5 Proactive Security

Distributed cryptography (or threshold cryptography) protects a private key against exposure by sharing it among a group of parties. An adversary has to intrude on a fraction of the parties to obtain the key. In case of a long-lived key, there is still a risk that an adversary breaks into one party after another during the lifetime of the key. Even when such break-ins can be detected, the exposure of a key share to the adversary cannot be undone. Proactively secure systems perform system rejuvenation steps periodically, in anticipation of successful break-ins, and render leaked key shares harmless to eliminate the long-time exposure problem.

5.1 Model

A proactively secure cryptosystem is a threshold cryptosystem using a group of \( n \) parties, where the shares of the parties are periodically refreshed [HJJ+97].

Recall that in an \((t + 1, n)\)-threshold (public-key) cryptosystem, every party holds a share of the private key, which is generated using secret sharing with a polynomial of degree \( t \). In order to execute a cryptographic operation, at least \( t + 1 \) parties must collaborate, and up to \( t \) parties may be faulty. Moreover, executing the operation does not leak any information about one party’s share to another party or require the parties to pool their shares.

For high-value keys with a long lifetime, however, there is a risk that an attack spreads through the system and over time affects all parties, although not all of them simultaneously. Proactive cryptosystems protect such keys by periodically refreshing the shares held by the parties, such that a share exposed in a particular period is useless to an adversary in subsequent periods, after a refreshment of the shares. The renewed shares still correspond to the same long-term private key, so that the long-term public key can remain unchanged.

In this section we assume for simplicity that the parties are synchronized and have access to a common clock and to a synchronous broadcast channel (in contrast to the rest of the course, which uses an asynchronous system). Furthermore, the parties are connected by secure channels, i.e., they can send private and authenticated point-to-point messages.

Time is divided into periods, determined by the common clock (for example, one day). Each time period consists of two phases: (1) a short refresh phase, during which the parties carry out the refresh protocol so that they hold fresh shares afterwards; and (2) a long computation phase, where the parties execute operations of the cryptosystem.

Infected parties should be rebooted and re-initialized by a trusted agent (e.g., from a read-only device) after a corruption has been discovered. We assume the adversary cannot impersonate a disinfected party, send messages in its name, or observe messages addressed to it. Proactive cryptosystems tolerate up to \( t \) corrupted parties during every period, but all parties may be corrupted over multiple periods. A corrupted party that has been disinfected in one period will be correct in the subsequent period, and a corrupted party that has not been disinfected remains corrupted also in the subsequent period (in particular, it may participate in the refresh protocol). In order not to leak secrets from past periods, it must be possible for a party to erase information permanently.
5.2 Proactive Refresh Tolerating Passive Attacks

The proactive resharing protocol below offers privacy but no robustness. This means that the corrupted parties follow the protocol, but they try to obtain more information than they are entitled to and leak this information to an adversary. The same security notion was already used for the distributed ElGamal cryptosystem in the lecture notes.

Recall the setting of discrete-log based public-key cryptosystems. Let $G = \langle g \rangle$ be a group of prime order $q$, such that $g$ is a generator of $G$. Suppose the discrete logarithm problem (DLP) in $G$ is hard, i.e., for a random $y \in G$, any probabilistic polynomial-time algorithm computes $x \in \mathbb{Z}_q$ such that $y = g^x$ only with negligible probability.

Suppose the $n$ parties hold a polynomial sharing of the private key $x$ for a cryptosystem with public key $y = g^x$. That is, there exists a polynomial $f(X) \in \mathbb{F}_q[X]$ of degree $t$ chosen at random under the condition that $f(0) = x$. Denote the coefficients of $f(X)$ by $f_0, f_1, \ldots, f_t$ such that $f(X) = \sum_{k=0}^t f_k X^k$, where $f_0 = x$.

Algorithm 1 (Proactive refresh [HJKY95]). At the begin of the refresh phase, every party $P_i$ holds a share $x_i = f(i) = \sum_{k=0}^t f_k i^k$ from the previous period. The refresh protocol consists of two steps:

1. Every party $P_i$ chooses uniformly at random a polynomial $a^{(i)}(X) \in \mathbb{F}_q[X]$ of degree $t$ subject to $a^{(i)}(0) = 0$. It generates shares $r_{ij} = a^{(i)}(j)$ for $j = 1, \ldots, n$, and sends $r_{ij}$ to $P_j$ as a private point-to-point message. (Observe that $P_i$ essentially acts as the dealer to share the value 0 using polynomial secret sharing.)

2. After receiving $n$ shares $r_{ji}$, for $j = 1, \ldots, n$, party $P_i$ computes its new share in $\mathbb{Z}_q$ as

$$x'_i = x_i + \sum_{j=1}^n r_{ji}.$$  

Then it erases all variables except $x'_i$.

At the end of the refresh phase, $P_i$ uses $x'_i$ as its new share for the computation phase of the period.

Theorem 2. Provided $n > 2t$, Algorithm 1 ensures that:

1. When the input shares $x_1, \ldots, x_n$ are a sharing of $x$, then the output shares $x'_1, \ldots, x'_n$ are also a sharing of $x$.

2. An adversary that observes the secrets of at most $t$ parties in every period learns no information about the private key $x$.

Proof (sketch). To show the first condition (correctness), observe that every party $P_i$ basically shares the value 0 as the dealer in a polynomial secret sharing scheme using $a^{(i)}(X)$. Given the previous sharing polynomial $f(X)$, the addition of all shares produces a new sharing polynomial

$$f'(X) = f(X) + \sum_{j=1}^n a^{(j)}(X).$$

And since $a^{(j)}(0) = 0$ for all $j = 1, \ldots, n$, it holds $f'(0) = f(0)$. 

2
In other words, suppose that there is a group $S$ of $t+1$ parties that could recover the private key from the previous sharing as $x = f(0) = \sum_{i \in S} \lambda^S_{0,i} x_i$, with Lagrange coefficients $\lambda^S_{0,i}$ for $i \in S$. Then the recover operation from the new shares gives

$$\sum_{i \in S} \lambda^S_{0,i} x_i' = \sum_{i \in S} \lambda^S_{0,i} \left( x_i + \sum_{j=1}^n r_{ji} \right)$$

$$= \sum_{i \in S} \lambda^S_{0,i} x_i + \sum_{i \in S} \sum_{j=1}^n \lambda^S_{0,i} a^{(j)}(i) = x + \sum_{j=1}^n a^{(j)}(0) = x.$$

To show the second condition (secrecy), suppose the adversary corrupts $t_{\text{prev}}$ parties in the previous period but not in the current period (these parties have been disinfected), $t_{\text{both}}$ parties in the previous period and in the current period (they remain corrupted during the refresh protocol), and $t_{\text{curr}}$ parties only in the current period (they may be corrupted already during the refresh protocol). The assumption means that $t_{\text{prev}} + t_{\text{both}} \leq t$ and $t_{\text{both}} + t_{\text{curr}} \leq t$.

For every possible value of $x$, since the shares $r_{ij}$ are sent privately and the adversary never learns more than $t$ shares of any polynomial $a^{(i)}$ that is generated by a correct $P_i$, all information that it observes is consistent with $x$. Hence, it learns no information about $x$. \hfill \Box

### 5.3 More Robust Proactive Refresh

Algorithm 1 can be made robust so that it tolerates an active or Byzantine adversary. One problem with the above protocol is that a corrupted party in step 1 may send inconsistent share values $r_{ij}$ or simply “share” a value $\neq 0$, so that the parties no longer hold a correct sharing of the private key.

The following extension [HJKY95] takes up the method of Feldman’s verifiable secret sharing protocol [Fel87] to prevent this attack. The robust refresh protocol adds five steps to Algorithm 1 after step 1:

a) Suppose $P_i$ uses polynomial $a^{(i)}(X) = \sum_{k=1}^t a_{ik} X^k$ (recall that $a^{(i)}(0) = 0$); then it additionally computes $A_{ik} = g^{a_{ik}}$ for $k = 1, \ldots, t$ and broadcasts the vector $[A_{i1}, \ldots, A_{it}]$.

b) After receiving the share $r_{ji}$ and the vector $[A_{j1}, \ldots, A_{jt}]$ from $P_j$, party $P_i$ verifies that they are consistent by computing (in $\mathbb{Z}_q$)

$$g^{r_{ji}} \equiv \prod_{k=1}^t (A_{jk})^{ik}.$$

If this check fails, $P_i$ broadcasts a complaint against $P_j$.

c) For every complaint that was broadcast against $P_i$ by some $P_j$, party $P_i$ reveals the share $r_{ij}$ by broadcasting $r_{ij}$.

d) Next, for all parties against which a complaint was broadcast, party $P_i$ verifies that the revealed shares satisfy the above equation. If any revealed share is not valid, then $P_i$ disqualifies the sending party.

e) Finally, every party determines a set $Q$ of parties that have not been disqualified and changes step 2 of Algorithm 1 to computing $x'_i = x_i + \sum_{j \in Q} r_{ji}$. 

3
The protocol is robust as long as \( n > 2t \) because the consistency checks ensure that every party \( P_j \), whose shares \( r_{ji} \) do not satisfy the equation for \( t + 1 \) or more indices \( i \) of correct parties, receives a complaint. This follows from the polynomial “verification in the exponent”: Even if a faulty \( P_j \) broadcasts arbitrary values \( \tilde{A}_{jk} = g^{\tilde{a}_{jk}} \), if no correct party complains, then its values have passed \( t + 1 \) or more checks that ensure

\[
g^{r_{ji}} = \prod_{k=1}^{t} (\tilde{A}_{jk})^{i_k} = g^{\sum_{k=1}^{t} \tilde{a}_{jk} i_k}.
\]

Hence, there are values \( \tilde{a}_{jk} \) that define a polynomial of degree \( t \) and represent a sharing of 0 as required.

Due to the synchronous broadcast channel that carries all messages during the verification phase, all correct parties compute the same value for \( Q \) and update their share using the same polynomials.

**Remarks.** Gennaro et al. [GJKR07] point out that this protocol still has some deficiencies and present a truly robust protocol that eliminates them.

Proactive cryptosystems in *asynchronous* networks also build on the principles presented here [CKLS02, ZSvR05].

**References**


