Robust Key-Evolving Public Key Encryption Schemes*

Wen-Guey Tzeng, Zhi-Jia Tzeng
Department of Computer and Information Science
National Chiao Tung University
Hsinchu, Taiwan 30050
E-mail:{tzeng,zjtzeng}@cis.nctu.edu.tw

Abstract

We propose a key-evolving paradigm to deal with the key exposure problem of public key encryption schemes. The key evolving paradigm is like the one used for forward-secure digital signature schemes. Let time be divided into time periods such that at time period j, the decryptor holds the secret key SK_j, while the public key PK is fixed during its lifetime. At time period j, a sender encrypts a message m as $\langle j, c \rangle$, which can be decrypted only with the private key SK_j. When the time makes a transit from period j to $j+1$, the decryptor updates its private key from SK_j to SK_{j+1} and deletes SK_j immediately. The key-evolving paradigm assures that compromise of the private key SK_j does not jeopardize the message encrypted at the other time periods.

We propose two key-evolving public key encryption schemes with z-resilience such that compromise of z private keys does not affect confidentiality of messages encrypted in other time periods. Assuming that the DDH problem is hard, we show one scheme semantically secure against passive adversaries and the other scheme semantically secure against the adaptive chosen ciphertext attack under the random oracle.

Keywords: forward secure, key evolving, public key encryption.

1 Introduction

Consider the situation that the sender Alice sends a message m to the decryptor Bob with Bob's public key PK. An attacker Carol who does not know Bob's private key SK at present time eavesdrops and records the ciphertext $c = E(PK, m)$. If Carol manages to get Bob's private key SK later on, she can get the message m no matter how much time has elapsed. The message m may carry information that is useful for a long period of time.

There are many ways to protect Bob's private key SK. One way is to replace Bob's public key when his private key is exposed. In this case every user has to update Bob's public key in its database when Bob replaces his public key.

*Research supported in part by the National Science Council grant NSC-89-2213-E-009-180 and by the Ministry of Education grant 89-E-FA04-1-4, Taiwan, ROC.

This is quite costly. Furthermore, it may not be practical since Bob may not be aware of losing his private key. Another way is to protect Bob's private key on a secure device, such as smartcards, so that key exposure is not possible. This is quite costly, too. The third way is to use a threshold scheme to distribute SK to n trusted agents (TA's) such that k out of them can fully recover SK. When Bob receives a ciphertext c, he uses the TA's to decrypt c in a distributed way. In this case TA's not only bear heavy load of computation, but also should stay on-line always to provide decryption service. There are of course other solutions. We cannot enumerate them all.

We propose a key evolving paradigm, like the one used in forward-secure digital signature schemes [3], to deal with the key exposure problem of public-key encryption schemes. Let time be divided into periods, starting with 0. The public key PK of the decryptor Bob is fixed for the whole lifetime. Bob's private key at time period j is SK_j, $j \geq 0$. When Alice wants to send message m to Bob at time period j, she encrypts m as $\langle j, c\rangle = E(PK, j, m)$ such that one can decrypt $\langle j, c\rangle$ only with SK_j. The key evolving paradigm is that when time runs from period j to period $j+1$, Bob updates his private key from SK_j to SK_{j+1} and then deletes SK_j immediately, possibly with help from a trusted agent TA. If Carol breaks into Bob's system during time period j and gets Bob's private key SK_j at that time period, she cannot get the private keys in the other time periods directly since they have been deleted. With the key evolving paradigm, even if Bob is not aware of losing his private key, he can be sure that only those ciphertexts encrypted in the time period are exposed. In the next time period, security of newly encrypted messages are guaranteed. We consider the situation that Bob always decrypts the received messages in a short time such that the ciphertext of time period j that arrives at time period j', $j' > j$ will be discarded or re-transmitted by the sender.

Our key evolving paradigm requires that the public key be fixed for the whole lifetime. The reason is that the public key is widely distributed to other users such that it is not practical to request users check validity and timeliness of the public key frequently and on-line. Thus, a public key should remain as stable as possible. On the other hand, the decryptor is a single point. It is reasonable to allow him to interact with a trusted agent for updating his private key. Key update occurs every time period and can be finished in a prompt, which shall not incur network and computational problems to the trusted agent. The key evolving paradigm adds to the array of mechanisms of protecting secrets, such as distributed, threshold, and proactive cryptography [7, 10, 14, 17, 23].

A key evolving paradigm is *z-resilient* if one cannot decrypt the ciphertext $\langle j, c\rangle$ even he gets the private keys $SK_{j_1}, SK_{j_2}, \ldots, SK_{j_z}$ of z time periods, for $j \neq j_l, 1 \leq l \leq z$. If the decryptor computes his private keys with help from a trusted agent, we assume that the trusted agent is not broken in by the attacker. To ensure this security, we distribute the secret to a set of TA's and protect them with threshold and proactive cryptography.

Our results. We propose two z-resilient public-key encryption schemes such that z times of key exposures do not release any information pertinent to ciphertexts that are encrypted with non-exposed private keys. One scheme is semantically secure against passive adversaries and the other is semantically secure against the chosen ciphertext attack under the random oracle model. The size of a public key is independent of the total number of time periods, but dependent on resilience. On the other hand, the size of a private key is a constant. Without counting the pre-computation time for each time period, both encryption and decryption operations take 2 modular exponentiations. The pre-computation time is independent of time period, but dependent on resilience. Therefore, our schemes are very efficient in key size and operations of encryption and decryption.

Though the key update of our schemes needs help from TA, nevertheless, TA can be treated as an escrow agent in case of the decryptor's emergent need to decrypt ciphertexts of previous time periods. To distribute trust of TA and provide security against transient (mobile) attackers, we propose distributed and proactive key update algorithms.

We also consider distributed decryption in our KE-ENC schemes in which the decryption key is shared among a set of decryptors. We stress that even all decryptors are broken in at time period j, only SK_j is exposed. In the next time period, the decryptors share a new SK_{j+1} and the security is restored if the adversary are transient.

2 Definitions and preliminaries

We provide definitions for a key-evolving public-key encryption scheme and discuss its security under various security models. In the definition, we may assume that there is a trusted agent TA, who holds some secret for updating private keys of the decryptor.

Definition 2.1 *A* key-evolving public key encryption *scheme* KE-ENC *consists of four algorithms* $\langle KG, UPD, E, D \rangle$*:*

1. *Key generation algorithm* KG*: it is a probabilistic polynomial-time algorithm that takes as input a security parameter* n *and possibly other parameters and returns a* base public key PK *and corresponding base private key* SK_0*. That is,* $KG(1^n) = (PK, SK_0, s)$*, where* s *is the system secret (trapdoor). If TA is involved, KG distributes* s *to it; otherwise, KG simply discards* s*.*

2. *Private key update algorithm* UPD*: it takes an input the public key PK, the private key* SK_{j-1} *of time period* $j-1, j \geq 1$*, and outputs the new private key* SK_j *of time period* j*. That is,* $UPD(PK, SK_{j-1}, j) = SK_j$*. If TA is involved, UPD interacts with TA to compute* SK_j*.*

3. *Encryption algorithm* E*: it takes as input the base public key PK, a message* m*, and the time-period indicator* j *and outputs the ciphertext* c *of* m *at time period* j*. We use* $E(PK, m, j) = \langle j, c \rangle$ *to denote the encryption. E might be probabilistic.*

4. *Decryption algorithm* D*: it takes as input the ciphertext* $\langle j, c \rangle$ *of time period* j *and the private key* $SK_{j'}$ *of time period* j' *and outputs* m *if and only if* $j = j'$*. That is,* $D(SK_j, E(PK, m, j)) = m$*.*

We set no limit on the number of time periods. Key evolving can continue till the limit set by the security parameter n*. The maximum number of time periods is* 2^n*.*

We assume a single TA for simplicity. In practicality, we distribute trust to multiple trusted agents such that each TA_i holds a share s_i of the system secret s. The decryptor with private key SK_{j-1} and the TA's together can compute SK_j in a secure way, such as, through the secure multi-party computation. We discuss this in Section 4.

Operation. The system first generates a key pair (PK, SK_0) to a decryptor by the key generation algorithm KG. If TA is used, the system also sets TA for interacting with the decryptor to update the private key at the end of each time period. The public key PK is treated identically to that in a standard encryption model for registration, distribution, revocation, etc. The time is divided into periods, starting at 0, which is some reference point of time, say, January 1, 2000 and each time period covers a week. Time periods serve as "time-stamps", which are known to all, that is, every body knows the number for the current time period. At the end of each time period $j-1$, the decryptor runs the algorithm UPD, possible with TA, to update his private key SK_{j-1} of time period $j-1$ to SK_j of time period j. The decryptor need immediately delete SK_{j-1} after getting SK_j for itself's protection. Therefore, an attacker who breaks in at time period j can get the decryptor's private key SK_j only. When a user wants to send a message m to the decryptor at time period j, he computes $E(PK, m, j) = \langle j, c \rangle$ and sends it to the decryptor. Note that the time period is part of the ciphertext. When the decryptor gets $\langle j, c \rangle$, he uses his private key SK_j to decrypt the ciphertext. It may be that the decryptor gets $\langle j, c \rangle$ at time period j', $j' > j$. In this case, the decryptor cannot decrypt it. Therefore, KE-ENC is better used for applications that decryption is done in a short period of time.

The key evolving scheme cannot guarantee the desired security if the decryptor does not delete the old private key after getting the new one. It would be better to ensure erasure of old private keys by some system mechanism.

Security. We consider the conventional security models for public-key encryption schemes. We first consider the attack from passive adversaries and then the adaptive chosen ciphertext attack, which is the strongest attack against an encryption scheme.

Let KG, E and D be the key generation, encryption and decryption algorithms of a public key encryption scheme. Also, let PK and SK be the public and private keys, respectively. A public key encryption scheme is *semantically secure against passive adversaries* if a passive adversary $\mathcal{A}$, which is probabilistic polynomial-time, cannot distinguish the ciphertexts of any two messages m_0 and m_1 [15] with a non-negligible advantage. In particular, for any passive adversary $\mathcal{A}$,

$$\Pr[KG(1^n) \to (PK, SK) : \mathcal{A}(PK, E(PK, m_b)) = b] = 1/2 + \epsilon(n),$$

where $\epsilon(n)$ is negligible and the probability is taken over b and the random coin tosses of KG, E and $\mathcal{A}$.

The adaptive chosen ciphertext attack on an encryption scheme works as follows [22]. An adversary $\mathcal{A}$ of the attack has two probabilistic polynomial-time algorithms A_1 and A_2. A_1 takes as input PK, makes some queries to the decryption oracle adaptively, and outputs two messages m_0 and m_1. Then, the encryption oracle randomly chooses a bit b and encrypts m_b as $c = E(PK, m_b)$. A_2 takes as input PK, c, m_0 and m_1, makes some queries to the decryption oracle DO in an adaptive way, and outputs b'. The decryption oracle DO takes as an input a ciphertext c', other than c, and returns its corresponding plaintext m'. If c' is invalid, DO outputs '?'. A public key encryption scheme is *semantically secure against the adaptive chosen ciphertext attack* if, for any adversary $\mathcal{A} = (A_1, A_2)$,

$$\Pr[KG(1^n) \rightarrow (PK, SK); A_1^{DO}(PK) \rightarrow (m_0, m_1) : \\ A_2^{DO}(PK, E(PK, m_b)) = b] = 1/2 + \epsilon(n),$$

where $\epsilon(n)$ is negligible and the probability is taken over b and all coin tosses of KG, E, A_1 and A_2.

For the resilience property, we consider the security of of the scheme after the attacker gets some private keys additionally.

Definition 2.2 (Resilience) *Assume a security model for public-key encryption scheme. A key evolving public-key encryption scheme* KE-ENC*=(KG, UPD, E, D) is* z-resilient *if the attacker cannot break the encryption scheme under the assumed security model even if he gets z private keys* SK_{j_1}, SK_{j_2},..., SK_{j_z}.

Since the attacker gets z private keys, breaking does not include obtaining the corresponding plaintext of a ciphertext that is encrypted with any of the private keys.

The *random oracle model* assumes that hash functions used in a scheme are "truly random" hash functions [4]. Although security under the random oracle model is not rigid, it does provide satisfactory security argument to related schemes in most cases [6].

Assumption. We need a standard assumption of solving the decisional discrete logarithm (DDH) problem [5]. Breaking our schemes is reduced to solving the DDH problem. Let G_q be a group of a large prime order q. Consider the following two distribution ensembles R and D:

- $R = (g_1, g_2, u_1, u_2) \in G_q^4$, where g_1 and g_2 are generators of G_q;
- $D = (g_1, g_2, u_1, u_2)$, where g_1 and g_2 are generators of G_q and $u_1 = g_1^r$ and $u_2 = g_2^r$ for $r \in Z_q$.

The *DDH problem* is to distinguish the distribution ensembles R and D. That is, we would like to find a probabilistic polynomial-time algorithm $\mathcal{A}$ such that

$$|\Pr[\mathcal{A}(R_n) = 1] - \Pr[\mathcal{A}(D_n) = 1]| = \epsilon(n)$$

is non-negligible, where R_n and D_n are the size-n distributions of R and D, respectively.

A *trapdoor one-way permutation* is a probabilistic polynomial-time algorithm $\mathcal{G}$ that takes as input 1^n and outputs a triple of algorithms (f, f^{-1}, d), where f and f^{-1} are inverses of each other and deterministic polynomial-time algorithms and d is a probabilistic polynomial-time algorithm. The range of $d(1^n)$ is a subset of $\{0,1\}^k$ and f and f^{-1} on the range of $d(1^n)$ are permutations. Furthermore, for any probabilistic polynomial-time algorithm $\mathcal{A}$,

$$\epsilon(n) = \Pr[\mathcal{G}(1^n) \rightarrow (f, f^{-1}, d); d(1^n) \rightarrow x; f(x) \rightarrow y : \mathcal{A}(f, d, y) = x]$$

is negligible. For convenience, we shall call f, not $\mathcal{G}$, a "trapdoor one-way" permutation.

1. Algorithm KG($1^n, z$):
   (a) Randomly an n-bit prime $p = 2q + 1$, where q is also a prime. All operations work over Z_p except being stated otherwise. Let G_q be the subgroup of order q in Z_p^* and g be a generator of G_q.
   (b) Randomly select a degree-z polynomial $f(x) = \sum_{i=0}^{z} a_i x^i \bmod q$.
   (c) Set the public key and base private key as
   $$PK = \langle g^{a_0}, g^{a_1}, \ldots, g^{a_z} \rangle \text{ and } SK_0 = \langle f(0) \rangle.$$
   (d) Let TA hold $f(x_j)$, for some random $x_j \in Z_q, 1 \leq j \leq z$.
2. Algorithm UPD(PK, SK_{j-1}): the decryptor Bob and TA together compute $SK_j = f(j)$ from their shares in a secure distributed way.
3. Algorithm E(PK, m, j): randomly select $k \in Z_q$, compute
   $$\alpha = g^k, \quad s = m \cdot \prod_{i=0}^{z} (g^{a_i})^{kj^i} = m \cdot g^{kf(j)}$$
   and return the ciphertext $\langle j, \alpha, s \rangle$.
4. Algorithm D($SK_j, \langle j, \alpha, s \rangle$): compute and return $m = s/\alpha^{f(j)}$.

Figure 1: Scheme 1 – discrete logarithm based KE-ENC with z-resilience and semantic security against passive adversaries.

3 Our protocols

We first propose a KE-ENC scheme, based on the discrete logarithm problem, with z-resilience and show that the scheme is semantically secure against passive adversaries. We then modify the scheme to become semantically secure against the adaptive chosen ciphertext attack under the random oracle model.

3.1 KE-ENC against passive adversaries

Our KE-ENC scheme with z-resilience and security against passive adversaries is Scheme 1, which is shown in Figure 1. The main idea is to treat $f(j)$ as the time stamp. The public key $g^{f(j)} = \prod_{i=0}^{z} (g^{a_i})^{j^i}$ at time period j can be computed from the base public key $PK = \langle g^{a_0}, g^{a_1}, \ldots, g^{a_z} \rangle$. Key updating involves in computing $f(j)$ from $f(j-1)$ with help from TA. We use the ElGamal-type encryption scheme for encryption and decryption.

Correctness. The correctness of decryption follows easily since

$$s/\alpha^{f(j)} = m \cdot \prod_{i=0}^{z} (g^{a_i})^{kj^i} / g^{kf(j)} = m \cdot g^{kf(j)} / g^{kf(j)} = m.$$

Efficiency. In each time period j, we can pre-compute

$$\prod_{i=0}^{z}(g^{a_i})^{j^i} = g^{a_0}(g^{a_1}(g^{a_2}(\cdots)^j)^j)^j = g^{f(j)}.$$

It needs $z+1$ modular exponentiations and z modular multiplications, which is independent of the time period j. After pre-computation, each encryption takes 2 modular exponentiations and 1 modular multiplication only. Each decryption takes 1 modular exponentiation and 1 modular division. Therefore, our scheme is very efficient in computation.

The public key consists of $(z+1)$ n-bit values and the private key consists of one n-bit value only, which are both independent of the total number of time periods. The number of time periods can be as large as 2^n.

Security. We now show that Scheme 1 is semantically secure against passive adversaries even the decryptor's system is broken in z times.

Theorem 3.1 *Assume that the DDH problem is hard. For Scheme 1, given public key $PK=\langle g^{a_0}, g^{a_1}, \ldots, g^{a_z}\rangle$ and z private keys $SK_{j_1}, SK_{j_2}, \ldots, SK_{j_z}$, no probabilistic polynomial-time adversary can distinguish the ciphertexts of any two messages m_0 and $m_1 \in G_q$ at time period j, $j \neq j_l, 1 \leq l \leq z$.*

Proof. We reduce the DDH problem to the distinguishing problem of m_0 and m_1 at time period j. Assume that m_0 and m_1 are distinguishable with a non-negligible advantage $\epsilon(n)$ by a probabilistic polynomial-time adversary $\mathcal{A}$. We construct another probabilistic polynomial-time adversary $\mathcal{B}$ to solve the DDH problem (with a non-negligible advantage $\epsilon(n)$).

Let $\langle g_1, g_2, u_1, u_2\rangle$ be the input to the DDH problem. We let $a = \log_{g_1} g_2$ and randomly select $b_{j_1}, b_{j_2}, \ldots, b_{j_z} \in Z_q$. There is a degree-$z$ polynomial $f'(x) = \sum_{i=0}^{z} a'_i x^i \bmod q$ that passes $k+1$ points $(j, a), (j_1, b_{j_1}), (j_2, b_{j_2}), \ldots, (j_z, b_{j_z})$. Note that $a = f'(j)$. Although we don't know a, we can compute $g^{a'_i}$, $0 \leq i \leq z$, by Lagrange's interpolation method. We set the public key $PK = \langle g^{a'_0}, g^{a'_1}, \ldots, g^{a'_z}\rangle$ with the base generator g_1 and z private keys $SK_{j_i} = \langle b_{j_i}\rangle$, $1 \leq i \leq z$. To pose a ciphertext challenge to $\mathcal{A}$, the encryption oracle randomly selects a bit b and computes the ciphertext $\langle j, u_1, m_b u_2\rangle$ of m_b. The adversary $\mathcal{B}$ outputs 1 for the input (g_1, g_2, u_1, u_2) if and only if $\mathcal{A}$'s output is equal to b for the challenge $(j, u_1, m_b u_2)$. We can see that if $(g_1, g_2, u_1, u_2) = (g_1, g_2, g_1^r, g_2^r)$, $r \in Z_q$, the ciphertext outputted by the encryption oracle has the right distribution for each m_0 and m_1. That is, $\Pr[\mathcal{A}(PK, \langle j, u_1, m_b u_2\rangle) = b] = 1/2 + \epsilon(n)$. If $(g_1, g_2, u_1, u_2) = (g_1, g_2, g_1^{r_1}, g_2^{r_2})$, $r_1, r_2 \in Z_q$, the distributions of ciphertexts for m_0 and m_1 are equal. Thus, $\Pr[\mathcal{A}(PK, \langle j, u_1, m_b u_2\rangle) = b] = 1/2$. Therefore, $\mathcal{B}$ solves the DDH problem with a non-negligible advantage $\epsilon(n)$. □

3.2 KE-ENC against adaptive chosen ciphertext attack

We modify Scheme 1 to become semantically secure against the adaptive chosen ciphertext attack under the random oracle model. The idea is to use randomness of hash functions. We construct a (probabilistic) trapdoor one-way permutation

$$h_{j,y}(k, r) = (g^k, r \cdot y^k, k \oplus H(j, r))$$

1. Algorithm $\mathrm{KG}(1^n, z)$:
   (a) Randomly select an n-bit prime $p = 2q + 1$, where q is also a prime. Let G_q be the subgroup of order q in Z_p^* and g be a generator of G_q.
   (b) Randomly select a degree-z polynomial $f(x) = \sum_{i=0}^{z} a_i x^i \bmod q$.
   (c) Set the public key and base private key as
   $$PK = \langle g^{a_0}, g^{a_1}, \ldots, g^{a_z} \rangle \text{ and } SK_0 = \langle f(0) \rangle.$$
   (d) Let TA hold $f(x_j)$, $x_j \in Z_q$, $1 \leq j \leq z$.
2. Algorithm $\mathrm{UPD}(PK, SK_{j-1})$: the decryptor Bob and TA together compute $SK_j = \langle f(j) \rangle$ from their shares in a secure distributed way.
3. Algorithm $\mathrm{E}^{H_1,H_2,H_3}(PK, m, j)$: randomly select $k \in Z_q$ and $r \in G_q$, compute
   $$\alpha = g^k,\ \beta_1 = r \cdot (\prod_{i=0}^{z} (g^{a_i})^{j^i})^k = r \cdot g^{f(j) \cdot k},$$
   $$\beta_2 = k \oplus H_1(j, r),\ s = m \oplus H_2(j, r, k),\ h = H_3(j, r, k, m),$$
   and return the ciphertext $\langle j, \alpha, \beta_1, \beta_2, s, h \rangle$.
4. Algorithm $\mathrm{D}^{H_1,H_2,H_3}(SK_j, \langle j, \alpha, \beta_1, \beta_2, s, h \rangle)$:
   (a) Compute $r = \beta_1 / \alpha^{f(j)}$, $k = \beta_2 \oplus H_1(j, r)$ and $m = s \oplus H_2(j, r, k)$.
   (b) Check whether $\alpha = g^k$ and $h = H_3(j, r, k, m)$. If it is so, return m; otherwise, return '?'.

Figure 2: Scheme 2 – discrete logarithm based KE-ENC with z-resilience and semantic security against the adaptive chosen ciphertext attack under the random oracle model.

for time period j, where $y = g^{f(j)}$ and the trapdoor is $f(j)$. The message is encrypted with one-time pad $H_2(j, r, k)$. The hash value $H_3(j, r, k, m)$ forces the querist to be aware of m.

The modified scheme, Scheme 2, is shown in Figure 2, in which H_1, H_2, H_3 are collision-resistant hash functions with output length dependent on n.

Correctness. We can see that

$$\beta_1 / \alpha^{f(j)} = r \cdot g^{kf(j)} / g^{kf(j)} = r,$$
$$\beta_2 \oplus H_1(j, r) = (k \oplus H_1(j, r)) \oplus H_1(j, r) = k, \text{ and}$$
$$s \oplus H_2(j, r, k) = (m \oplus H_2(j, r, k)) \oplus H_2(j, r, k) = m.$$

Efficiency. Encryption and decryption computation is similar to that of Scheme 1. After pre-computation, each encryption takes 2 modular exponentiations, 1 modular multiplication and 3 hash operations. Each decryption takes

1 modular exponentiation, 1 modular division and 3 hash operations. Again, computation time is independent of the time period j. Therefore, this scheme is as efficient as Scheme 1.

Again, Scheme 2's public key consists of $(z+1)$ n-bit values and private key consists of only one n-bit value, which are both independent of the total number of time periods.

Security. Assume the random oracle model, which postulates that H_1, H_2 and H_3 are "truly random" hash functions. We show that Scheme 2 is semantically secure against the adaptive chosen ciphertext attack.

Theorem 3.2 *Assume the random oracle model and that the discrete logarithm problem is hard. For Scheme 2, given public key $PK = \langle g^{a_0}, g^{a_1}, \ldots, g^{a_z} \rangle$ and z private keys $SK_{j_1}, SK_{j_2}, \ldots, SK_{j_z}$, no probabilistic polynomial-time adversary with access to the decryption oracle is able to find m_0 and m_1 in G_q such that their ciphertexts at time period j, $j \neq j_l, 1 \leq l \leq z$, are computationally distinguishable.*

Proof. Let n be the security parameter (or complexity measure). Assume that the scheme does not have the desired properties. That is, there is an adversary $\mathcal{A}$, knowing z private keys and querying the decryption oracle adaptively, that can distinguish the ciphertexts of two messages m_0 and m_1 with a non-negligible probability. $\mathcal{A}$ has two probabilistic polynomial-time procedures A_1 and A_2. A_1 takes as input the public key and z private keys, makes some queries to the decryption and hash oracles, and outputs two messages m_0 and m_1. A_2 takes as input the public key PK, z private keys, m_0, m_1, and the ciphertext $E(PK, m_b, j)$, queries the decryption and hash oracles adaptively, and outputs b', where b is a random bit. We say that A attacks the scheme successfully if $\Pr[b = b'] = 1/2 + \epsilon(n)$ for some non-negligible function $\epsilon(n)$, where the probability is taken over b and the internal coin tosses of A_1, A_2, KG and z private keys. We show that we can use $\mathcal{A}$ to construct a probabilistic polynomial-time algorithm $\mathcal{B}$ for solving the discrete logarithm problem with a non-negligible probability.

The proof idea is that by the random oracle paradigm, one cannot get hash results without querying the hash oracles H_1, H_2 and H_3 except with a negligible probability. It means that one has to know the input to the hash oracles. Therefore, one has to know r and k, and thus can solve the discrete logarithm problem, which contradicts to the assumption.

Assume that $\mathcal{A}$ makes q_d queries to the decryption oracle and q_h queries to the hash oracles within time bound $t(n)$ and gains an $\epsilon(n)$ advantage. Without loss of generality, let the hash results of H_1, H_2 and H_3 be of length n. Our simulation for the hash and decryption oracles is as follows. The simulator S maintains a table T for the random oracles for consistency. When $\mathcal{A}$ makes a hash query (j', r') to H_1, S checks whether (j', r') is already in T. If (j', r') is in T, it returns the hash result in T; otherwise, it returns a random value c and puts $\langle (j', r'), c \rangle$ into T. The same is done for the queries to H_2 and H_3. For the decryption oracle, if $(j', \alpha', \beta_1', \beta_2', s', h')$ is queried, S checks whether $c_1 = h_1(j', r')$, $c_2 = h_2(j', r', k')$ and $c_3 = h_3(j', r', k', m')$ are in T and m' is the plaintext of $(j', \alpha', \beta_1', \beta_2', s', h')$. We can verify the later by checking whether $\alpha' = g^{k'}$, $\beta_1' = r' \cdot (g^{f(j')})^{k'}$, $\beta_2' = k' \oplus c_1$, $s' = m' \oplus c_2$ and $h' = c_3$. If it is so, it returns m'; otherwise, it returns '?', which means that the input ciphertext is invalid. Note that S may be wrong on returning '?' since $(j', \alpha', \beta_1', \beta_2', s', h')$

may be valid. But, this occurs with probability $1/2^{3n}$ only due to the random oracle model.

Let E_1 be the event that $\mathcal{A}$ does make a query (j, r) to the H_1 hash oracle, or a query (j, r, k) to the H_2 hash oracle, or a query (j, r, k, m) to the H_3 hash oracle. Let E_2 be the event that $\mathcal{A}$ makes a valid decryption query $(j', \alpha', \beta'_1, \beta'_2, s', h')$, but without querying (j', r')), (j', r', k'), and (j', r', k', m') to hash oracles, where $r' = \beta'_1 / y^{\log_g \alpha'}$, $k' = \beta'_2 \oplus H_1(j', r')$, $m' = s' \oplus H_2(j', r', k')$, and $y = g^{f(j)}$. Then, we have

$$\Pr[E_2] \leq q_d 2^{-3n} \leq t(n)2^{-3n}.$$

Since without querying the hash oracles on proper r, k and m, $\mathcal{A}$ has no advantage in distinguishing m_0 from m_1. Thus, we have

$$\Pr[A_2(PK, SK's, m_0, m_1, E(PK, m_b, j)) = b | \neg(E_1 \vee E_2)] = 1/2.$$

Since $\mathcal{A}$ has $\epsilon(n)$ advantage in guessing b, we have

$$\begin{aligned}
1/2 + \epsilon(n) &= \Pr[A_2(PK, SK's, m_0, m_1, E(PK, m_b, j)) = b] \\
&= \Pr[A_2(\cdot) = b | E_2] \Pr[E_2] \\
&\quad + \Pr[A_2(\cdot) = b | \bar{E}_2 \wedge E_1] \cdot \Pr[E_1 \wedge \bar{E}_2] \\
&\quad + \Pr[A_2(\cdot) = b | \bar{E}_2 \wedge \bar{E}_1] \cdot \Pr[\bar{E}_1 \wedge \bar{E}_2] \\
&\leq t(n)2^{-3n} + \Pr[E_1] + (1/2)\Pr[\bar{E}_1].
\end{aligned}$$

This implies that $\Pr[E_1] \geq 2\epsilon(n) - t(n)2^{-3n+1}$.

The algorithm $\mathcal{B}$ of solving the discrete logarithm problem works as follows. On input (g, y), it randomly selects a degree-z polynomial $f'(x) = \sum_{i=0}^{z} a_i x^i \bmod q$. It sets the decryption key SK_j of time period j as $f'(j)$ and $SK_{j_i} = f'(j_i)$ as the given exposed decryption key of time periods j_i, $1 \leq i \leq z$. $\mathcal{B}$ calls $A_1(g^{a_0}, g^{a_1}, \ldots, g^{a_z}, SK_{j_1}, SK_{j_2}, \ldots, SK_{j_z})$ to find m_0 and m_1, while using the simulator S to answer queries. It then randomly selects r, c_1, c_2 and c_3 and feeds the ciphertext $\langle j, c \rangle = \langle j, y, ry^{SK_j}, c_1, c_2, c_3 \rangle$ to A_2, where c_1, c_2 and c_3 are consistent with S's answers to hash queries. Finally, $\mathcal{B}$ outputs k if (j, r), (j, r, k), or (j, r, k, m) have been queried to the hash oracles and $g^k = y$, where k is either queried in (j, r, k) or (j, r, k, m), or is equal to $c_1 \oplus H_1(j, r)$.

By the above argument, A_2 queries (j, r), (j, r, k) or (j, r, k, m) of appropriate form with probability $\Pr[E_1] = 2\epsilon(n) - t(n)2^{-3n-1}$. Therefore, the probability of $g^k = y$ is $2\epsilon(n) - t(n)2^{-3n+1}$ at least, which is non-negligible. □

4 Key evolving with TA

In our schemes, key evolving is done with help from the TA, which holds $f(x_1), f(x_2), \ldots, f(x_z)$. We consider distributed and proactive methods to evolve private keys of the decryptor.

4.1 Distributing TA's secret

Assume that there are z TA's and each TA_i holds a share $f(x_i)$, $1 \leq i \leq z$, where x_i's are distinct and large enough so that the maximum time period never reaches them. At time period $j-1$, the decryptor Bob holds $SK_{j-1} = f(j-1)$. Bob and TA's would like to compute $SK_j = f(j)$, which shall be known to Bob only. Assume that each pair of Bob and TA's share a private channel by which secret information can be passed between them.

We treat Bob as TA_0 and let $x_0 = j-1$. By the Lagrange interpolation method, the polynomial that passes shares of Bob and TA's is

$$f(x) = \sum_{k=0}^{z}(f(x_k) \cdot \prod_{0\le i\ne k\le z} \frac{x-x_i}{x_k-x_i}).$$

TA_k can compute $s_k = f(x_k) \cdot \prod_{0\le i\ne k\le z} \frac{j-x_i}{x_k-x_i}$, $0 \le k \le z$. Therefore, $f(j) = \sum_{k=0}^{z} s_k$. Our goal is that TA's together compute $f(j)$ and only TA_0 knows $f(j)$. Furthermore, each TA_i does not reveal any information about its share $f(x_i)$. Note that $x_0, x_1, \ldots, x_z$ are known to all TA's. The distributed protocol D-UPD for computing $f(j)$ securely is as follows.

1. Each TA_l, $1 \le l \le z$, selects a degree-z polynomial $h_l(x) = \sum_{i=1}^{z} a_{l,i}x^i + s_l$ over Z_q and sends $h_l(x_i)$ to TA_i, $0 \le i \le z$, via the private channel between them. Let $F(x) = \sum_{i=1}^{z} h_i(x)$.
2. Each TA_l, $0 \le l \le z$, computes its share $F(x_l) = \sum_{i=1}^{z} h_i(x_l)$.
3. Each TA_l, $1 \le l \le z$, sends $F(x_l)$ to TA_0 via the private channel between them.
4. TA_0 then computes the constant coefficient $\sum_{i=1}^{z} s_i$ of $F(x)$ from $F(x_0)$, $F(x_1)$,..., $F(x_z)$ by the Lagrange interpolation method and his private key $SK_j = f(j) = s_0 + \sum_{i=1}^{z} s_i$ at time period j.

Correctness follows the protocol easily. In the protocol, each TA_i does not have any information about another TA_j's share $f(x_j)$.

We can make the computation verifiable by letting each TA_l publish $g^{a_{l,0}}$, $g^{a_{l,1}}$,..., $g^{a_{l,z}}$ [13]. Each TA_i, $0 \le i \le z$, verifies whether he receives the right share $h_l(x_i)$ from TA_l, $1 \le l \le z$, by checking $g^{h_l(x_i)} = \prod_{k=0}^{z} g^{a_{l,k}{x_i}^k}$.

4.2 Proactivizing TA's shares

We use proactive cryptography to protect TA's shares further. Let PS_i, $1 \le i \le n$, be n proactive servers. In practicality, we can make TA's as proactive servers. We proactivize each TA_i's share $f(x_i)$ into PS_i's. by the proactive (t,n)-secret sharing scheme [16]. Each evolving time period $j-1$ is divided into sub-periods $p_1, p_2, \ldots, p_b$, where $p_1, p_2, \ldots, p_{b-1}$ are refresh sub-periods in which the shares of $f(x_1), f(x_2), \ldots, f(x_z)$ are refreshed among PS_i's. In the last sub-period p_b, $f(j-1)$ of the decryptor is updated to $f(j)$.

For simplicity, we let $t = n$. Let $r_i(x)$ be a degree-(n-1) polynomial over Z_q with constant coefficient $f(x_i)$, $1 \le i \le z$. Each PS_j, $1 \le j \le n$, holds a share $r_i(j)$ for the share $f(x_i)$, $1 \le i \le z$, and refreshes them in every sub-time period p_i, $1 \le i \le b-1$, as follows.

1. Each PS_l, $1 \le l \le n$, selects n degree-(n-1) polynomials $r_{l,i}(x)$ over Z_q, $1 \le i \le n$, whose constant coefficients are all 0. PS_l sends $r_{l,i}(j)$, $1 \le i \le n$, to PS_j, $1 \le j \le n$, via the private channel between them.

2. After receiving shares from other proactive servers, PS_l, $1 \leq l \leq n$, updates its shares to $r'_i(l) = r_i(l) + \sum_{j=1}^{n} r_{j,i}(l)$.

To update the decryptor Bob's decryption key $f(j-1)$, we assume Bob as PS_0 and $x_0 = j-1$. Let

$$\rho_{j,k} = \prod_{0 \leq i \neq k \leq z} \frac{j - x_i}{x_k - x_i} \text{ and } \lambda_l = \prod_{1 \leq i \neq l \leq n} \frac{-i}{l-i},$$

where $0 \leq k \leq z$ and $1 \leq l \leq n$. We have

$$\begin{aligned} f(j) = \sum_{k=0}^{z} \rho_{j,k} f(x_k) &= \sum_{k=1}^{z} \sum_{l=1}^{n} \rho_{j,k} \lambda_l r_k(l) + \rho_{j,0} f(x_0) \\ &= \sum_{l=1}^{n} (\sum_{k=1}^{z} \rho_{j,k} \lambda_l r_k(l)) + \rho_{j,0} f(x_0). \end{aligned}$$

Let $s_l = \sum_{k=1}^{z} \rho_{j,k} \lambda_l r_k(l)$, which can be computed by PS_l, $1 \leq l \leq n$. Our proactive key update scheme P-UPD for computing $SK_j = f(j)$ in the subperiod p_b of time period j is as follows.

1. Each PS_l, $1 \leq l \leq n$, selects a degree-n polynomial $h_l(x) = \sum_{i=1}^{n} a_{l,i} x^i + s_l$ over Z_q and sends $h_l(x_i)$ to PS_i, $0 \leq i \leq n$, via the private channel between them. Let $F(x) = \sum_{i=1}^{n} h_i(x)$.
2. Each PS_l, $0 \leq l \leq n$, computes its share $F(x_l) = \sum_{i=1}^{n} h_i(x_l)$.
3. Each PS_l, $1 \leq l \leq n$, sends $F(x_l)$ to PS_0 via the private channel between them.
4. PS_0 then computes the constant coefficient $\sum_{l=1}^{n} s_l$ of $F(x)$ from $F(x_0)$, $F(x_1)$,..., $F(x_n)$ by the Lagrange interpolation method and his private key $SK_j = f(j) = \rho_{j,0} f(x_0) + \sum_{l=1}^{n} s_l$.

Correctness follows the above equations. Again, PS_i does not have any information about another PS_j's shares $r_1(j), r_2(j), \ldots, r_z(j)$. We can also make the computation verifiable by the verifiable secret sharing method as that in Section 4.1.

5 Distributed KE-ENC schemes

It is sometimes desirable to have distributed decryption in which the decryption key is shared among a set of decryptors B_i, $1 \leq i \leq n$.

We assume that each decryptor B_i holds a share s_i of the secret key $SK_j = f(j)$ via a polynomial $t(x) = \sum_{k=1}^{z} a_i x^i + f(j) \bmod q$ such that $s_i = t(i)$. For Scheme 1, on receiving a ciphertext (α, s), B_i computes the partial plaintext $m_i = \alpha^{s_i} \bmod p$. With $z+1$ partial plaintexts $m_{i_1}, m_{i_2}, \ldots, m_{i_{z+1}}$, one can computes the plaintext

$$m = s / \prod_{k=1}^{z+1} (m_{i_k})^{\lambda_k} \bmod p = s / \alpha^{f(j)} \bmod p,$$

where λ_k's are appropriate Lagrange coefficients. This method can be used in Scheme 2 also since knowing $\alpha^{f(j)} \bmod p$ is sufficient to decrypt the ciphertext encrypted at time period j.

6 Discussion

Due to technical difficulty, the decryptor's key evolving in our schemes need help from TA. We think that it is difficult to construct a key-evolving encryption scheme without TA. The reason is that a public key encryption scheme has a trapdoor that is necessary in decrypting ciphertext. Thus, the decryptor need have the trapdoor. On the other hand, this trapdoor is sufficient in key evolving in both forward and backward directions. Therefore, break-in to the decryptor results in obtaining the trapdoor, and thus all private keys. This is unlike the key-evolving digital signature scheme in which the signer need only construct a "verifiable" signature for a message.

7 Conclusion

We have proposed two KE-ENC schemes to deal with the key exposure problem of public key cryptosystems. Our schemes are semantically secure against passive adversaries under the standard model and the adaptive chosen ciphertext attack under the random oracle model, respectively. To distribute trust and provide security against transient attackers, we propose distributed key update D-UPD and proactive key update P-UPD schemes.

By the techniques of [8], we can modify Scheme 1 to become semantically secure against the adaptive chosen ciphertext attack under the standard model before breaking. However, it only retains security against passive adversaries after break-in.

The main drawback of these schemes is that key update needs help from TA. It would be interesting to find a KE-ENC scheme, in which key update can be done by the decryptor alone in a single-decryptor mode.

References

[1] M. Abdalla, L. Reyzin, "A new forward-secure digital signature scheme", *Proceedings of Advances in Cryptology – Asiacrypt 2000*, Lecture Notes in Computer Science 1976, pp.116-129, Springer-Verlag, 2000.

[2] J. Anzai, N. Matsuzaki, T. Matsumoto, "A quick group key distribution scheme with "entity revocation"", *Proceedings of Advances in Cryptology – Asiacrypt 99*, Lecture Notes in Computer Science 1716, pp.333-347, Springer-Verlag, 1999.

[3] M. Bellare, S.K. Miner, "A forward-secure digital signature scheme", *Proceedings of Advances in Cryptology – Crypto 99*, Lecture Notes in Computer Science 1666, pp.431-448, Springer-Verlag, 1999.

[4] M. Bellare, P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols", *Proceedings of the First ACM Conference on Computer and Communications Security*, pp.62-73, 1993.

[5] D. Boneh, "The decision Diffie-Hellman problem", *Proceedings of the Third Algorithmic Number Theory Symposium*, Lecture Notes in Computer Science 1423, pp.48-63, Springer-Verlag, 1998.

[6] R. Canetti, O. Goldreich, S. Halevi, "The random oracle methodology revisited", *Proceedings of the 30th ACM Annual Symposium on Theory of Computing*, pp.209-218, 1998.

[7] R. Canetti, R. Gennaro, A. Herzberg, D. Naor, "Proactive security: long-term protection against break-ins", *CryptoBytes* 3(1), 1997.

[8] R. Cramer, V. Shoup, "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack", *Proceedings of Advances in Cryptology – Crypto '98*, Lecture Notes in Computer Science 1462, pp.13-25, Springer-Verlag, 1998.

[9] I. Damgård, "Towards practical public key cryptosystems secure against chosen ciphertext attacks", *Proceedings of Advances in Cryptology – Crypto 91*, Lecture Notes in Computer Science 576, pp.445-456, Springer-Verlag, 1991.

[10] Y. Desmedt, Y. Frankel, "Threshold cryptosystems", *Proceedings of Advances in Cryptology – Crypto 89*, Lecture Notes in Computer Science 435, pp.307-315, Springer-Verlag, 1989.

[11] W. Diffie, P.C. Van Oorschot, M.J. Weiner, "Authentication and authenticated key exchanges", *Designs, Codes and Cryptography* 2, pp.107-125, 1992.

[12] T. ElGamal, "A public-key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Transactions on Information Theory* 31(4), pp.469-472, 1985.

[13] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing", *Proceedings of the 28th IEEE Annual Symposium on the Foundations of Computer Science*, pp.427-437, 1987.

[14] P. Gemmel, "An introduction to threshold cryptography", *CrytoBytes* 2(7), 1997.

[15] S. Goldwasser, S. Micali, "Probabilistic encryption", *Journal of Computer and System Sciences* 28, pp.270-299, 1984.

[16] A. Herzberg, S. Jarcki, H. Krawczyk, M. Yung, "Proactive secret sharing, or how to cope with perpetual leakage", *Proceedings of Advances in Cryptology – Crypto 95*, Lecture Notes in Computer Science 963, pp.339-352, Springer-Verlag, 1995.

[17] I. Ingemarsson, G.J. Simmons, "A protocol to set up shared secret schemes without the assistance of a mutualy trusted party", *Proceedings of Advances in Cryptology – Eurocrypt 90*, Lecture Notes in Computer Science 473, pp.266-282, Springer-Verlag, 1990.

[18] C.H. Lim, P.J. Lee, "Another method for attaining security against adaptively chosen ciphertext attacks", *Proceedings of Advacnes in Cryptology – Crypto 93*, Lecture Notes in Computer Science 773, pp.420-434, 1993.

[19] T. Pedersen, "A threshold cryptosystem without a trusted party", *Proceedings of Advances in Cryptology – Eurocrypt 91*, Lecture Notes in Computer Science 547, pp.522-526, Springer-Verlag, 1991.

[20] T. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing", *Proceedings of Advances in Cryptology – Crypto 91*, Lecture Notes in Computer Science 576, pp.129-140, Springer-Verlag, 1991.

[21] D. Pointcheval, J. Stern, "Security proofs for signature schemes", *Proceedings of Advances in Cryptology – Eurocrypt 96*, Lecture Notes in Computer Science 1070, pp.387-398, Springer-Verlag, 1996.

[22] C. Rackoff, D. Simon, "Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack", *Proceedings of Advances in Cryptology – Crypto 91*, Lecture Notes in Computer Science 576, pp.433-444, Springer-Verlag, 1991.

[23] A. Shamir, "How to share a secret", *Communications of the ACM* 22(11), pp.612-613, 1979.