

Secure Key-Evolving Protocols for Discrete Logarithm Schemes

Cheng-Fen Lu and ShiuhPyng Winston Shieh*

Computer Science and Information Engineering Department
National Chiao Tung University, Taiwan 30050
{cflu,ssp}@csie.nctu.edu.tw

Abstract. This paper addresses the security and efficiency of key-evolving protocols. We identify forward-secrecy and backward-secrecy as the security goals for key-evolving and present two protocols to achieve these goals. The first protocol is operated in Z_p^* and is efficient for the secret-key holder; the second one is operated in Z_n^* , and is efficient for the public-key holder. For both protocols, we provide proofs and analysis for correctness, security and efficiency.

Keywords: Key Management, Key-Evolving, Forward-Secrecy, Backward-Secrecy.

1 Introduction

Over the past 20 years, public key cryptography has provided many signature schemes and public key cryptosystems. However, if a signing key or decryption key is compromised, it is regarded a total break of the system. To avoid this undesirable situation, one common practice is assigning each key a certain usage period and updating the key when entering a new period. Therefore, the security and efficiency of key updating or key-evolving becomes an important topic for key management [11].

Key-evolving is commonly employed for symmetric cryptosystems, where the sender and the receiver of a message share a common master key. Instead of using this shared key directly, they use it to derive a set of sub-keys and each sub-key is valid for a certain period or for a specific application [1].

In this paper, we focus on the key-evolving of asymmetric key systems, where the communication parties no longer possess the same master key. Instead, the secret key holder (the decryptor or the signer) generates the public and secret key base and the public key holder (the encryptor or the verifier) retrieves this public key base. In the beginning, the public key base is first signed by a Certificate Authority (CA). And then it is retrieved and verified by the public key holder.

Without a key-evolving protocol, the certificate retrieval and verification operations must be performed periodically. With a key-evolving protocol, the certificate retrieval and verification operation is performed only once for the public

* This work is supported in part by Ministry of Education, National Science Council of Taiwan, and Lee & MTI Center, National Chiao Tung University.

key base. Thereafter, both parties update their corresponding public or secret keys for the future periods.

The goal of secure key-evolving is to reduce the damage in case of a secret key compromise. The corresponding security notions have been investigated in papers of key management and key agreement [12,8,14]. There the properties such as forward-secrecy, backward-secrecy, key-independence were defined. Informally, forward-secrecy refers to that the compromise of one or several secret keys does not compromise previous secret keys. Likewise, backward-secrecy refers to that the compromise of one or several secret keys does not compromise future secret keys. Key-independence means the secret keys used in different periods are basically independent. Thus, even if the attacker finds out the secret key of a certain period, it gives him little advantage in finding the secret keys of other periods.

One related concept of the forward-secrecy is the *perfect forward-secrecy*, which assures that the “compromise of long-term keys does not compromise past session keys” [7,11]. Also in recent papers of *forward-secure* signature, signatures schemes with forward-secrecy properties were proposed [4,9]. The related concept of the backward-secrecy is the resistance to the *known-key attack*, which assures the compromise of past sessions keys will allow neither a passive adversary to compromise future session keys nor an active adversary to impersonate in the future [5,17,16,11]. Recently, Tzeng proposed two public key encryption schemes which also enjoy the property of key-independence but they require the help of the extra trusted agent (TA) in initial key distribution and at each key-evolving [15].

The security notions for the key-evolving protocol have been little investigated. Thus, we redefined the above security notions in the key-evolving setting. Besides, we presented and analyzed two key-evolving protocols for discrete logarithm schemes. The advantage of these key-evolving protocols is that no trusted agent is needed and they are applicable for both public key encryption schemes and signature schemes.

This paper is organized as follows. Section 2 describes the model and definitions. Section 3 presents the protocol based on the difficulty of computing discrete logarithm in Z_p^* . Section 4 presents the protocol based on the difficulty of factoring a large composite n . Section 5 discusses the possible extensions of the protocols. Section 6 concludes this paper.

2 Model and Definitions

Let (PKB, SKB) denote the pair of the public/secret key base, which is used to derive the key-pair in i -th period, (PK_i, SK_i) .

Definition 1 A key-evolving protocol consists of three algorithms (KG, g, f) :

1. *Public/Secret Key Base Generation Algorithm* $KG(k)$

Given a security parameter k , the secret key holder generates the public/secret key base (PKB, SKB) . SKB is kept secret at the secret key holder and PKB is then distributed to the users in the form of public certificate.

2. *Public Key-Evolving Algorithm $g()$*
 Given the period i and PKB , the public key holder computes $PK_i = g(PKB, i)$.
3. *Secret Key-Evolving Algorithm $f()$*
 Given the period i and SKB , the secret key holder computes $SK_i = f(SK_{i-1})$ or $f(SKB, i)$.

Next, some desirable properties of the key-evolving protocols are presented as follows:

Definition 2 *A key-evolving protocol is forward-secret if the compromise of SK_i will not compromise SK_j for all $j < i$.*

Definition 3 *A key-evolving protocol is backward-secret if the compromise of SK_i will not compromise SK_j for all $j > i$.*

Definition 4 *A key-evolving protocol is key-independent if it is forward-secret and backward-secret.*

Definition 5 *A key-evolving protocol is t -bounded key-independent if*

1. *The compromise of a set of secret keys C and $|C| > t$ will compromise all the secret keys.*
2. *The compromise of a set of secret keys C and $|C| \leq t$ does not help to compromise additional secret keys.*

3 Protocol 1

The first protocol is based on the difficulty of computing discrete logarithm in Z_p^* . It applies a technique of Feldman for the construction of a non-interactive verifiable secret sharing scheme [6]. This technique uses the secret sharing scheme of Shamir [13], where a polynomial $f(x)$ of degree t is employed. Polynomials have the following properties:

- I. Given $t + 1$ distinct points on the polynomial $f(x)$ degree t , namely $(x_0, y_0), (x_1, y_1), \dots, (x_t, y_t)$ and $y_i = f(x_i)$, all the $t + 1$ coefficients can be determined. In other words, the polynomial can be uniquely determined.
- II. Given t distinct points on the polynomial $f(x)$ degree t , namely $(x_1, y_1), (x_2, y_2), \dots, (x_t, y_t)$ and $y_i = f(x_i)$, the $t + 1$ coefficients cannot be uniquely determined.

The Protocol is presented in Figure 1, consisting of three algorithms:

First, the secret key holder executes the key base generation algorithm and publishes the public key base $PKB = (P_0, P_1, \dots, P_t) = (g^{a_0}, g^{a_1}, \dots, g^{a_t})$, where a_0, a_1, \dots, a_t are random. In contrast, $SKB = (a_0, a_1, \dots, a_t)$, the set of coefficients of $f(x)$, is kept secret. Also he publishes a hash function h that works as a simple randomizer and takes the index of the period to a number in Z_q , i.e. $h(i) \in Z_q$. Depending on the application, this $h(\cdot)$ can be required to be collision-free or be a permutation in Z_q .

1. Public/Secret Key Base Generation Algorithm $KG(1^n, t)$
 - (1) The secret key holder chooses a n -bit prime $p = 2q + 1$ with that q is a prime of at least 160-bits long, i.e. $q > 2^{160}$. Let G_q denote the subgroup of the quadratic residues modulo p and g the generator of G_q .
 - (2) The secret key holder chooses a t -degree polynomial $f(x)$ with its coefficients randomly chosen from Z_q and $f(x) = \sum_{j=0}^t (a_j x^j) \pmod{q}$. $SKB = (a_0, a_1, \dots, a_t)$, the coefficients of $f(x)$, will be kept secret and used to derive the secret keys for later periods.
 - (3) The secret key holder publishes the public key base information and a hash function, including:

$$PKB = (P_0, P_1, \dots, P_t) = (g^{a_0}, g^{a_1}, \dots, g^{a_t}),$$

and $h : N \rightarrow Z_q$.

2. Public Key-Evolving Algorithm
 Given the index i , the public key holder will update its public key of period i as follows:

$$PK_i = \prod_{j=0}^t (P_j)^{h(i)^j} \pmod{p}.$$

3. Secret Key-Evolving Algorithm
 Given the index i , the secret key holder will update the key of period i as:

$$SK_i = f(h(i)) \pmod{q}.$$

Fig. 1. Key-evolving protocol using Feldman’s technique

The public key holder retrieves and verifies PKB and $h(\cdot)$. Suppose that he needs the public key of the period i . He would then compute $PK_i = \prod_{j=0}^t (P_j)^{h(i)^j} \pmod{p}$.

SKB is the long-term system secret, protected separately from SK_i at the secret-key holder. Its protection follows the paper of Shamir secret-sharing scheme [13]. At the beginning of the period i , the secret key holder evaluates the secret key as $SK_i = f(h(i)) \pmod{q}$. The analysis of this protocol is as follows:

Correctness. The relation of $PK_i = g^{SK_i}$ is established because

$$\begin{aligned} PK_i &= \prod_{j=0}^t (P_j)^{h(i)^j} \pmod{p} = \prod_{j=0}^t (g^{a_j})^{h(i)^j} \pmod{p} \\ &= g^{\sum_{j=0}^t a_j \cdot h(i)^j} \pmod{q} = g^{f(h(i))} \pmod{q} = g^{SK_i}. \end{aligned}$$

Security. The following lemmas summarize the security property.

Lemma 1. *The ability of the attacker, who is able to compute the corresponding SK of a given PK, is equivalent to the solving of the Discrete Logarithm problem in Z_p^* .*

Proof: (\Leftarrow) This is trivial because $SK = \log_g PK$.

(\Rightarrow) We can build a DL-oracle using this attacker. Suppose we want to find $x = \log_g y$. We choose $PK = yg^a$ and give it the attacker. If the attacker outputs the secret key SK corresponding to PK , we can compute $x = \log_g y = \log_g (PK/g^a) = \log_g PK - a = SK - a$. Q.E.D.

Conjecture 1 *The protocol in Figure 1 is t -bounded key-independent.*

Remark:

Given $t + 1$ sets of keys of $(SK_{i_0}, i_0), (SK_{i_1}, i_1), \dots, (SK_{i_t}, i_t)$, the attacker can determine the coefficients of $f(x)$ and thus the secret keys of all periods.

If less than $t + 1$ secrets keys are compromised, then the polynomial for the secret key-evolving can not be unique determined, i.e. it has at least $q \geq 2^{160}$ free choices for the coefficients. Therefore, the only way to compute one extra secret key corresponding to a public key seems to compute the discrete logarithm in Z_p^* , which is intractable.

The interesting problems include (1) building a DL oracle using the successful attacker of breaking t -bounded key-independence property and (2) using the successful attacker to construct the oracle for Diffie-Hellman (DH) or Decisional Diffie-Hellman (DDH) problem.

Efficiency. For the secret key holder, this protocol is very efficient, requiring only the evaluation of a t -degree polynomial over Z_q . For the public key holder, this protocol requires t modular exponentiations and t modular multiplications. However, to reduce the on-line computation, the t modular exponentiations can be pre-computed.

4 Protocol 2

This protocol is based on the difficulty of computing discrete logarithms in Z_n^* , where n is the product of several large primes. It uses the same technique of Maurer-Yacobi in the design of non-interactive public-key distribution system [10]. Recently, this technique was also suggested by Anderson to build a forward-secret signature scheme [2]. All operations are performed in Z_n^* , where factoring n is hard and g is the element of the maximal order in Z_n^* . A lemma from Maurer and Yacobi showed that every square modulo n has a discrete logarithm to the base of g . And the knowledge of the factorization of n is the trapdoor for solving the discrete logarithm problem; namely,

1. For the secret key holder, with the trapdoor knowledge of the factors of n , he can compute $SK_i = \log_g(PK_i)$.
2. For someone without this trapdoor knowledge, given PK_i , the computation of SK_i is intractable.

This protocol is presented in Figure 2, consisting of three algorithms. In the beginning, the secret key holder first executes the key base generation

1. Public/Secret Key Base Generation Algorithm

The secret key holder chooses a k -bit composite $n = p_1 p_2 \cdots p_r$ where the factorization of n is intractable and $(p_1 - 1)/2, (p_2 - 1)/2, \dots, (p_r - 1)/2$ are pairwise relatively prime. Next, he chooses an element g of the maximal order in Z_n^* .

Then he broadcasts the public key base $PKB = (n, g, H, PK_0)$, where the following requirements are met.

- (a) $SKB = (p_1, p_2, \dots, p_r)$, the set of factors of n , is kept secret to the secret key holder,
- (b) $H(\cdot) : \{0, 1\}^* \rightarrow Z_n^*$ is a cryptographic hash function,
- (c) PK_0 is a random number.

2. Public Key-Evolving Algorithm

Given the period i , the public key holder evaluates for $j = 1$ to i ,

$$PK_j = (H(PK_{j-1}))^2 \pmod n \in QR_n.$$

3. Secret Key-Evolving Algorithm

In the beginning of period i , the secret key holder first compute the public key by one hash-and-square

$$PK_i = (H(PK_{i-1}))^2 \pmod n \in QR_n.$$

With the knowledge of the factors of n , the secret key holder computes

$$SK_i = \log_g(PK_i).$$

Fig. 2. Key-Evolving based on Maurer-Yacobi Scheme

algorithm and publishes the public key base $PKB = (n, g, H, PK_0)$, where n and g are defined as above, H is a cryptographic hash function, and PK_0 is a random number. The secret key base SKB , consisting of the factors of n , is kept secret.

The public key holder retrieves and verifies the public key base PKB . Suppose that he needs the public key of the period i . He would then perform a series of hash-and-square operations to get $PK_1 = (H(PK_0))^2 \pmod n$, $PK_2 = (H(PK_1))^2 \pmod n$, \dots , $PK_i = (H(PK_{i-1}))^2 \pmod n$. If he could store some of these intermediate values, he can reduce the hash-and-square computations.

SKB is the long-term system secret, protected separately from SK_i at the secret holder. Its protection follows the paper of Maurer and Yacobi [10]. At the beginning of the period i , the secret key holder performs the hash-and-square $PK_i = (H(PK_{i-1}))^2 \pmod n$ and then calculates the corresponding secret key $SK_i = \log_g(PK_i)$ with his trapdoor knowledge. The analysis of this protocol is as follows:

Correctness. $PK_i = g^{SK_i}$ is established because SK_i is computed by the secret key holder using the factors of n (trapdoor information).

Security. We will show that the security of this protocol depends on the following relations between the computing discrete logarithm in Z_n^* and factoring n [3,10,11].

- I. If the discrete logarithm problem in Z_n^* can be solved efficiently, then n can be factored efficiently.
- II. If n can be factored efficiently and if the discrete logarithm in every $Z_p^*, p|n$ can be solved efficiently, then the discrete logarithm problem in Z_n^* can be solved efficiently.

The following lemmas summarize the security property. First, we show that successful attacker can be used to construct a DL oracle in Z_n^* . Next, we state a lemma from Maurer and Yacobi about the factorization of n using a DL-oracle [10]. For detailed algorithm, please refer to the original paper. Finally, we prove the security in the random oracle model.

Lemma 2. *The ability of the attacker, who is able to compute the corresponding SK of a given PK, is equivalent to the solving of the Discrete Logarithm problem over Z_n^* .*

Proof: (\Leftarrow) This is trivial because $SK = \log_g PK$.

(\Rightarrow) We can build a DL-oracle using this attacker. Suppose we want to find $x = \log_g y$. We choose $PK = yg^a$ and give it the attacker. If the attacker outputs the secret key SK corresponding to PK , we can compute $x = \log_g y = \log_g (PK/g^a) = \log_g PK - a = SK - a$. Q.E.D.

Lemma 3. [10] *Let n be the product of distinct odd primes p_1, \dots, p_r and let g be primitive in each of the prime fields $GF(p_i)$ for $1 \leq i \leq r$. Then computing discrete logarithms modulo n to the base of g is at least as difficult as factoring n completely.*

Theorem 1. *In the random oracle model, the key-evolving protocol in Figure 2 is key-independent if factoring n is hard.*

Proof: We will show how to factor n if the successful attacker breaks the key-independent property. First, we will build a DL oracle using the successful attacker. Next, we follow the algorithm of Maurer-Yacobi to factor n using this DL oracle.

Given y in the maximal cyclic group of Z_n^* , we will use the successful attacker to compute $x = \log_g y$. First, we will generate a set of random numbers $\{r_i | i = 1, 2, \dots, T + 1\}$. And we control the random oracle to output $\{r_1, r_2, \dots, r_{k-1}\}$ at the first $k - 1$ queries. Thus, the following relations hold.

$$\begin{array}{ll}
 r_1 = H(PK_0), & PK_1 = r_1^2 \\
 r_2 = H(PK_1), & PK_2 = r_2^2 \\
 \vdots & \vdots \\
 r_{k-1} = H(PK_{k-2}), & PK_{k-1} = r_{k-1}^2.
 \end{array}$$

At the beginning of the k periods, we force the random oracle to output yg^a , i.e.

$$yg^a = H(PK_{k-1}), PK_k = y^2g^{2a}.$$

For periods from $k + 1$ to $T + 1$, the random oracle proceeds as usual.

Second, we give (PK_i, SK_i) and the above public key set to the attacker A . If he succeeds to break forward-secrecy and forward-secrecy, A would return (PK_j, SK_j) for some $j \neq i$.

The probability that $j = k$ is $\frac{1}{T}$. If this happens, we have $SK_j = SK_k = \log_g PK_k = \log_g y^2g^{2a} = 2\log_g y + 2a = 2x + 2a$. Because $PK_j = PK_k$ is a square in Z_n^* , SK_j will be even. Though the order $\lambda(n)$ is unknown, the discrete logarithm of y can be computed as follows:

$$x = \frac{1}{2}SK_j - a.$$

Given this DL oracle, we follow Maurer-Yacobi's method to factor n . Therefore, if factoring n is hard, this key-evolving protocol is key-independent in the random oracle model. Q.E.D.

Efficiency. This protocol is especially efficient for the public key holder because only i hash-and-square operations are needed and this can be pre-computed. Also if the public key holder stores some intermediate values, the load of the public key-evolving is further reduced. Thus it is suitable for public key holders with low computation power such as mobile devices or smart cards. The load of secret key holder is mainly the computation of discrete logarithm of $Z_p^*, p|n$. The computation of discrete logarithm relies on the choices of p [10]; for example, one can employ Pohlig-Hellman algorithm when $p|n, (p - 1)/2$ is chosen to be the product of primes of moderate size.

5 Possible Extensions

The possible extensions of the protocols include: First, the immediate problem is to prove the conjecture 1 using stronger assumption such as DH or DDH assumption or using DL assumption in a restricted model.

Second, can we provide a public-key holder efficient protocol in Z_p^* instead of in Z_n^* ? And is it possible to use this key-evolving model for specific algebraic structure or RSA related schemes?

Third, since each DL key pair is associated with a period, it will be desirable to provide period-stamping or time-stamping service of the public key. Furthermore, applying these two party protocols to multi-party computation would be an interesting task.

6 Conclusion

We first discussed about the importance of key-evolving and then identified several security notions, namely forward-secrecy, backward-secrecy, and key independence. Next, two concrete constructions were proposed. One protocol is

based on a technique of Feldman, and it is efficient for the secret-key holder; the other one is a variant of the Maurer-Yacobi protocol, and it is efficient for the public-key holder. Besides, we provide proofs and analysis for correctness, security and efficiency. Finally, we pointed out the possible extensions for future work.

References

1. M. Abdalla and M. Bellare. Increasing the lifetime of a key: a comparative analysis of the security of re-keying techniques. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT ’ 2000*, Kyoto, Japan, 2000.
2. R.J. Anderson. Two remarks on public key cryptology. In *Rump Session Eurocrypt’97*.
3. E. Bach. Discrete logarithm and factoring. *Report no. UCB/CSD 84/186, Comp. Sc. Division (EECS), University of California, Berkeley*, June 1984.
4. M. Bellare and S. K. Miner. A forward-secure digital signature scheme. In *Proc. 19th International Advances in Cryptology Conference – CRYPTO ’99*, pages 431–448, 1999.
5. D. E. Denning and M. S. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(7):533–536, 1981.
6. P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Symposium on Foundations of Computer Science (FOCS)*, pages 427–437. IEEE Computer Society Press, 1987.
7. C. G. Guenther. An identity-based key-exchange protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology - EuroCrypt ’89*, pages 29–37, Berlin, 1989. Springer-Verlag. Lecture Notes in Computer Science Volume 434.
8. Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *Proceedings of the 7th ACM conference on Computer and communications security (CCS-00)*, pages 235–244. ACM Press, 2000.
9. H. Krawczyk. Simple forward-secure signatures from any signature scheme. In Sushil Jajodia and Pierangela Samarati, editors, *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS-00)*, pages 108–115. ACM Press, 2000.
10. U. M. Maurer and Y. Yacobi. A non-interactive public-key distribution system. *Designs, Codes and Cryptography*, vol. 9, no. 3:305–316, 1996.
11. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. Boca Raton, 1997.
12. A. Perrig. Efficient collaborative key management protocols for secure autonomous group communication. In *International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC ’99)*, July 1999.
13. A. Shamir. How to share a secret. *Communication of ACM*, pages 612–613, (Nov. 1979).
14. M. Steiner, G. Tsudik, and M. Waidner. Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769–780, August 2000.
15. W. Tzeng and Z. Tzeng. Robust key-evolving public key encryption schemes. Record 2001/009, Cryptology ePrint Archive, 2001.

16. Y. Yacobi. A key distribution s “paradox”. In Alfred J. Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - Crypto '90*, pages 268–273, Berlin, 1990. Springer-Verlag. Lecture Notes in Computer Science Volume 537.
17. Y. Yacobi and Z. Shmueli. On key distribution systems. In Gilles Brassard, editor, *Advances in Cryptology - Crypto '89*, pages 344–355, Berlin, 1989. Springer-Verlag. Lecture Notes in Computer Science Volume 435.