February 15, 1999
RT0305
Computer Science 15 pages

Research Report

Periodical Multi-Secret Threshold Cryptosystems

Masayuki Numao

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Limited Distribution Notice

This report has been submitted for publication outside of IBM and will be probably copyrighted if accepted. It has been issued as a Research Report for early dissemination of its contents. In view of the expected transfer of copyright to an outside publisher, its distribution outside IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or copies of the article legally obtained (for example, by payment of royalities).

Periodical Multi-Secret Threshold Cryptosystems

Masayuki Numao

Tokyo Research Laboratory, IBM Japan, Ltd.
1623-14, Shimo-Tsuruma, Yamato, Kanagawa 242-8502, JAPAN
numao@jp.ibm.com

Abstract. A *periodical multi-secret threshold cryptosystem* enables a sender to encrypt a message by using a cyclical sequence of keys which are shared by n parties and periodically updated. The same keys appear in the same order in each cycle, and thus any subset of $t+1$ parties can decrypt the message only in the periodical time-frames, while no subset of t corrupted parties can control the system (in particular, none can learn the decryption key). This scheme can be applied to a timed-release cryptosystem whose release time is determined when the number of share update phases equals the period of the sequence. The system is implemented by sharing a pseudo-random sequence generator function. It realizes $n \geq 3t+1$ robustness, and is therefore secure against an adversary who can corrupt at most one third of the parties.

1 Introduction

The concept of "timed-release crypto" was first introduced by May [May93] and further studied by Rivest et al. [RSW96]. Its goal is to encrypt a message so that it cannot be decrypted by anyone, not even the sender, until a predetermined amount of time has passed. According to [RSW96], there are two known approaches for implementing it:

1. Use trusted agents who promise not to reveal certain information until a specified time.
2. Use "time-lock puzzles"-computational problems that cannot be solved without running a computer for at least a certain amount of time.

The problem with the first approach is that the user has to totally trust the agents in all matters from the maintenance of the key to the provision of the decryption service at the specified time. Moreover, since there is no direct correspondence between the key and the time, the handling of the decryption time by the agents is purely operational. The second approach is even less practical, because the decryption time is calculated from the amount of computational steps. Therefore, the receiver has to start the decryption process as soon as he receives the encrypted message, and has to use the best computer available, since the decryption might otherwise be delayed.

In this paper, we will propose an alternative method for realizing "timed-release crypto," using a threshold cryptosystem [Des88] [DF90] that shares a function (possibly a pseudo-random sequence generator) for generating a sequence of random values with some period. Note that in any pseudo-random number generator, the sequence is repeated after a specific period. Our idea is to use the period of the sequence as the time needed to obtain a pair of encryption and decryption keys, so that if the user encrypts the message by using the public key, the corresponding secret key is available only in a periodical timeframe. Since the key pairs are periodically available in the same timeframe, this scheme does not satisfy the original definition of a timed-release key. But it does if the sender encrypts the message by using the encryption key in some timeframe and sends the message in the next timeframe. In this case, the receiver has to wait until one cycle of the period passes to get the decryption key.

Our approach bears some similarity to proactive secret sharing [OY91] [HJKY95] [FGMY98], in the sense that each party updates its share by multiparty computation in each time frame. But the difference is that the proactive scheme aims at maintaining one secret for a long period, whereas in our scheme the secrets themselves are updated so that different secrets appear in different timeframes.

Although our scheme maintains multiple secrets, it differs from the $(c, d; k, n)$-multi-secret sharing scheme [FY92], in which k different secrets are maintained among n parties in such a way that at least d parties are necessary to recovery all k secrets, whereas no subset of c parties can deduce anything. In our system, the secret (seed) is maintained in a $(t, n)-$threshold scheme, but it is updated by a function-sharing scheme that results in the generation of a sequence of secrets.

Our approach is also different from the proactive random number generator [CH94], because it deals only with the generation of fresh random values that are computationally independent of the previous states, but cannot maintain the period of the sequence, which is essential to timed-release key generation.

1.1 Our results

We define a new class of threshold cryptosystems that can be used for timed-release crypto, and describe an efficient and robust implementation. More precisely, in this paper:

1. We define periodical multi-secret threshold cryptosystems as a class of threshold cryptosystems.
2. We implement a periodical multi-secret threshold cryptosystem by sharing a pseudo-random sequence generation (PRSG) function.
3. We show that our implementations have t-resiliency in $n \geq 3t + 1$, where n and t are the total number of parties and the number of parties corrupted by a mobile adversary, respectively.

1.2 Organization of the paper

The paper is organized as follows. In section 2, we define our periodical multi-secret threshold cryptosystem. Since we have to share a pseudo-random sequence generator function, we give a multiparty protocol for a sequence generator in section 3. Then, in section 4, we describe periodical multi-secret threshold cryptosystems that we implemented by sharing a linear congruence generator (LCG) and a Blum-Blum-Shub (BBS) generator based on the ElGamal encryption scheme. Finally, we present our conclusions in section 5. In appendix A, we explain some of the cryptographic techniques that are used in our protocols as basic tools.

2 Model and Definition

2.1 Periodical Multi-secret Threshold Cryptosystem

Let E be a public key scheme defined by three protocols: key-pair generation, encryption, and decryption. Key update is then introduced as an additional function of key-pair generation to update the key-pair from (EK_t, DK_t) to (EK_{t+1}, DK_{t+1}).

A periodical multi-secret threshold cryptosystem $T\bigcup_{TpE}$ for scheme E distributes the operation of key generation (update), and decryption among a set of n parties $P_1, \ldots, P_n$. Let DK_t and $DK_{i,t}$ be a secret key and its share for party P_i in timeframe t respectively; then (EK_t, DK_t) forms a key-pair in timeframe t. $T\bigcup_{T_pE}$ is defined by two protocols:

$T\bigcup_{T_p}$**-KeyUpdate,** a randomized protocol that takes a previous share $DK_{i,t-1}$ as private input for party P_i, and returns as public output the public encryption key EK_t for timeframe t and as private output for party P_i a value $DK_{i,t}$, such that (1) $DK_{1,t}, \ldots, DK_{n,t}$ constitute a k-out-of-n threshold secret sharing of DK_t, which is a secret key for timeframe t corresponding to the public key EK_t, and (2) $DK_t = DK_{t'}$ only when $t' = t \pmod{T_p}$, where T_p is the period specified by the system.

$T\bigcup_{T_p}$**-Decrypt,** a protocol in which each party P_i takes as public input a ciphertext $C = E_{EK_t}(M)$ and as secret input his share $DK_{i,t'}$ and returns as public output the message M only when $t' = t \pmod{T_p}$.

2.2 Communication Model

We assume that all the communication links are secure (i.e., private and authenticated), which allows us to focus on high-level aspects of the protocols.

2.3 Adversary

We assume that an adversary, A, can corrupt up to t out of the n parties in the network. Since the system changes its internal states by updating the parties' shares, the ability to deal with only a static adversary is not sufficient to provide security. A "mobile malicious adversary" is allowed to move among parties over time with the limitation that it can only control up to t parties in a timeframe. Here we assume that the adversary is static within a timeframe and moves to other parties in the next timeframe.

2.4 Resiliency

The resiliency of a distributed protocol is defined by comparing it with its corresponding centralized protocol: t-resiliency means that the protocol will compute a correct output even in the presence of a malicious adversary who can corrupt up to t parties (robustness), and that the adversary cannot obtain any information other than what he can obtain from the centralized protocol (privacy).

3 Multiparty Computation for Sequence Generation

In this section, we will describe a multiparty protocol for the sequence generation, since we want to share a pseudo-random sequence generation function to update the secret keys. As sequence generators, we will use the linear congruence generator and the BBS generator (see appendix A.1), because they are simple (need only one multiplication per next number generation) and the periods can be determined by the parameters (the coefficients, modulo, and seed).

On the basis of the multiparty protocol [BGW88] [CCD88], every computation can be distributed in a secure and robust way. The authors of the above papers proved that the bound of the privacy threshold is $k < n/2$, where n is the total number of parties and k is the number of parties colluding to obtain the secret. They also proved that the robustness threshold is $k' < n/3$, where k' is the number of parties corrupted by a malicious adversary. Since the above papers assume a computationally unbounded adversary, the share verification process is carried out by error-correcting codes.

By accepting the use of encryption (which satisfies only weaker notions of security), we can employ a more practical non-interactive verification scheme using homomorphic commitment. In this paper, we will use a verifiable secret-sharing scheme (VSS) proposed by Feldman [Fel87] as a basic tool (see appendix A.3). Let $a, b \in Z_p$ be two secrets that are shared by using polynomials $f_a(x), f_b(x)$ of order k, respectively, and let $c \in Z_q, c \neq 0$ be some constant. The values $a + b$ and $c \cdot a$ can be simply computed by having each party perform the same operation on its shares $f_a(i) + f_b(i)$ and $c \cdot f_a(i)$, respectively; this results in the sharing of new polynomials $f_{a+b}(x) = f_a(x) + f_b(x)$ and $f_{ca}(x) = c \cdot f_a(x)$, whose free coefficients are $a + b$ and $c \cdot a$, respectively. Feldman's VSS is also easy to perform, because the correctness of the resulting shares is checked by using the publicly known values (commitments) computed as follows: $g^{a_i+b_i} = g^{a_i} g^{b_i}$ and $g^{c \cdot a_i} = (g^{a_i})^c$.

3.1 Degree Reduction

To share the multiplication $a \cdot b$, three computing steps are necessary: (1) share multiplication, (2) randomization, and (3) degree reduction, because step (1) generates a new polynomial $h(x) = f_a(x)f_b(x)$, which is of order $2k$ and not irreducible [BGW88].

Here, we show a VSS-based robust degree reduction protocol. Let $f(x)$ be a polynomial of order k to share the secret s. The share $s_i = f(i)$ was distributed by a dealer to party i by means of Feldman's VSS. Now, the parties want to reduce the threshold from k to k' $(k' < k)$ (without relying on the dealer, of course). First, each party i generates a random polynomial $f^{(i)}(x)$ of order k' whose free coefficient is set to s_i, and shares the polynomial by using Feldman's VSS. Then, other parties verify their shares sent from party i and announce whether they are correct or not. If party i has more than k correct announcements, it is qualified. Finally, after $k+1$ qualified parties are determined, each party j computes his new share using the Lagrange interpolation $s'_j = \sum_{i \in \Lambda} \lambda_i f^{(i)}(j)$, where Λ_i denotes a set of qualified parties and λ_i denotes the coefficient computed by the Lagrange interpolation. Note that the coefficients are constant when the set is determined. The protocol is given below:

3.2 Protocol for Robust Degree Reduction

Input of Party i: $f(i)$ of a polynomial $f(x)$ of order k representing a secret s.

Public Input: $\alpha_i = g^{f_i}$ for $0 \le i \le k$, where f_i is the i-th coefficient of the $f(x)$. Note that $\alpha_0 = g^s$.

Protocol

1. Party i shares the value of $f(i)$ by means of a random polynomial $f^{(i)}(x)$ of order $k'(k' < k)$ such that $f^{(i)}(0) = f(i)$. Then she broadcasts $\beta_j^{(i)} = g^{f_j^{(i)}}$ for $0 \le j \le k$, where $f_j^{(i)}$ is the j-th coefficient of the $f^{(i)}(x)$, as public checking parts for VSS. Note that $\beta_0^{(i)} = g^{f^{(i)}(0)} = g^{f(i)}$, which is publicly computable from the initial input of α_j for $0 \le j \le k$.
2. Party j receives the share $f^{(i)}(j)$ from i, and then announces i-correct or i-wrong according to whether the equation
$$g^{f^{(i)}(j)} \overset{?}{=} \prod_{m=0}^{k} {\beta_m^{(i)}}^{j^m} \pmod{p}$$
holds or not.
3. A set Λ of good parties of cardinality $k+1$ is defined as one in which the parties have more than k i-corrects. (If more than $k+1$ parties have more than k i-corrects, those with smaller id numbers are chosen.) Party j then computes $\sum_{i \in \Lambda} \lambda_i f^{(i)}(j)$, which results in the sharing of a polynomial

$f'(x) = \sum_{i \in \Lambda} \lambda_i f^{(i)}(x)$ of order k'. Public values for VSS checking are also computed as follows:

$$\alpha'_j = g^{\sum_{i \in \Lambda} \lambda_i f_j^{(i)}} = \prod_{i \in \Lambda} (\beta_j^{(i)})^{\lambda_i}, \ for\ 0 \leq j \leq k'.$$

Lemma 3.2 The protocol 3.2 has the following properties:
(Robustness) The new shares computed at the end of the protocol correspond to the secret s. (That is, any subset of $k' + 1$ of the new shares can reconstruct the secret s, and an adversary who can control up to k' parties cannot alter the result.)
(Privacy) Apart from the value g^s, an adversary who can control up to k' cannot learn anything about the secret.
(Freshness) The new share of each party is independent of its old share.

Sketch of Proof (Robustness) We can assume that there are $k + 1 > k'$ honest parties. Thus at least $k + 1$ parties are announced as i-correct by $k + 1$ parties. Thus the honest parties can compute the new share. By VSS, the party i can check that the share from party j lies on a polynomial which represents party i's original share. It can also confirm that the new secret is the same as the original secret by comparing α_0 and α'_0.
(Privacy) We can construct a simulator that, given the input of the corrupted parties, simulates the process of local secret sharing by party j. We can assume that party j is honest and that parties i for $1 \leq i \leq k'$ are corrupted. Using the k' shares $f^{(j)}(i)$ for $1 \leq i \leq k'$, together with the secret s, the simulator can set $k' + 1$ equations and determine the polynomial of order $k' + 1$. It then distributes the shares and VSS checking parts that have the same distribution as in the real protocol.
(Freshness) Party j's new share is computed as $\sum_{i \in \Lambda} \lambda_i f^{(i)}(j)$, where $f^{(i)}(j)$ is generated by party i by using random coefficients.

3.3 Robust Sequential Multiplication

The difficulty in applying Feldman's VSS to multiplication is that we cannot compute $g^{a \cdot b}$ from g^a and g^b. Thus, in some protocol [GRR98], the parties need to prove that the new published commitment $g^{a_i \cdot b_i}$ is really obtained from $(g^{a_i})^{b_i}$ (or $(g^{b_i})^{a_i}$), by using the zero-knowledge proof of equality of discrete-logs [Cha90], which proves that $DL_g(g^{b_i}) = DL_{g^{a_i}}(g^{a_i \cdot b_i})$ without requiring any information about b_i to be provided. However, the above ZKIP requires a verifier and interaction between the parties and a verifier, which reduces the advantages of non-interactive verification by Feldman's VSS.

Cerecedo et al. [CMI93] showed that VSS can be applied in a special case of multiplication where one of the multipliers is generated by the joint-shared-secret protocol. We will first review this protocol before proposing the general multiplication protocol:

Let $f(x) = a_0 + \cdots + a_k x^k$ be a polynomial of order k used to share the value $a = a_0$. Party i holds the value $f(i) \bmod q$, and the values $\alpha_m = g^{a_m} \bmod p$ for $0 \leq m \leq k$ are publicly known. She then generates $r^{(i)}(x) = r_0^{(i)} + \cdots + r_k^{(i)} x^k$ of order k and $z^{(i)}(x) = z_1^{(i)} x + \cdots + z_{2k}^{(i)} x^{2k}$ of order $2k$ for Joint-Random-VSS and Joint-Zero-VSS (see appendix A.4), respectively, which means that party j holds values $r^{(i)}(j), z^{(i)}(j)$ and the values $\beta_m = g^{r_m^{(i)}}, (0 \leq m \leq k)$ and $\gamma_m = g^{z_m^{(i)}}, (0 \leq m \leq 2k)$ are publicly known. Now let $y^{(i)}(x) = f(x) r^{(i)}(x) + z^{(i)}(x)$ be a polynomial of order $2k$ that is used to share the value $a \cdot r_0^{(i)}$.

The problem is how party i can validate the shares $y^{(i)}(j)$ that she distributes to party j. Let $y^{(i)}(x) = y_0^{(i)} + y_1^{(i)} x + \cdots + y_{2k}^{(i)} x^{2k}$; then public commitment values for VSS checking are:

$$
\begin{aligned}
&\delta_0^{(i)} = g^{y_0^{(i)}} = g^{a_0 \cdot r_0^{(i)}} = \alpha_0^{r_0^{(i)}} \quad (\mathrm{mod}\ p),\\
&\delta_1^{(i)} = g^{y_1^{(i)}} = g^{a_0 \cdot r_1^{(i)} + a_1 \cdot r_0^{(i)} + z_1^{(i)}} = \alpha_0^{r_1^{(i)}} \alpha_1^{r_0^{(i)}} g^{z_1^{(i)}} \quad (\mathrm{mod}\ p),\\
&\vdots\\
&\delta_{2k}^{(i)} = g^{y_{2k}^{(i)}} = g^{a_k \cdot r_k^{(i)} + z_2 k^{(i)}} = \alpha_k^{r_k^{(i)}} g^{z_{2k}^{(i)}} \quad (\mathrm{mod}\ p),
\end{aligned}
$$

all of which can be locally computed by i and published. Thus party j can verify his share $y^{(i)}(j) = f(j) r^{(i)}(j) + z^{(i)}(j)$ by computing

$$
g^{(f(j) r^{(i)}(j) + z^{(i)}(j))} \stackrel{?}{=} \prod_{m=0}^{2k} \delta_m^{(i)^{j^m}} \quad (\mathrm{mod}\ p).
$$

All the verified parties' shares are then summed up to share the product $r \cdot a = \sum_{i \in \Lambda} r^{(i)} \cdot a$, where Λ denotes the set of qualified parties, and its public VSS commitment values are computed as follows:

$$
\delta_j = g^{\sum_{i \in \Lambda} y_j^{(i)}} = \prod_{i \in \Lambda} \delta_j^{(i)}, \ for\ 0 \leq j \leq 2k.
$$

Now, note that the result is a product of a and $r = \sum_{i \in \Lambda} r^{(i)}$, where $r^{(i)}$ is a random number generated by party i. But suppose that $r^{(i)}$ is the share of the polynomial $h(x)$ of order $2k$ whose free coefficient is another secret s. Then s is recovered by Lagrange interpolation: $s = \sum_{i \in \Lambda} \lambda_i r^{(i)}$. This means that we can apply the above scheme to the multiplication of two jointly shared (not random) values. The protocol is defined as follows:

3.4 Protocol for Robust Sequential Multiplication

Input of Party i: $f(i)$ of a polynomial $f(x)$ of order k representing a secret a, and $h(i)$ of a polynomial $h(x)$ of order $2k$ representing a secret s.

Public Input: $\alpha_i = g^{f_i}$ for $0 \leq i \leq k$, and $\delta_i = g^{h_i}$ for $0 \leq i \leq 2k$, where f_i and h_i denote the i-th coefficients of $f(x)$ and $h(x)$, respectively.

Protocol

1. Using VSS, party i shares the value of $h(i)$ by means of a random polynomial $h^{(i)}(x)$ of order k such that $h^{(i)}(0) = h(i)$. Note that $g^{h(i)}$ is already publicly computable from the public input δ.
2. Using VSS, party i shares 0 by means of a random polynomial $z^{(i)}(x)$ of order $2k$ such that $z^{(i)}(0) = 0$.
3. Party i broadcasts $\delta_j^{(i)} = g^{h_j^{(i)}}$ for $0 \leq j \leq 2k$, where $h_j^{(i)}$ denotes the j-th coefficient of a polynomial computed by $h^{(i)}(x) = f(x)h^{(i)}(x) + z^{(i)}(x)$.
4. Party j first checks the shares $h^{(i)}(j)$, $z^{(i)}(j)$ received from i, then checks the multiplication $f(j)h^{(i)}(j) + z^{(i)}(j)$ by confirming that $g^{(f(j)h^{(i)}(j)+z^{(i)}(j))} \overset{?}{=} \prod_{m=0}^{2k} \delta_m^{(i)^{j^m}} \pmod{p}$. If all the checks are validated, she announces i-correct; otherwise, she announces i-wrong.
5. Party i is included in a good set of parties Λ if more than $2k$ parties announced i-correct. Then, party j computes her share as $\sum_{i \in \Lambda} \lambda_i h^{(i)}(j)$, which results in sharing of a polynomial $h'(x) = \sum_{i \in \Lambda} \lambda_i h^{(i)}(x)$ of order $2k$ that carries the secret $h'(0) = a \cdot s$. Public values are also computed as follows:

$$\delta_j = g^{(\sum_{i \in \Lambda} \lambda_i h_j^{(i)})} = \prod_{i \in \Lambda} (\delta_j^{(i)})^{\lambda_i}, \; for\, 0 \leq j \leq 2k.$$

Lemma 3.4 The above protocol is t-resilient for a malicious adversary when the total number of parties is $n \geq 3t + 1$. After the protocol has been carried out, each party's new share is independent of its old share.

Sketch of Proof (Robustness) We can assume that there are $2k+1$ honest parties. Thus at least $2k+1$ parties are announced to be i-correct by $2k+1$ parties, and therefore the honest parties can compute the product.
(Privacy) We can construct a simulator that, given the input of the corrupted parties, simulates the process of local secret sharing. And the outputs have the same distribution as in the real protocol.
(Freshness) The proof is similar to that of Lemma 3.2.

4 $T\bigcup_{T_p\, EG}$ Based on PRSG and the ElGamal Encryption Scheme

Our goal is to implement function sharing of a pseudo-random sequence generator to obtain a periodical sequence of secrets, and to form a threshold cryptosystem using the secrets. We will use polynomial-based secret sharing over the ring Z_N, where N is the product of large primes. We assume that the number of parties n is smaller than any prime divisors of N, in order to apply the Lagrange interpolation formula. This assumption is not always satisfied in a linear congruence generator (LCG), and is thus an additional constraint.

4.1 $T \bigcup_{T_p} EG$ Protocol Based on LCG

Here we give an implementation of the periodical multi-secret threshold cryptosystem based on LCG-based key sequence generation (see appendix A.1) and the ElGamal encryption scheme.

It is well known that using an LCG as PRSG for the secret key is very dangerous, because it is possible to predict when some part of the sequence will become available. But in our application, the sequence is hidden in the exponent of the public key. Thus, because of the difficulty of calculating a discrete logarithm, the secret key is not predictable. Recently, the LCG used in the DSS has been attacked by solving the simultaneous modular linear equations obtained from two sets of DSS signatures [BGM97]. But such an attack cannot be used with our application.

Protocol for $T \bigcup_{T_p(LCG)} EG$-KeyUpdate

Input to Party i $y^{[0]}(i)$ of polynomial $y^{[0]}(x)$ of order $2k$ representing a seed $s^{[0]}$, $f(i)$ of polynomial $f(x)$ of order k representing a, and $h(i)$ of polynomial $h(x)$ or order $2k$ representing b. A dealer is necessary to select an appropriate a, b, and seed ($s^{[0]}$), because the period of LCG depends on N, a, b, and $s^{[0]}$.

Public Input A composite number N and an element $g \in Z_p^*$ of order N, where all divisors of N are larger than the number of the parties (n). $\alpha = g^{f_i}$ for $0 \leq i \leq k$, where f_i denotes the i-th coefficient of $f(x)$. $\beta_i = g^{h_i}$ and $\delta_i^{[0]} = g^{y_i^{[0]}}$ for $0 \leq i \leq 2k$, where h_i and $y_i^{[0]}$ denote the i-th coefficients of $h(x)$ and $y^{[0]}(x)$, respectively.

Protocol

1. First, parties perform the robust sequential multiplication protocol (section 3.4) to multiply the polynomials $y^{[m]}(x)$ and $f(x)$. Then they replace their share $y^{[m]}(i)$ by the result share of $s^{[m]} \cdot a$, which is represented by a polynomial of order $2k$.
2. Then parties add $h(i)$ to their intermediate results to obtain the shares of $a \cdot s^{[m]} + b$.
3. Finally, parties erase all the previous shares and intermediate results. The shares are carried by a polynomial $y^{[m+1]}(x)$ of order $2k$. which represents a secret of $s^{[m+1]} \overset{def}{=} s^{[m]} \cdot a + b \pmod{N}$.

Protocol for *EG*-Encrypt This protocol is common to all ElGamal-based encryption schemes: Since the public key $Y^{[m+1]} = g^{y_0^{[m+1]}} \bmod p$ is publicly known, the user can encrypt his message without informing the key-generating parties. The message M is encrypted, by using random $K \in Z_p$, as $(A, B) \overset{def}{=} (g^K, {Y^{[m+1]}}^K M) \pmod{p}$.

Protocol for $T\bigcup_{T_p(LCG)} EG$-Decrypt

Input for all parties A ciphertext (A, B).

Protocol

1. Each party P_i sends to the receiver who is requesting decryption of the message the partial decryption $A_i = A^{y^{[m]}(i)} \bmod p$ and proves to the receiver that $DLog_A A_i = DLog_g g^{y^{[m]}(i)}$ by using ZKIP [Cha90].
2. The receiver reconstructs the message by computing

$$M = \frac{B}{\prod_{i \in \Lambda} A_i^{\lambda_i}} \bmod p,$$

where Λ is a qualified set of parties of cardinality $2k+1$.

Theorem 4.1 The above protocols are t-resilient for a mobile malicious adversary when the number of parties is $n \geq 3t+1$.

Sketch of Proof We can assume that there are $2k+1$ honest parties.
(Robustness) The protocol for KeyUpdate consists of multiplication and addition of two shares. Lemma 3.4 is used for the VSS-supported multiplication, and the result is verifiably shared by $2k+1$ honest parties. The robustness of the decryption protocol is obvious: unless a bad party passes the ZKIP with an incorrect partial decryption, the message is correctly decrypted.
(Privacy) Given the input of the corrupted parties as an adversary's view, we can construct a simulator that simulates the key update process and outputs an arbitrary Y as a public key with the same distribution as in the real protocol.

4.2 $T\bigcup_{T_p} EG$ Protocol Based on BBS-PRSG

Here we give another implementation of the periodical multi-secret threshold cryptosystem based on BBS-based key sequence generation (see appendix A.1) and the ElGamal encryption scheme.

The BBS is a secure bit generator when used as the PRSG, but here we will use the whole value, which like LCGs, is also predictable if some part of the sequence is available. But again, in our application, the values are hidden in the exponent of the public key. Now the question is, when an adversary sees the sequence: $g^y, g^{y^2}, \ldots, g^{y^{2^i}}$, can he predict the next sequence? The decisional Diffie-Helman (DDH) assumption says that, given $< P, Q, g, g^a, g^b >$, there is no efficient algorithm that distinguishes g^c and $g^{a \cdot b}$. Thus, applying the DDH assumption where $a = b (= y)$, the adversary cannot have any information about the next sequence.

Protocol for $T\bigcup_{T_p(BBS)} EG$-KeyUpdate

Input to Party i $y^{[0]}(i)$ of polynomial $y^{[0]}(x)$ of order $2k$ representing a seed $s^{[0]}$. A dealer is necessary to select an appropriate seed ($s^{[0]}$), because the period of BBS depends on N and $s^{[0]}$.

Public Input A composite number N that is a product of two primes and an element $g \in Z_p^*$ of order N, where any divisors of N are larger than the number of the parties (n). $\delta_i^{[0]} = g^{y_i^{[0]}}$ for $0 \leq i \leq 2k$, where $y_i^{[0]}$ denotes the i-th coefficient of $y^{[0]}(x)$.

Protocol

1. The parties first perform the robust degree reduction protocol (section 3.2) to reduce the degree of $y^{[m]}(x)$ from $2k$ to k. The resulting polynomial is set to $y'^{[m]}(x)$.
2. The parties then perform the robust sequential multiplication protocol (section 3.4) for multiplying the polynomials $y'^{[m]}(x)$ and $y^{[m]}(x)$ in order to share ${s^{[m]}}^2 \bmod N$.
3. The parties finally erase all the previous shares and intermediate results. The shares are carried by a polynomial $y^{[m+1]}(x)$ of order $2k$. which represents a secret of $s^{[m+1]} \overset{def}{=} {s^{[m]}}^2 \pmod{N}$.

Protocol for $\boldsymbol{T \bigcup_{T_p(BBS)} EG}$-Decrypt The protocol is the same as $T \bigcup_{T_p(LCG)} EG$-Decrypt (section 4.1).

Theorem 4.2 The above protocols are t-resilient for a mobile malicious adversary when the number of parties is $n \geq 3t + 1$.
The proof is similar to that for Theorem 4.1.

5 Conclusions

In this paper we have defined the periodical multi-secret threshold cryptosystem and given an implementation that consists of (1) a t-resilient key sequence generation and (2) a t-resilient encryption/decryption scheme. For (1), we designed a protocol for sharing a pseudo-random sequence generation function based on Feldman's VSS. We also gave a robust protocol for sequential multiplication. For (2), we used the ElGamal encryption scheme, which makes it easy to collectively generate a public key while the private key is implicitly maintained.

There is an opinion that from the viewpoint of trust relationships, once the user relies on distributed servers to enforce the timing of a decryption, timing can be also operationally maintained by the servers. But we argue that there is a big difference between the operationally maintained timing and the timing kept by the key refreshment process, because we can apply security theory (resiliency and proactiveness) to the latter, while for the former the security is also maintained operationally.

By using different schemes for (1) and (2) and combining them, we can define various types of cryptographic system. For example, if we replace the encryption scheme by a signature scheme in (2), we can construct a time-restricted signature scheme, where the user has to visit more than k servers in a timeframe to get a signature. This might be used for contents metering.

We can also replace the PR-sequence generator with another sequence generator to control the period more easily. Actually, the BBS and linear congruential generators have a problem in that their period is too long, since it is dependent on the modulus, which should be large in order to maintain security. Thus we need to find an appropriate sequence generator whose period is relatively short and easy to control, yet whose sequence cannot be predicted by a malicious adversary.

Although we assume that the parameters and seed for the PRSG are fed by a dealer, we can eliminate the dealer by using Joint-Random-VSS to generate such values. But to realize this, we should consider the distributed checking mechanism for the period of the sequence.

References

[BGM97] Bellare, M., Goldwasser, S., and Micciancio, D., "Pseudo-Random Number Generation within Cryptographic Algorithms: the DSS Case," *Advances in Cryptology - CRYTO'97,* Lecture Notes in Computer Science 1294, Springer-Verlag, 1997.

[BBS86] Blum, L., Blum, M., and Shub, M., "A Simple Unpredictable Pseudo-random Number Generator," *SIAM Journal on Computing,* Vol. 15, No. 2, pp.364-383, 1986.

[BGW88] Ben-Or, M., Goldwasser, S., and Wigderson, A., "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation," *Proceedings of the 20th ACM Symposium on Theory of Computing,* pp.1-10, 1988.

[CCD88] Chaum, D., Crepeau, C. and Damgård, I., "Multiparty Unconditionally Secure Protocols," *Proceedings of 20th ACM Symposium on Theory of Computing,* pp.11-19, 1988.

[CH94] Canetti, R. and Herzberg, A., "Maintaining Security in the Presence of Transient Faults," *Advances in Cryptology - CRYPTO'94,* Lecture Notes in Computer Science 839, Springer-Verlag, pp.425-438, 1994.

[Cha90] Chaum, D., "Zero-Knowledge Undeniable Signature," *Advances in Cryptology - EUROCRYPTO'90,* Lecture Notes in Computer Science 473, Springer-Verlag, pp.458-464, 1991.

[CMI93] Cerecedo, M., Matsumoto, T., and Imai, H., "Efficient and Secure Multiparty Generation of Digital Signatures Based on Discrete Logarithms," *IEICE Transaction on Fundamentals,* E76-A(4):522-533, 1993.

[Des88] Desmedt, Y., "Society and Group Oriented Cryptography: A New Concept," *Advances in Cryptology - CRYPTO'87,* Lecture Notes in Computer Science 293, Springer-Verlag, pp.120-127, 1988.

[DF90] Desmedt, Y. and Frankel, Y., "Threshold Cryptosystems," *Advances in Cryptology - CRYPTO'89,* Lecture Notes in Computer Science 435, Springer-Verlag, pp.307-315, 1990.

[Fel87] Feldman, P, “A Practical Scheme for Non-interactive Verifiable Secret Sharing,” *Proceedings of the IEEE 28th Annual Symposium on Foundation of Computer Science,* pp.427-437, 1987.

[FGMY98] Frankel, Y., Gemmell, P., MacKenzie, P. D., and Yung, M., “Optimal-Resilience Proactive Public-Key Cryptosystems,” *Proceedings of the IEEE 38th Annual Symposium on Foudation of Computer Sciences,* 1997.

[FY92] Franklin, M., and Yung, M., “Communication Complexity of Secure Computation,” *Proceedings of the 24th ACM Symposium on Theory of Computing,* 1992.

[GRR98] Gennaro, R., O.Rabin, M., and Rabin, T., “Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography,” *Proceedings of the 17th ACM Symposium on Principles of Distributed Computing,* 1998.

[HJKY95] Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M., “Proactive Secret Sharing or: How to Copy With Perpetual Leakage,” *Advances in Cryptology - CRYPTO’95,* Lecture Notes in Computer Science 963, Springer-Verlag, pp.339-352, 1995.

[May93] May T. C., “Timed-Release Crypto,” Informal memo referred to by [RSW96].

[OY91] Ostrovsky, R. and Yung, M., “How to Withstand Mobile Virus Attacks,” *Proceedings of the 10th ACM Symposium on Principle of Distributed Computing,* 1991.

[Sha79] Shamir, A., “How to Share A Secret,” *Communications of the ACM 22,* 1979.

[Ped91a] Pedersen, T., “Distributed Provers with Applications to Undeniable Signatures,” *Advances in Cryptology - EUROCRYPT’91,* Lecture Notes in Computer Science 547, Springer-Verlag, 1991.

[Ped91b] Pedersen, T., “Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing,” *Advances in Cryptology - CRYPTO’91,* Lecture Notes in Computer Science 576, Springer-Verlag, pp.129-140, 1991.

[Ped91c] Pedersen, T., “A Threshold Cryptosystem without a Trusted Party,” *Advances in Cryptology - EUROCRYPT’91,* Lecture Notes in Computer Science 547, Springer-Verlag, pp.522-526, 1991.

[RSW96] Rivest, L.R., Shamir, A., and Wagner, D. A., “Time-Lock Puzzles and Time-Released Crypto,” MIT Technical Paper, 1996.

A Basic Tools

Here, we describe some of the existing tools that we use in our protocol.

A.1 Pseudo-random Sequence Generators (PRSG)

A random number sequence generator is defined by the form

$$X_n = f(X_{n-1}).$$

Here we will consider two well-known types of generators, linear congruential generators and BBS generators.

Linear congruential generators (LCG) A linear congruential generator is defined by the form

$$X_i = (aX_{i-1} + b) \bmod N,$$

where the parameters a, b, and N are constants. If they are properly chosen, the generator has an $(N-1)$ period, which we call the maximal period generator.

Blum-Blum-Shub (BBS) generators [BBS86] A BBS generator is defined by the form

$$X_i = X_{i-1}^2 \bmod N,$$

where modulus N is the product of two large prime numbers, p and q, that are congruent to $3 \bmod 4$ (N is a Blum integer). If we specify the additional properties that $p_1 = (p-1)/2$, $p_2 = (p-3)/4$, $q_1 = (q-1)/2$, and $q_2 = (q-3)/4$ are all primes, then the period is a divisor of $2p_2q_2$.

A.2 Secret Sharing by Polynomial Interpolation [Sha79]

Shamir's secret sharing realized a $(k,n)-$threshold scheme according to which any $k-1$ parties have no information about the secret, while k can recover the secret. Suppose the secret to be shared is $s \in Z_q$. The dealer generates a polynomial of order $k-1$ by randomly choosing its coefficients $a_1, \ldots, a_n$ and sets the secret to its free coefficient $a_0 = s$:

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{k-1}x^{k-1}.$$

The dealer then distributes the share $s_i = f(i) \bmod q$ to the party $P_i, (1 \leq i \leq n)$ via a secure channel. By using Lagrange interpolation, the set of shareholders Λ of cardinality k determines

$$f(x) = \sum_{i \in \Lambda} \lambda_i f(i), \quad where \quad \lambda_i = \prod_{j \in \Lambda, j \neq i} (x-j)(i-j)^{-1}$$

thus reconstructing the secret $s = a_0 = f(0)$.

A.3 Verifiable Secret Sharing by Feldman [Fel87]

The verifiable secret sharing (VSS) scheme enables the receivers of shares to check whether the dealer has distributed the correct shares. Feldman proposed an efficient non-interactive VSS scheme that uses a homomorphic encryption function: Assume that a secret space is defined over a prime field Z_q. When a dealer generates a polynomial of order $k-1$, he broadcasts the values $\alpha_i = g^{a_i} \bmod p$, for $0 \leq i \leq n$, where p is a prime such that $q|p-1$ and $g \in Z_p^*$ is an element of order q. The receiver i can check that his/her share s_i is really the value of $f(i)$ by calculating

$$g^{s_i} \overset{?}{=} \prod_{j=0}^{k-1} \alpha_j^{i^j} \bmod p$$

If the number of corrupt parties (t) is less than a half of the total (n), where $n \geq 2t+1$, then the incorrect shares are recoverable in a $(t+1, n)-$threshold VSS scheme.

A.4 Joint Random/Zero Secret Sharing [BGW88][Ped91a][Ped91b]

Sometimes it is convenient to generate and share a random value without a dealer. This can be done by means of the following protocol. A zero secret can also be shared in the same way. First, each party acts as a dealer of a random local secret or zero: the party i randomly generates a polynomial $f_i(x)$ of order $k-1$ whose free coefficient is set to a random value r_i or zero, and distributes the value $f_i(j)$ to party P_j. Then, each party sums up its shares in order to share a new function $g(x) = \sum_i f_i(x)$ whose free coefficient is $\sum_i r_i$ or zero. This protocol can be assisted by Feldman's VSS, which enables every party to verify that its share is correct. We will refer to these protocols as Joint-Random-VSS and Joint-Zero-VSS.