Hwang's Scheme

T.-C. Yeh *et al.* have proposed a scheme based on the famous S/Key scheme, which can be applied to smart cards. Herein Yeh-Shen-Hwang's scheme is explained.

Definitions and Notations

The following definitions and notations are used throughout this subsection.

- U refers to the computer user who employs the protocol for authentication.
- S refers to the server that authenticates users.
- K is the user's secret key/password.
- H is a conventional hash function. For example, $H(x)$ means x is hashed once.
- N is the maximum allowable number of login attempts.
- $C(= N - t)$ is an initial counter value.
- t is an integer indicating the number of authentication sessions.
- $SEED, D$ represent random numbers.
- $\oplus$ represents a bit-wise exclusive-or (XOR) operation.
- p_t denotes $H^{N-t}(K \oplus SEED)$, where $N - t$ is the number of hash iterations. For example, $H^3(K \oplus SEED)$ denotes $H(H(H(K \oplus SEED)))$.
- The expression '$s \rightarrow t$: u' represents the fact that s sends u to t.

3.1 Stolen-Verifier Attacks on Existing Schemes

Yeh-Shen-Hwang's scheme has three stages: the registration stage, the login stage, and the authentication stage. The registration stage is performed only once, and the login stage and the authentication stage are executed every time the user logs in to the system. These three stages are described below.

Registration Stage

In the registration stage, Yeh-Shen-Hwang's scheme operates as follows.

1. $U \leftarrow S$: $SEED$.
2. $U \leftarrow S$: N , $SEED \oplus D$, $H(D)$.
3. $U \rightarrow S$: $p_0 \oplus D$.

Initially, S issues a smart card containing a pre-shared secret $SEED$ to U, where $SEED$ is a large random number generated by S. Then S generates a random number D, calculates $SEED \oplus D$ and $H(D)$, and sends those data with N to U. After receiving them, U extracts D by $(SEED \oplus D) \oplus SEED$. D is then hashed and compared with the received $H(D)$. If it is correct, U is convinced of S's identity. Then U calculates $p_0 = H^N(K \oplus SEED)$, XORs p_0 with D, and then sends its data to S.

Login Stage

To login, U executes as follows. In the tth login, U has her/his smart card containing $SEED$, and S is storing p_{t-1} with the counter $C(= N - t)$.

1. $U \leftarrow S$: C, $SEED \oplus D$, and $H(D) \oplus p_{t-1}$.
2. $U \rightarrow S$: $p_t \oplus D$.

For the tth login, S generates a random number D, calculates $SEED \oplus D$ and $H(D) \oplus p_{t-1}$, and then sends those data to U. After receiving them, U retrieves D by

$(SEED \oplus D) \oplus SEED$, $H(D)$ by $(H(D) \oplus p_{t-1}) \oplus p_{t-1}$ using $p_{t-1} = H^{N-(t-1)}(K \oplus SEED)$. Then U calculates $H(D)$ and compares it to the retrieved $H(D)$. If they match, S is authenticated, and then U calculates and sends $p_t \oplus D$ using $p_t = H^{N-t}(K \oplus SEED)$ to S.

Authentication Stage

When S receives $p_t \oplus D$, the system XORs the received data and D to retrieve p_t. Next the retrieved data is hashed once and is compared with the stored p_{t-1}. If they match, U is authenticated. Finally, S updates p_t with C.

Attack on Yeh-Shen-Hwang's Scheme

Yeh-Shen-Hwang's scheme is also vulnerable to a hybrid attack combining the forgery attack and the stolen-verifier attack. Its procedure is shown below.

First, an attacker A intercepts the following data for the $(i-1)$th login.

$$\begin{aligned} U \leftarrow S &: N-(i-1),\ \ SEED \oplus D_{i-1}, \\ &\quad H(D_{i-1}) \oplus p_{i-2}. \\ U \rightarrow S &: p_{i-1} \oplus D_{i-1}. \end{aligned}$$

Then S stores p_{i-1} in place of p_{i-2} if $H(p_{i-1}) = p_{i-2}$. Next, A steals p_{i-1} in S, retrieves $SEED$ by $(SEED \oplus D_{i-1}) \oplus (p_{i-1} \oplus D_{i-1}) \oplus p_{i-1}$.

In the ith login, A disturbs and intercepts the following communication.

$$\begin{aligned} U \leftarrow S &: N-i,\ SEED \oplus D_i\ ,\ H(D_i) \oplus p_{i-1}. \\ U \rightarrow S &: p_i \oplus D_i. \end{aligned}$$

Then A retrieves D_i by $(SEED \oplus D_i) \oplus SEED$, extracts p_i by $(p_i \oplus D_i) \oplus D_i$. Next A sends $p_i \oplus D_i \oplus X$ in place of $p_i \oplus D_i$ to S, where X is a arbitrary number generated by A. After receiving $p_i \oplus D_i \oplus X$, S rejects U.

When U is rejected, U sends the service request again. Then S sends the following data to U for U's request.

$$N - i,\ SEED \oplus D'_i\ ,\ H(D'_i) \oplus p_{i-1}.$$

Then A disturbs those data, retrieves D'_i by $(SEED \oplus D'_i) \oplus SEED$, calculates $H(D'_i) \oplus p_i$ by using retrieved D'_i and the extracted p_i. Next A sends the following data to U.

$$N - (i+1),\ SEED \oplus D'_i\ ,\ H(D'_i) \oplus p_i.$$

After receiving them, U replies as follows.

$$p_{i+1} \oplus D'_i.$$

Then A disturbs it, extracts p_{i+1}, and sends the following data to S.

$$p_i \oplus D'_i.$$

When S receives it, S verifies p_i and U is authenticated. Then S stores p_i in place of p_{i-1}. In the ith login, S perceives their communications are as follows.

$$\begin{aligned} &U \leftarrow S : N - i,\ SEED \oplus D'_i\ ,\ H(D'_i) \oplus p_{i-1}. \\ &U \rightarrow S : p_i \oplus D'_i. \end{aligned}$$

In the $(i+1)$th login, A can impersonate U as follows.

$$\begin{aligned} A \leftarrow S : &N - (i+1),\ SEED \oplus D_{i+1}, \\ &H(D_{i+1}) \oplus p_i. \\ A \rightarrow S : &p_{i+1} \oplus D_{i+1}. \end{aligned}$$

First, S generates a random number D_{i+1}, calculates $SEED \oplus D_{i+1}$ and $H(D_{i+1}) \oplus p_i$ using the stored $SEED$ and p_i. Then S sends those data with $N-(i+1)$. After receiving them, A retrieves D_{i+1} by $(SEED \oplus D_{i+1}) \oplus SEED$, creates and sends $p_{i+1} \oplus D_{i+1}$ using the extracted p_{i+1}. After receiving $p_{i+1} \oplus D_{i+1}$, S verifies p_{i+1}. It will be true and A will be authenticated. Therefore, A can impersonate U.

3.1.4 Cryptanalysis on Lin-Chang Authentication Scheme

Many authentication schemes have been proposed, and the Lamport's scheme [1] and the S/Key [5], [6] are representation of one-time password authentication schemes appiled hash functions. For mobile commerce, T.-C. Yeh *et al.* have proposed an authentication scheme using smart cards [24], which solves some kinds of attacks. Though, T.-C. Yeh *et al.*'s scheme does not satisfy the low computation requirement. In 2004, M.-H. Lin and C.-C. Chang have proposed a one-time password authentication scheme [39], in which the user entrusts almost hash operations to the server. However, the Lin-Chang scheme is vulnerable a stolen-verifier attack.

Lin-Chang Scheme

Lin-Chang scheme is a one-time password authentication protocol. In this subsection, this scheme is illustrated.

Definitions and Notations

The following definitions and notations are used throughout this subsection.

- U refers the user.
- S refers to the server that authenticates users.
- $||$ denotes concatenation.
- $\oplus$ represents a XOR operation.
- h is a cryptographic hash function. $h(x)$ means x is hashed once, and $h^2(x)$ means x is hashed twice.
- K is a secret key is generated by U.
- $SEED$ is an unique secret and is a large random number generated by S.

- SK is a session key, equals $D||T$, where D is a random number and T is a timestamp.
- IK is an initial key, equals $K \oplus SEED$.
- N is the number of logins.
- p is an initial password, is calculared as $p_i = h^{N-i}(IK)$
- The expression '$s \rightarrow t$: u' represents the fact that s sends u to t.

Registration Stage

For access to the system, the user receives an unique secret, the number of logins, and initial passwords from the server and stores those data in her/his smart card. The registration process is below.

1. S issues a smart card containing $SEED$ to U.
2. S generates SK and computes $SEED \oplus SK$.
3. $U \leftarrow S$: $SEED \oplus SK$.
4. U extracts SK by XORing the received data and the stored $SEED$ and generates K. Next, U computs IK, and stores it. Then, U decides on N, and computes $IK \oplus SK$, $N \oplus SK$.
5. $U \rightarrow S$: $IK \oplus SK$, $N \oplus SK$.
6. S extracts IK and N from the received data and computes p_0, p_1, and p_2. Then, S stores p_0 with N.
7. $U \leftarrow S$: $p_0 \oplus SK$, $p_1 \oplus SK, p_2 \oplus SK$.
8. U obtains p_0, p_1, and p_2 from the received data. Then, U computes the hashed N times of IK and compares with p_0. If the two values are equivalent, U can be sure of the authenticity of S and stores p_0, p_1, p_2, and N in her/his smart card.

Login and Authentication Stage

To log in, the user executes the tth login and authentication stage. The login and authentication process is below.

1. S generates SK and computes $SEED \oplus SK$. Next, S computes p_{t-1} and $p_{t-1} \oplus SK$.
2. $U \leftarrow S$: $p_{t-1} \oplus SK$, $SEED \oplus SK$.
3. U extracts SK by XORing the received $p_{t-1} \oplus SK$ and the stored p_{t-1}. Then, U checks T of the session key SK. If the timestamp is valid, U obtains $SEED$ from the received data. If the obtained data is the same as the stored $SEED$ in her/his smart card, U can be certain of S's identity and computes $p_t \oplus SK$ with the stored p_t.
4. $U \rightarrow S$: $p_t \oplus SK$.
5. S extracts p_t and computes the hash value of p_t. If the hashed value and p_{t-1} are equivalent, S can be sure of U's identity. Next, S updates N with $N - t$ and computes p_{t+1} using p_0 of U. S performs the XOR operation on p_{t+1} and SK
6. $U \leftarrow S$: $p_{t+1} \oplus SK$.
7. U obtains p_{t+1} from the received data and stores it in her/his smart card for the next login.

Stolen-Verifier Attack on Lin-Chang Scheme

Before the tth authentication session, the attacker steals p_{t-1} from server's database. Next, the attacker intercepts communication data in the tth authentication session. Then the attacker can obtain the tth session key SK_t by XORing the stolen p_{t-1} and the intercepted $p_{t-1} \oplus SK_t$. Similarly, the attacker can get initial passwords p_t, p_{t+1} using the obtained SK_t and the intercepted data: $p_t \oplus SK$ and $p_{t+1} \oplus SK$.

In the $(t+1)$th authentication session, the server sends $p_t \oplus SK_{t+1}$, $SEED \oplus SK_{t+1}$.

Then, the attacker obtains SK_{t+1} by XORing the gotten p_t the received $p_t \oplus SK_{t+1}$. Next, the attacker computes and sends $p_{t+1} \oplus SK_{t+1}$. Then, the attacker authenticated the server because the hash value of p_{t+1} is the same as p_t. Thus, the stolen-verifier attack succeeds.

3.2 Authentication Scheme against Impersonation Attacks

In the SAS method [19], the user uses a one-way function five times. However, this function has high overhead, because a one-way function applies hash functions or common-key cryptosystems. It's desirable to reduce the number of times this function is used in the SAS method because authentication methods are now being applied to mobile communications. Accordingly, the authors have proposed the SAS-2 (SAS ver.2) [23], [27] which reduces such overhead. H.Y. Chien and J.K. Jan have proposed the ROSI (RObust and SImple authentication protocol) [28], which eliminates the stolen-verifier problem. In a stolen-verifier problem, an attacker steals the user's verifier from the server. In the same manner, the attacker can steal other data from the user's personal information. Here, 2GR (Two-Gene-Relation password authentication protocol), which eliminates such problem without increasing the processing load, is proposed.

3.2.1 2GR Protocol

In existing password authentication methods, the server can calculate the user's authentication data because the server stores the user's verifiers and unties received masking data with the verifiers. Therefore, those methods are vulnerable to theft attacks. 2GR eliminates such problems by using the verifiers that are created by no one but user. This method can't do a mutual authentication because if the user can au-

thenticate the server, an attacker can impersonate the server using a technique such as a stolen-verifier attack.

The 2GR has two phases: the registration phase and the authentication phase. The registration phase is performed only once, and the authentication phase is executed every time the user logs in to the system. These two phases are described below.

Definitions and Notations

The following definitions and notations are used throughout this section.

- U refers the computer user who employs the protocol for authentication.
- S refers to the server that authenticates users.
- ID is the user's identity.
- P is the user's password.
- h is one-way hash function. 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.
- $\oplus$ represents a bitwise exclusive OR (XOR) operation.
- $||$ denotes concatenation.
- G_i denotes $h(ID, P, N_i)$. Then G_i is calculated in various ways using ID, P, and N_i. For example, $G_i = h(ID||P \oplus N_i)$.
- D_i denotes $h(G_{i-1}, G_i)$. Then D_i is calculated in various ways using G_{i-1} and G_i. For example, $D_i = h(G_{i-1}||G_i)$.
- The expression '$s \Rightarrow t$: u' represents the fact that s sends u to t through a private and authenticated channel. For example, s delivers to t directly or encrypts and sends u with certification.
- The expression '$s \rightarrow t$: u' represents the fact that s sends u to t through a common

channel such as the Internet.

Registration Phase

Figure 3.5 shows the initial registration phase of 2GR.

1. U inputs the ID and P. Then U generates N_0, N_1, N_2 and stores N_1, N_2. Next U calculates gene data G_0, G_1, and G_2, where

$$G_0 = h(ID, P, N_0) \ ,$$
$$G_1 = h(ID, P, N_1) \ ,$$
$$G_2 = h(ID, P, N_2) \ ,$$

and calculates the relation data D_1 and D_2, where

$$D_1 = h(G_0, G_1) \ ,$$
$$D_2 = h(G_1, G_2) \ .$$

2. U sends ID, G_0, D_1, and D_2 to S.

   In existing schemes, U sends to S through a private and authenticated channel. For example, s delivers to u directly or encrypts and sends communication data with certification. In 2GR, U can send to S through an authenticated channel in which communication data must be certified then 2GR is secure if those data are tapped because U's password and large random numbers are secret. For example, U can send communication data with certification or open to the public such as a certification authority.

3. S stores the received data for subsequent authentication.

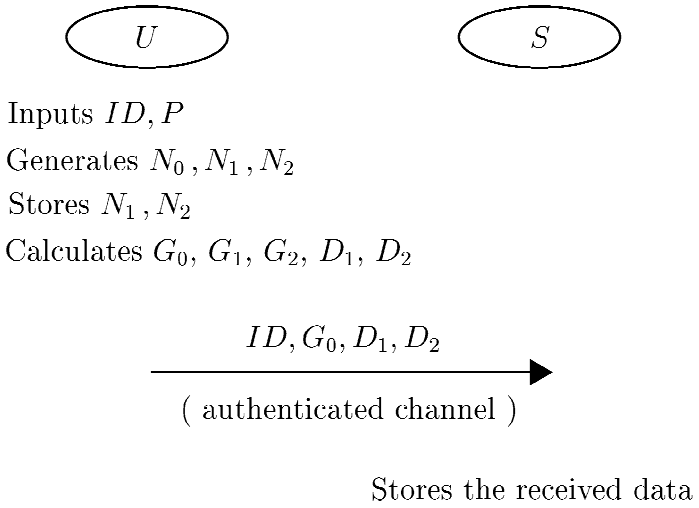

Fig. 3.5 Registration phase of 2GR

Authentication Phase

To log in, U executes the ith authentication session of 2GR protocol. Then U is storing N_i and N_{i+1}, and S stores G_{i-1}, D_i, and D_{i+1} with ID. Figure 3.6 shows the ith authentication phase of 2GR.

1. U inputs ID and P. Then U generates and stores N_{i+2}. Next U calculates G_i, G_{i+1}, G_{i+2}, and D_{i+2}, where

$$\begin{aligned} G_i &= h(ID, P, N_i) \ , \\ G_{i+1} &= h(ID, P, N_{i+1}) \ , \\ G_{i+2} &= h(ID, P, N_{i+2}) \ , \\ D_{i+2} &= h(G_{i+1}, G_{i+2}) \ . \end{aligned}$$

2. $U \rightarrow S$: ID, G_i, and D_{i+2}.
3. S calculates $h(G_{i-1}, G_i)$ using the received G_i. Then S compares the calculated data and the stored D_i. If they match, U is authenticated, and S stores G_i and D_{i+2} in place of G_{i-1} and D_i.

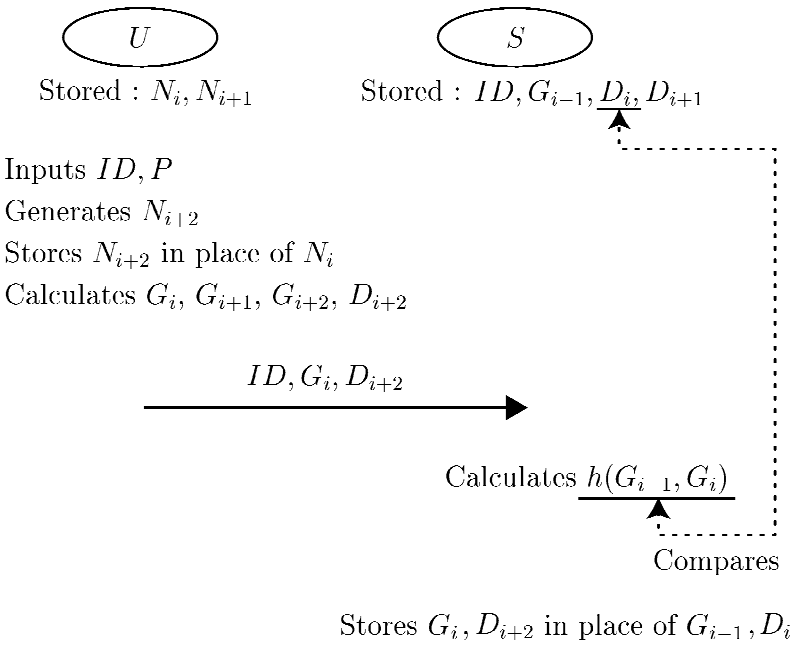

Fig. 3.6 The ith authentication phase of 2GR

3.2.2 Security and Performance Analysis

The 2GR method eliminates previously identified types of attacks and is the most efficient of the one-time password methods.

Security Considerations

One-time password authentication methods are vulnerable to certain kinds of attacks and problem: the replay attack [6], the impersonation attack [26], and the stolen-verifier problem [25], [28]. The 2GR method is secure against these problems.

Replay Attack

2GR throws the authentication data away after use, and those data can't be used after authentication session. 2GR is secure against the replay attack because an attacker can't impersonate if she/he replays the communication data.

Theft Attack

Existing schemes suffer from vulnerability to hybrid impersonation attacks, which is combined the forgery attack and the theft attacks. 2GR eliminates those attacks.

When U tries to be authenticated by S on the ith authentication session, it is assumed that an attacker A has intercepted transmission data and the verifiers stored from S before the ith authentication session. Then the attacker has G_0, G_1, $\cdots$, G_{i-1}, D_1, D_2, $\cdots$, and D_{i+1}. In the ith authentication session, the user sends the following authentication data.

$$\begin{aligned} G_i &= h(ID, P, N_i) \ , \\ D_{i+2} &= h(G_{i+1}, G_{i+2}) \ , \\ &\text{and } ID. \end{aligned}$$

If A takes the offensive, she/he has to send $G'_i = h(ID, P, N'_i)$ in place of G_i. However, A can't change its data because S verifies D_i using the received G_i. Next A tries sending the following data in the ith authentication session.

$$\begin{aligned} G_i &= h(ID, P, N_i) \ , \\ D'_{i+2} &= h(G'_{i+1}, G'_{i+2}) \ , \\ &\text{and } ID. \end{aligned}$$

Then U is authenticated and A can change D_{i+2} with D'_{i+2}. In the next session, A want to send G'_{i+1} in place of G_{i+1}. However, A can't change its data because S verifies D_{i+1} using the received G_{i+1}. Next A tries sending the following data.

$$\begin{aligned} G_{i+1} &= h(ID, P, N_{i+1}) \ , \\ D'_{i+3} &= h(G'_{i+2}, G'_{i+3}) \ , \\ &\text{and } ID. \end{aligned}$$

Then U is authenticated and A can change D_{i+3} with D'_{i+3}. In the $(i+2)$th authentication session, S authenticates the user by comparing the stored D'_{i+2} and calculates

$h(G_{i+1}, G_{i+2})$ using the stored G_{i+1} and the received G_{i+2}. However, those data don't match. Then A can change G_{i+2} with another data. However, S can't successfully perform the certification even if A changes G_{i+2}. Thus A can't impersonate U. Therefore, 2GR is secure.

Server-Modification Attack

If an attacker modifies the verifiers G_{i-1}, D_i, D_{i+1} in S, she/he can impersonate U. However, malicious modification of the verifiers is more unrealistic than theft because the 'writing' is under high restriction than the 'reading' in most of systems. If an attacker wants to impersonate U, she/he needs high authorization than the user to modify the verifiers. When the attacker has higher authoritative than U, S has a problem because the system gives high authorization to the attacker.

Denial of Service Attack

One-time password authentication methods are vulnerable to kind of the denial of service attack [21]. In 2GR method, an attacker can't impersonate the user even if she/he offenses the system using theft attacks. However, an attacker can forge the communication data and modify the verifiers. When an attacker modifies the verifiers, the user is denied by the system. To apply, 2GR needs a countermeasure to the malicious modification.

Performance Considerations

The performance of SAS-2, ROSI, and 2GR is evaluated here. Table 3.1 summarizes performance evaluations of SAS-2, ROSI, and 2GR in the ith authentication phase.

These performance evaluations are done in terms of data storage, number of hashing

Table 3.1 Performance evaluations of SAS-2, ROSI, and 2GR

	SAS-2	ROSI	2GR
Data storage on U	N_i	$h(x\|\|ID) \oplus P, VR_i^1$	N_i, N_{i+1}
Data storage on S	ID, VS_i	ID, VR_i^2	$ID, G_{i-1}, D_i, D_{i+1}$
Number of hashing operations by U	3(4)	4	4
Number of hashing operations by S	1(2)	4	1
Number of XORing operations by U	4	4(5)	0
Number of XORing operations by S	2	3(4)	0
Theft attacks	$\times$	$\times$	$\bigcirc$

(x): x denotes a number using a mutual authentication.

$\times$ means the method is vulnerable to theft attacks.

$\bigcirc$ means the method is endurable to theft attacks.

operations, number of XORing operations, and about theft attacks. The 2GR has to store two or four data in data storages. However, this is not an issue because security, processing speed and load are more important than storage capacity. In number of hashing operations, the 2GR method is the same as SAS-2 and ROSI. Moreover, the 2GR method is far superior to SAS-2 and ROSI in number of XORing operations. In particular, 2GR eliminates theft attacks and has strong security compared with the other methods.

2GR has another characteristic: 2GR can use a low security level channel in the registration phase. In SAS-2 and ROSI, the registration data has to be concealed and can't be tampered with. However, 2GR is secure even if the registration data is tapped.

The S/Key one-time password system [5] is based on Lamport's method and has high security. However, this system has two practical difficulties: high hash overhead and the requirement of resetting the verifier. 2GR has solved those problems without any decrease in the security level.

2GR method proves only the user because an attacker can impersonate the server by stealing server's secrets. However, the mutual authentication is necessary for 2GR method when the method is applied to many systems. Some systems certify the server's right by the server's service responses. Otherwise, 2GR can append a mutual authentication. In 2GR method, the server proves the user by using the password. If 2GR uses other data such as a IP address or times, it is possible that the user can confirm the server.

3.2.3 Algorithm Variations of 2GR

In Sect. 4, 2GR generates D where

$$D_i = h(G_{i-1}, G_i)$$

for the ith authentication session. Then 2GR can use other calculations for D with as follows.

$$\begin{aligned}
D_i &= h(G_{i-1}, G_k) : k > i-1 \ , \\
D_i &= h(G_{i-1}, G_i, \cdots, G_l) : l > i-1 \ , \\
D_i &= h(G_{i-1}, G_i, D_m) : m > i-2 \ , or \\
D_i &= h(G_{i-1}, G_i, D_{i-1}, \cdots, D_n) : n > i-1 \ .
\end{aligned}$$

Those rules are realized without an increase in operations. If 2GR adopts those rules, the number of data stored increases. Table 3.2 summarizes the increase in the number

of data stored in 2GR variations.

Table 3.2 Increase number of data storage in 2GR variations

	user side	server side
rule (1)	$++2$	$++2$
rule (2)	$[++0]$	$++1$
	$++1$	
rule (3)	$++1$	$++1$
rule (4)	$++1$	$++1$

$++x$: x denotes a increase number when k, l, m, or n increases one.

$[++y]$: y denotes a increase number when l increases from i to $i+1$.

However, those rules eliminate attacks such as theft attacks. Moreover, 2GR can create D by using a combination of these rules.

3.3 Authentication Scheme against Denial of Service Attacks

One-time password authentication scheme suffers from denial of service attacks. In this section, a denial of service attack and a solution is discussed.

3.3.1 Impersonation attack on 2GR Protocol

One-time password authentication schemes suffers from stolen-verifier attacks because almost one-time password authentication schemes use verifier for masking communication data. 2GR [40] scheme, in which masking operation is not used, solves impersonation attack using the stolen verifier. However, 2GR suffers from a denial

of service attack easily because the communication data are not masked and are not protected.

2GR Protocol

The 2GR has two phases: the registration phase and the authentication phase. The registration phase is performed only once, and the authentication phase is executed every time the user logs in to the system. These two phases are described below.

Definitions and Notations

The following definitions and notations are used throughout this section.

- U refers the computer user who employs the protocol for authentication.
- S refers to the server that authenticates users.
- ID is the user's identity.
- P is the user's password.
- h is one-way hash function. 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.
- $\oplus$ represents a bitwise exclusive OR (XOR) operation.
- $||$ denotes concatenation.
- G_i denotes $h(ID, P, N_i)$. Then G_i is calculated in various ways using ID, P, and N_i. For example, $G_i = h(ID||P \oplus N_i)$.
- D_i denotes $h(G_{i-1}, G_i)$. Then D_i is calculated in various ways using G_{i-1} and G_i. For example, $D_i = h(G_{i-1}||G_i)$.
- The expression '$s \Rightarrow t$: u' represents the fact that s sends u to t through a private and authenticated channel. For example, s delivers to t directly or encrypts and

sends u with certification.

- The expression '$s \rightarrow t$: u' represents the fact that s sends u to t through a common channel such as the Internet.

Registration Phase

In the registration phase of 2GR, the following operationa are performed.

1. U inputs the ID and P. Then U generates N_0, N_1, N_2 and stores N_1, N_2. Next U calculates gene data G_0, G_1, and G_2, where

$$G_0 = h(ID, P, N_0) \ ,$$
$$G_1 = h(ID, P, N_1) \ ,$$
$$G_2 = h(ID, P, N_2) \ ,$$

and calculates the relation data D_1 and D_2, where

$$D_1 = h(G_0, G_1) \ ,$$
$$D_2 = h(G_1, G_2) \ .$$

2. U sends ID, G_0, D_1, and D_2 to S.

   In 2GR, U can send to S through an authenticated channel in which communication data must be certified then 2GR is secure if those data are tapped because U's password and large random numbers are secret. For example, U can send communication data with certification or open to the public such as a certification authority.

3. S stores the received data for subsequent authentication.

Authentication Phase

To log in, U executes the ith authentication session of 2GR protocol. Then U is storing N_i and N_{i+1}, and S stores G_{i-1}, D_i, and D_{i+1} with ID.

1. U inputs ID and P. Then U generates and stores N_{i+2}. Next U calculates G_i, G_{i+1}, G_{i+2}, and D_{i+2}, where

$$\begin{aligned} G_i &= h(ID, P, N_i) \ , \\ G_{i+1} &= h(ID, P, N_{i+1}) \ , \\ G_{i+2} &= h(ID, P, N_{i+2}) \ , \\ D_{i+2} &= h(G_{i+1}, G_{i+2}) \ . \end{aligned}$$

2. $U \rightarrow S$: ID, G_i, and D_{i+2}.
3. S calculates $h(G_{i-1}, G_i)$ using the received G_i. Then S compares the calculated data and the stored D_i. If they match, U is authenticated, and S stores G_i and D_{i+2} in place of G_{i-1} and D_i.

Cryptanalysis on 2GR Protocol

2GR scheme suffers from a denial of service attack. Its attack is explained below.

When U tries to be authenticated by S on the ith authentication session, it is assumed that an attacker A has intercepted transmission data and the verifiers stored from S before the ith authentication session. Then the attacker has G_0, G_1, $\cdots$, G_{i-1}, D_1, D_2, $\cdots$, and D_{i+1}.

In the ith authentication session, U sends the following authentication data.

$$\begin{aligned} G_i &= h(ID, P, N_i) \ , \\ D_{i+2} &= h(G_{i+1}, G_{i+2}) \ , \\ &\text{and } ID. \end{aligned}$$

A intercepts communication data, which are sent from U. Next, she/he send the following data to S for U.

$$\begin{aligned} G_i &= h(ID, P, N_i) \ , \\ D'_{i+2} &= h(G'_{i+1}, G'_{i+2}) \ , \\ &\text{and } ID. \end{aligned}$$

Then U is authenticated and A can success to change D_{i+2} with D'_{i+2}.

In the next session, U sends the following authentication data.

$$\begin{aligned} G_{i+1} &= h(ID, P, N_{i+1}) \ , \\ D_{i+3} &= h(G_{i+2}, G_{i+3}) \ , \\ &\text{and } ID. \end{aligned}$$

A forges the U's authentication data to the following data.

$$\begin{aligned} G_{i+1} &= h(ID, P, N_{i+1}) \ , \\ D'_{i+3} &= h(G'_{i+2}, G'_{i+3}) \ , \\ &\text{and } ID. \end{aligned}$$

Then U is authenticated and A can change D_{i+3} with D'_{i+3}.

In the $(i+2)$th authentication session, U sends the following authentication data.

$$\begin{aligned} G_{i+2} &= h(ID, P, N_{i+1}) \ , \\ D_{i+4} &= h(G_{i+3}, G_{i+4}) \ , \\ &\text{and } ID. \end{aligned}$$

Then S stores G_{i+1}, D'_{i+2}, and D'_{i+3}, and she/he receives U's data. Next, S calculates $h(G_{i+1}, G_{i+2})$ using the stored G_{i+1} and the received G_{i+2}. The calculated data and the stored D'_{i+2} are different, and U is denied the service. A cannot impersonate U, but she/he can disturb U's service easily.

3.3.2 An Improvement of 2GR Protocol

The 2GR scheme suffers from denial of service attack. An improved 2GR solves its attack. The improved scheme has two phases: the registration phase and the authentication phase. The registration phase is performed only once, and the authentication phase is executed every time the user logs in to the system. These two phases are described below.

Definitions and Notations

The following definitions and notations are used throughout this section.

- U refers the computer user who employs the protocol for authentication.
- S refers to the server that authenticates users.
- ID is the user's identity.
- P is the user's password.
- h is one-way hash function. 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.
- $\oplus$ represents a bitwise exclusive OR (XOR) operation.
- $||$ denotes concatenation.
- G_i denotes $h(ID, P, N_i)$. Then G_i is calculated in various ways using ID, P, and N_i. For example, $G_i = h(ID||P \oplus N_i)$.
- D_i denotes $h(G_{i-1}, G_i)$. Then D_i is calculated in various ways using G_{i-1} and G_i. For example, $D_i = h(G_{i-1}||G_i)$.
- V_j denotes the jth verifier, which is created using server's strong secret key.
- The expression '$s \Rightarrow t$: u' represents the fact that s sends u to t through a private and authenticated channel. For example, s delivers to t directly or encrypts and

sends u with certification.

- The expression '$s \rightarrow t$: u' represents the fact that s sends u to t through a common channel such as the Internet.

Registration Phase

Figure 3.7 shows the initial registration phase of improved 2GR.

1. U inputs the ID and P. Then U generates N_0, N_1, N_2 and stores N_1, N_2. Next U calculates gene data G_0, G_1, and G_2, where

$$G_0 = h(ID, P, N_0) \ ,$$
$$G_1 = h(ID, P, N_1) \ ,$$
$$G_2 = h(ID, P, N_2) \ ,$$

   and calculates the relation data D_1 and D_2, where

$$D_1 = h(G_0, G_1) \ ,$$
$$D_2 = h(G_1, G_2) \ .$$

2. U sends ID, G_0, D_1, and D_2 to S.
3. S stores the received data for subsequent authentication. Then S generates V_1 using S's strong secret key.
4. S sends V_1 to U.
5. U stores the received data.

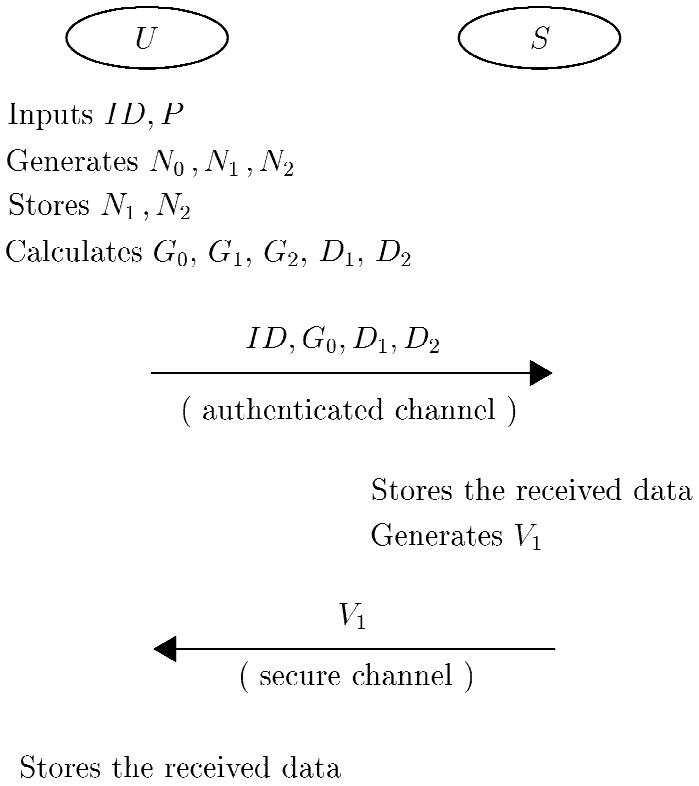

Fig. 3.7 Registration phase of the improved 2GR

Authentication Phase

To log in, U executes the ith authentication session of 2GR protocol. Then U is storing N_i, N_{i+1}, V_j, and S stores G_{i-1}, D_i, D_{i+1} with ID. Figure 3.8 shows the ith authentication phase of improved 2GR.

1. U inputs ID and P. Then U generates and stores N_{i+2}. Next U calculates G_i, G_{i+1}, G_{i+2}, and D_{i+2}, where

$$\begin{aligned} G_i &= h(ID, P, N_i) \ , \\ G_{i+1} &= h(ID, P, N_{i+1}) \ , \\ G_{i+2} &= h(ID, P, N_{i+2}) \ , \\ D_{i+2} &= h(G_{i+1}, G_{i+2}) \ , \\ c_3 &= h(V_j, D_{i+2}) \ . \end{aligned}$$

2. $U \rightarrow S$: ID, $c_1 = G_i$, $c_2 = D_{i+2}$, and c_3.

3. S calculates $h(G_{i-1}, G_i)$ using the received G_i. Then S compares the calculated data and the stored D_i. If they match, U is authenticated. Next, S generates V_j, and calculates $h(V_j, c_2)$. If the calculated data and the received c_3 are the same, c_2 is not forged. Then, S stores G_i and D_{i+2} in place of G_{i-1} and D_i.

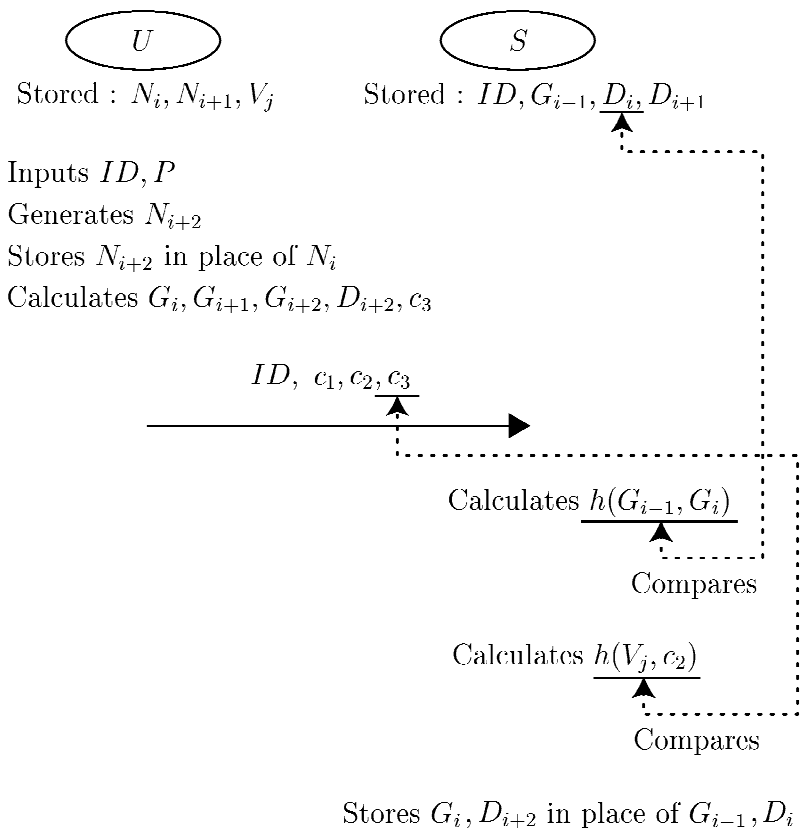

Fig. 3.8 The ith authentication phase of the improved 2GR

3.3.3 Security and Performance Analysis

Improved 2GR method eliminates denial of service attack and is the most efficient of the one-time password schemes.

Security Considerations

One-time password authentication methods are vulnerable to certain kinds of attacks and problem: the replay attack [6], the impersonation attack [26], and the stolen-verifier problem [25], [28]. The improved 2GR has the same security with the 2GR scheme except for the denial of service attack. The 2GR scheme is vulnerable to the denial of service attack. The improved 2GR solves this attack.

Replay Attack

The improved 2GR throws the authentication data away after use, and those data can't be used after authentication session. The proposal scheme is secure against the replay attack because an attacker can't impersonate if she/he replays the communication data.

Theft Attack

When U tries to be authenticated by S on the ith authentication session, it is assumed that an attacker A has intercepted transmission data and the verifiers stored from S before the ith authentication session. Then the attacker has G_0, G_1, $\cdots$, G_{i-1}, D_1, D_2, $\cdots$, and D_{i+1}. In the ith authentication session, U sends the following authentication data.

$$
\begin{aligned}
c_1 &= G_i \ , \\
c_2 &= h(G_{i+1}, G_{i+2}) \ , \\
c_3 &= h(V_j, c_2) \ , \\
&\text{and } ID.
\end{aligned}
$$

If A takes the offensive, she/he has to send $G_i' = h(ID, P, N_i')$ in place of G_i. However,

A can't change its data because S verifies D_i using the received G_i. Next A tries sending the following data in the ith authentication session.

$$\begin{aligned} c_1 &= G_i \ , \\ c_2' &= h(G_{i+1}', G_{i+2}') \ , \\ c_3' &= h(V_j, c_2') \ , \\ &\text{and } ID. \end{aligned}$$

A can forge c_2 with c_2', which is calculated by A. However, A do not know S's strong secret key, and she/he cannot create c_3'. In the worst case, S's secret key is stolen by A. Then, A can send the above data, U is authenticated, and A can change D_{i+2} with D_{i+2}'.

In the next session, U sends the following authentication data.

$$\begin{aligned} c_1 &= G_{i+1} \ , \\ c_2 &= h(G_{i+2}, G_{i+3}) \ , \\ c_3 &= h(V_j, c_2) \ , \\ &\text{and } ID. \end{aligned}$$

A want to send G_{i+1}' in place of G_{i+1}. However, A can't change its data because S verifies D_{i+1} using the received G_{i+1}.

$$\begin{aligned} c_1 &= G_{i+1} \ , \\ c_2' &= h(G_{i+2}', G_{i+3}') \ , \\ c_3' &= h(V_j, c_2') \ , \\ &\text{and } ID. \end{aligned}$$

If A know S's secret key, U is authenticated and A can change D_{i+3} with D_{i+3}'.

In the $(i+2)$th authentication session, U sends the following authentication data.

$$\begin{aligned} c_1 &= G_{i+2} \ , \\ c_2 &= h(G_{i+3}, G_{i+4}) \ , \end{aligned}$$

$$c_3 = h(V_j, c_2) \ ,$$
$$\text{and } ID.$$

When receiving those data, S authenticates U by comparing the stored $D'_{i+2} = h(G'_{i+1}, G'_{i+2})$ and calculates $h(G_{i+1}, G_{i+2})$ using the stored G_{i+1} and the received G_{i+2}. However, those data don't match. Then A can change G_{i+2} with another data. However, S is stored G_{i+1} and cannot successfully perform the certification even if A changes G_{i+2}. Thus A can't impersonate U, and the improved 2GR is secure. In this case, S's secret key is stolen, and the system is useless. U can know that S is insecure.

Server-Modification Attack

If an attacker modifies the verifiers G_{i-1}, D_i, D_{i+1} in S, she/he can impersonate U. However, malicious modification of the verifiers is more unrealistic than theft because the 'writing' is under high restriction than the 'reading' in most of systems. If an attacker wants to impersonate U, she/he needs high authorization than the user to modify the verifiers. When the attacker has higher authoritative than U, S has a problem because the system gives high authorization to the attacker.

Denial of Service Attack

When U tries to be authenticated by S on the ith authentication session, it is assumed that an attacker A has intercepted transmission data and the verifiers stored from S before the ith authentication session. Then the attacker has G_0, G_1, $\cdots$, G_{i-1}, D_1, D_2, $\cdots$, and D_{i+1}. In the ith authentication session, the user sends the following authentication data.

$$c_1 = G_i \ ,$$
$$c_2 = h(G_{i+1}, G_{i+2}) \ ,$$

$$c_3 = h(V_j, c_2) \ ,$$
$$\text{and } ID.$$

If A disturbs U, she/he has to send

$$c_1 = G_i \ ,$$
$$c_2' = h(G_{i+1}', G_{i+2}') \ ,$$
$$c_3' = h(V_j, c_2') \ ,$$
$$\text{and } ID.$$

A can forge c_2 with c_2', which is calculated by A. However, A do not know S's strong secret key, and she/he cannot create c_3'. Therefore, U's authentication request is not denied.

Performance Considerations

The improved 2GR has similar costs to the 2GR scheme. For the security, the costs of the proposed scheme increase in number to the 2GR. In the improved 2GR, U has to store V_j, c_3 is sent with other authentication data, and c_3 has to be calculated by U and S. They are not much costs.

3.4 Conclusion

In this chapter, the stolen-verifier attacks are discussed. The author success attacking [37] to an imprementation [35] of Payravian-Zunic's scheme [18].

Here, the authentication schemes against theft attacks are proposed. Those authentication schemes use one-way functions. Those scheme can apply other cryptographies [46], [53].

Chapter 4

Applications Using Secure Authentication Schemes

In the chapter 2 and 3, simple and secure authentication schemes are proposed. Those schemes can be applied various systems. Particularly, those schemes are simple and have little costs. Thus, mobile and ubiquitous systems using proposal schemes are very useful.

4.1 Application for Key-Free Systems

The SAS(Simple And Secure password authentication protocol) scheme [19] uses a one-way function five times. This function has high overhead. It's desirable to reduce the number of this function, which is used in the SAS scheme, because authentication schemes are now being applied to mobile communications and Internet protocols. SAS-2(Simple And Secure password authentication protocol, ver.2), applying one-way function only three times by using two verifiers and another for masking. This reduces hash overhead is by about 40% in comparison with the SAS.

Here we introduce a key-free system as one type of application systems. These systems open and shut the key using the SAS-2 scheme, 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.

The key-free system 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 when the user opens or shuts a lock such as a door. We describe these two phases below.

4.1.1 Definitions and Notations

The following definitions and notations are used throughout this subsection.

- User is the computer user who employs the protocol for authentication.
- Server is the server that authenticates users.
- ID is the user's identity.
- S 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.

4.1.2 Registration Phase

Figure 4.1 shows the initial registration phase of the key-free system. Then a lock has its own identity ID, the key's identity K, and secret key S.

1. The lock generates and stores a random number N_1, and calculates $A_1 = X(ID \oplus K, S \oplus N_1)$ using the generated N_1 and input data.
2. The lock sends ID, K, and A to the key through a secure channel.

3. The key stores A with ID and K for subsequent authentication.

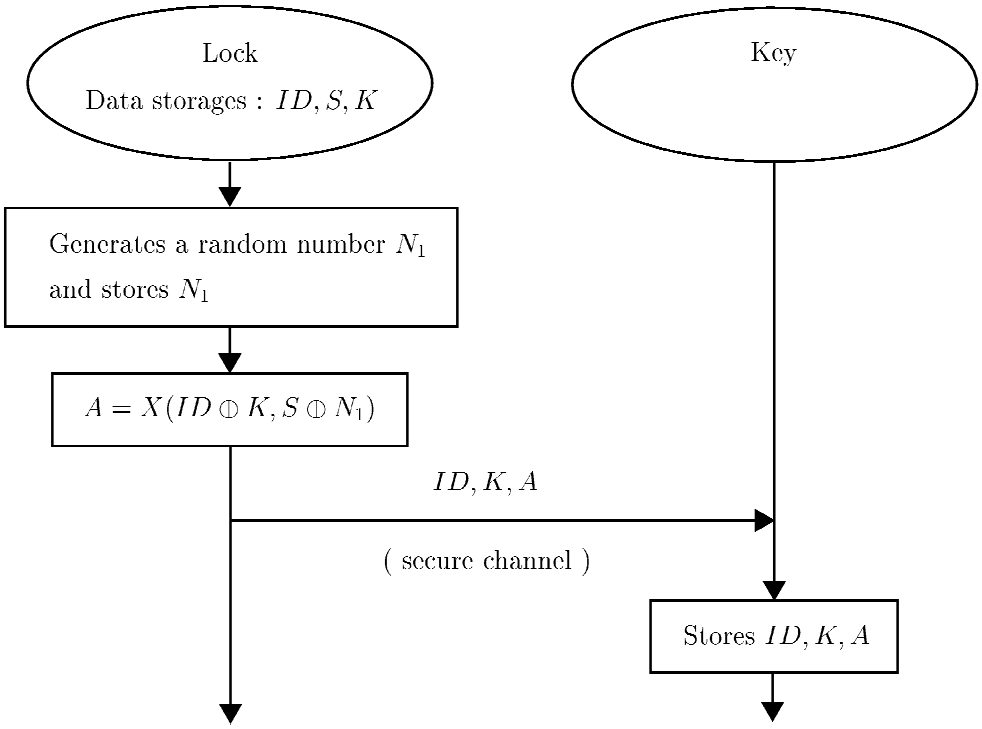

Fig. 4.1 Registration phase of the key-free system.

4.1.3 Authentication Phase

When the user wants to open or shut 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 lock has stored ID, K, S, and N_i. Figure 4.2 shows the ith authentication phase of the key-free system.

1. The key sends the key's identity K to the lock.
2. The lock calculates $A = X(ID \oplus K, S \oplus N_i)$ using stored ID, S, 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 $C = X(ID \oplus K, S \oplus N_{i+1})$ and $F(C) = F(ID \oplus K, C)$ using the generated N_{i+1} and stored data. Then the lock

computes $\alpha = C \oplus (F(C) + A)$ and $\beta = F(C) \oplus A$.

3. The lock sends ID, α and β to the key through a common network such as the Internet.
4. 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 do not match, the lock is rejected, and the key fails to lock. If they match, the lock is authenticated and the next process is executed.
5. The key stores C in place of A for next authentication session and calculates γ equals $F(ID \oplus K, F(C))$.
6. The key sends γ to the lock through a common network.
7. The lock calculates $F(ID \oplus K, F(C))$ and compares the received $\gamma = F(ID \oplus K, F(C))$. If they match, the key is authenticated and the lock opens or shuts. If they do not match, the user tries again.

4.1 Application for Key-Free Systems

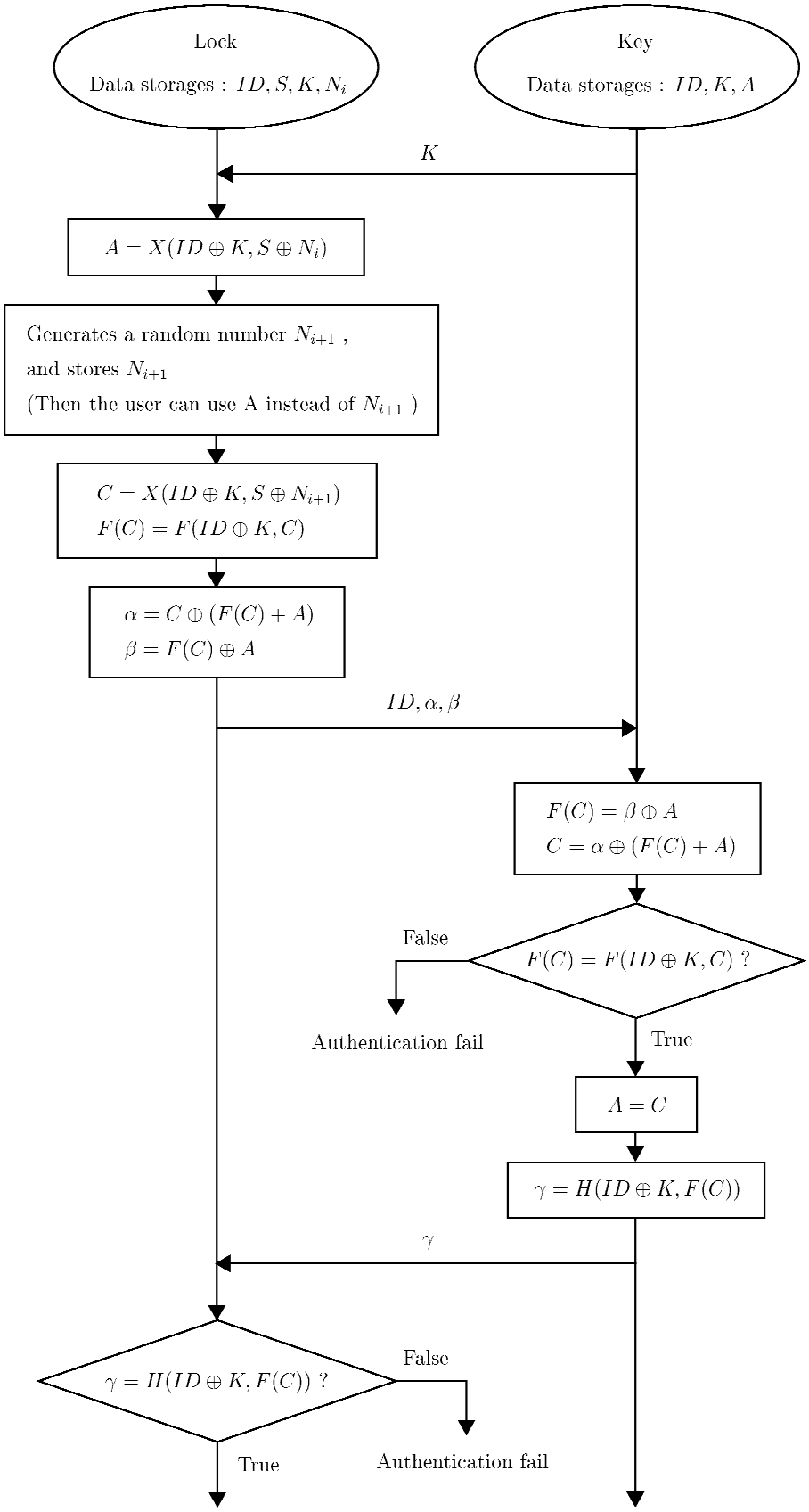

Fig. 4.2 The ith authentication phase of the key-free system.

4.2 A convenient micropayment scheme with multi-vendor

The Internet and mobile communications have been developing, and lots of Internet services are offered, and electronic money systems are used in those services. For circulating such electronic money, several electronic payment schemes [7], [8], [12], [13] are proposed. However, those schemes have calculation costs because they use public-key cryptosystems. This can be accomplished 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 [4] or SHA-1 [17] are employed.

The Millicent scheme [9] uses a hash function and symmetric encryption and doesn't use public-key cryptosystems. This scheme uses scrip, which is similar to a debit card, but customer's secret information is looked furtively from the scrip. Moreover, the vendor can change the sum of money. The MicroMint scheme [13] is based on hash collisions. However, the customer has to store combination of hash collisions, and the attacker can use customer's money by seraching the combination of hash collisions. For the security, the MicroMint scheme has to use secure combinations, but this has calculation costs.

The SAS-Coin (Simple And Secure Coin) [20] based on one-time password authentication protocol SAS (Simple And Secure password authentication protocol) [19] has been proposed. This SAS authentication protocol is superior to the Lamport scheme [1] and S/Key scheme [5] about calculation costs. However, the SAS suffers from vulnerability to some attacks [21]. The SAS-Coin is vulnerable to the impersonation attack by changing the communication data. Moreover, this system covers only one vendor in an account because the system verifies the money data by using a particular verifier. In this section, a convenient payment protocol with multi-vendor SAIFU(Secure

Agreement Identification for Flexible Users) [47] is proposed.

4.2.1 Definitions and Notations

The following definitions and notations are used throughout this paper.

- C refers the customer who employs the protocol for using electronic money.
- B refers to the broker that is same as Internet bank and manages money.
- V refers an vendor who offers some services.
- ID_X is X's identity or login name.
- SM is the sum of money which is necessary to buy V's commodities or to use V's services.
- BM is C's balance.
- AM is supplement when C adds to BM.
- h is a conventional hash function. For example, $h(x)$ means x is hashed once.
- $\oplus$ represents a bitwise exclusive OR (XOR) operation.
- $||$ denotes concatenation.
- m is masking function (e.g. $m(x, y) = x \oplus y$).
- i is an integer indicating the number of payment sessions.
- N_{Xi} represents X's random number corresponding to the ith authentication.
- G_{Xi} denotes $h(N_i)$ calculated by X, where $h(N_i)$ may be calculated from a combination of a random number and other data such as user's password.
- M_iX_Y represents a masking data corresponding to the ith authentication, and X retrieves Y's masked information from this masking data.
- H_iX_Y represents an authentication code corresponding to the ith authentication, and X authenticates Y using this authentication code.
- The expression '$s \Rightarrow t$: C' represents the fact that B sends C to t through a private,

authenticated channel. For example, B delivers to t directly or encrypts and sends C with certification.

- The expression '$s \rightarrow t$: C' represents the fact that B sends C to t through a common channel such as the Internet.

4.2.2 The SAIFU Protocol

The SAIFU protocol consist seven phases: the customer registration phase, the vendor registration phase, the supplement phase, the renewal phase, the payment phase, the direct debit phase, and the synchronous confirmation phase. The customer registration process and the vendor registration process are performed only once, and the supplement phase is executed when the customer want to add to her/his balance. The renewal process is executed when a new vendor joins the service or the direct debit process has been successful. The payment phase is executed every time the customer buys the vendor's commodities or uses the vendor's services. The direct debit process is executed after the payment process, and the synchronous confirmation process may be executed when the customer wants to confirm the broker's stores. These seven phases are described below.

Customer Registration Phase

For use of the service, the customer registers her/his identity and the verifier to the broker. The customer registration process is below.

1. C inputs ID_C and generates N_{C1} and N_{C2}. Next C calculates G_{C1} and G_{C2}. Then C stores G_{C1} and G_{C2} in her/his secret memory such as a smart card.
2. $C \Rightarrow B$: ID_C, G_{C1}, and G_{C2}.
3. B stores G_{C1} and G_{C2} with ID for subsequent authentication. Next B generates

N_{B0}, N_{B1} and calculates G_{B0}, G_{B1}, and $h(G_{C1}, G_{B1})$. Then B stores the calculated data.

4. $B \Rightarrow C$: G_{B0} and $h(G_{C1}, G_{B1})$.
5. C stores the received data.

Vendor Registration Phase

For participation of the service, the vendor registers her/his identity and the verifier. The vendor registration process is below.

1. V generates N_{V1} and calculates G_{V1}. Then V stores the calculated data.
2. $V \Rightarrow B$: ID_V and G_{V1}.
3. B stores the received data.

Supplement Phase

When the customer wants to supply to her/his balance, she/he can transfer money or points into her/his account. C storing G_{Ci}, G_{Ci+1}, G_{Bi-1}, and $h(G_{Ci}, G_{Bi})$, and B is storing G_{Ci}, G_{Ci+1}, G_{Bi-1}, G_{Bi}, G_{Bi+1}, G_{Vi}, $h(G_{Ci}, G_{Bi})$, and $h(G_{Ci+1}, G_{Bi+1})$. The supplement process is below.

1. C calculates

$$M_i B_C = m(G_{Ci+1}, G_{Ci+2}) \ and$$
$$H_i B'_C = h(G_{Ci+1}, G_{Ci+2}, AM).$$

2. $C \rightarrow B$: $AM, M_i B_C$, $H_i B'_C$ with supplement request, when AM may be masked for the security.
3. B retrieves G_{Ci+2} from $m(G_{Ci+1}, G_{Ci+2})$ and calculates $h(G_{Ci+1}, G_{Ci+2}, AM)$.

Then B compares the calculated data and the received $H_iB'_C$. If they are the same, B confirms AM whether AM is the same C's transfer money. If they are the same, B calculates $BM + AM$ and stores the calculated data in place of BM. Next B calculates $h(G_{Ci+2}, BM)$.

4. $B \rightarrow C$: $h(G_{Ci+2}, BM)$.
5. C calculates $h(G_{Ci+2}, BM)$, and the calculated data and the received data. If they are the same, C stores $BM + AM$ in place of BM.

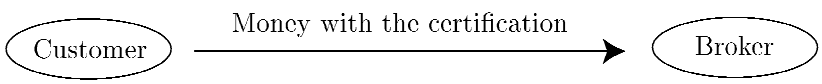

Fig. 4.3 The procedure of the supplement phase.

Renewal Phase

For customer purchase or customer use, the broker registers or renews the authentication data to vendors. Then V is storing G_{Vi}, and B is storing G_{Ci}, G_{Ci+1}, G_{Bi-1}, G_{Bi}, G_{Vi}, $h(G_{Ci}, G_{Bi})$, and BM. The renewal process is below.

1. B generates N_{Bi+1}. Then B calculates and stores the following data.

$$
\begin{aligned}
M_iV_B &= m(h(G_{Vi}, ID_V), BM), \\
M_iC_B &= m(G_{Bi-1}, G_{Bi}), \\
M_iC_V &= m(G_{Vi}, h(G_{Ci}, G_{Bi-1})), \\
M_iC_{CB} &= m(h(G_{Ci}, G_{Bi}), h(G_{Ci+1}, G_{Bi+1})), \\
H_iC_{VB} &= h(G_{Bi}, G_{Vi}, h(G_{Ci+1}, G_{Bi+1}), ID_V), \\
H_iV_C &= h(G_{Ci}, G_{Bi}, G_{Vi}), \ \textit{and} \\
H_iV_B &= h(h(G_{Vi}, ID_V), M_iV_B, M_iC_B, M_iC_V, \\
&\qquad M_iC_{CB}, H_iC_{VB}, H_iV_C).
\end{aligned}
$$

2. $B \rightarrow V$: M_iV_B, M_iC_B, M_iC_V, M_iC_{CB}, H_iC_{VB}, H_iV_C, H_iV_B, with ID_C.
3. V calculates $h(h(G_{Vi}, ID_V), M_iV_B, M_iC_B, M_iC_V,$ $M_iC_{CB}, H_iC_{VB}, H_iV_C)$ and compares the calculated data and the received H_iV_B. If they are the same, B is authenticated. Next V retrieves BM from M_iV_B, and V stores BM and the received data.

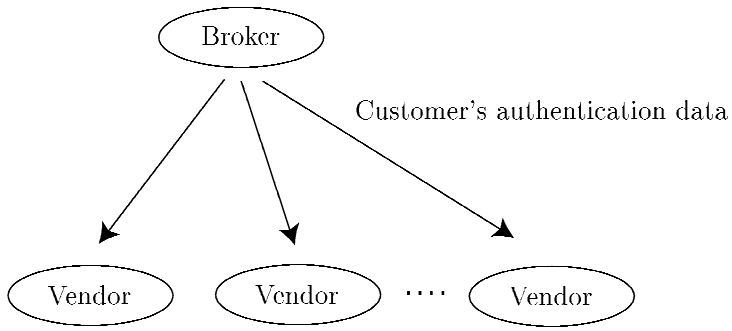

Fig. 4.4 The procedure of the renewal phase.

Payment Phase

To buy commodities or use services, C executes the ith payment session of the SAIFU protocol. Then C is storing G_{Ci}, G_{Ci+1}, G_{Bi-1}, $h(G_{Ci}, G_{Bi})$, and BM, and V is storing G_{Vi}, M_iC_B, M_iC_V, M_iC_{CB}, H_iC_{VB}, H_iV_C, BM, with ID_C.

1. $C \rightarrow V$: Commodities request of Service request with ID_C.
2. V confirms SM whether SM is lower than BM. Then V calculates $H_iC_{VD} = h(h(G_{Vi}, ID_C), SM)$.
3. $V \rightarrow C$: ID_A, M_iC_B, M_iC_V, M_iC_{CB}, H_iC_{VB}, H_iC_{VD} and SM, when SM may be masked for the security.
4. After receiving V's response, C calculates $h(G_{Ci}, G_{Bi-1})$ and retrieves G_{Bi}, G_{Vi}, and $h(G_{Ci+1}, G_{Bi+1})$ from the received M_iC_B, M_iC_V, M_iC_{CB}. Next, C calculates

$h(G_{Bi}, G_{Vi}, h(G_{Ci+1}, G_{Bi+1}), ID_V)$ and compares the calculated data and the received H_iC_{VB}. If they are the same, V and B are authenticated. Then C calculates $h(h(G_{Vi}, ID_V), SM)$ and compares the calculated data and the received H_iC_{VD}. If they are the same, C calculates the following data.

$$\begin{aligned} & h(G_{Ci}, G_{Bi}, G_{Vi}), \\ M_iB_C &= m(G_{Ci+1}, G_{Ci+2}), \\ H_iB_C &= h(G_{Ci+1}, G_{Ci+2}, SM), \ \textit{and} \\ H_iV_{CD} &= h(h(G_{Vi}, ID_C), M_iB_C, H_iB_C, SM). \end{aligned}$$

5. $C \rightarrow V$: $h(G_{Ci}, G_{Bi}, G_{Vi})$, M_iB_C, H_iB_C, and H_iV_{CD}.
6. V compares the storing H_iV_C and the received $h(G_{Ci}, G_{Bi}, G_{Vi})$. If they are the same, C is authenticated. Next, V calculates $h(h(G_{Vi}, ID_C), M_iB_C, H_iB_C, SM)$, and compares the calculated data and the received H_iV_{CD}. If they are the same, the received data M_iB_C and H_iB_C are not forged, and V stores them.

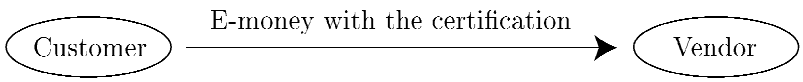

Fig. 4.5 The procedure of the payment phase.

Direct Debit Phase

For the next payment phase, V directs debit the money. Then V storing ID_V, ID_C, M_iB_C, H_iB_C, BM, SM, and B is storing G_{Ci}, G_{Ci+1}, G_{Bi-1}, G_{Bi}, G_{Bi+1}, G_{Vi}, $h(G_{Ci}, G_{Bi})$, and $h(G_{Ci+1}, G_{Bi+1})$, and BM. The direct debit process is below.

1. V generates N_{Vi+1} and calculates G_{Vi+1},

$$m(h(G_{Vi}), SM),$$

$$M_iB_V = m(G_{Vi}, G_{Vi+1}),\ and$$
$$H_iB_V = h(h(G_{Vi+1}), M_iB_C, H_iB_C).$$

2. $V \rightarrow B$: ID_V, ID_C, M_iB_V, $m(h(G_{Vi}), SM)$, H_iB_V, M_iB_C, H_iB_C, and SM, when SM may be masked for the security.
3. First, B retrieves G_{Vi+1} from the received M_iB_V. Next, B calculates $h(h(G_{Vi+1}), M_iB_C, H_iB_C)$ and compares the calculated data and the received H_iB_V. If they are the same, V is authenticated. Next B calculates $m(h(G_{Vi}), SM)$ and confirms SM whether SM is lower than BM. Then B retrieves G_{Ci+2} from the received M_iB_C, and authenticates the retrieved data by comparing the calculated $h(G_{Ci+1}, G_{Ci+2})$ and the received H_iB_C. If they are the same, V's and C's verifiers are stored, and $BM - SM$ is stored in place of BM. Then B calculates $h(G_{Vi}, G_{Vi+1})$.
4. $B \rightarrow V$: $h(G_{Vi}, G_{Vi+1})$.
5. V compares the calculated $h(G_{Vi}, G_{Vi+1})$ and the received data. If they are the same, B's mutual authentication is succeed, and $BM - SM$ is stored in place of BM.

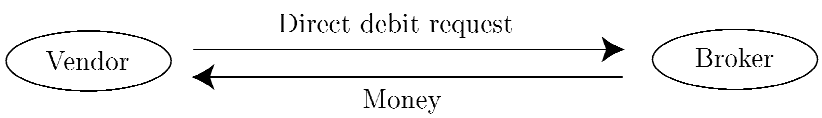

Fig. 4.6 The procedure of the direct debit phase.

Synchronous Confirmation Phase

For the next payment phase, the customer can confirm her/his verifiers, which is stored in the broker. Then C is storing G_{Ci}, G_{Ci+1}, G_{Ci+2}, G_{Bi-1}, G_{Bi}, G_{Vi},

$h(G_{Ci}, G_{Bi})$, $h(G_{Ci+1}, G_{Bi+1})$, M_iB_C, H_iB_C, H_iC_{VD}, BM, SM, and B is storing G_{Ci}(or G_{Ci+2}), G_{Ci+1}, G_{Bi-1}, G_{Bi}, G_{Bi+1}, G_{Vi} (or G_{Vi+1}), $h(G_{Ci}, G_{Bi})$, $h(G_{Ci+1}, G_{Bi+1})$, BM, SM. The synchronous confirmation process is below.

1. $C \rightarrow B$: ID_C, M_iB_C, H_iB_C, and SM, when SM may be masked for the security.
2. B checks the receiving data using the stored C's verifiers G_{Ci}, G_{Ci+1} (or G_{Ci+1}, G_{Ci+2}); B computes $h(m(M_iB_C, G_{Ci}), G_{Ci}, SM)$ (or $h(m(M_iB_C, G_{Ci+1}), G_{Ci+1}, SM)$) and compares the computed data and the received H_iB_C. If they are the same, C's verifiers are renewed, and B calculates $h(G_{Bi}, G_{Vi})$.
3. $B \rightarrow C$: $h(G_{Bi}, G_{Vi})$.
4. C compares the calculated $h(G_{Bi}, G_{Vi})$ and the received data. If they are the same, C's verifier is confirmed, and C removes the stored G_{Ci}, G_{Bi-1}, $h(G_{Ci}, G_{Bi})$, M_iB_C, H_iB_C, and SM. Then C stores $BM - SM$ is stored in place of BM. If this confirmation is failed, C sends ID_A and H_iC_{VD} to B, and deprives V's confidence.

4.2.3 A Practical Billing Protocol

The customer can bill more flexible without through the broker in the payment phase.

If the customer want to pay more faster, she/he can use customer-vendor billing protocol in which another balance BM' is used between the customer and the vendor for reducing the communication costs and synchronous costs of authentication data such as the customer's balance. This protocol consists only two phases: the supplement phase and the payment phase. The supplement phase is executed when the customer want to add to her/his balance, and the payment phase is executed every time the customer buys the vendor's commodities or uses the vendor's services. These two phases are described below.

4.2 A convenient micropayment scheme with multi-vendor

In the supplement phase, the procedures of the above Payment Phase of SAIFU protocol are executed. Then C pays BM to V, and C and V stores $BM' + BM$ in place of BM'. Next C's G_{Cj} is shared between C and V, where j is an integer indicating the number of payment sessions between C and V.

To buy commodities or use services, C executes the jth payment session of the customer-vendor billing protocol. Then C and V are storing G_{Cj} and BM'.

1. C calculates G_{Cj+1},

$$M_jV_C = m(G_{Cj}, G_{Cj+1}),\ and$$
$$H_jV_C = h(G_{Cj}, G_{Cj+1}, SM).$$

2. $C \to V$: ID_C, M_jV_C, H_jV_C, and SM.
3. V retrives G_{Cj+1} from M_jV_C. Then V calculates $h(G_{Cj}, G_{Cj+1}, SM)$ and compares the calculated data and the received H_jV_C. If they are the same, V stores $BM' - SM$ in place of BM' and calculates $h(G_{Cj+1}, BM')$ using the calculated BM'.
4. $V \to C$: $h(G_{Cj+1}, BM')$
5. C calculates $BM' - SM$ and $h(G_{Cj+1}, BM' - SM)$. Next C compares the calculated data and the received data. If they are the same, C stores $BM' - SM$ in place of BM'

In the customer-vendor billing protocol can be used without money data BM' and SM if the customer prepay the money such as the monthly fees.

4.2.4 Security and Performance Analysis

The proposal payment scheme SAIFU is secure and useful.

Security Considerations

The attacker can't retrieve secret data in the SAIFU scheme because communication data are masked or hashed using the secret shared data. Moreover, the attacker can't replay communication data because the same verification data aren't sent.

In the Millicent [9], customer's secret information is looked furtively from the scrip. The SAIFU scheme masks the customer's secrets with the secret shared data between the customer and the broker, and those secrets aren't leaked out other persons.

The MicroMint scheme [13] uses combinations of the hash collisions for the money, and the combinations can be calculated others. The SAIFU scheme communicates hash value of the money with the secret shared data, and the attacker can't create those communication data.

In the SAS-Coin [20], communication data can be forged because the secret data is masked with the same data. The SAIFU scheme employs different secret data for masking, and the attacker can't forge the secret data and money data.

Performance Considerations

The SAIFU scheme doesn't symmetric cryptosystems for encrypting communication data such as the Millicent [9], and stores few data than the MicroMint scheme [13]. Moreover, the SAIFU hash costs are remarkably lower than the costs of the SAS-Coin [20] because the SAIFU doesn't use the coin which is generated by hash iterations.

In the SAIFU scheme, the same secret data can be used for plural vendors, and

the customer doesn't need vendor's database. Moreover, the customer can confirm the synchronous of the secret shared data, and can disclose an injustice vendor.

4.3 Conclusion and Other Applications

The SAS-2 authentication scheme [42], [45] is very simple and can be applied to key free systems [42]. The key-free system has synchronous problem, but the synchronous problem can be solved using the past verifier [50], [51]. Moreover, the key-free system become more convenience by managing spare key [48]. The SAS-2 scheme can be applied other systems: a billing system [30], cncryption communicaion systems [31], a simultaneous multiple access system [32], and wireless LAN systems [54].

The SAS-3 (Simple And Secure password authentication protocol, version 3) [41] is an improvement of the SAS-2. The SAIFU(Secure Agreement Identification for Flexible Users) [44] is applied the SAS-3, in which many agents can authenticate the user flexibly. This scheme can be applied to systems, in which the customers are managed plual vendors, such as payment systems.

The 2GR scheme [34], [40] is secure granted that the verifier is stolen from the server, but user's secret has to be protected. This can be solved using private memories [38]. Moreover, the 2GR scheme is secure against theft attacks and can be applied peer-to-peer communications [52], in which many users share personal information mutually.

Chapter 5

Conclusion

The Internet and mobile communications have been developing, and authentication schemes are proposed for protecting those communications. Authentication schemes apply one-way functions for the security. Those authentication schemes suffer from vulnerability to various attacks.

In this thesis, attacks using communication data and stolen-verifier are discussed, and those solutions are proposed. Authentication data communicated through the Internet can be easily intercepted. The attacker can forge authentication data using the intercepted communication data. Authentication schemes must not create similar authentication data in other sessions. Stolen-verifier attacks are more active attacks. Those attacks become difficult if the stored verifier is protected or other secure data is used. Proposal scheme create authentication data using two data: the verifier stored in the server and the secret data stored or memorized the user. Thus, the attacker cannot impersonate the user using the stored verifier. Moreover, applications using the proposed authentication schemes are presented in this thesis. Those proposed authentication schemes are not limited to apply those presented applications and can be applied various systems.

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. Moreover, I appreciate his concerns for my future works and health.

My grateful appreciation is extended to Professor Kazunori Shimamura, Professor Makoto Iwata, Professor Mamoru Okada, and Professor Takao Nishitani for their kindness and valuable advice.

Grateful thanks are also extended to Professor Lawrence Hunter for his helpful English teaching towards the writing of my papars.

Thanks a lot to Mr. Tomoya Nakahara for his helpful in the realizing of application systems. 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 and the Course of Frontier Engineering during my studies at Kochi University of Technology.

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

References

[1] L. Lamport, "Password authentication with insecure communication," Commun. ACM, vol.24, no.11, pp.770–772, 1981.

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

[3] A. Shimizu, "A dynamic password authentication method by one-way function," System and Computers in Japan, vol.22, no.7, pp.32–40, July 1991.

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

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

[6] N. Haller and R. Atkinson, "On internet authentication," Request For Comments 1704, Oct. 1994.

[7] R. Anderson, C. Manifavas, and C. Sutherland, "Netcard - a practical electronic cash system," Computer Laboratory, Cambridge, 1995.

[8] B. Cox, J.D. Tygar, and M. Sirbu, "NetBill security and transaction protocol," http://www.ini.cmu.edu/net-bill/pubs/Usenix.html, 1995.

[9] M. Manasse, "The Millicent protocol for inexpensive electronic commerece," http://www.w3.org/Conferences/ WWW4/Papers/246/, 1995.

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

[11] H. Dobbertin, A. Bosselaers, and B. Preneel, "RIPEMD-160: A strengthened version of RIPEMD," Fast Software Encryption, Third International Workshop, Lec-

ture Notes in Computer Science 1039, pp.71–82, Springer-Verlag, 1996.

[12] R.Hauser, M. Steiner, and M. Waidner, "Micropayments based on iKP," IBM Zurich, 1996.

[13] R. Rivest and A. Shamir, "PayWord and MicroMint: Two simple micropayment schemes," MIT Laboratory for Computer Science, May 1996.

[14] C. J. Mitchell and L. Chen, "Comments on the S/Key user authentication scheme," ACM Operating Systems Review, vol.30, no.4, pp.12–16, Oct. 1996.

[15] S.-M. Yen and K.-H. Liao, "Shared authentication token secure against replay and weak key attacks," Inf. Process. Lett., vol.62, pp.77–80, 1997.

[16] 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, Aug. 1998.

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

[18] M. Peyravian and N. Zunic, "Methods for protecting password transmission," Computers & Security, vol.19, no.5, pp.466–469, 2000.

[19] 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.

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

[21] 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, Sept. 2001.

[22] T. Kamioka and A. Shimizu, "The examination of the security of SAS one-time password authentication," IEICE Technical Report, OFS2001-48, Nov. 2001.

[23] T. Tsuji, T. Kamioka, and A. Shimizu, "Simple and secure password authentication protocol, ver.2 (SAS-2)," IEICE Technical Report, OIS2002-30, Sept. 2002.

[24] T.-C. Yeh, H.-Y. Shen, and J.-J. Hwang, "A secure one-time password authentication scheme using smart cards," IEICE Trans. Commun., vol.E85-B, no.11, pp.2515–2518, Nov. 2002.

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

[26] T. Tsuji and A. Shimizu, "An impersonation attack on one-time password authentication protocol OSPA," IEICE Technical Report, ISEC2002-81, Nov. 2002.

[27] T. Tsuji and A. Shimizu, "Algorithm variations of SAS-2," IEICE Technical Report, IN2002-149, Dec. 2002.

[28] H.-Y. Chien and J.-K. Jan, "Robust and simple authentication protocol," Comput. J., vol.46, no.2, pp.193–201, Feb. 2003.

[29] T. Tsuji, S. Fujimoto, and A. Shimizu, "High-speed protocol on one-time password authentication," IEICE Technical Report, OIS2003-4, May 2003.

[30] T. Kamioka, M. Kishida, T. Tsuji, and A. Shimizu, "A billing protocol for mobile contents interchange using SAS," IEICE Technical Report, OIS2003-5, May 2003.

[31] F. Ohgaki, T. Konishi, T. Tsuji, and A. Shimizu, "An encryption communication method with SAS," IEICE Technical Report, OIS2003-6, May 2003.

[32] S. Kawamura, S. Nagai, T. Tsuji, and A. Shimizu, "Authentication method for simultaneous multiple access," IEICE Technical Report, OIS2003-7, May 2003.

[33] T. Tsuji and A. Shimizu, "An impersonation attack on one-time password authentication protocol OSPA," IEICE Trans. Commun., vol.E86-B, no.7, pp.2182–2185, Jul. 2003.

[34] T. Tsuji and A. Shimizu, "Two-gene-relation password authentication protocol

(2GR)," IEICE Technical Report, OIS2003-15, Jul. 2003.

[35] C.-C. Yang, T.-Y. Chang, J.-W. Li, and M.-S. Hwang, "Security Enhancement for Protecting Password Transmission," IEICE Trans. Commun., vol.E86-B, no.7, pp.2178–2181, Jul. 2003.

[36] T. Tsuji and A. Shimizu, "Cryptanalysis on Yeh-Shen-Hwang's password authentication scheme," IEICE Technical Report, OIS2003-26, Sep. 2003.

[37] T. Tsuji and A. Shimizu, "Comments on improved Peyravian-Zunic's password authentication schemes," IEICE Technical Report, OIS2003-57, Nov. 2003.

[38] T. Tsuji, F. Ohgaki, and A. Shimizu, "Secure authentication schemes using computable private memories," IEICE Technical Report, OIS2003-77, Jan. 2004.

[39] M.-H. Lin and C.-C. Chang, "A secure one-time password authentication scheme with low-computation for mobile communications," ACM Operating System Review, vol.38, no.2, pp.76–84, Apr. 2004.

[40] T. Tsuji and A. Shimizu, "One-time password authentication protocol against theft attacks," IEICE Trans. Commun., vol.E87-B, no.3, pp.523–529, Mar. 2004.

[41] T. Tsuji and A. Shimizu, "Simple and secure password authentication protocol, Version 3 (SAS-3)," IEICE Technical Report, OIS2003-89, Mar. 2004.

[42] T. Tsuji and A. Shimizu, "A one-time password authentication protocol for mobile communications and internet protocols," IEICE Trans. Commun., vol.E87-B, no.6, pp.1594–1600, Jun. 2004.

[43] T. Tsuji and A. Shimizu, "Cryptanalysis on one-time password authentication schemes using counter value," IEICE Trans. Commun., vol.E87-B, no.6, pp.1756–1759, Jun. 2004.

[44] T. Tsuji and A. Shimizu, "Secure agreement identification for flexible users (SAIFU)," IEICE Technical Report, OIS2004-16, Jul. 2004.

[45] A. Shimizu and T. Tsuji, "Simple and secure password authentication protocol and

its implementation," Proceedings of International Conference Next Era Information Networking 2004 (NEINE'04), pp.2–11, Sep. 2004.

[46] T. Tsuji and A. Shimizu, "Improvement on 2GR authentication scheme," Proceedings of International Conference Next Era Information Networking 2004 (NEINE'04), pp.12–21, Sep. 2004.

[47] T. Tsuji and A. Shimizu, "A convenient micropayment scheme with multi-vendor," IEICE Technical Report, CQ2004-69, Sep. 2004.

[48] T. Nakahara, T. Tsuji, and A. Shimizu, "SAS-2 key-free application system," IEICE Technical Report, CQ2004-70, Sep. 2004.

[49] T. Nakahara, T. Tsuji, and A. Shimizu, "A study on synchronous problems in SAS-2 protocol," IEICE Technical Report, OIS2005-109, Mar. 2005.

[50] S. Inoue, T. Nakahara, T. Tsuji, and A. Shimizu, "Resolving asynchronous problem of SAS-2 key-free application system," Proceedings of International Conference Next Era Information Networking 2005 (NEINE'05), pp.69–78, Sep. 2005.

[51] S. Inoue, T. Nakahara, T. Tsuji, and A. Shimizu, "Resolving asynchronous problem of SAS-2 key-free application system," IEICE Technical Report, CQ2005-37, Sep. 2005.

[52] Y. Nishida, T. Tsuji, and A. Shimizu, "Application of one-time password authentication protocol to peer-to-peer network," IEICE Technical Report, CQ2005-38, Sep. 2005. (Japanese edition)

[53] T. Tsuji, T. Nakahara, and A. Shimizu, "A one-time password authentication method," IEICE Technical Report, Jan. 2006

[54] I. Okada, T. Nakahara, T. Tsuji, and A. Shimizu, "A secure roaming protocol for wireless LAN," IEICE Technical Report, Mar. 2006. (To be presented in Japanese)