JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 22, 1125-1143 (2006)

Secure Authentication Protocols Resistant to Guessing Attacks*

JIA-NING LUO, SHIUHPYNG SHIEH+ AND JI-CHIANG SHEN+
Department of Information and Telecommunication
Ming Chuan University
Taoyuan, 333 Taiwan
E-mail: deer@mcu.edu.tw
+*Department of Computer Science*
National Chiao Tung University
Hsinchu, 300 Taiwan
E-mail: ssp@csie.nctu.edu.tw

Users are normally authenticated via their passwords in computer systems. Since people tend to choose passwords that can be easily remembered, the systems are under the threat of guessing attacks. Many authentication and key distribution protocols have been proposed to protect user passwords from guessing attacks. However, these protocols either are limited to some specific environments or incur high computation and communication costs. In the paper, we first specify five common forms of guessing attacks, which are used to determine whether a protocol is vulnerable to those attacks. Based on these common forms, some guidelines are provided for developing secure protocols that can be used in both symmetric and asymmetric cryptosystems to defend against guessing attacks. Finally, we enhance the well-known authentication system Kerberos and propose two authentication and key distribution protocols, which are both resistant to guessing attacks.

Keywords: network security, authentication, guessing attack, cryptography, protocol

1. INTRODUCTION

In many computer systems, users are authenticated via passwords which they choose. Unfortunately, people tend to choose easy-to-remember passwords [16], which are vulnerable to "guessing attacks." A malicious attacker can guess such passwords using the words in a machine-readable dictionary. To reduce the danger of guessing attacks, authentication mechanisms may use longer passphrases or provide automatic checking of user passwords against known dictionary lists. However, due to human limitations, people tend to use easy-to-remember passwords or variants of words from dictionary. With the rapid development of semiconductor technology, the time needed for both guessing and verification decreases, the probability of successful guessing increases. All the above increase the need for a new scheme to resist guessing attacks.

Guessing attacks can be classified into two categories: on-line and off-line attacks. In on-line guessing attacks, attackers try to guess user passwords and verify them inter-

Received April 16, 2004; revised August 9, 2004 & February 17, 2005; accepted May 30, 2005.
Communicated by Ja-Ling Wu.
* This work was supported by III and National Science Council of Taiwan, R.O.C., under grant No. NSC 95-2221-E-130-028.

actively. Generally, servers can detect an attack from continuous authentication failures. In contrast, in off-line guessing, an attacker eavesdrops on authentication messages and tries to find a quantity X that is derived from poorly chosen passwords in a known way. The attacker can use lots of computers to guess passwords, convert them in a known way, and verify if X is produced. Because such a verification procedure is performed off-line, the servers cannot detect the attack. Since most security systems can detect on-line guessing attacks, this paper focuses on off-line guessing attacks.

There are two approaches to solving the problem of guessing attacks. The first one is to prevent users from using poorly chosen passwords. This is difficult because human's memory has its limitation. The other one is to frustrate off-line guessing attacks directly. That is, to ensure the information available to attackers is sufficiently unpredictable. In other words, the messages in authentication protocols should only contain enough information for the intended recipients to accept them, and there should be insufficient redundancy for attackers to attempt off-line guessing. Many existing authentication protocols, such as Network Encrypted Key Exchange [2], Kerberos [11], the Secure Network Protocol (SNP) [21], Needham-Schroeder shared key protocol [17], Otway-Rees protocol [19] and Neuman-Stubblebine protocol [18] are vulnerable to off-line guessing attacks [26]. Many enhanced authentication protocols have been proposed to resist guessing attacks, including those in [2, 7-10, 13-15, 28]. However, these approaches are either limited to specific environments or incur high computation and communication costs.

In this paper, we will first discuss common types of guessing attacks, which can determine whether a protocol is vulnerable to guessing attacks. Based on these common types, we will propose some useful guidelines for developing protocols that are secure against guessing attacks. Based on these guidelines, we enhance the well-known Kerberos protocol so that it can resist the guessing attacks and also propose two authentication and key distribution protocols for the trusted third-party model. The first protocol assumes that all the principals know the public key of the trusted server, and the second one does not. In the first protocol, we minimize the use of random numbers as well as the number of encryption operations, and eliminate the requirement of timestamps. In the second protocol, exponential key exchange is used to negotiate a new encryption key that is used to distribute the session key.

2. RELATED WORKS

Many authentication protocols to resist off-line guessing attacks have been proposed [2, 7, 8, 10, 15, 24]. We can classify them into two categories, based on the principals involved in communication. The protocols in the first category are based on the *two-party model*. In this model, two communication principals use a shared common secret to authenticate each other and negotiate a session key. The EKE (*Encrypted Key Exchange*) protocol and its variants [1, 2] belong to this category. The protocols in the other category are based on the *trusted third-party model*. In this model, all the principals trust an authentication server and share their own secrets with the server only. Many protocols, such as ours and those proposed in [8, 10, 15] belong to this category.

2.1 Two-Party Model

Suppose that *A* (Alice) and *B* (Bob) share a common secret (the password) and intend to derive/exchange a secure session key. In order to prevent guessing attacks from succeeding, Bellovin and Merritt proposed a concise protocol, called Encrypted Key Exchange (EKE) [2]. Based on different underlying cryptosystems, such as RSA [20], ElGamal [6], and Diffie-Hellman [5], several variants of this protocol have been developed. The generic EKE protocol is susceptible to *Denning-Sacco attack* [4]. This attack was first illustrated by Denning and Sacco in their critique of Needham and Schroeder's paper [17]. In this attack, an attacker somehow obtains one of the session keys distributed in one run of the EKE protocol. Armed with this knowledge, the attacker can mount a guessing attack on the password. There is a variant of EKE, called Exponential Key Exchange EKE, which is resistant to the Denning-Sacco Attack. However, EKE has the limitation that the two participators must share a common secret before communication begins. This limitation confines use of the EKE protocol to be used in specific environments.

2.2 Trusted Third-Party Model

Gong *et al*. proposed a protocol [8] of the trusted third-party model type, the GLNS protocol, which provides protection against guessing attacks. Due to the introduction of *nonces* and a *confounder*, attackers cannot derive the message contents encrypted by the password because the messages are protected by nonces and a random session key. Gong proposed an enhancement [7] of the GLNS protocol, which reduces the number of message transmissions and does not require timestamps. Keung and Siu [10] proposed another protocol which resists both replay and off-line password guessing attacks. Their protocol focuses on enlarging the search space to provide better security and minimizing the amount of encryption. Kwon, Kang, and Song [15] proposed another protocol for mutual authentication and key distribution. Later, Yeh, Sun, and Hwang [25] enhanced the three-party EKE protocol [23] to resist both undetectable on-line and off-line password guessing attacks.

Within these protocols, many random numbers and cryptographic operations are performed to prevent guessing attacks. It is assumed that all the principals know the public key of the trusted server before communication begins. In some situations, however, it may be difficult for users to get the public key of the trusted server (e.g., in mobile environments). In section 5, we will present a more efficient authentication protocol that does not require that the communication principals know the public key of the trusted server.

3. COMMON FORMS OF GUESSING ATTACKS

In this section, we will discuss five common types of guessing attacks: *simple guessing attacks*, *cascade guessing attacks*, *insider guessing attacks*, *replay guessing attacks*, and *partition attacks*. These common types are helpful for diagnosing whether a protocol is vulnerable to guessing attacks. Then, we will propose some guidelines for

developing protocols that are secure against guessing attacks. For convenience, the notations shown in Table 1 will be used to discuss the protocols.

Table 1. Notations.

A	System principal A
B	System principal B
S	Trusted Server
$A \rightarrow B : m$	A sends a message m to B
m, n	Concatenation of message m and n
Ka, Kb	Passwords of A and B
K	Session key between A and B
Ks	Public key of the Trusted Server
$\{m\}_k$	Encrypt m using key k
$[m]_k$	Decrypt m using key k
$\oplus$	Bit-wise exclusive-or operation (XOR)

3.1 Simple Guessing Attacks

If authentication protocols containing predicable information are encrypted with poorly chosen passwords, they will be vulnerable to guessing attacks. Guessing attacks of this form are called *simple guessing attacks*. Many existing protocols, such as SNP (Secure Network Protocol) [21], and the Needham-Schroeder secret key protocol [17], are vulnerable to simple guessing attacks. The SNP protocol proposed by Shieh and Yang is a nonce-based authentication and key distribution protocol for open network systems. The SNP protocol is shown in Fig. 1.

1. $C \rightarrow S : \{C, S, crand\}_{K_c}$
2. $S \rightarrow AS : S, \{C, S, srand, \{C, S, crand\}_{K_c}\}_{K_s}$
3. $AS \rightarrow S : \{AS, K_{ss}, (srand + 1), \{K_{ss}, (crand + 1)\}_{K_c}\}_{K_s}$
4. $S \rightarrow C : \{K_{ss}, (crand + 1)\}_{K_c}$

Fig. 1. SNP protocol.

In the SNP protocol, *AS*, *C*, and *S* represent the authentication server, client, and server, respectively. *Crand* and *srand* are random numbers generated by the client and server, respectively. K_s is the shared secret between the server *S* and authentication server *AS*. K_c is the shared secret between the client *C* and *AS*. K_{ss} is the session key used by *C* and *S* to communicate with each other. Using the keys from a series of guesses against the password K_c, the attacker can iteratively decrypt the first message to derive a set of names of for guessing *C*. If one of the derived names is identical to *C*, he has made a correct guess. The simple guessing attack is described as follows:

- The attacker captures all the messages of one run of the SNP protocol.
- The following steps are performed iteratively until all possible candidates of password K_c are tested:

1. Pick a candidate $\overline{K_c}$.
2. Derive plaintext $\{\overline{C}, \overline{S}, \overline{crand}\}$ by computing $[\{C, S, crand\}_{K_c}]_{\overline{K_c}}$ from the first message $\{C, S, crand\}_{K_c}$ captured.
3. Compare $\overline{C}$ with C.

A match in the last step indicates a correct guess of the password K_c.

3.2 Cascade Guessing Attacks

To prevent simple guessing attacks, a message encrypted with a password should be made sufficiently unpredictable to attackers. An attacker may be able to guess the message encrypted with a poorly chosen password, but he will not be able to directly verify whether his guess is correct if the message content is unpredictable. If the attacker can find any relationship among all parts of the authentication message, he can successfully verify his guess based on that relationship. We present in Fig. 2 a typical trusted third-party and challenge-response protocol to demonstrate this type of guessing attacks called a *cascade guessing attack*. In the protocol shown in Fig. 2, a trusted host S serves as a mediator between the two clients A and B to achieve mutual authentication.

1. $A \rightarrow B : \{A, B, na\}_{K_s}, ra$
2. $B \rightarrow S : \{A, B, na\}_{K_s}, \{B, A, nb\}_{K_s}$
3. $S \rightarrow B : \{na, K\}_{K_a}, \{nb, K\}_{K_b}$
4. $B \rightarrow A : \{na, K\}_{K_a}, \{ra + 1, rb\}_K$
5. $A \rightarrow B : \{rb + 1\}_K$

Fig. 2. Demonstration protocol 1.

In this protocol, *na*, *nb*, *ra*, and *rb* are random numbers generated by the originator of the message in which they first appeared. The key K_s is the public key of the server S, and the keys K_a and K_b shared with S are the secret keys of clients A and B, respectively. The session key K used by A and B is generated by S. In message 4, we can get a possible value of $ra + 1$ by decrypting the second part ($\{ra + 1, rb\}_K$) with the session key K which is derived from our guess in message 3. Since *ra* appears in message 1 in plaintext, we can successfully verify the relationship between *ra* and the derived $ra + 1$. The cascade guessing attack is as follows:

- The attacker captures all the messages of one run of the protocol.
- The following steps are performed iteratively until all possible candidates of password K_a are tested:

1. Pick a candidate $\overline{K_a}$.
2. Derive plaintext $\{\overline{na}, \overline{K}\}$ by computing $[\{na, K\}_{K_a}]_{\overline{K_a}}$ from $\{na, K\}_{K_a}$, which is the first part of captured message 1.
3. Derive plaintext $\{\overline{ra+1}, \overline{rb}\}$ by computing $[\{ra+1, rb\}_K]_{\overline{K}}$ from $\{ra + 1, rb\}_K$, which is the second part of captured message 4.

4. Compare $\overline{ra+1}$ with $ra + 1$.

A match in the last step indicates a correct guess of the password K_a.

3.3 Insider Guessing Attacks

According to demonstration protocol 1, it is vulnerable to cascade guessing attacks if the session keys are directly encrypted with poorly chosen passwords. A revision of demonstration protocol 1 is presented in Fig. 3. It is resistant to cascade guessing attacks.

1. $A \rightarrow B : \{A, B, na1, na2\}_{K_s}, ra$
2. $B \rightarrow S : \{A, B, na1, na2\}_{K_s}, \{B, A, nb1, nb2\}_{K_s}$
3. $S \rightarrow B : \{na1, K \oplus na2\}_{K_a}, \{nb1, K \oplus nb2\}_{K_b}$
4. $B \rightarrow A : \{na1, K \oplus na2\}_{K_a}, \{ra + 1, rb\}_K$
5. $A \rightarrow B : \{rb + 1\}_K$

Fig. 3. Demonstration protocol 2.

Unlike protocol 1, a random number $na2$ is included in the authentication messages. The random number $na2$ prevents the attacker from deriving the session key K because he cannot get the exact value of $na2$ to resolve K from $K \oplus na2$ and, therefore, is unable to verify whether his guess is corrects. Therefore, protocol 2 can resist cascade guessing attacks.

Although we assume that the communicating principals, A and B, trust each other, each principal might try to guess the other's password with the aid of the residue of a successful transaction. This is called the *insider guessing attack*. To show that protocol 2 is vulnerable to insider guessing attacks, we assume that principal B is malicious and try to guess principal A's password. In protocol 2, B can decrypt the second part of message 3 and perform an exclusive-or (XOR) operation on $K \oplus nb2$ with $nb2$ to derive the session key K. Then he can make a guess about K_a and decrypt the first part of message 3 to get the guessing value of $na1$ and $K \oplus na2$. After XORing $K \oplus na2$ with K to obtain the guessing value of $na2$, B can try to construct the first part of message 2 with $na1$, $na2$ and the public key K_s of the server. If the constructed message is identical to the real one, B can correctly guess K_a and succeed to performing the insider guessing attack. The insider guessing attack is shown as follows:

- The malicious principal B records all the messages of one run of the protocol.
- The following steps are performed iteratively until all possible candidates of password K_a are tested:

1. Pick a candidate $\overline{K_a}$.
2. Derive plaintext $\{\overline{na1}, \overline{K \oplus na2}\}$ by computing $[\{na1, K \oplus na2\}_{K_a}]_{\overline{K_a}}$ from $\{na1, K \oplus na2\}_{K_a}$ which is the first part of received message 3.
3. Compute $\overline{na2} = \overline{K \oplus na2} \oplus K$.
4. Construct $\{A, B, \overline{na1}, \overline{na2}\}_{K_s}$.

5. Compare $\{A, B, \overline{na1}, \overline{na2}\}_{K_s}$ with $\{A, B, na1, na2\}_{K_s}$, which is the first part of received message 2.

A match in the last step indicates a correct guess of the password K_a.

3.4 Replay Guessing Attacks

Insider guessing attacks can be prevented by introducing a sufficiently large random number, called a *confounder*, into authentication messages that will be encrypted with the public key of the server. The purpose of the confounder is to prevent an attacker from constructing authentication messages. The value of the confounder can be ignored by the legitimate recipient of the message in which it appears. A revision of protocol 2 with the cofounder $na3$ is shown in Fig. 4. Demonstration protocol 3 is resistant to insider guessing attacks because a malicious principal B cannot construct the first part of message 2 to verify that his guess is correct without knowing the confounder $na3$.

1. $A \rightarrow B : \{A, B, na1, na2, na3\}_{K_s}, ra$
2. $B \rightarrow S : \{A, B, na1, na2, na3\}_{K_s}, \{B, A, nb1, nb2, nb3\}_{K_s}$
3. $S \rightarrow B : \{na1, K \oplus na2\}_{K_a}, \{nb1, K \oplus nb2\}_{K_b}$
4. $B \rightarrow A : \{na1, K \oplus na2\}_{K_a}, \{ra + 1, rb\}_K$
5. $A \rightarrow B : \{rb + 1\}_K$

Fig. 4. Demonstration protocol 3.

In this protocol, the server S cannot determine whether the received request message (message 2) is fresh. An attacker who has captured the old messages of one run of the protocol can masquerade as principal B by re-sending the old message 2 to the server. The server S decrypts message 2, selects a new session key K and then replies with a new message 3. The only difference between the old message 3 and the new one is the session key K, so the attacker can mount a guessing attack, in this case, a *replay guessing attack*. The attacker can guess K_a (or K_b) by decrypting the first (or second) part of both the old message 3 and the new one, and comparing the random number $na1$ (or $nb1$) in the two messages. Consequently, the attacker can successfully verify that his guess is correct and perform the replay guessing attack. The reply guessing attack is shown as follows:

- The attacker captures all the messages of one run of the protocol.
- The attacker masquerades as B and replays the old message 2, $\{A, B, na1, na2, na3\}_{K_s}, \{B, A, nb1, nb2, nb3\}_{K_s}$ to get a new reply message 3: $\{na1, K \oplus na2\}'_{K_a}, \{nb1, K \oplus nb2\}'_{K_b}$.
- The following steps are performed iteratively until all possible candidates of password K_a have been tested:

1. Pick a candidate $\overline{K_a}$.
2. Derive plaintext $\{\overline{na1}, \overline{K \oplus na2}\}$ by computing $[\{na1, K \oplus na2\}_{K_a}]_{\overline{K_a}}$ from $\{na1, K \oplus na2\}_{K_a}$, which is the first part of the captured old message 3.

3. Derive plaintext $\{\overline{na1}', \overline{K \oplus na2}'\}$ by computing $[\{na1, K \oplus na2\}'_{K_a}]_{\overline{K_a}}$ from $\{na1, K \oplus na2\}'_{K_a}$, which is the first part of the received new message 3.
4. Compare $\overline{na1}$ with $\overline{na1}'$.

A match in the last step indicates a correct guess of the password K_a.

3.5 Partition Attacks

To prevent someone from using a secret key to encrypt future session keys, Li Gong proposed the "secret public key protocol," which uses two randomly generated public key pairs k_1 and k_2 to encrypt the session key K [7]. The secret public key protocol is shown in Fig. 5.

1. $A \rightarrow B : na, \{k_1\}_{K_a}$
2. $B \rightarrow S : na, \{k_1\}_{K_a}, nb, \{k_2\}_{k_b}$
3. $S \rightarrow B : \{A, B, cs1, K, \{n_a\}_{k_a}\}_{k_1}, \{B, A, cs2, K, \{n_b\}_{k_b}\}_{k_2}$
4. $B \rightarrow A : \{A, B, cs1, K, \{n_a\}_{k_a}\}_{k_1}, \{n_a\}_K, nb$
5. $A \rightarrow B : \{nb\}_K$

Fig. 5. Secret public key protocol.

In this protocol, the randomly generated public key pair k_1 and k_2 is encrypted by means of the two communicating principals' passwords K_a and K_b. The server, which knows passwords K_a and K_b, can derive the correct k_1 and k_2. In the third message, S uses these public keys to encrypt a new session key K and sends the encrypted message back to the communicating principals, A and B. Finally A and B use their private keys corresponding to k_1 and k_2 to derive the session key K from the messages $\{A, B, cs1, K, \{n_a\}_{k_a}\}_{k_1}$ and $\{B, A, cs2, K, \{n_b\}_{k_b}\}_{k_2}$.

This protocol cannot resist the *partition attack* proposed by Bellovin and Merritt [1]. Before describing the partition attack, we will first look at the key generation part of the well-known RSA algorithm:

1. Generate two large random primes, p and q, of approximately equal size such that their product $n = pq$ is of the required bit length m, e.g., $m = 1024$ bits.
2. Compute $n = pq$ and $\phi(n) = (p - 1)(q - 1)$.
3. Choose the public exponent e, where $\gcd(\phi(n), e) = 1$ and $1 < e < \phi(n)$.
4. Compute the secret exponent $d \equiv e^{-1} \bmod \phi(n)$.

Fig. 6. RSA key generation algorithm.

In the key generation algorithm, the public key is (e, n), and the private key is (d, n). The public component e is always odd because $\phi(n)$ is even and $\gcd(\phi(n), e) = 1$. Furthermore, the value of e is less than $\phi(n)$. According to these properties, an attacker who has captured several old messages can choose one candidate password K_a' to verify that

his guess is correct by checking the range of the derived k_1'. At each round of verification, if k_1' is even or greater than n, the attacker knows his guess is wrong and picks another candidate to continue the attack. Since each verification will partition the remaining candidate password space into two parts, the decrease in the key space is logarithmic. This attack will succeed if K_a is a poorly chosen password and is called as the *partition attack*.

4. GUIDELINES FOR DEVELOPING PROTOCOLS FOR DEFENDING AGAINST GUESSING ATTACKS

In section 3, we introduced some protocols for defending against off-line guessing attacks and specified five common types of such attacks. Having analyzed these types, we will propose some guidelines for developing protocols that are resistant to guessing attacks. First, some generic guidelines for reducing the threat of guessing attacks will be given. Second, we will clarify the properties of guessing attacks in symmetric and asymmetric cryptosystems, and propose respective guidelines.

4.1 Guidelines

Problems with guessing attacks result from poorly chosen passwords. In many cases, we cannot avoid using poorly chosen passwords as encryption keys. This gives attackers opportunities to guess the passwords. Preventing attackers from making guesses is difficult, but preventing verification is possible if we encrypt the data carefully with poorly chosen passwords. Below, we give the first guideline.

Guideline 1 Do not encrypt predictable texts with poorly chosen passwords directly.

If we do not encrypt predictable texts with poorly chosen passwords, the attacker cannot verify his guess directly. In other words, messages encrypted with poorly chosen passwords should be unintelligible to attackers and meaningful to the intended recipients.

In authentication protocols, the data encryption keys, such as session keys, are generally well-chosen random numbers, which can resist guessing attacks. However, the protocols remain vulnerable to cascade guessing attacks if the encryption keys are not used properly in the authentication messages. Now, we will give a guideline concerning encryption keys.

Guideline 2 Do not encrypt keys, such as session keys, with poorly chosen passwords directly.

In section 3, we gave an example which demonstrates the vulnerability of encrypting session keys with poorly chosen passwords directly. In the example, the attacker can decrypt the message with his guessing key to get a possible session key, and verify whether the derived key is equal to the real by decryption of the following messages, since the session key was used to encrypt the communication data.

We know that if poorly chosen passwords are used as encryption keys, the protocols, will be vulnerable to guessing attacks. This vulnerability to guessing attacks can be con-

tagious, because a poorly chosen password will compromise an otherwise well-chosen private key, since an attacker can mount many other attacks on the protocol based on the compromised private key. Therefore, we will give another guideline concerning poorly chosen passwords.

Guideline 3 Avoid using poorly chosen passwords as encryption keys if it is not necessary to do so.

If possible, we should try not to use poorly chosen passwords as encryption keys because of the potential risk. We can use other elements, such as random numbers or hash functions of something, instead. Decreasing the probability of using poorly chosen passwords as encryption keys will improve the security of protocols.

To avoid using poorly chosen passwords as encryption keys, we can use another method for constructing the encryption keys:

$$EncryptionKey = password \oplus seed.$$

The *password* is chosen by the user, and the *seed* is a random number stored in the client's computer. During system setup, the encryption key is shared by the user and is a random number.

4.2 Guidelines for Public Key Systems

Many protocols have been proposed to authenticate each other and negotiate a session key through by using both public key and secret key systems. In this kind of protocol, each principal shares a secret (or password) with the authentication server. For example, an initial request may be as follows:

$$1.\ A \rightarrow S : \{A, B, K_a\}_{K_s}.$$

An attacker cannot decrypt the request message directly, but he may try to construct an initial request message and compare it with the eavesdropped one. Initially, the attacker knows the public key K_s of the server and the principal names A and B. He then guesses a possible K_a' and constructs a request message. If the constructed message is identical to the captured one, he got the correct K_a. In fact, this attack is a simple type of guessing attack. This kind of attack can be prevented by inserting a confounder like the following one into the public key encrypted message:

$$1.\ A \rightarrow S : \{A, B, na, K_a\}_{K_s}.$$

Since the attacker does not know *na*, he is unable to mount a guessing attack on the message. Next, we offer a guideline for public key systems.

Guideline 4 Inhibit reconstruction of the original message by inserting confounders into the public key encrypted message.

After the client sends an initial request message to the authentication server, the server usually sends a reply in which the session key may be included. It should be noted that if the initial request message does not contain a timestamp, then the message can be replayed. Since the server cannot determine the message's freshness, a new response is always replied to the attacker. If the server's response message is encrypted with poorly chosen passwords, it may be vulnerable to replay guessing attacks. Next, we will give another guideline for preventing this kind of guessing attacks.

Guideline 5 A server's reply to a client's request should not include the components taken from the request message if the server's reply is encrypted with passwords.

4.3 Guidelines for Secret Key Systems

In an authenticated protocol based on secret key systems, two communication principals A and B share one secret to authenticate each other. If A and B want to mutually authenticate each other through the only secret, they must confirm some relationship such as challenge and response through the secret. Since the attacker can guess the secret if it is poorly chosen, the protocol messages are unshaded for him if we assume that his guess is correct. Guessing attacks on secret key systems can be prevented by introducing authenticators into encrypted messages. An authenticator is a secret shared by the two parties who wish to communicate with each other. It can be used to help receivers authenticate their messages and can prevent an attacker from verifying that his guess is correct. For example, user A may securely send a message M to server S as follows:

1. $A \rightarrow S : \{A_u, M\}_{K_a}$.

Here, A_u is the authenticator, and K_a is the poorly chosen password. Server S can derive A_u and M by decrypting the message received with K_a, and can verify whether A_u is identical to the one he has or not. If it is, then the server S will believes that the message M was really sent by user A. Though the authenticator A_u could be a poorly secret chosen because it is memorable, guessing attacks will not succeed. This is because an attacker cannot verify that his guess is correct by only using this encrypted message without knowledge of A_u and M.

5. PROPOSED SOLUTIONS

In section 4, we summarized several guidelines for developing secure protocols for defending against guessing attacks. Based on these guidelines, we will propose three protocols that can resist guessing attacks. The first one is an enhancement of the Kerberos protocol. The other two protocols are new and are designed to provide better security and efficiency.

5.1 Enhanced Kerberos Protocol

The Kerberos protocol, as shown in Fig. 7 is a trusted third-party authentication protocol based on secret key system. The Kerberos protocol versions 4 and 5 were described

1. $c \rightarrow AS : c, TGS$
2. $AS \rightarrow c : \{K_{c,tgs}, \{T_{c,tgs}\}_{K_{tgs}}\}K_c$
3. $c \rightarrow TGS : \{A_c\}_{K_{c,tgs}}, \{T_{c,tgs}\}_{K_{tgs}}$
4. $TGS \rightarrow c : \{K_{c,s}, \{T_{c,s}\}_{K_s}\}_{K_{c,tgs}}$
5. $c \rightarrow s : \{A_c\}K_{c,s}, \{T_{c,s}\}_{K_s}$

Fig. 7. Kerberos authentication protocol.

in [22] and [11, 12], respectively. Both of them are vulnerable to cascade guessing attacks.

If the user key K_c used in the Kerberos protocol is a poorly chosen password, an attacker can mount a cascade guessing attack on the protocol: After guessing K_c, an attacker can decrypt message 2 with the guessed K_c to get the $K_{c,tgs}$. Consequently, he can use the derived $K_{c,tgs}$ to decrypt the first part of message 3 to get the authenticator A_c. The A_c contains recognizable data such as the client's name, network address and the timestamp, the attacker can determine whether his guess is successful. To solve this problem, we negotiate a new shared secret key through the public key distribution system. The new shared secret key instead of the poorly chosen password K_c is then used to encrypt message 2, the Kerberos protocol hence is resistant to guessing attacks. The enhanced Kerberos protocol is presented in Fig. 8.

1. $C \rightarrow AS : C, TGS, \{e + jn\}_{K_e}, n$
2. $AS \rightarrow C : \{K_{c,tgs}, \{T_{c,tgs}\}_{K_{tgs}}\}_{K_{c,as}}, \{K_{c,as}\}_{E_c}$
3. $C \rightarrow TGS : \{A_c\}_{K_{c,tgs}}, \{T_{c,tgs}\}_{K_{tgs}}$
4. $TGS \rightarrow C : \{K_{c,s}, \{T_{a,s}\}_{K_s}\}_{K_{c,tgs}}$
5. $C \rightarrow S : \{A_c\}K_{c,s}, \{T_{c,s}\}_{K_s}$

Fig. 8. Enhanced Kerberos authentication protocol.

The difference between the original protocol and our enhanced protocol is in the first two messages. In the enhanced Kerberos protocol, C randomly generates an RSA key pair, $E_c = (e, n)$ and $D_c = (d, n)$, where e is the public key component, d is the correspondent private key and n is the modulous.

When the protocol starts, C encrypts the public key by using his password K_c, and sends it to AS. AS uses it to encrypt further messages that only A can decrypt because only he knows the private key. To prevent the partition attack described in section 3.5 from suceeding, we do not encrypt the public key component e by using the password K_c directly. Assume the required bit length of the symmetric cipher is m, where $2^m > n$. Let $x = \left\lfloor \frac{2^m}{n} \right\rfloor$; we choose a random number j in the range $[0, x - 1]$ and send $\{e + jn\}_{K_c}, n$ to AS. AS can obtain e by calculating $(e + jn) \bmod n = e$.

Upon receipt of message 1, AS decrypts $\{e + jn\}_{K_c}$ to obtain the public key component e. AS generates a shared secret key $K_{c,as}$ and encrypts it with E_c using asymmetric encryption algorithms. In the meantime, the original message 2 is encrypted with $K_{c,as}$ instead of K_c. Now, only C can decrypt the second part of message 2 with the generated

private key (d, n) and get $K_{c,as}$ to continue the sequential procedure, which is same as in the original Kerberos protocol. Therefore, this protocol is secure against off-line guessing attacks, since no poorly chosen password is used to encrypt predictable information.

5.2 A Trusted Third-Party Protocol for Public Key Systems

Based on our guidelines, we will propose an authentication and key distribution protocol in this subsection. It is assumed that all the participators know the public key of the trusted server. The proposed protocol is shown in Fig. 9.

1. $A \rightarrow B : \{A, na, \{A, B, ra\}_{K_a}\}_{K_s}$
2. $B \rightarrow S : \{A, na, \{A, B, ra\}_{K_a}\}_{K_s}, \{B, nb, \{A, B, rb\}_{K_b}\}_{K_s}$
3. $S \rightarrow B : \{A, B, K \oplus h(na \oplus K_a)\}_{ra}, \{B, A, K \oplus h(nb \oplus K_b)\}_{rb}$
4. $B \rightarrow A : \{A, B, K \oplus h(na \oplus K_a)\}_{ra}, \{h(rb)\}_K$
5. $A \rightarrow B : \{h(rb) + 1\}_K$

Fig. 9. Proposed authentication protocol for public key systems.

In the proposed protocol, S is the trusted authentication server, and the public key K_s of S is known by each principal in the system. Principals A and B share their secrets, K_a and K_b, with S. Four random numbers (na, nb, ra, rb) are generated by A and B, respectively, in the protocol. ra and rb are used as randomly chosen encryption keys, and na and nb are used for challenge-response. After the authentication procedure is completed, A and B negotiate a session key K to communicate with each other securely.

In this protocol, the encryptor ra is protected by the user password K_a in message 1. The session key K can only be derived from $K \oplus h(na \oplus K_a)$ or $K \oplus h(nb \oplus K_b)$. Even if an attacker can guess the encryptor ra, he cannot get the session key K or user password K_a, because he does not know both na and K_a, and thus, cannot derive $K \oplus h(na \oplus K_a)$. If the attacker picks one candidate K_a' in message 1, he cannot verify that his guess is correct. For the sake of brevity, we will not describe the protocol in detail.

Simple guessing and *cascade guessing attacks* will not be successful because no poorly chosen password is used as an encryption key. Though message 2 may be replayed because S cannot determine its freshness, no additional information will be helpful for performing a *replay guessing attack* based on S's reply. Knowing the session key is not helpful for mounting a guessing attack on user passwords in the protocol, so insider guessing attacks also will not be successful. Compared with other similar work [7, 8], we minimize the use of random numbers as well as the total number of encryption operations, and we eliminate the need for a timestamp.

5.3 A Trusted Third-Party Protocol without a Public Key System

In some environments, it is difficult to assume that all users know the public key of the trusted third-party. In Fig. 10, another trusted third-party key distribution protocol is shown in which the server's public key is not needed. The function $h(x)$ is a one-way hash function.

1. $A \rightarrow B : A, (\alpha^{X_a} \bmod \beta) \oplus h(K_a \oplus seed_a), ra$
2. $B \rightarrow S : A, B, (\alpha^{X_a} \bmod \beta) \oplus h(K_a \oplus seed_a), (\alpha^{X_b} \bmod \beta) \oplus h(K_b \oplus seed_b)$
3. $S \rightarrow B : (\alpha^{Y_a} \bmod \beta) \oplus h(K_a \oplus seed_a), \{A, B, K\}_{K_{a,s}}, (\alpha^{Y_b} \bmod \beta) \oplus h(K_b \oplus seed_b), \{B, A, K\}_{K_{b,s}}$
4. $B \rightarrow A : (\alpha^{Y_a} \bmod \beta) \oplus h(K_a \oplus seed_a), \{A, B, K\}_{K_{a,s}}, \{ra, rb\}_K$
5. $A \rightarrow B : \{rb + 1\}_K$

Fig. 10. Proposed authentication protocol without a public key system.

Principals A and B share their secrets, $K_a \oplus seed_a$ and $K_b \oplus seed_b$, with S. Before the protocol starts, the two large prime numbers α and β are generated for long-term use. X_a, X_b, Y_a and Y_b are random numbers used for Diffie-Hellman exponential key exchange [5]:

$$K_{a,s} = ((\alpha^{X_a} \bmod \beta)^{Y_a}) \bmod \beta = \alpha^{X_a Y_a} \bmod \beta,$$

$$K_{b,s} = ((\alpha^{X_b} \bmod \beta)^{Y_b}) \bmod \beta = \alpha^{X_b Y_b} \bmod \beta.$$

The generated encryption keys ($K_{a,s}$ and $K_{b,s}$) are only used to protect the session key K. Though an attacker can guess $K_a \oplus seed_a$ (or $K_b \oplus seed_b$) and extract both $\alpha^{X_a} \bmod \beta$ and $\alpha^{Y_a} \bmod \beta$ using his guessed K_a (or K_b), he cannot calculate $K_{a,s}$ (or $K_{b,s}$) and find a relationship that he can use to verify that his guess is correct. User passwords are not used in the protocol except to protect the exponential message, which means that the attacker has no other opportunity to perform an off-line guessing attack. Moreover, this protocol is not vulnerable to *man-in-the-middle attacks*, since exponential messages, such as $\alpha^{X_a} \bmod \beta$, are protected by means of user passwords.

6. DISCUSSION

In this section, we will describe some common attacks and explain why our proposed protocols can resist these attacks. We will also make a comparison between our protocols and other similar protocols. Moreover, we will discuss implementation issues.

6.1 Security Analysis

Trivial Attacks In general, there are two types of trivial attacks: *replay attacks* and *substitution attacks*. A replay attack is an attack in which an intruder successfully impersonates somebody else by replaying one or more old messages collected previously. A substitution attack is an attack in which an intruder successfully impersonates somebody else by substituting one or more messages during the authentication phase. Many advanced attacks are based on these two types of attacks. Since all of our protocols are nonce-based instead of timestamp-based protocols, substitution and replay attacks based on the use of old authentication messages can be easily defended against.

Oracle Session Attacks An *oracle session attack* was described by Bird [3]. In such an attack, an intruder starts two separate authentication sessions with two different service providers, such that he is able to use the messages in one authentication session to successfully impersonate a particular user in the other session. This kind of attack can be

effectively blocked if the encrypted messages involved in each run of the protocol are different from or logically linked with one another. In our proposed authentication protocols, the session key is encrypted with a pre-negotiated random key. Without knowing the pre-negotiated random key, the intruder cannot attack the protocols successfully.

Comparisons In this section, we will compare our protocols with other similar protocols, such as the GLNS compact protocol [8], GLNS nonce protocol [8], Gong's optimal protocol [7], Keung-Siu's protocol [10], and Kwon-Kang-Song's protocol [15]. We will show that our proposed protocol with a public key system has the advantage of employing fewer random numbers, and that our protocol without a public key system has the advantage of not needing the authentication server's public key. A comparison is shown in Table 2.

Table 2. Comparison of various authentication protocols.

	# of messages	Needs timestamp	Needs server's public key	# of random numbers	# of exponentiation
GLNS compact protocol	5	Yes	Yes	8	2
GLNS nonce protocol	7	No	Yes	8	2
GLNS optimal protocol	5	No	Yes	10	2
Keung-Siu Protocol	5	No	Yes	6	2
Kwon-Kang-Song protocol	5	No	Yes	6	2
Proposed protocol with a public key system	5	No	Yes	4	2
Proposed protocol without a pubic key system	5	No	No	6	4

6.2 Analysis of Protocol Implementation

In the above sections, we considered cryptographic primitives as abstract operations. In this section, we will consider two issues related to protocol implementation: key generation and message padding.

6.2.1 Cryptographic algorithms and key generation

Most authentication protocols use cryptography algorithms to encrypt communication messages. For example, Kerberos is based on secret key technology, and all Kerberos V4 implementations use DES.

In some implementations, a password string may not be used to encrypt message contents directly in a secret key system. A key generation function converts the password

string into an encryption key. For example, in the well-known encryption library "SSLeay" [27], the DES key-generation function is des_string_to_key(). If a key generation function accepts a fixed-length input, then guessing attacks may be successful, not because the password is used by a human, but because the space for likely candidates is small enough to make brute-force search feasible. As another example, in the traditional UNIX system, the passwd() function accepts only an eight-byte input. A password "12345678qazwsxytrfg" might be considered to be secure, but in fact it will be truncated, resulting in the easily broken password "12345678." In other cases, implementations may use hash algorithms (for example, MD5) as key-generation functions to permit longer passphrases and reduce the above risk.

Due to their limited memory capacity, people are tempted to use easy-to-remember passwords which are used to generate keys. In these cases, the key spaces of likely candidates may be small enough to make brute-force search feasible. Therefore, mechanisms for blocking guessing attacks are still needed.

Now we will consider public key systems. In many public key implementations, private key files are protected by passwords. If an intruder can get the private key file, he will only need to guess the password rather than solve a factoring problem. Thus, the robustness of such an implementation is based on a user-chosen password, not on the lengths of prime numbers.

6.2.2 Message padding and checksum

In block-cipher systems, messages are encrypted block by block. The block-length is fixed, and we assume that it is n bytes long. If the length of a plain-text message is i byte(s) (which is less than n), then an extra *padding-string* is appended to the plain-text message before encryption is performed. The padding-string may be filled with zeros, fixed formats, or random values.

Password guessing is easier if a padding algorithm is used in the implementation. Consider the last message of demonstration protocol 1, as shown in Fig. 2. Assume that the variable $rb + 1$ requires storing four bytes. If we use DES to encrypt this message, we will need to append four extra padding bytes, such as "01 02 03 04," to the message because the plain-text length of a DES block cipher is eight bytes. An attacker can choose a candidate password $\overline{K}$ to decrypt the captured message and compare the end of the plaintext with the padding form.

Moreover, if a checksum is used before encryption is performed in the implementation, the situation is similar to the padding problem. An attacker can verify that his guess is correct by calculating a new checksum value and comparing it with the old one.

For the reasons given above, checksum should not be used before encryption in the implementation, and the padding string should be filled with random numbers.

7. CONCLUSION

In this paper, we have discussed in detail the problem of guessing attacks. First, we explored five types of the guessing attacks: *simple guessing attacks*, *cascade guessing attacks*, *insider guessing attacks*, *replay guessing attacks* and *partition attacks*. These

forms are very useful for determining whether a protocol is vulnerable to the guessing attacks. Then, we proposed some guidelines for developing protocols that are secure against guessing attacks. Finally, we proposed an enhanced version of the Kerberos protocol and two new authentication protocols, all of which can resist guessing attacks.

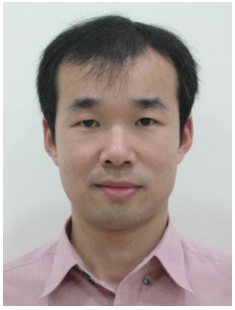
REFERENCES

1. S. Bellovin and M. Merritt, "Encrypted key exchange: password-based protocols secure against dictionary attacks," in *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 1992, pp. 72-84.
2. S. Bellovin and M. Merritt, "Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password file compromise," in *Proceedings of the 1st ACM Conference on Computer and Communication Security*, 1993, pp. 244-250.
3. R. Bird, I. Gopal, A. Herzberg, P. A. Janson, S. Kutten, R. Molva, and M. Yung, "Systematic design of a family of attack-resistant authentication protocols," *IEEE Journal on Selected Areas in Communications*, Vol. 11, 1993, pp. 679-693.
4. D. E. Denning and G. M. Sacco, "Timestamps in key distribution systems," *Communications of the ACM*, Vol. 24, 1981, pp. 533-536.
5. W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, Vol. IT-11, 1976, pp. 644-654.
6. T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, Vol. 31, 1985, pp. 469-472.
7. L. Gong, "Optimal authentication protocols resistant to password guessing attacks," in *Proceedings of the 8th IEEE Computer Security Foundation Workshop*, 1995, pp. 24-29.
8. L. Gong, M. Lomas, R. Needham, and J. Saltzer, "Protecting poorly chosen secrets from guessing attacks," *IEEE Journal on Selected Areas in Communications*, Vol. 11, 1993, pp. 648-656.
9. B. T. Hsieh, H. M. Sun, and T. Hwang, "Cryptanalysis of enhancement for simple authentication key agreement algorithm," *Electronic Letters*, Vol. 38, 2002, pp. 20-21.
10. S. Keung and K. Siu, "Efficient protocols secure against guessing and replay attacks," in *Proceedings of the 4th International Conference on Computer Communications and Networks*, 1995, pp. 105-112.
11. J. T. Kohl, B. C. Neumann, and T. Ts'o, "The evolution of the Kerberos authentication system," *Distributed Open Systems*, 1994, pp. 78-94.
12. J. T. Kohl, "The evolution of the Kerberos authentication service," in *Proceedings of the EurOpen Conference*, 1991, pp. 295-313.
13. T. Kwon, M. Kang, S. Jung, and J. Song, "An improvement of the password-based authentication protocol (K1P) on security against replay attacks," *IEICE Transactions on Communications*, Vol. E82-B, 1999, pp. 991-997.
14. T. Kwon, M. Kang, S. Jung, and J. Song, "Authentication key exchange protocols resistant to password guessing attacks," in *Proceedings of the IEE Communication*, Vol. 145, 1998, pp. 304-308.

15. T. Kwon, M. Kang, and J. Song, "An adaptable and reliable authentication protocol for communication networks," in *Proceedings of the IEEE INFOCOM*, 1997, pp. 737-744.
16. R. Morris and K. Thompson, "Password security: a case study," *Communications of the ACM*, Vol. 22, 1979, pp. 594-597.
17. R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, Vol. 21, 1978, pp. 993-999.
18. B. C. Neuman and S. G. Stubblebine, "A note on the use of timestamps as nonces," *ACM SIGOPS Operating Systems Review*, Vol. 27, 1993, pp. 10-14.
19. D. Otway and O. Rees, "Efficient and timely mutual authentication," *ACM SIGOPS Operating System Review*, Vol. 21, 1978, pp. 8-10.
20. R. L. Rivest, A. Shamir, and L. Adleman, "A method of obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, Vol. 21, 1978, pp. 120-126.
21. S. P. Shieh and W. H. Yang, "An authentication and key distribution system for open network systems," *ACM SIGOPS Operating Systems Review*, 1996, pp. 32-41.
22. J. G. Steiner, B. C. Neuman, and J. I. Schiller, "Kerberos: an authentication service for open network systems," in *Proceedings of the Winter USENIX Conference*, 1988, pp. 191-201.
23. M. Steiner, G. Tsudik, and M. Waidner, "Refinement and extension of encrypted key exchange," *ACM SIGOPS Operating Systems Review*, Vol. 29, 1995, pp. 22-30.
24. G. Tsudik and E. Van Herreweghen, "Some remarks on protecting weak keys and poorly-chosen secrets from guessing attacks," in *Proceedings of the 12th IEEE Symposium on Reliable Distributed Systems*, 1993, pp. 136-141.
25. H. T. Yeh, H. M. Sun, and T. N. Hwang, "Efficient three-party authentication and key agreement protocols resistant to password guessing attack," *Journal of Information Science and Engineering*, Vol. 19, 2003, pp. 1059-1070.
26. H. T. Yeh, H. M. Sun, and T. Wang, "Security analysis of the generalized key agreement and password authentication protocol," *IEEE Communications Letters*, Vol. 5, 2001, pp. 462-463.
27. E. A. Young, "SSLeay encryption library," http://www2.psy.uq.edu.au/~ftp/Crypto/.
28. M. Zhang, "Analysis of the SPEKE password-authenticated key exchange protocol," *IEEE Communications Letters*, Vol. 8, 2004, pp. 63-65.

Jia-Ning Luo (羅嘉寧) is an Assistant Professor of Department of Information and Telecommunications Engineering of Ming Chuan University. He received the B.S. degree in Electrical Engineering and M.S. degree in Computer Science Engineering from Tatung University, and Ph.D. degree in Computer Science from National Chiao Tung University. Jia-Ning Luo is interested in distributed systems and network security.

Shiuhpyng Shieh (謝續平) is a Professor at Department of Computer Science of National Chiao Tung University (NCTU). He has worked as advisor to many institutes, such as National Security Bureau, GSN-CERT/ CC, National Information and Communication Security Task Force. Before joining NCTU, Dr. Shieh participated in the design and implementation of the B2 Secure XENIX at IBM, Federal Sector Division, Gaithersburg, Maryland. He also designed and developed NetSphinx, a network security product, for Formosoft Inc., which is awarded 1999 network product of the year, Taiwan. Dr. Shieh received the M.S. and Ph.D. degrees in Electrical and Computer Engineering from the University of Maryland, College Park. He is a senior member of IEEE, and an editor of ACM Transactions on Information and System Security, Journal of Computer Security, and Journal of Information Science and Engineering. He was on the organizing committees of numerous conferences, such as ACM conference on Computer and Communications Security, IACR Asiacrypt. Dr. Shieh published over a hundred academic articles, including papers, patents, and books. Recently he received the Outstanding Research Award from National Chiao Tung University for his academic achievement in research, and the Outstanding Achievement Award from Executive Yuan of Taiwan. His research interests include internetworking, distributed operating systems, and network security



Ji-Chiang Shen (沈志強) is an project manager of Lite-On Technology Corporation. Mr. Shen received the B.S. degree in Department of Electrical Engineering from National Tsing Hua University and M.S. degree in Computer Science from National Chiao Tung University, Taiwan.