Practical Key Recovery Schemes

Sung-Ming Yen

Laboratory of Cryptography and Information Security (LCIS)
Department of Computer Science and Information Engineering
National Central University
Chung-Li, Taiwan 320, R.O.C.
yensm@csie.ncu.edu.tw

Abstract. In this paper, the Bell Labs key recovery scheme is extensively modified to enable a user to request on-line key recovery service when the file decryption key is forgotten or lost. New practical and important requirements of key recovery are also considered in the proposed schemes, for example, the key recovery server and any intruder over the communication channel should not learn the key to be reconstructed. Furthermore, the necessary authenticity and secrecy between a user and the key recovery server should be provided.

Keywords: Active attack detectability, Cryptography, Dictionary attack, Key escrow, Key recovery, Off-line attack, On-line attack.

1 Introduction

Since 1994, the topic of key escrowed encryption and communication has been widely noticed and studied [1,2,3,4,5]. However, this technique has never been widely employed because of the privacy issue. Until 1996, some researchers changed their attention on escrowed encryption into the commercial applications in an alternative scenario, say the commercial *key recovery* [6,7,8]. The topic of commercial key recovery is not only nonconflicting but also can be identified as a necessary component for the applications of data security service. Evidently, it is a very important and practical issue of how to get survive when using a hard to remember (and to guess) password in a real security system.

1.1 The Classification of Keys

Here we classify passwords (or keys) to be remembered by a person into three different types depending on the complexity required to recover them by the attacker and the difficulty required to remember them by the owner.

simple password : This category of passwords are more easy to remember by the owner, but they are still difficult to be guessed by nonprofessional attacker. However, a professional hacker may sometimes figure out the weak

V. Varadharajan and Y. Mu (Eds.): ACISP 2001, LNCS 2119, pp. 104–114, 2001.

password P_a (if it is poorly chosen) with some probability via the *off-line dictionary attack* if a copy of $f(P_a)$ is accessible where $f()$ is any one-way function. In this paper, a simple password means a password that should at least be resistant with large probability against the guessing and matching attack from some collected dictionaries. Tools for preventing poorly chosen weak password from guessing attacks, i.e., to filter out fatal or inappropriate weak passwords, are available in [9,10,11]. In [9], it was reported that prior to the experiment, it is believed to find a large percentage of the collected passwords (selected by a novice) in the dictionaries and common word lists. However, surprisingly, only 1 out of 5 passwords were found in the dictionaries. The standard dictionary used in [9] has about or more than 25000 words. So, it is expected that each of the matched passwords needs $\frac{25000}{2}$ comparisons with words in the dictionary before been identified. Notice that while an off-line guessing approach may be feasible, an on-line approach however will cause an unreasonable delay and most of the time this attack will be easily detected. Evidently, the cost of on-line (and off-line) guessing attack on an already sieved (using the above mentioned tools) simple password will be extremely high since a much larger dictionary and more sophisticated manipulation will be necessary.

strong password : Theoretically, passwords of this category are selected to be random numbers. So, they are basically be quite difficult to be figure out by the attacker but they are also quite hard to remember by the owner.

pseudo strong password : A password "*pass*" for practical and high security requirement usage often falls into this category which can be considered as a mixture of the above two categories of passwords. It avoids or at least complexes greatly the off-line dictionary attack against professional hackers but it also brings the risk of forgetting the passwords to the owner. Another typical approach of selecting a pseudo strong password is by concatenating two or more simple passwords and it is sometimes called a passphrase.

1.2 Review of the Bell Labs Key Recovery Scheme

To compromise with the requirement of using more secure pseudo strong passwords or keys and the requirement of recovering any forgotten passwords or keys, Maher at Bell Labs developed a crypto key recovery scheme [6].

• The protocol

In the Bell Labs key recovery scheme [6], the key recovery server has its secret key x_s and the related public key $y_s = \alpha^{x_s} \bmod P$, where P is a large prime. Each user, say A, registers to this server through a physical manner and will be given the server's public key.

The working key generation and working key recovery protocol can be briefly reviewed as follows. Here the working key refers to the file encryption key or any login password used in a remote login procedure.

(1) A**:** Each time user A wishes to encrypt his important file, he computes the file encryption key K (it is assumed to be a *strong password*) as

$$K = h(\alpha^{pass} \bmod P || y_s^{pass} \bmod P)$$

where $h()$ is any secure one-way hash function, e.g., [12,13,14], and "*pass*" is the password (it is assumed to be a *pseudo strong password*) selected by the user. Besides the ciphertext $E_K(M)$, $\alpha^{pass} \bmod P$ is also stored. The encryption function $E()$ is assumed to be performed by using any symmetric-key cipher.

(2) $A \rightarrow S$**:** When the user A forgets his password, he tries to recover the file encryption key K by delivering $\alpha^{pass} \bmod P$ to the key recovery server.

(3) $A \leftarrow S$**:** The recovery server computes $T = (\alpha^{pass})^{x_s} \bmod P$ and sends the result T to the user.

(4) A**:** The file encryption key K can be recovered by the user as $h(\alpha^{pass} \bmod P || T)$.

• Some remarks on the Bell Labs protocol

Two important but overlooked issues of the above Bell Labs protocol are given below.

(a) In the step (2), the value $\alpha^{pass} \bmod P$ should be delivered to the recovery server by A through an *authenticated* channel.

(b) In the step (3), the value T should be sent back to user A via a *secure* channel.

However, no solution has been provided in the original protocol to meet the above two important and necessary requirements. Otherwise, a physical face-to-face approach will be necessary for each query and this reduces the practicality of commercial key recovery.

In the Bell Labs protocol, the recovery server will learn the password or key K to be reconstructed. Therefore, multiple recovery servers will be necessary and the file encryption key K can be computed as a combination of many subkeys K_i (e.g., $K = K_1 \oplus K_2 \oplus K_3$ for $i = 1, 2, 3$) and each of which should be recovered when the user forgets his password. The above development aims to prevent any single server or a subset of servers in collusion to obtain the key K. To have a satisfying security level, the number of servers should at least be three. There are two drawbacks for this arrangement. First, system performance may be worse than its original scheme. The second drawback, the more critical issue for the case of using multiple key recovery servers, is that the applicability of the key recovery will be reduced. Based on the above construction, when one of the servers is not accessible or becomes faulty, then the forgotten key K can theoretically never be reconstructed.

Another issue to be pointed out is that in the step (1) of file encryption key generation procedure, K can be generated as either $h(\alpha^{pass} \bmod P)$ or $h(y_s^{pass} \bmod P)$ when $y_s^{pass} \bmod P$ or $\alpha^{pass} \bmod P$, respectively will be stored with the ciphertext $E_K(M)$.

2 The Model of a Practical Key Recovery

Since the problem of how to recover a forgotten password or key is basically a practical issue. Therefore, a practical consideration of what a key recovery scheme should provide is necessary. From the view point of practicality, the requirements of a good key recovery scheme are identified and listed in the following.

(1) The key recovery protocol should be performed at the user's location through an on-line process interacted with the recovery server. For this purpose, the on-line process must also provide both secrecy and authenticity.
(2) The key recovery server should not know the exact passwords or keys to be recovered by the user. This implies that a simple key backup approach does not match the requirement of a good key recovery.
(3) An attacker cannot try to impersonate to be a specific valid user without being detected and recovers that user's passwords or keys via the assistance from the key recovery server which acting as an oracle. In another word, any on-line impersonating and guessing attack should be detectable by the recovery server.
(4) In the real world of using digital systems, any user may have many passwords or keys to remember, however the user does not have to keep a cleartext backup of them in order to prevent forgetfulness.
(5) Even if the user loses his local copy of the most important personal secret information, the key recovery scheme should also enable the server to assist the user to recover his password or key. Although, in this situation, it maybe requires the user to return back to the recovery server physically and performs the recovery process.

The requirement (1) implies that the user can request a key recovery service through an on-line process performed from a remote location instead of returning back to the recovery server physically. To realize this useful functionality, a key recovery scheme itself should also provide both authenticity and secrecy in order to avoid impersonation (to be discussed in the requirement (3)) and to protect the recovered passwords or keys.

As to the requirement (2), since the secret information to be recovered may be a password to log into a computer system connected to the Internet. It is not reasonable to enable the server to know the exact secret information.

The requirement (3) is crucial because that the stored information (e.g., the value of $\alpha^{pass} \bmod P$ in the Bell Labs scheme) in the user's machine used to recover the forgotten password or keys may be accessible to an internal attacker in some situations. However, it is not reasonable for an attacker with access to the above stored information to recover the passwords or keys via the assistance from the trusted recovery server. This will open a backdoor of a key recovery system. Therefore, this suggests that the stored information for key recovery to be user specific or under the protection by using an important password or key of the specific user in order to avoid impersonation. Furthermore, since a remote key recovery process will be performed, any on-line guessing attack if happen

should be detectable by the recovery server. The key recovery scheme should guarantee that the recovery server will not be used as an oracle by an attacker impersonating to be a specific valid user without being detected and trying to recover that user's passwords or keys.

The requirement (4) says that key recovery server should provide promising service to its users to help them to reconstruct important information. Otherwise, a trivial solution that each user keeps a backup of all his passwords or keys in a *secret* (although it is difficult to define precisly) place is enough.

The requirement (5) is to make the scheme be robust enough. After registering to the recovery server, each user may select or receive some long term personal secret information which will be used to help recovering the working passwords or keys. However, it is not reasonable that the passwords or keys cannot be recovered forever if that long term personal information is lost.

Some issues of chosing and using passwords and keys are described in the following. Usually it is strongly suggested that a person should not keep only a single password or key and uses this same password for every system that he has access. Often, it is suggested that a lift cycle of a password should not exceed three or four months, especially for remote login passwords used to enter a system that does not have strong intrusion detection measures. At the same time, people are educated to avoid of using poorly chosen weak passwords. For the problem of file protection, people sometimes use different passwords or keys for files of different security classifications. Therefore, each person will have a moderately amount of *pseudo strong* passwords or keys to remember. In this paper, we focus our attention on considering the problem of how to recover any of such pseudo strong passwords or keys if they will be forgotten under the above mentioned practical environment.

3 The Proposed Key Recovery Scheme – KRS-1

In this section, the first proposed key recovery scheme, named the KRS-1, is given to modify the Bell Labs scheme in order to enhance both security and functionary.

3.1 The Protocol of KRS-1

The key recovery server S has its secret key x_s and the related public key $y_s = \alpha^{x_s} \bmod P$, where P is a large prime and α is a primitive root of P. For security reasons, P is often selected to be a *safe prime* of the form $P = 2q + 1$ where q is also a large prime. A prime where $P - 1$ has only small factors is called smooth. Smooth primes should be avoided because they allow a much faster discrete logarithm computation [15].

When each user, say A, registers to the server, he selects a personal long term password P_a and gives it to the server S. Noticeably, the password P_a is assumed to be a *simple password*. This will enable the user to remember his long term password in a more easy way and makes the key recovery scheme be

robust enough. The server stores each user's identity and the related personal long term password in a secure table and gives the above server's public key to the user. The server's public key and its certificate can also be retrieved through the network when required.

The working key (it can be a file encryption key or a login secret to access a remote machine) generation and recovery protocol goes as follows.

(1) A**:** Each time user A wishes to encrypt his important file, he computes the file encryption key K (it is assumed to be a *strong password*) as

$$K = h(y_s^{pass} \bmod P)$$

where $h()$ is any secure one-way hash function, e.g., [12,13,14], and "*pass*" is a file accessing password selected by the user for a specific classification of files.

As previously described, *pass* should be a *pseudo strong password* so that an off-line dictionary attack is infeasible or at least with excessive cost that will make the attack meaningless. Besides the ciphertext $E_K(M)$,

$$R = E_{P_a}(\alpha^{pass} \bmod P)$$

(or just as $R = (\alpha^{pass} \bmod P) \oplus P_a$) is also stored. Of course, if two or more files share a common encryption key K (or file accessing password *pass*), then the value of R can also be shared.

(2) $A \rightarrow S$**:** When the user A forgets his file accessing password *pass*, he tries to recover the file encryption key K by retrieving $\alpha^{pass} \bmod P$ from R using his long term personal password P_a and computes

$$V = \alpha^{pass} \cdot \alpha^{r_1} \bmod P$$

where r_1 is a random integer selected by user A. User A then computes

$$\begin{cases} c_1 = \alpha^{r_2} \bmod P \\ c_2 = y_s^{r_2} \cdot V \bmod P \\ h(V, P_a) \end{cases}$$

where r_2 is a random integer selected by A and $\{c_1, c_2\}$ are the ciphertext of V produced by using the ElGamal encryption scheme [16]. Finally, the user sends c_1, c_2, and $h(V, P_a)$ along with his identity ID_a to the recovery server.

(3) $A \leftarrow S$**:** The key recovery server first decrypts V from $\{c_1, c_2\}$ by using its secret key x_s and checks the data integrity and originality via the assistance of $h(V, P_a)$. If the above verification is correct, then the key recovery server computes

$$\begin{aligned} T &= V^{x_s} \bmod P \\ &= (\alpha^{pass} \cdot \alpha^{r_1})^{x_s} \bmod P \\ &= y_s^{pass} \cdot y_s^{r_1} \bmod P. \end{aligned}$$

The server then returns T to the user.

(4) A**:** Since A knows the random integer r_1, he can recover

$$y_s^{pass} \equiv T \cdot y_s^{-r_1} \pmod{P},$$

then the file encryption key K can be recovered as $h(T \cdot y_s^{-r_1} \bmod P)$.

3.2 Security Analysis of the KRS-1 Protocol

The reason of sending V in the ciphertext version instead of the cleartext version is to counteract the following off-line *verifiable text attack* on the long term password P_a (it is a simple password). If the passive attacker can intercept both V and $h(V, P_a)$, then he can perform an *off-line* dictionary attack trying to recover P_a if P_a is poorly chosen. In the above protocol, the attacker can intercept $T = V^{x_s} \bmod P$ from the step (3), however to derive V from the intercepted value T requires the server's secret key x_s.

An interesting problem for the above protocol is that if the user A still remembers his long term password P_a, then why not just storing the encrypted version of a working key as $E_{P_a}(K)$ or $E_{P_a}(pass)$ using P_a as the protection key? Therefore, the file encryption key K or file accessing password $pass$ can be recovered when required without the assistance from the recovery server.

Recall that P_a is a simple password. If the above approach is employed, then the attacker who has access to both the ciphertext $E_K(M)$ and the encrypted version of key $E_{P_a}(K)$ can conduct an *off-line* exhaustive search and test on the possible long term password P_a.

On the other hand, in the proposed key recovery protocol, when an active attacker has both the knowledge of $E_K(M)$ and $R = (\alpha^{pass} \bmod P) \oplus P_a$ he has to conduct the following *on-line* password guessing attack. However, the attack can be detected and can be prevented by some precautions. The attacker should try a guessed long term password P'_a and computes

$$G = R \oplus P'_a = (\alpha^{pass} \bmod P) \oplus P_a \oplus P'_a.$$

The attacker then computes $V = G \cdot \alpha^r \bmod P$ and its ciphertext $\{c_1, c_2\}$ and the keyed hash $h(V, P'_a)$. Finally, the attacker sends ID_a, $\{c_1, c_2\}$, and $h(V, P'_a)$ to the recovery server. If the guessed P'_a is identical to P_a, then the integrity check will be correct and the server will return T, otherwise the active attack will be detected and the recovery server will take some suitable countermeasures. Possible countermeasures include: (1) to keep a log file and to delay the following attempts; (2) to advise the legal user to change his P_a. It should be emphasized that for a simple password based protocol, the on-line password guessing attacks are always possible. The main concern is that the protocol should be active attack detectable.

In fact, the above on-line guessing attack can be modified to send ID_t, $\{c_1, c_2\}$, and $h(V, P_t)$ to the recovery server where ID_t and P_t are the identity and the personal password, respectively of the active attacker. The server will not detect the existence of an attack. If the recovered $h(T \cdot y_s^{-r} \bmod P)$ is

equivalent to $h(y_s^{pass})$ and can be used to decrypt $E_K(M)$ correctly, then the guessed P'_a is correct and now the recovery server acts like an oracle.

However, the above scenario is not exactly the case of a usual on-line guessing attack. First, the attacker now conducting an unusual (in terms of its high frequency) service request will reveal its identity. Different application environments may take their own appropriate countermeasures, e.g., to reveal the attacker's identity or to restrict the number of service provided within a predetermined period of time. Second, a large amount of service request (each itself will cost the attacker some service charge) will be necessary in order to figure out the possible P_a. Recall that even a password chosen by a novice it can only be identified with a probability of about 0.2 [9] and it will take a huge amount of on-line test, say on average $\frac{25000}{2}$. As suggested previously, P_a should be sieved by some tools to rule out poorly chosen passwords. This precaution may complex the on-line guessing attack extensively. Therefore, such kind of on-line attack will be extremely unreasonable for commercial key recovery since the total amount of cost (paid to the recovery server) may exceed the profit of the attack or the cost to find *pass* via an exhaustive search.

4 The Key Recovery Scheme Based on RSA – KRS-2

In this section, the second proposed key recovery scheme, named the KRS-2, is given which is based on the blinding of the RSA system.

4.1 The Protocol of KRS-2

The key recovery server S selects two large secret primes p and q, and publishes $n = p \cdot q$. Also, the server chooses a base number α with order $\phi(n)$. The server publishes its RSA [17] public encryption exponent e and keeps privately the decryption exponent d such that $e \cdot d \equiv 1 \pmod{\phi(n)}$ where $\phi(n) = (p-1) \cdot (q-1)$. When each user, say A, registers to the server, he selects a personal long term password P_a and his identity ID_a such that $\gcd(ID_a, \phi(n)) = 1$. When encoded into an integer, the identity should be unique. Alternatively, a random number (often called a salt) is chosen for each identity and an extended identity is computed by using a cryptographic hash function on the identity and the salt to obtain a unique ID_a. The details of generating unique identity numbers are out of the scope of this paper. It is the same as in the KRS-1 that the password P_a is assumed to be a simple password. The server stores each user's identity and the personal long term password in a secure table.

The working key generation and recovery protocol goes as follows.

(1) ***A*:** Each time user A wishes to encrypt his important files, he computes the file encryption key K as

$$K = h(\alpha^{pass} \bmod n)$$

where $pass$ is a pseudo strong file accessing password selected by the user. Besides the ciphertext $E_K(M)$,

$$R = \alpha^{pass \cdot ID_a} \bmod n$$

is also stored. Of course, if two or more files share a common encryption key K (or file accessing password $pass$), then the value of R can also be shared.

(2) $A \rightarrow S$**:** When the user A forgets his file accessing password $pass$, he computes

$$V = \alpha^{pass \cdot ID_a} \cdot r_1^{ID_a} \bmod n$$

where r_1 is a random integer in $[1, n-1]$ selected by user A so that $\gcd(r_1, n) = 1$. The user then computes

$$\begin{cases} r_2^e \bmod n \\ h(V, r_2, P_a) \end{cases}$$

where r_2 is a random integer in $[1, n-1]$ selected by the user A. User A then sends V, $r_2^e \bmod n$, and $h(V, r_2, P_a)$ along with his identity ID_a to the recovery server.

(3) $A \leftarrow S$**:** The recovery server first decrypts r_2 using the decryption key d and checks the integrity and data originality via $h(V, r_2, P_a)$. If the above checking is correct, then the key recovery server computes

$$\begin{aligned} T &= V^{ID_a^{-1}} \bmod n \\ &= (\alpha^{pass \cdot ID_a} \cdot r_1^{ID_a})^{ID_a^{-1}} \bmod n \\ &= (\alpha^{pass} \cdot r_1) \bmod n \end{aligned}$$

where ID_a^{-1} is the multiplicative inverse of ID_a modulo $\phi(n)$. The server then returns T to the user.

(4) A**:** Since A knows the random integer r_1, he can recover

$$\alpha^{pass} \equiv T \cdot r_1^{-1} \pmod{n},$$

where r_1^{-1} is the multiplicative inverse of r_1 modulo n. Then, the file encryption key K can be recovered as $h(T \cdot r_1^{-1} \bmod n)$.

4.2 Security Analysis of the KRS-2 Protocol

One major difference to the previous KRS-1 protocol is that in the KRS-2 protocol the value of V is delivered to the recovery server in the cleartext version. The reason of not sending $V^e \bmod n$ is that V can be easily obtained via $V = T^{ID_a} \bmod n$, where T can be intercepted from the step (3).

If the protocol is modified to remove the inclusion of r_2 such that V is sent in the ciphertext version as $V^e \bmod n$, the integrity and originality check value $h(V, P_a)$ is sent in the step (2), and T is protected by $t = (T \cdot P_a) \bmod n$ when received from the recovery server in the step (3). The protection of T is to

avoid the direct derivation of V by computing $V = T^{ID_a} \bmod n$. However, it can be easily verified that an off-line *verifiable text attack* still applies to the above modified protocol. The attacker tries a guessed long term password P'_a and computes $T = (t \cdot P'^{-1}_a) \bmod n$. Based on this T, the attacker obtains $V = T^{ID_a} \bmod n$, then the intercepted integrity and originality check value $h(V, P_a)$ can be employed to verify the correctness of P'_a.

Alternatively, in the KRS-2 protocol, the inclusion of random integer r_2 is to prevent the possible verifiable text attack. Because the random integer r_2 selected by user A is sent in the ciphertext version, therefore it disables the attacker to guess and verify a possible P_a via the assistance of intercepted $h(V, r_2, P_a)$. To have a successful guess, the attacker should try both r_2 and P_a at the same time. However, since r_2 is a random integer in $[1, n-1]$ and this makes the off-line guessing attack infeasible.

The situation of having the recovery server acting as an oracle in order to derive other person's long term password P_a (in fact, it has already been described on how to prevent this drawback) can be avoided in the KRS-2 protocol. In the KRS-2 protocol, in order to recover the working key $K = h(\alpha^{pass} \bmod n)$ from the value $R = \alpha^{pass \cdot ID_a} \bmod n$, the active attacker can only pretend to be user A by sending ID_a, V, $r_2^e \bmod n$, and $h(V, r_2, P'_a)$ to the recovery server. This will however enable the server to identify a possible attack if $h(V, r_2, P'_a)$ is not correctly computed because of an incorrect P'_a.

5 Conclusions

From the view point of security engineering, commercial and even non-commercial key recovery is a necessary key management function in order to provide a complete and sound security application environment. Without a satisfying and practical key recovery solution, any research of strong cryptography can be in vain for many situations. This is especially true for the cases where tamper proof hardware for storing passwords or keys are not accessible. Furthermore, access of the tamper proof hardware also needs passwords. Because strong and secure cryptography always needs strong passwords or keys to apply and this implies a high risk of losing anything valuable if the keys are lost or forgotten. This situation applies to both individual requirement and organization requirement.

Since key recovery is considered to be a practical issue. Practical and important requirements of key recovery are first pointed out in this paper. Then, two commercial key recovery schemes based on these identified requirements are proposed. In the future research, other key recovery requirements for more complex environments or largely different applications will be considered. Key recovery schemes based on these different models will also be developed.

6 Acknowledgments

This research was supported in part by the National Science Council of the Republic of China under contract NSC 89-2213-E-008-055. The author wish to

thank the anonymous referees for their many valuable and constructive suggestions to improve both the technical contents and presentation skills. Some suggestions will be considered in the full version of this article.

References

1. NIST, "Escrowed Encryption Standard," *Federal Information Processing Standards Publication (FIPS PUB) 185*, 1994.
2. D.E. Denning, "The US key escrow encryption technology," *Computer Communication Magazine*, vol. 17, no. 7, pp. 453–457, July 1994.
3. D.E. Denning and M. Smid, "Key escrowing today," *IEEE Communication Magazine*, pp. 58–68, Sept. 1994.
4. D.E. Denning, "Key Escrow Encryption – the third paradigm," *Computer Security Journal*, vol. 11, no. 1, pp. 43–52, 1995.
5. D.E. Denning and D.K. Branstad, "A taxonomy for key escrow encryption systems," *Commun. ACM*, vol. 39, no. 3, pp. 34–40, 1996.
6. D.P. Maher, "Crypto backup and key escrow," *Commun. ACM*, vol. 39, no. 3, pp. 48–53, 1996.
7. S.T. Walker, S.B. Lipner, C.M. Ellison, and D.M. Balenson, "Commercial key recovery," *Commun. ACM*, vol. 39, no. 3, pp. 41–47, 1996.
8. S.T. Walker, S.B. Lipner, C.M. Ellison, D.K. Branstad, and D.M. Balenson, "Commercial key escrow: Something for everyone, now and for the future," *TISR #541*, Trusted Information Systems, Apr. 16, 1995.
9. E.H. Spafford, "Observing reusable password choices," *Purdue University Technical Report CSD–TR 92–049*, 31 July 1992.
10. E.H. Spafford, "Observations on reusable password choices," In *Proc. of the 3rd Security Symposium*, Usenix, September 1992.
11. E.H. Spafford, "OPUS: Preventing weak password choices," *Computers & Security*, vol. 11, no. 3, pp. 273–278, 1992.
12. R. Rivest, "The MD5 message digest algorithm," *RFC 1321*, Apr. 1992.
13. FIPS 180-1, "Secure Hash Standard," NIST, US Department of Commerce, Washington D.C., April 1995.
14. Y. Zheng and J. Pieprzyk and J. Seberry, "HAVAL - a one-way hashing algorithm with variable length of output," *Advances in Cryptology – AUSCRYPT'92*, Lecture Notes in Computer Science, Vol.718, Springer-Verlag, pp.83–104, 1993.
15. S.C. Pohlig and M.E. Hellman, "An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance," *IEEE Trans. on Inform. Theory*, vol. 24, no. 1, pp. 106–110, January 1978.
16. T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. on Inform. Theory*, vol. 31, no. 4, pp. 469–472, July 1985.
17. R.L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystem," *Commun. of ACM*, vol. 21, no. 2, pp. 120–126, 1978.