Reliable Performance Evaluation of Rekeying Algorithms in Secure Multicast

Abdulhadi Shoufan, Ralf Laue, and Sorin A. Huss
Integrated Circuits and Systems Laboratory, Department of Computer Science
Technische Universität Darmstadt, Germany
{shoufan|laue|huss}@iss.tu-darmstadt.de

Abstract

While a vast number of solutions to multicast group rekeying were published in the last years, a common base to evaluate these solutions and compare them with each other is missing. This paper presents a unified way to evaluate the performance of different rekeying algorithms running on the server side. A rekeying simulator estimates rekeying costs from a system point of view, which allows a reliable comparison between different rekeying algorithms. For this purpose, new system metrics related to rekeying performance are defined: the Rekeying Quality of Service (RQoS) and the Rekeying Access Control (RAC). By means of four simulation modes, these metrics are estimated by the simulator in relation to both the group size and its dynamics. A simulator prototype, implemented in Java, demonstrates the merit of this unified assessment method by means of a comprehensive case study.

1. Introduction

Secure multicast raises a scalability problem for key management in large dynamic groups. The session key, denoted as group key, must be updated every time a group member leaves the group or a new one joins it. Updating the group key, denoted as rekeying, ensures forward access control regarding leaving members and backward access control concerning the joining ones. Updated group keys must be delivered to group members efficiently. A naive scheme encrypts the new group key with the different keys of all members and then sends the resulting data as rekeying message. This solution does not scale well for large groups, as rekeying costs depend linearly on the group size.

This problem has been thoroughly addressed in the last decade to reduce the rekeying costs and to achieve a scalable group key management. A large variety of architectures, protocols, and algorithms have been proposed in literature, see [1-14].

Regardless of the quality of the proposed approaches, the reader of related publications misses a way to compare the results of these solutions to each other. This is attributed mainly to vastly different ways of estimating rekeying costs by different researchers and to the application of highly diverse metrics to express these costs. In the scope of this work, this problem is denoted as the rekeying performance evaluation problem.

In this paper an approach is presented, which provides a reliable rekeying performance evaluation, which, in turn, allows for a meaningful comparison of different rekeying algorithms. The evaluation reliability is based on a concept, which describes rekeying costs on system level independent of the evaluated rekeying algorithms and of the underlying cryptographic primitives or execution platform. As an implementation of this concept, the rekeying simulator is proposed, which supports the execution of different rekeying algorithms with uniform simulation parameters. The simulator estimates uniform cost metrics and presents simulation results in the same diagram for comparison.

The rekeying simulator ReKSim considers only the costs of cryptographic operations required for rekeying on the server side, which dominate total costs in most cases. Including other factors is part of future work.

To the authors’ knowledge, neither the rekeying performance evaluation problem has been addressed, nor solutions to it have been proposed in other work to the date of this publication.

The rest of the paper is organized as follows. Section 2 details the rekeying performance evaluation problem. Section 3 represents the proposed approach of reliable rekeying performance evaluation. The rekeying simulator is then described in Section 4. A case study simulation is presented in Section 5.

2. Rekeying Performance Evaluation Problem

Rekeying is a complex task with multiple dependencies and output quantities, which need to be considered to describe its cost. The versatile possibilities of selecting factors, metrics, and the way to estimate
rekeying costs resulted in an enormous amount of non-comparable research results. The performance evaluation problem in multicast rekeying can be specified by the following three points. The related work given is only for representation, not for completeness.

1. Non-unified definition of input quantities affecting the performance. These quantities vary considerably in literature. Examples for input quantities considered in related work are group size [1, 2], number of join and disjoin requests [3, 4], and observation times and rekeying intervals [5, 6]. In addition, many solutions estimate rekeying costs in relation to algorithm-specific parameters, such as tree height in LKH and OFT [1, 7].

2. Non-unified consideration of output metrics representing the performance. Examples for used metrics are the number of required encryptions [1, 3], the number of messages per minute [8], the number of updated keys [2], and the request processing time [4].

3. Non-unified performance estimation method. Three categories can be identified so far. These are analytical approaches [1, 7], simulation based methods [2, 9], and measurements of implemented prototypes [4, 11].

Analytical approaches are always based on simplified theoretical models and relate to special cases such as full balanced trees. The performance may often be expressed only for borderline cases by abstract numbers of some primitive operations. Simulation is mostly used to prove a proposed analytical investigation without further model enhancements to include sophisticated effects such as group dynamics. Results delivered by measurements are strongly affected by the underlying cryptographic primitives, their implementations, and by the platforms they run on.

### 3. Reliable Rekeying Performance Evaluation

Rekeying presents a solution for group key management in secure multicast. As an essential step in the process of joining and removing members, rekeying performance directly influences the efficiency of this process with major effects on the system behavior. The faster a member can be removed, the higher is the system security. The faster a member can be joined, the higher is the system quality of service. The more efficiently the rekeying, the larger the groups that can be supported, and the more members may be joined and removed per time unit. Accordingly, the significance of rekeying performance estimation results from the fundamental effects of this performance on the system behavior in respect of the following properties.

1. The amount of quality of service that can be offered to a joining member.
2. The amount of security against a removed member.
3. Scalability in terms of supportable group sizes.
4. Group dynamics in terms of maximal supportable join/disjoin rates.

![Figure 1. Abstraction model for rekeying evaluation](image-url)

A representation of rekeying performance using the above listed properties delivers a more understandable and reliable means to evaluate different rekeying algorithms. This advantage, on the one hand, stems from the abstraction, which associates rekeying performance with metrics and parameters independent of the rekeying algorithm. On the other hand, the rekeying evaluation is decoupled from underlying cryptographic primitives and execution platforms. Figure 1 illustrates this situation by representing the task of rekeying evaluation as a 4-layer abstraction model. By separating the rekeying layer from lower ones a reliable and efficient evaluation is enabled. This results from the fact, that rekeying algorithms, for the purpose of their evaluation, do not need to perform any cryptographic operations. They only estimate the kind and the number of these operations and provide corresponding data to the evaluation layer as abstract rekeying costs. The actual rekeying costs in time units are then calculated based on these data under consideration of the timing specification of the cryptography and the platform layers. As will be seen later, these timing specifications should be uniquely set for different rekeying algorithms.

#### 3.1. Rekeying cost metrics and group parameters

**Definition 1:** A Required Join Time $T_{J_{sys}}$ is a system parameter defined as the maximal allowable rekeying time needed to join a member.

**Definition 2:** An Actual Join Time $T_{J}$ specifies a join request and is defined as the sum of the waiting time
\( W_j \) of the join request in the system queue and the rekeying time \( RT_J \) consumed by a rekeying algorithm to grant this request:

\[
T_J = W_J + RT_J
\]

**Definition 3:** Rekeying Quality of Service (RQoS) specifies a join request and is defined as the difference between the required join time of the system and the actual join time of this request:

\[
RQoS = T_J^{sys} - T_J
\]

**Definition 4:** A Required Disjoin Time \( T_D^{sys} \) is a system parameter defined as the maximal allowable rekeying time needed to disjoin a member.

**Definition 5:** An Actual Disjoin Time \( T_D \) specifies a disjoin request and is defined similarly to \( T_D \) as follows:

\[
T_D = W_D + RT_D
\]

**Definition 6:** Rekeying Access Control (RAC) specifies a disjoin request and is defined as the difference between the required disjoin time of the system and the actual disjoin time of this request:

\[
RAC = T_D^{sys} - T_D
\]

**Definition 7:** Maximal Group Size \( n_{max} \) represents the supportable group size without deterioration of the system requirements of QoS and access control.

**Definition 8:** Maximal Join/Disjoin rates \( \lambda_{max}/\mu_{max} \) represent the maximal group dynamics which can be supported without deterioration of the system requirements of QoS and access control.

### 3.2. Evaluation criteria and simulation modes

Depending on the proposed metrics and parameters, rekeying algorithms may be evaluated by checking the following criteria:

1. For a rekeying algorithm to function correctly, it must feature RQoS and RAC values, which are equal to or greater than zero.
2. Considering two rekeying algorithms, which satisfy criterion 1, the algorithm that supports a higher \( n_{max} \) features a higher scalability.
3. Considering two rekeying algorithms, which fulfill criterion 1, the algorithm that supports higher \( \lambda_{max}/\mu_{max} \) features higher join/disjoin dynamics.

To verify these criteria four simulation modes are proposed as illustrated in Table 1. Note that the system parameter \( N_{max} \) given in this table represents the desired group size, which is needed by some rekeying algorithms to set up the data structures. It differs from the actual supportable \( n_{max} \) according to Definition 7.

**3.2.1. Transient Simulation.** This simulation mode estimates the current values of the group size \( n(t) \), of the rekeying quality of service RQoS(t), and of the rekeying access control RAC(t). By this means, the behavior of rekeying algorithms over long time periods can be observed. For this purpose, an initial group size \( n_0 \), a join rate \( \lambda \), a disjoin rate \( \mu \), and the desired simulation time \( t_{sim} \) are set by the user. Similarly to other modes, the transient simulation receives the system parameters \( T_J^{sys} \) and \( T_D^{sys} \) and the timing parameters, see Definition 22. The transient simulation builds the foundation of all other simulation modes.

### Table 1. Simulation modes

<table>
<thead>
<tr>
<th>System parameters</th>
<th>Transient</th>
<th>Scalloplity</th>
<th>Join Dynamics</th>
<th>Disjoin Dynamics</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_0, \lambda, \mu )</td>
<td>( T_J^{sys}, T_D^{sys}, n_{max} )</td>
<td>( C_d, C_\alpha, C_b, C_i )</td>
<td>( RAC(t) )</td>
<td>( RAC(t) )</td>
</tr>
<tr>
<td>( n_{max} \lambda \mu )</td>
<td>( n_{max} \lambda \mu )</td>
<td>( n_{max} \lambda \mu )</td>
<td>( n_{max} \lambda \mu )</td>
<td>( n_{max} \lambda \mu )</td>
</tr>
<tr>
<td>Simulation parameters</td>
<td>( \lambda_{max}, \mu_{max} )</td>
<td>( RQoS(t) )</td>
<td>( RQoS(t) )</td>
<td>( RQoS(t) )</td>
</tr>
<tr>
<td>Variable</td>
<td>( n )</td>
<td>( \lambda )</td>
<td>( \mu )</td>
<td>( \mu )</td>
</tr>
</tbody>
</table>

**3.2.1. Scalability Simulation.** The importance of this simulation mode results from the significance of the scalability problem in group rekeying. The scalability simulation investigates the effect of the group size on the system behavior, which is caused by the terms \( RT_J \) and \( RT_D \) in (1) and (3). Each value of \( n \) in a user-definable range serves as an initial group size for a transient simulation, which is run in a fixed user-definable observation interval \( T_o \). From all the resulting values of RQoS and RAC for this simulation point, only the worst case values are considered, i.e. \( RQoS_{min} \) and \( RAC_{min} \). The scalability simulation helps to estimate the maximal group size \( n_{max} \).

**3.2.3. Join/Disjoin Dynamics Simulation.** High join rates result in shorter inter-arrival times of join requests and more rekeying computations. This causes longer waiting times for join and disjoin requests according to the terms \( W_J \) and \( W_D \) in (1) and (3). Thus, higher join rates do not only affect RQoS, but also RAC. The join dynamic simulation represents a way to investigate theses dependencies. The user defines an initial group size \( n_0 \), a disjoin rate \( \mu \), and a fixed observation interval \( T_o \). In addition, a simulation range for the join rate \( \lambda \) is entered. For each value of \( \lambda \) a transient simulation over \( T_o \) is started as in the scalability simulation. With the help of join dynamics simulation the maximal join rate \( \lambda_{max} \) for a rekeying algorithm may be determined.
The disjoin dynamics simulation is similar to the join dynamics simulation and can be exploited to calculate the maximal disjoin rate $\mu_{\text{max}}$.

4. Rekeying Simulator (RekSim)

The Rekeying Simulator is mainly composed of two interfaces and three components, as depicted in Figure 2. The User Interface (UI) enables users to evaluate different rekeying algorithms by selecting these algorithms and setting the desired parameters. For designers of new rekeying algorithms a Programming Interface (PI) is provided for the integration of new algorithms into the simulator. In addition, groups with special dynamic behavior, which does not follow Poisson distribution, can be supported by means of a special programming interface module.

The component Request Generator builds a rekeying request list depending on the selected group and simulation parameters. An entry of this list keeps information on the request type, join or disjoin, the identity of the member to be joined or removed, and the arrival time of this request. The Algorithm Manager then selects and configures the rekeying algorithms according to the user settings. It coordinates all the functions of the simulator and controls the rekeying algorithms. Based on the rekeying cost data delivered from the rekeying algorithms and the entered timing parameters, the Performance Evaluator finally estimates the rekeying performance of each algorithm and prepares its output for a graphical presentation.

For clarity, the next sections detail the three simulator components in an order corresponding to the data flow as depicted Figure 2. For the lack of space, a complete presentation of all functions as pseudo-code can not be given here. Thus, the presentation is limited to some essential functions only.

4.1 Request Generator

The Request Generator produces a rekeying request list RRL(T) by executing the Request Generator Process based on three subprocesses: the Arrival Process to generate join/disjoin arrival lists $A_J(T)/A_D(T)$ and the Join/Disjoin Identity Selection Processes to generate member identities. For a formal description a terminology specific to the Request Generator is presented first.

Definition 9: A Rekeying Request is a 3-tuple $(\text{type}, \text{ID}, \text{t}_\text{a})$. type indicates the request type, which may either be join (J) or disjoin (D). ID represents the member identity to be joined (IDJ) or removed (IDD). $t_a$ describes the arrival time of a join/disjoin request $t_{aJ}$/$t_{aD}$, measured from the start point of the simulation run.

\[ \text{RRL}(T) \text{ is an ordered set of rekeying requests, where } \text{RRL}(T)_{\text{join}} = (\Delta t_{J}(1), \Delta t_{J}(2), \ldots, \Delta t_{J}(i), \ldots, \Delta t_{J}(h)), \text{ and } \]
\[ \text{RRL}(T)_{\text{disjoin}} = (\Delta t_{D}(1), \Delta t_{D}(2), \ldots, \Delta t_{D}(j), \ldots, \Delta t_{D}(k)). \]

Definition 10: A Rekeying Request List over T, RRL(T), is an ordered set of rekeying requests, which arrive during a defined time interval T. The requests in the list are ordered according to their arrival times.

Definition 11: A join arrival list over T, $A_J(T)$, is an ordered list of inter-arrival times relating to all join requests generated during a given time interval T: $A_J(T) = (\Delta t_{J}(1), \Delta t_{J}(2), \ldots, \Delta t_{J}(i), \ldots, \Delta t_{J}(h))$, where $\Delta t_{J}(i)$ indicates the inter-arrival time of the i-th join request in the interval T, and
\[ \sum_{i=1}^{h} \Delta t_{J}(i) \leq T \] (5)

Definition 12: A disjoin arrival list over T, $A_D(T)$, is an ordered list of inter-arrival times relating to all disjoin requests generated during a given time interval T: $A_D(T) = (\Delta t_{D}(1), \Delta t_{D}(2), \ldots, \Delta t_{D}(j), \ldots, \Delta t_{D}(k))$, where $\Delta t_{D}(i)$ indicates the inter-arrival time of the i-th disjoin request in the interval T, and
\[ \sum_{i=1}^{k} \Delta t_{D}(i) \leq T \] (6)

Definition 13: A member identity ID is a natural number between 0 and $N_{\text{max}} - 1$.

Definition 14: A complete multicast group $M$ is defined as a set of all member identities:
\[ M = \{ID(i)\}, i = 0 \div (N_{\text{max}} - 1) \]

Definition 15: A joined multicast subgroup $M_J$ is defined as the subset of all given identities. At the start of a simulation with an initial group size $n_0$, $M_J$ is defined as follows:
\[ M_J = \{ID(i)\}, i = 0 \div (n_0 - 1) \]

Definition 16: A potential multicast subgroup $M_D$ is defined as the subset of all identities, which can still be given to new members. At the start of a simulation with an initial group size $n_0$, $M_D$ is defined as follows:
\[ M_D = \{ID(i)\}, i = n_0 \div (N_{\text{max}} - 1) \]
4.1.1. Request Generator Process GenReqList. This process generates the rekeying request list RRL(T) as given in Algorithm 1. First, the arrival process GetArrivalLists(T) is called to produce join and disjoin arrival lists \( A_J(T) \) and \( A_D(T) \). According to the inter-arrival times in these lists, the arrival times for the individual requests are determined. Depending on the request type, the member identity is then obtained by calling GetJoinID or GetDisjoinID. Then, the RRL(T) is updated by the new rekeying request 3-tuple. After processing all entries in \( A_J(T) \) and \( A_D(T) \), the RRL(T) is sorted by increasing arrival time. Note that the request generator is transparent to the simulation mode. Utilizing the generator for different simulation modes will be described in the scope of the Algorithm Manager.

Algorithm 1. GenReqList

**Input:** T  
**Output:** RRL(T)  
1. GetArrivalLists(T) \( \rightarrow A_J(T), A_D(T) \)  
2. \( i := 1, j := 1, t_a := 0, t_d := 0; \)  
3. **do**  
4. if \( \Delta t_J(i) > \Delta t_D(j) \) then  
5. \( t_a := t_a + \Delta t_D(j) \)  
6. GetDisjoinID \( \rightarrow ID_D \)  
7. \( j := j + 1; \)  
8. else  
9. \( t_d := t_d + \Delta t_J(i) \)  
10. GetJoinID \( \rightarrow ID_J \)  
11. \( i := i + 1; \)  
12. **end if**  
13. Update RRL(T)  
14. while \( i \leq h \ or \ j \leq k \)  
15. Sort RRL(T)  
16. **return** RRL(T)

4.1.2. Arrival process GetArrivalLists. According to related work on modeling multicast member dynamics [15], the rekeying simulator assumes, as default, inter-arrival times, which follow an exponential distribution for join and disjoin requests with the rates \( \lambda \) and \( \mu \), respectively. The corresponding cumulative distribution functions are given by:

\[
F_J(\Delta J) = 1 - e^{-\lambda \Delta J} \quad F_D(\Delta D) = 1 - e^{-\mu \Delta D} \quad (7)
\]

To generate an exponentially distributed random variate based on uniform random numbers between 0 and 1, the inverse transformation technique can be used. Accordingly, if \( r \) represents such a random number, the inter-arrival times of a join and disjoin request can be estimated as:

\[
\Delta t_J = -\frac{1}{\lambda} \ln r \quad \Delta t_D = -\frac{1}{\mu} \ln r \quad (8)
\]

Algorithm 2 outlines the arrival process.

**Algorithm 2. GetArrivalLists**

**Input:** T  
**Output:** \( A_J(T), A_D(T) \)  
1. \( \Sigma \Delta t_J := 0, \quad \Sigma \Delta t_D := 0 \)  
2. while \( \Sigma \Delta t_J \leq T \) do  
3. Generate \( r \)  
4. Determine \( \Delta t_J \) according to (8)  
5. \( \Sigma \Delta t_J := \Sigma \Delta t_J + \Delta t_J \)  
6. Update \( A_J(T) \)  
7. while \( \Sigma \Delta t_D \leq T \) do  
8. Generate \( r \)  
9. Determine \( \Delta t_D \) according to (8)  
10. \( \Sigma \Delta t_D := \Sigma \Delta t_D + \Delta t_D \)  
11. Update \( A_D(T) \)  
12. **return** \( A_J(T), A_D(T) \)

4.1.3. Join/Disjoin identity selection processes. To join a member, the group manager may select any available identity number \( ID_J \) from the potential multicast subgroup \( M_J \). A possible selection strategy may rely on choosing the smallest available \( ID_J \), which allows some order in the group management. In contrast, selecting a leaving \( ID_D \) from \( M_J \) is inherently indeterministic, as a group owner can not forecast which member will leave the group. To select an \( ID_D \) the following method is proposed.

The \( ID_D \)'s of \( M_J \) are associated with continuous successive indices from 0 to \( m-1 \), where \( m \) is the number of all \( ID_D \)'s in \( M_J \). To select an \( ID_D \), first a uniform zero-one random number \( r \) is generated. Then an index \( j \) is determined as \( mr \). In a last step, the \( ID_D \) is selected, which is indexed by this \( i \).

4.2. Algorithm Manager

This component acts as a coordinator within the simulator and fulfills the main tasks of user interface management, algorithm control, and simulation execution. For this purpose the Algorithm Manager reads in the user settings and calls the request generator. It then passes the request list to the selected rekeying algorithms and collects the rekeying cost data. These data are then sent to the Performance Evaluator. For further descriptions, some basic concepts are introduced first.

**Definition 17:** Abstract Rekeying Cost (ARC) is a 5-tuple \( (G, E, H, M, S) \), which specifies the costs of a rekeying request in terms of the amount of cryptographic operations needed to grant this request by a rekeying algorithm. The elements of the ARC are specified in Table 2.

**Definition 18:** A Rekeying Cost List RCL(T) is a rekeying request list RRL(T), see Definition 10, which is extended by the abstract rekeying cost ARC for each request.
Table 2. Abstract Rekeying Costs notation

<table>
<thead>
<tr>
<th>ARC Element</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td># Generated cryptographic keys</td>
</tr>
<tr>
<td>E</td>
<td># Symmetric encryptions</td>
</tr>
<tr>
<td>H</td>
<td># Cryptographic hash operations</td>
</tr>
<tr>
<td>M</td>
<td># Message authentication codes</td>
</tr>
<tr>
<td>S</td>
<td># Digital signatures</td>
</tr>
</tbody>
</table>

Definition 19: A Complex Rekeying Cost List CRCL(T) is a set of rekeying cost lists generated over the same interval under different group conditions: CRCL(T) = {RCL1(T), RCL2(T),...}. In the simulation modes of scalability and join/disjoin dynamics, an RCL(T) is generated for each n, λ, or μ value in the desired simulation range, respectively.

Algorithm 3. DoTranSim

Input: All settings given in Table 1 for transient simulation, set of rekeying algorithms to be evaluated.

Output: A RCL(tim) for each rekeying algorithm
1. GenReqList(tim) → RRL(tim)
2. for each rekeying algorithm do
3. Initialize the group with n members
4. while RRL(tim) is not empty do
5. Send a rekeying request to the algorithm
6. Get corresponding ARC
7. Add ARC to RCL(tim)
8. return RCL(tim) of all algorithms

Algorithm 3 represents the process of transient simulation DoTranSim. The request generator process is resumed to generate a request list RRL(tim) for the desired simulation time period. For each selected rekeying algorithm, the Algorithm Manager performs two main steps. Firstly, the rekeying algorithm is requested to initialize a group n₀. Secondly, each rekeying request of RRL(tim) is sent to the rekeying algorithm, which then estimates the corresponding abstract rekeying cost ARC. As mentioned in Section 3.2, other simulation modes are based on the transient simulation and are similar with regard to their functionality. Therefore, only the scalability simulation is described here by means of Algorithm 4.

Algorithm 4. DoScalSim

Input: All settings given in Table 1 for scalability simulation, set of rekeying algorithms to be evaluated.

Output: A CRCL(T₀) for each rekeying algorithm
1. n := n₀;
2. for each rekeying algorithm do
3. while n ≤ n₀ do
4. DoTranSim for T₀ and n= n → RCL(T₀)
5. n := n + ∆;
6. Add RCL(T₀) to CRCL(T₀)
7. return CRCL(T₀) of all algorithms

4.3. Performance Evaluator

This component receives a set of RCL(T) or CRCL(T) and calculates both system metrics RQoS and RAC with respect to time, group size, or dynamics.

Definition 20: A Performance Simulation Point (PSP) is a 3-tuple (x, RQoS, RAC), where x is the variable, which the RQoS and RAC are related to, e.g. t_sim in the transient simulation. Depending on the simulation mode x, RQoS, and RAC are interpreted as illustrated in Table 1. Note that in a transient simulation RQoS and RAC are not defined for a disjoin and join request, respectively.

Definition 21: A Rekeying Performance List (RPL) is a set of performance simulation points. RPL = {PSP} = {x₁, RQoS₁, RAC₁}, {x₂, RQoS₂, RAC₂},...

Definition 22: A Timing Parameter List (TPL) is a 5-tuple (Cg, Ce, Ch, Cm, Cs), where the tuple elements are defined as given in Table 3.

Recall that the timing parameters reflect the performance of cryptographic algorithms and of the platform on an abstraction level, which allows for a reliable evaluation of different rekeying algorithms.

Table 3. Timing parameters

<table>
<thead>
<tr>
<th>TPL Element</th>
<th>Meaning: Time cost of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cg</td>
<td>Generating one cryptographic key</td>
</tr>
<tr>
<td>Ce</td>
<td>One symmetric encryption</td>
</tr>
<tr>
<td>Ch</td>
<td>One cryptographic hash operation</td>
</tr>
<tr>
<td>Cm</td>
<td>One message authentication code</td>
</tr>
<tr>
<td>Cs</td>
<td>One digital signature</td>
</tr>
</tbody>
</table>

The Performance Evaluator executes processes, which combine a rekeying cost list RCL(T) or a complex rekeying cost list CRCL(T) with a timing parameter list TPL to produce a rekeying performance list PRL for a specific rekeying algorithm. For each rekeying request in RCL(T)/CRCL(T) the actual join/disjoin time is established according to relations (1) and (3). The waiting and rekeying times for a join or disjoin request may be determined as follows.

\[ RT_{J/D} = G \cdot C_g + E \cdot C_e + H \cdot C_h + M \cdot C_m + S \cdot C_s \]  
\[ W_{J/D} = \sum_{i=1}^{m} RT_i \]  
\[ W_{J/D} = 0 \]  

m represents the number of all requests waiting in the system queue or being processed at the arrival of the request at hand. Subsequently, RQoS and RAC may be calculated for a join or disjoin request according to (2) or (4), respectively.
In case of a transient simulation the Performance Evaluator executes the process EvalTranSimResults. For each join/disjoin request in the input list RCL(T), a performance simulation point PSP is provided. The other simulation modes deliver a CRCL(T). The Performance Evaluator generates one performance simulation point PSP for each RCL(T). The first element of the PSP tuple represents a $n$, $\lambda$, or $\mu$ value. The second element represents the minimal rekeying quality of service RQoS$_{\text{min}}$ of all join requests in the observation time for the corresponding $n$, $\lambda$, or $\mu$ value. Similarly, the third element represents RAC$_{\text{min}}$ of all disjoin requests. The corresponding process is denoted as EvalNonTranSimResults. For brevity, the processes executed by the Performance Evaluator are not detailed here.

5. Implementation and Case Study

A prototype of the rekeying simulator was implemented in Java using the Eclipse SE environment. A graphical user interface enables users to select rekeying algorithms and to enter simulation settings. Figure 3 shows one register of the simulation setup window. To illustrate the advantages of the proposed tool RekSim, a simulation case study is performed to evaluate tree rebalancing in the LKH algorithm.

LKH is based on the management of a key tree on the rekeying server. As an effect of multiple disjoin processes, the key tree may get out of balance. Several solutions have been proposed to rebalance the tree in this case. The first contribution originates from Moyer [11] who introduced two methods to rebalance the key tree, an immediate and a periodic rebalancing. Only a cost analysis after one disjoin request is given for the first method. The periodic rebalancing is not analyzed. Moharrum [12] presented a method for rebalancing based on sub-trees. A comparison with the solution of Moyer is drawn, but not with the original LKH. Rodeh [13] applied AVL-tree rebalancing methods to key trees. However, no backward access control is guaranteed in this solution. Goshi [14] proposed three algorithms for tree rebalancing. Simulation results are provided, which assume equally likely join and disjoin behavior. However, this condition in itself ensures tree balancing, because a new member can be joined at the leaf of the most recently removed member. The same applies to the simulation results by Lu [9].

From this description, it is obvious that a comprehensive analysis is needed to justify the employment of rebalancing, which is associated with extra rekeying costs resulting from shifting members between tree leaves. The simulation tool RekSim offers this possibility by allowing a simultaneous evaluation of two LKH algorithms (with and without rebalancing) under complex conditions. Especially the effect of the disjoin rate is of interest in case of rebalancing. Therefore, a disjoin dynamics simulation is performed under the following conditions: $T_{\text{sys}}=100\text{ms}$, $N_{\text{max}}=65,536$, $C_g=C_e=C_b=C_m=1\mu s$, $C_s=15\text{ms}$, $n_0=4096$, $\lambda=10^{-1}$, $T_o=1s$, $\mu_{\text{start}}=1^{-1}$, $\mu_{\text{stop}}=20^{-1}$, $\Delta \mu=1^{-1}$. The simulation result clearly unveils that rebalancing degrades both RQoS and RAC values. This deterioration of performance increases with an increasing disjoin rate. Simulation results are depicted in Figure 4 and 5.

Figure 4. RQoS in rebalanced vs. non-rebalanced LKH

The results of this case study unambiguously demonstrate that additional rekeying costs associated with
rebalancing exceed the performance gain achieved by it. Consequently, rebalancing is not advantageous for LKH trees, at least under the given simulation conditions.

Figure 5. RAC in rebalanced vs. non-rebalanced LKH

Currently, related work argues for rebalancing as a way to prevent tree degradation, which results in linear rekeying costs with respect to the group size in the rather extreme case of a very high disjoin rate. The main point, which is disregarded in this argumentation, is that the group size in such rare cases becomes very small and is then almost equal to the LKH tree height in the balanced case.

6. Conclusion

An assessment methodology and an associated simulation tool were presented as a novel method to deal with the rekeying performance evaluation problem. By means of the underlying concept of abstraction a reliable and meaningful evaluation of different rekeying algorithms is provided. A case study illustrated the advantage of this simulator in analyzing yet unanswered questions relating to rekeying. In its first prototype, the simulator RekSim considers rekeying costs in terms of cryptographic operations to be run on the server side. Other cost factors, such as tree traversing in LKH, will be addressed in future work.

7. References


