Dynamic Threshold Cryptosystem without Group Manager

Andreas Noack and Stefan Spitz

Horst Görtz Institut für IT-Sicherheit
Ruhr-Universität Bochum

Abstract. In dynamic networks with flexible memberships, group signatures and distributed signatures are an important problem. Dynamic threshold cryptosystems are best suited to realize distributed signatures in dynamic (e.g. meshed) networks. Without a group manager or a trusted third party even more flexible scenarios can be realized.
Gennaro et al. [6] showed, it is possible to dynamically increase the size of the signer group, without altering the public key. We extend this idea by removing members from the group, also without changing the public key. This is an important feature for dynamic groups, since it is very common, e.g. in meshed networks that members leave a group.
Gennaro et al. used RSA and bi-variate polynomials for their scheme. In contrast, we developed a DL-based scheme that uses ideas from the field of proactive secret sharing (PSS). One advantage of our scheme is the possibility to use elliptic curve cryptography and thereby decrease the communication and computation complexity through a smaller security parameter.
Our proposal is an efficient threshold cryptosystem that is able to adapt the group size in both directions. Since it is not possible to realize a non-interactive scheme with the ability to remove members (while the public key stays unchanged), we realized an interactive scheme whose communication efficency is highly optimized to compete with non-interactive schemes. Our contribution also includes a security proof for our threshold scheme.

Keywords: Public-key cryptosystem, threshold cryptosystem, dynamic adding, dynamic removing, dynamic membership, secret sharing, proactive.

1 Introduction

Anonymous and autonomous group oriented cryptosystems have several basic requirements such as the anonymity of group members to outsiders and no need for a trusted third party. Additionally it is required to need at least a distinct number of cooperating group members to accomplish group operations, in which the secret key of the group is involved.

Furthermore, to call the system autonomous, it is mandatory to be able to add and remove members *without* a trusted third party. A subset of the group

decides on adding new or removing current members, whereby the public and secret key of the group remains untouched, even when the group size changes.

Applications for those cryptosystems are firstly dynamic networks like mobile ad-hoc networks or mesh networks, in which mobile nodes constantly leave and join. Threshold signatures, threshold decryption and secure routing are examples for the crypotgraphic usage.

Second, it can be used in applications distributed among a network, for example distributed intrusion detection systems (IDS). In distributed IDS, many network sensors communicate jointly with an automatic or manual warning system. Scenarios without a trusted third party become more and more important, since the network complexity inreases due to new types of networks (i.e. ad-hoc, mesh networks) and a centralized TTP would introduce a single point of failure. We considered threshold and identity based cryptography to find an appropriate solution for these scenarios.

1.1 Related Work

In 1984, Shamir [14] proposed the concept of id-based cryptography. In this concept, a user is identified by a publicly known parameter such as an e-mail address, IP address or name. This identifier can be used in conjunction with the public key of the group to encrypt data destined to or verify signatures originating from this user. A trusted third party who generates secret keys that correspond to the public key of the group *and* the ID of the user, is necessary. While Shamir provided the id-based digital signature scheme, Boneh and Franklin [2] as well as Cocks [3] independently introduced a scheme for id-based encryption in 2001 which are based on Weil pairings respectively the problem of quadratic residues.

The adding of members in id-based schemes is done by generating a secret key (Trusted Third Party) that fits to the public key of the group and the identifier of the new member. Removing a member can either be done by using certificate revocation lists or by assigning a period of validity to the distributed identifier as suggested by [2].

In 2003 Libert and Quisquater [10] introduced a threshold cryptosystem based on pairings with the ability to efficiently revocate users within the group. To accomplish this task they used the idea of a so-called *semi-trusted mediator* (mostly a synonym for TTP) proposed by Boneh, Ding, Tsudik and Wong [1] in 2001. The private keys are shared between the TTP and each group member. When a member wants to perform a decryption or signing operation, a message-specific token needs to be received from the mediator. Revocating a member is quite simple: The mediator does not issue these tokens to removed members any longer.

An alternative to id-based cryptography is the idea of using threshold schemes. In 1979, Shamir [13] presented the idea of "How to Share a Secret". In his scheme, the secret is divided and distributed among several entities. This secret can be reconstructed when some of the entities work together. Desmedt and Frankel [5] used this idea to design a threshold cryptosystem based on ElGamal in 1989. During 1991, Pedersen [11] presented another threshold scheme based

on Shamir's idea and the ElGamal cryptosystem. In this scheme, the distributed key generation needs no group manager or trusted third party. Each member of the group has an equal part in generating the groups public and secret key. An additional property of this scheme is the verifiability of all member shares (VSS - Verifiable Secret Sharing). The group's size was still fixed to the preassigned parameters, hence no possibility for adding and removing members was implemented. In 2005 Saxena, Tsudik and Yi [12] presented a scheme with the option of dynamically adding members to the group. Their scheme uses bi-variate polynomials, which allowed them to design a non-interactive threshold scheme. The key establishment can be performed by a set of founding members using the method of joint secret sharing [7] while adding members is possible through a subset of already established members of the group. However, the removing of group members is still not possible in their designed system.

Another approach on threshold schemes is based on RSA. In 1991 Desmedt and Frankel [4] initiated the study of using RSA for threshold schemes presenting a non-robust and non-interactive threshold scheme. Similar to the ElGamal-based scheme of [5], adding or removing a member is not possible in their scheme due to the fixed parameter choices. Gennaro et al. [8] introduced some ideas to provide robustness to RSA threshold schemes in 1996, with Shoup [15] presenting a robust RSA threshold signature scheme in 1999. Recently, during 2008 Gennaro et al. [6] developed a dynamic RSA threshold scheme. They utilize bi-variate polynoms for adding new members while being dependent on a trusted third party in the initial key distributing phase. In addition to the dynamic member adding, their scheme provides robustness and is non-interactive.

In 1995, Herzberg et al. [9] developed a method to increase the security of a (k, n) threshold scheme by decreasing the time window during which an adversary must compromise $\geq k+1$ shares of the secret. In this proactive secret sharing scheme (PSS) the shares of all members are periodically renewed so that information gained in one time period is useless after the share renewal. Their scheme is based on a constant number of members with no option to add or remove.

1.2 Our Contribution

The threshold scheme [10] uses id-based cryptography and provides the possibility to add and remove members dynamically, while being dependent on a TTP. Scheme [12] is based on ElGamal and allows the adding of members without the help of a TTP. In the RSA-based scheme [6], the adding of members requires a TTP without providing an option of removing members. Table 1 shows the comparison of these schemes (and their predecessors) with our scheme.

To the best of our knowledge, there is currently no threshold cryptosystem (neither id-based, ElGamal-based nor RSA-based) which allows the removing of members without the use of a TTP. Certificate revocation lists (i.e., for id-based schemes) could be used to replace a TTP, but this possibilty will not be considered due to the unbounded size of those lists and their impact to the anonymity property.

Table 1. Comparison of referenced schemes

Scheme	Establish Group	Add Member	Remove Member
Desmedt, Frankel [4] (RSA)	○	–	–
Gennaro et al. [6] (RSA)	○	•	–
Boneh, Franklin [2] (ID)	○	○	(○)
Libert, Quisquater [10] (ID)	○	○	○
Pedersen [11] (ElG.)	•	–	–
Saxena et al. [12] (ElG.)	•	•	–
This Scheme (ElG.)	•	•	•

ELG = ElGamal-based, RSA = RSA-based, ID = id-based
• without TTP, ○ with TTP, (○) with CRL/Timestamp, – not possible

Our contribution is an ElGamal-based anonymous and autonomous threshold scheme. It is based on techniques derived from the share renewal method in PSS, which we adapted to realize a method for dynamic adding and removing members without a TTP, while the public key remains unchanged. Predefining a maximum number of group members is no longer necessary either.
The removing feature for groups without a trusted third party is very important, when applying those schemes in today's dynamic decentralized networks. Due to the nature of these networks, it is common that members leave the group, e.g. caused by a topology change (mesh-network) or a connection loss. Schemes that do not support removing of members from the group are hence not suitable for dynamic networks.

Moreover, it is possible to use ECC instead of ElGamal for our scheme. This will most likely reduce the communication and computation complexity in comparison to RSA-based schemes because of the smaller security parameter (160 bit vs. 1024 bit).

2 Threshold Scheme without Group Manager

We have searched for a group oriented cryptosystem that has the following properties: it is autonomous, anonymous and provides security against inside and outside adversaries.

ID-based systems do not accomplish the autonomous and anonymous property. A group manager is necessary in the known schemes based on pairings and no anonymity is present due to the fact that each party has an own public key. Group signature schemes need a group manager for the possibility to identify signers (signer ambiguous) and for adding or removing members. We came to the conclusion to deploy threshold schemes, since a group manager is not mandatory here and anonymity can be provided as well. Additionally there is a threshold for the minimal number of cooperating parties, who can recover the secret key, which can be valuable for practical demands. A (k, n)-threshold scheme has n members from which at least $k + 1$ members are necessary to recover the group secret. This scheme is extended to a public key cryptosystem with a shared secret key SK and a single public key PK.

There are two possibilities to generate the public and secret key pair and distribute SK: Either using a TTP or a distributed key generation. In the first case, the TTP generates the (PK, SK) pair, splitting the secret key into n shares and distributes them securely to each member. This procedure represents the idea of Shamir's secret sharing [13]. Our introduced scheme uses the second possibility where all members cooperate in the key generation.

2.1 Distributed Key Generation

In this section we show how a group $U = \{P_1, \ldots, P_n\}$ can cooperate to establish a public key $PK = (p, q, g, A)$ and corresponding secret key $SK = (a_0)$, such that at least $k + 1$ members are needed to utilize SK. The distributed key generation is divided into two operations, the key generation and distribution. Prior to key generation, all members P_i need to agree on the public parameters p, q, g such that:

$$p, q \in \mathbb{P} : p = 2q + 1$$
$$g \in \mathbb{Z}_p^* : ord(g) = q$$

After agreeing on the public parameters and the desired threshold value k, each member P_i of group U generates a key pair for the El-Gamal encryption scheme as follows:

$$\begin{aligned} &u_{i,0} \in_R \mathbb{Z}_q^* \\ &A_i := g^{u_{i,0}} \mod p \end{aligned} \tag{1}$$

with A_i being a part of the public key parameter A. The value $u_{i,0}$ is called *member secret* of P_i. It represents a part of the group's secret key a_0 and is distributed among all members of U. This is done by creating a k-resilient (k, n) secret sharing scheme with the idea of Shamir[13].

$$\begin{aligned} &\{s_1, \ldots, s_k\} \in_R \mathbb{Z}_q^* \\ &f_i(x) = s_k x^k + s_{k-1} x^{k-1} + \cdots + s_1 x + u_{i,0} \mod q \\ &u_{i,j} := f_i(j),\ \forall j : P_j \in U \\ &\text{Send } \langle A_i, u_{i,j} \rangle \text{ to } P_j \text{ (confidentially)}, \forall P_j \in U \end{aligned} \tag{2}$$

Each member P_i chooses a random polynomial of $deg(f_i) = k$ and computes n *member shares* $u_{i,j} = f_i(j)$ of its own secret polynomial. Then the member shares, as well as the values A_i, are sent confidentially to each P_j. After the distribution each member computes its so-called *secret share* and public key parameter A.

$$\begin{aligned} A &= \prod_{i=1}^{n} A_i \mod p \\ a_i &= \sum_{j=1}^{n} u_{j,i} \equiv \sum_{j=1}^{n} f_j(i) \mod q \end{aligned} \tag{3}$$

Each P_i is now in possession of $PK = (p, q, g, A)$ and a secret share a_i of $SK = (a_0)$. The secret share a_i is the function evaluation of $f(x) = \sum_{j=1}^{n} f_j(x)$ mod p at position i (homomorphism property).

2.2 Adding Members

The adding of a specific user P_{n+1} to a pre-established group U with n members is explained in detail within this section. The following conditions must hold when a user is added to the group

- The public key $PK = (p, q, g, A)$ and the secret key $SK = (a_0)$ remain unchanged.
- The secret share a_{n+1} and polynomial f_{n+1} is only known to P_{n+1}.
- The new a_{n+1} of P_{n+1} is a valid and fully functional secret share of the group U.

Both, adding and removing a member is based on the share renewal technique used in PSS. PSS updates already distributed shares of all n members to provide proactive security. This update is done in three simple steps with the details given in [9]: First, each P_i picks a new random polynom f_i' with $f_i'(0) = 0$. Then, P_i sends $u_{i,j} = f_i'(j)$ to P_j, $\forall i, j$. In the final step, each P_i computes his updated share $a_{i,new} = a_{i,old} + \sum_j u_{j,i}$, $\forall j, i$.

We adapted this technique to implement an efficient adding and removing of members. While adding, $k + 1$ members split off a part of their secret and share this part with the new user. Removing a member is done by computing and redistributing the member's secret to some remaining members, as shown in 2.3.

The adding of a user is done in three phases. Phase 1 covers necessary pre-computations of group U_h. In phase 2 the new user P_{n+1} computes his secret share a_{n+1} and member secret $u_{n+1,0}$. Then, member shares $u_{n+1,i}$ for each P_i in U are generated by P_{n+1}. After the distribution of these shares, the remaining group members insert the shares from P_{n+1} into their local databases during phase 3. The details of each phase are described below.

Phase 1a - Secret Sharing: During phase 1a, $k + 1$ members of U form a so called *helper group* U_h. These members cooperate in the generation of a new member secret $u_{n+1,0}$ for P_{n+1}. For this, all $P_i \in U_h$ cede a part of their member secret $u_{i,0}$ and submit it secretly to P_{n+1}. A new polynomial is generated and the differences between the values generated from the new and the old polynomial are also distributed secretly. The details of this process executed by all $P_i \in U_h$

are shown below:

$$
\begin{aligned}
&share_i \in_R \boldsymbol{Z}_q^* \\
&u_{i,0} = u_{i,0} - share_i \\
&s_t \in_R \boldsymbol{Z}_q^*,\ \forall t = \{1, \ldots, k\} \\
&f_i'(x) := s_k x^k + s_{k-1} x^{k-1} + \cdots + s_1 x + u_{i,0} \mod q \\
&\delta_i[j] := f_i'(j) - f_i(j) \mod q,\ \forall j : P_j \in U \\
&f_i(x) = f_i'(x)
\end{aligned} \tag{4}
$$

Each member $P_i \in U_h$ is now in possession of a new $f_i(x)$ and a new member secret $u_{i,0}$. All P_i then send (confidentially) $\langle \text{share}_i,\ \delta_i,\ u_{i,n+1} \rangle$ to P_{n+1}.

Phase 1b - Share Computation: In this phase, the member shares $u_{i,n+1}$ ($\forall i \in U$) for the new member P_{n+1} are computed. During the distributed key generation all n members were needed to compute their member shares, but with the help of the Lagrangian interpolation only $k+1$ members need to cooperate to generate these n shares. The Lagrangian interpolation is defined as:

$$
L(\alpha, \beta) \stackrel{def}{=} \prod_{\gamma \in U, \gamma \neq \beta} \frac{\alpha - \gamma}{\beta - \gamma}
$$

The communication progress is similar to a chain reaction. Without loss of generality, we start with member P_1, incrementing step-wise until P_{k+1} is reached. The necessary $k+1$ members are allowed to be in U_h. Phase $1a$ and phase $1b$ can be run simultaneous, since the operations do not depend on each other. P_1 computes:

$$
\begin{aligned}
&r \in_R \boldsymbol{Z}_q^* \\
&\omega_j := u_{j,1} * L(n+1, j) + r \mod q,\ \forall j : P_j \in U \setminus U_h \\
&\Omega = \{\omega_1 | \omega_2 | \ldots | \omega_j\}
\end{aligned} \tag{5}
$$

The array Ω contains a subset of all computed member shares of P_i, $\forall i \in U \setminus U_h$ for user P_{n+1} which is sent (confidentially) to the next member within the chain. It is necessary that the second user in the chain is a member of the group U_h, so that P_2 is not able to recover the used r. In addition, r is sent (confidentially) to P_{n+1}.

Now, members P_i, $\forall i \in \{2, \ldots, k+1\}$ proceed with:

$$
\begin{aligned}
&\omega_j = \omega_j + (u_{j,i} * L(n+1, j)) \mod q,\ \forall j : P_j \in U \setminus U_h \\
&\text{Send } \Omega \text{ to } P_{i+1} \text{ (confidentially)}
\end{aligned} \tag{6}
$$

The last member P_{k+1} sends Ω confidentially to P_{n+1}.

Phase 2 - Reconstructing Shares: At the start of phase 2 the (yet to be added) new user P_{n+1} receives the messages from phase $1a$ and phase $1b$, hence

$\langle \text{share}_i,\ \delta_i,\ u_{i,n+1} \rangle$, $\forall i \in U_h$ and $\langle \Omega \rangle$.

When P_{n+1} has received all $h+1$ messages, the following is done:

$$
\begin{aligned}
& u_{i,n+1} = \omega_i - r,\ \forall i : P_i \in U \setminus U_h \\
& u_{n+1,0} = \sum_{i:P_i \in U_h} share_i \mod q \\
& s_t \in_R \boldsymbol{Z}_q^*,\ \forall t = \{1, \ldots, k\} \\
& f_{n+1}(x) := s_k x^k + s_{k-1} x^{k-1} + \cdots + s_1 x + u_{n+1,0} \mod q \\
& a_{n+1} := \sum_{i:P_i \in U \cup P_{n+1}} u_{i,n+1} \mod q \\
& \Delta = \{\delta_1 | \delta_2 | \ldots | \delta_i\},\ \forall i : P_i \in U_h \\
& \Lambda = \{u_{n+1,1} | u_{n+1,2} | \ldots | u_{n+1,n}\}, \text{with } f_{n+1}(i) = u_{n+1,i}
\end{aligned} \tag{7}
$$

Then P_{n+1} sends $\langle \Delta\ ,\ \Lambda \rangle$ via broadcast (confidentially).

Phase 3 - Final Computation: During phase 3 each member P_i receives the broadcast from P_{n+1} and computes:

$$
\begin{aligned}
& u_{j,i} = u_{j,i} + \delta_j[i] \mod q,\ \forall j : P_j \in U_h \\
& u_{n+1,i} = \Lambda[i] \\
& a_i = \sum_{j=1}^{n+1} u_{j,i} \mod q
\end{aligned} \tag{8}
$$

All members now possess the member share of P_{n+1} and the updated member shares from the members of U_h. Then, they compute their new secret share a_i adding the user P_{n+1} to the group.

2.3 Removing Members

Removing a member P_r is executed in three steps. First, the member secret $u_{r,0}$ is recovered using the Lagrangian interpolation by at least $k+1$ members. This secret is then shared between some members. From there on, the removing of a member is similar to the adding, with the details of each of the three phases shown below.

Phase 1 - Secret Recovery/Share Computation: The user P_r shall be removed from the group U. For this, $k+1$ members need to agree on the removing, forming the group U_{k+1}, with one of the members designated as P_z. Each of the members P_i, $\forall i \in \{1, \ldots, k+1\}$ (w.l.o.g.) computes:

$$
\omega_i := L(0, i) * u_{r,i} \mod q \tag{9}
$$

With this, it is possible to reconstruct the secret of P_r by computing $u_{r,0} = \sum_i^{k+1} \omega_i$ later on. A helper group U_h' with at least 2 members is formed. It is

mandatory, that member P_z is in U'_h. Ideally, the remaining $h-1$ members are from U_{k+1} as well, which will reduce the number of communications necessary for this phase. Now, $h-1$ users P_i from U'_h will do the following operations and then send the results to the remaining user P_z of this group:

$$\begin{aligned}
&share_i \in_R Z_p^* \\
&u_{i,0} = u_{i,0} + share_i \mod q \\
&s_t \in_R \boldsymbol{Z}_q^*,\ \forall t = \{1, \ldots, k\} \\
&f'_i(x) := s_k x^k + s_{k-1} x^{k-1} + \cdots + s_1 x + u_{i,0} \mod q \\
&\delta_i(x) := f'_i(x) - f_i(x) \mod q \\
&f_i(x) = f'_i(x)
\end{aligned} \tag{10}$$

In comparison with the corresponding adding phase $1a$, the computations differ in the computation of the differences (δ_i), because δ_i is now a polynomial. Therefore the data volume is much lower during removing phase, since the polynomial has only $k+1$ values (coefficents). The difference values during the adding phase consist of n values. Now, $\langle \omega_j, \text{share}_i, \delta_i \rangle$, $\forall j \in \{1, \ldots, k+1\}$ and $\forall i \in U'_h$ is sent (confidentially) to P_z.

Phase 2 - Difference Transfer: During phase 2 all necessary computations for the removing of member P_r are executed. First, member P_z reconstructs the member secret of P_r by computing

$$u_{r,0} = \sum_{i=1}^{k+1} \omega_i \mod q \tag{11}$$

Without loss of generality: $i = 1, \ldots, k+1$. Then, P_z computes δ_z similar to the other users of the helper group U'_h.

$$\begin{aligned}
&share_z = u_{r,0} - \sum_{i=1}^{h} share_i \mod q \\
&u_{z,0} = u_{z,0} + share_z \mod q \\
&s_t \in_R \boldsymbol{Z}_q^*,\ \forall t = \{1, \ldots, k\} \\
&f'_z(x) := s_k x^k + s_{k-1} x^{k-1} + \cdots + s_1 x + u_{z,0} \mod q \\
&\delta_z(x) := f'_z(x) - f_z(x) \mod q \\
&f_z(x) = f'_z(x) \\
&\Delta = \{\delta_1(x) | \delta_2(x) | \ldots | \delta_i(x)\},\ \forall i : P_i \in U'_h
\end{aligned} \tag{12}$$

At the end of this phase, P_z broadcasts $\langle \Delta \rangle$ to all members of U.

Phase 3: Each member P_i receives the broadcasts and computes:

$$\begin{aligned}
&u_{j,i} = u_{j,i} + \delta_j(i) \mod q,\ \forall j : P_j \in U'_h \\
&a_i = \sum_{j : P_j \in U \setminus P_r} u_{j,i} \mod q
\end{aligned} \tag{13}$$

Similar to the adding, each member (except P_r) is now in possession of $h+1$ new member shares and the updated secret shares of the polynomial

$$f(x) = \sum_{i:P_i \in U \setminus P_r} f_i(x) \mod p$$

The secret share a_r is no longer valid, hence P_r is not any longer able to cooperate with members $P_i \in U$ to reconstruct the secret a_0 using the Lagrangian interpolation. Finally each P_i deletes P_r's entry $u_{r,i}$ from its local database removing the last trace of the membership.

3 Security Proof

3.1 Distributed Key Generation

Our scheme is based on Shamir's Secret Sharing Scheme. Hence, we start with a security proof of this scheme to lay the ground for our improvements.

Each party P_i generates such a scheme with a random polynomial $f_i(x)$ and distributes shares $u_{i,j} = f_i(j)$, $\forall j \in U$ confidentially to all other parties. With the homomorphism property, the polynomial $f(x) = \sum_{j=1}^{n} f_j(x) \mod q$ has the shares $a_i = \sum_{j=1}^{n} u_{j,i} \equiv \sum_{j=1}^{n} f_j(i) \mod q$.

Each user P_i knows the own share a_i, but has no knowledge of $f(x)$. If Shamir's Secret Sharing Scheme is secure and all shares are sent confidentially, our scheme is secure against adversaries that own up to k shares.

Security Proof for Shamir's Secret Sharing Scheme: The scheme is secure, when it is not possible to recover the polynomial $f(x)$ (or at least the last coefficent from $f(x)$, the secret group key a_0) with less than $k+1$ shares a_i $(i > 0)$. The degree of $f(x)$ is k. We will show, that an attacker owning k shares $< i, a_i >$ $(i > 0)$ is not successful in recovering the secret a_0.

Theorem: For $k+1$ shares $< i, a_i >$ $(i > 0)$ with pairwise different i, there is exactly one polynomial $f(x)$ of degree $\leq k$ with $f(i) = a_i$, $\forall i \in \{1, \ldots, k+1\}$ (w.l.o.g).

Proof: A proof by contradiction states that there are two different polynomials $g(x)$ and $h(x)$ with deg(g) $\leq k$ and deg(h) $\leq k$, that can be created with $k+1$ shares $< i, a_i >$ $(i > 0)$. The difference polynomial $\Delta(x) = g(x) - h(x)$ must therefore have at least $k+1$ zero points. $\Delta(i) = 0$, $\forall i \in \{1, \ldots, k+1\}$ (w.l.o.g). Hence $\Delta(x)$ has a degree of maximal k, since $g(x)$ and $h(x)$ are of degree $\leq k$, $\Delta(x)$ can have a maximum of k zero points. Because $\Delta(x)$ has at least $k+1$ zero points (but may have k zero points, degree $\leq k$), $\Delta(x)$ must be the null polynomial. That means, that $g(x) = h(x)$.

An attacker who owns only k shares $< i, a_i >$ from $f(x)$ can interpolate a polynomial $p(x)$ with a maximum degree of $k-1$ (Lagrangian interpolation).

This polynom has the following property:

$$f(i) = p(i),\ \forall i \in \{1, \ldots, k+1\} \text{ (w.l.o.g)}$$

But additionally it must hold with a probability of $\Pr \approx 1 - \frac{1}{2^l}$, that:

$$f(i) \neq p(i),\ \forall i \notin \{1, \ldots, k+1\} \text{ (w.l.o.g)}$$

This is because $f(x) \neq p(x)$, which follows from the fact that $f(x)$ has degree k and $p(x)$ can have a maximal degree of $k-1$. If the coefficients from $f(x)$ over $\boldsymbol{Z}_p^*$ ($p \in \boldsymbol{P}$) are uniformly distributed, the probability for $f(i) = p(i)$, $i \notin \{1, \ldots, k+1\}$ is equal to the probability that a distinct value v with $f(i) = v$ is hit. The probability is $\Pr[f(i) = v, \forall i \in \boldsymbol{Z}_p^*] \approx \frac{1}{2^l}$.

Shamir's Secret Sharing Scheme is secure against attackers that know only k shares, since $f(x)$ cannot be recovered with less than $k+1$ shares. Thus our distributed key generation scheme is also secure against attackers, knowing only k shares, because it is not possible to recover $f(x)$ with less than $k+1$ shares $a_i = \sum_{j=1}^{n} u_{j,i}$, when all shares were transmitted confidentially.

3.2 Adding Members

Phase 1a - Secret Sharing: With the knowledge of $(h-1 = k)$ share$_i$ values, an adversary gains no advantage, because the adversary can not recover $u_{i,0}$, nor the resulting $u_{n+1,0} = \sum \text{share}_i \mod q$. In the case of the new $u_{i,0}$ values, the former state is unknown (only the difference share$_i$ is known) and in the case of $u_{n+1,0}$ one fully random value share$_i$ is missing.

The differences δ_i give no knowledge about the former state of the secret shares $u_{i,j}$ to which they will be applied. Hence they may be even transmitted in plaintext, although this is not done here (explanation in security issues).

Phase 1b - Share Computation: The first user adds a random value r to the secret products $u_{j,1} * L(n+1, j)$. A subsequent user (w.l.o.g.) P_2 is prevented from recovering the $u_{j,1}$ values due to the addition of the random value r, which is transmitted confidentially to P_{n+1}. When P_2 knows any of the $u_{j,1}$ values, she can recover the value r and with r all other $u_{j,1}$ values. Since the ω_j are created $\forall j \in U \setminus U_h$ and user P_2 is member of U_h (by definition), she does not get knowledge of any value from $u_{j,1}$ ($j \in U \setminus U_h$).

Additionally to this method, the messages are transmitted confidentially (unicast encryption) to prevent a former user to eavesdrop the changes of the following user and hereby recovering her secrets.

Phase 2 - Reconstructing Shares: To proof the security of the remaining scheme, it is necessary to show that P_{n+1} can not gain any other information from $\omega_i - r$ than $u_{i,n+1}$. ω_i is a sum of $k+1$ products of a secret ($u_{j,i}$) and a known value $L(n+1, j)$. It is not possible to disassemble this sum into the seperate addends, since all possibilities have the same probability. The secret values $u_{n+1,i}$ will be sent encrypted to each user P_i, so it is clear that no one, who is not allowed to, gets the knowledge of these values.

Phase 3 - Final Computation: If some of these values are faked or compromised, the scheme will not work. In this case, all changes since the last working state (probably since the last member adding) can be undone.

3.3 Removing Members

The proof for the security of the removing is quite similar to the adding of members. Equal techniques are applied, so that we will only give proofs, where differences between both schemes exist.

Phase 1 - Secret Recovery/Share Computation: $k+1$ members work together to recover the secret $u_{r,0}$ of user P_r, who should be removed from the group. This is done by computing $\omega_i := L(0,i) * u_{r,i} \mod q$, which can be sent in cleartext, since there is no need to keep the shares $u_{r,*}$ of user P_r secret.

Furthermore it is possible to send difference polynomials $\delta_i(x)$ instead of the difference values for each user in U. Each user can recover $u_{r,0}$, but this will not impact the scheme negatively, hence there is no need so save the privacy of $u_{r,0}$ (or $f_r(x)$).

Phase 2 - Difference Transfer: There may be the need to keep P_z away from altering the $\delta_i(x)$ in such a way, that the sum stays equal but the individual values differ. In Section Appendix B, there is a propsal for an option to make changes of these values evident.

4 Conclusion

This is the first time, a threshold cryptosystem with the ability to remove members without changing the public key is proposed. Furthermore, our scheme does not depend on a trusted third party, neither for the group establishment nor the adding or removing of members.

For adding and removing members, we extended the ideas from Herzberg et al. [9] that were used to refresh member shares. Our cryptographic contribution is an threshold cryptosystem that exceeds trivial extensions (for adding and removing members) of Herzberg's scheme in efficency aspects.

Due to its dynamic, our scheme can be deployed in emerging technologies such as mesh or ad-hoc networks, where members join and leave frequently. Without a TTP, our scheme fits to all kinds of decentralized networks where the location is not fixed to a specific area.

In addition to a security proof of our scheme, we made a proposal for using the scheme for encryption and shared decryption in dynamic groups (Appendix A). Besides that, the common application of our scheme will clearly be the threshold signature (based on DL - in contrast to [6]).

When using a threshold signature with our scheme, the security property non-repudiation is provided for the group. All group members have the same functionality and equal knowledge, so that there is in fact no distinguished

member. No single entity (e.g. TTP) has knowledge of the groups secret key. The group cannot deny that at least $k+1$ members worked together to compute the signature. Therefore an outside entity can be sure that a valid signature has been created by the group.

For our scheme, an ECC security parameter can be used, since there are no known attacks on Shamir's Secret Sharing Scheme that would lead to higher security parameters.

The reduction of the complexity of dynamic threshold schemes and providing an efficient non-iteractive DL-based threshold signature scheme optimized for our scheme is an open challenge.

References

1. Dan Boneh, Xuhua Ding, Gene Tsudik, and Chi Ming Wong. A method for fast revocation of public key certificates and security capabilities. In *SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium*, pages 22–22, Berkeley, CA, USA, 2001. USENIX Association.
2. Dan Boneh and Matt Franklin. Identity-based encryption from the Weil pairing. *Lecture Notes in Computer Science*, 2139:213–??, 2001.
3. Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *Proceedings of the 8th IMA International Conference on Cryptography and Coding*, pages 360–363, London, UK, 2001. Springer-Verlag.
4. Yvo Desmedt and Yair Frankel. Shared generation of authenticators and signatures (extended abstract). In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 457–469, London, UK, 1992. Springer-Verlag.
5. Yvo G. Desmedt and Yair Frankel. Threshold cryptosystems. In *CRYPTO '89: Proceedings on Advances in cryptology*, pages 307–315, New York, NY, USA, 1989. Springer-Verlag New York, Inc.
6. Rosario Gennaro, Shai Halevi, Hugo Krawczyk, and Tal Rabin. Threshold rsa for dynamic and ad-hoc groups. Cryptology ePrint Archive, Report 2008/045, 2008. `http://eprint.iacr.org/`.
7. Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Lecture Notes in Computer Science*, 1592:295+, 1999.
8. Rosario Gennaro, Tal Rabin, Stanislav Jarecki, and Hugo Krawczyk. Robust and efficient sharing of RSA functions. *Journal of Cryptology: the Journal of the International Association for Cryptologic Research*, 13(2):273–300, 2000.
9. Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. *Lecture Notes in Computer Science*, 963:339–352, 1995.
10. Benoît Libert and Jean-Jacques Quisquater. Efficient revocation and threshold pairing based cryptosystems. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 163–171, New York, NY, USA, 2003. ACM.
11. Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In *EUROCRYPT*, pages 522–526, 1991.
12. Nitesh Saxena, Gene Tsudik, and Jeong Hyun Yi. Efficient node admission for short-lived mobile ad hoc networks. In *ICNP '05: Proceedings of the 13TH IEEE International Conference on Network Protocols*, pages 269–278, Washington, DC, USA, 2005. IEEE Computer Society.
13. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
14. Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.
15. Victor Shoup. Practical threshold signatures. *Lecture Notes in Computer Science*, 1807:207–??, 2000.

Appendix A: Encryption and Shared Decryption

Encryption: A user outside the group knows the public key (p, g, A) of the group U.

$$\begin{aligned}
&\text{choose a plaintext: } m \\
&b \in_R \{2, \ldots, p-2\} \\
&B := g^b \mod p \\
&\text{ciphertext: } c := m * A^b \mod p
\end{aligned}$$

The user sends the ciphertext $< c, B >$ to the group U.

Decryption - Phase 1: We assume that P_i begins the decryption of $< c, B >$. Since $k+1$ users are needed, to decrypt the ciphertext, P_i asks for the help of k chosen users P_j with $i \neq j$.
To prevent an impersonation attack, P_i has to proof that she knows a_i, her secret part of the group key. This can be accomplished by a Zero-Knowledge Proof ($p' \in \mathbb{P}$ with $p' > p$, $< g' >= \mathbb{Z}^*_{p'}$):

$$\begin{aligned}
&t \in_R \mathbb{Z}^*_{p'} \\
&T := g'^t \mod p' \\
&S := t - \text{Hash}(A_i, g', p', T, < c, B >, U_{help})\ a_i \mod p' - 1
\end{aligned}$$

P_i sends $(< T, S >, < c, B >, U_{help})$ to all users P_j, she wants to be helped from. The subgroup of users in U that decrypt cooperatively is called U_{help}, whereby U_{help} also includes P_i.

Decryption - Phase 2: All users P_j in U_{help} verify the ZK-Proof from P_i by checking:

$$T \stackrel{?}{=} g^S A_i^{\text{Hash}(A_i, g', p', T, < c, B >, U_{help})} \mod p'$$

Then each user P_j in U_{help} computes the partial decryption and encrypts it with the personal key A_i of P_i:

$$\begin{aligned}
&m_j := B^{L(0,j)a_j} \mod p \\
&r \in_R \mathbb{Z}^*_{p'} \\
&R_j := g'^r \mod p' \\
&d_j := A_i^r m_j \mod p'
\end{aligned}$$

where $L(0, j)$ is the Lagrangian interpolation with:

$$L(\alpha, \beta) \stackrel{def}{=} \frac{\prod_{\gamma \in U_{help}, \gamma \neq \beta} (\alpha - \gamma)}{\prod_{\gamma \in U_{help}, \gamma \neq \beta} (\beta - \gamma)} \mod q$$

All users in U_{help} send their encrypted shares $< d_j, R_j >$ to P_i.

Decryption - Phase 3: P_i decrypts all shares $< d_j, R_j >$ with:

$$m_j = R_j^{p'-1-a_i} d_j \mod p'$$

and then computes the plaintext m from $< c, B >$ with:

$$m = c * \frac{1}{\prod_{\forall j \in U_{help}} m_j} \mod p$$

It is important to note, that $|U_{help}|$ is $k+1$.

Appendix B: Security Issues

Distributed Key Generation: During the distribution of the $(A_i, u_{i,j})$ an attacker could collect all messages easily reconstructing the group secret with the help of the lagrangian interpolation or even modify the values to his liking. To prevent this, each message from P_i to P_j is secured in the following way:

$$\Big\langle Nonce_i,\ A_i,\ enc_{pk_j}\big(u_{i,j} | H(A_i,\ u_{i,j},\ Nonce_i)\big)\Big\rangle$$

using the public key of P_j. With this, replay attacks are prevented as well. Additionally, the member pair of (P_i, P_j) now possess a means to communicate confidentially, using $u_{i,j}$ for the encryption. In the rare $(\frac{1}{2^q})$ case of $u_{i,j} = 0$, the value of 1 is used as key.

Adding Members: During the adding of members, the values of ω_i and δ_i need to be sent confidential as well. Otherwise an attacker could impersonate the user P_{n+1}. To prevent this kind of attack, in phase 1a the members of U_h compute $\delta_i[j]$ in the following way:

$$\delta_i[j] := f_i'(j) - f_i(j) * u_{j,i} \mod q$$

Likewise, during phase 1b the ω_i are computed with:

$$\omega_j := (u_{j,1} * L(n+1, j) + r) * u_{j,i} \mod q$$

Now, during phase 2 the newly added member P_{n+1} sends his shares of $u_{n+1,i}$ to all members of U. In this case, an attacker could modify these values, preventing a correct adding of the new member. Therefore, these values are secured in the following manner:

$$\Lambda = \{u_{n+1,1} * u_{1,n+1} | u_{n+1,2} * u_{2,n+1} | \ldots | u_{n+1,n} * u_{n,n+1}\}$$

Removing Members: During the removing the computed $\delta_i(x)$ need to be authentic to prevent adversaries to alter the coefficents in such a way that the whole system does not detect the changes while applying the normal decryption or signing operations. Changes in the polynomial become evident when another user leaves or joins the group (if a user with a changed polynomial belongs to U_h).

To prevent this, a signature is applied to the difference polynomial $\delta_i(x)$:

$$\begin{aligned} & k \in_R \boldsymbol{Z}_q^* \\ & r = g^k \mod p \\ & s = \left[H(\delta_i(x)) - a_i r\right] k^{-1} \mod q \\ & \sigma = < r, s > \end{aligned}$$

Whereby $H()$ is a cryptographic hash function, which is publicly known. σ can be verified by (El Gamal Signature):

$$g^{H(\delta_i(x))} \stackrel{?}{=} A_i^r r^s \mod p$$

Additional notes: Naturally, the values that were encrypted using $u_{i,j}$ by the sender, are decrypted with the value of $(u_{i,j})^{-1} \mod q$ by the recipient.