Secure Efficient Group Key Management For Few-to-Many Applications

Nathalie Weiler
Swiss Federal Institute of Technology (ETH Zürich), Switzerland
weiler@tik.ee.ethz.ch

ABSTRACT

Multimedia applications, news distribution and service discovery protocols are examples of group applications that typically involve more than one sender in the distribution process. This paper proposes few-to-many Semsomm, a secure and scalable group key management scheme for such applications. The main strategy of Semsomm is twofold. First, through the use of a multiple encryption scheme, there is no longer the need to trust the intermediate nodes of the multicast distribution tree. They are used as untrusted relaying nodes in order to overcome the need to re-key the entire group upon each membership change. Second, the traffic encryption key is periodically renewed and redistributed to legitimate group members, thus inhibiting any collusion attack. It is shown that Semsomm scales to very large groups while preserving perfect forward secrecy of the multicasted information, i.e. only actual members of the group can understand it.

KEY WORDS

Secure Group, Key Agreement, Internet and Software Control, Protection, and Security, Security Protocols.

1 Introduction

The success of multicast capable overlay networks for the Internet and similar mass communication networks will be determined by their ability both to preserve the privacy of their users and to secure the network infrastructure of their operators. IP multicast does not support closed user groups, everyone can listen to all traffic of a specified multicast group which IP address is universally known – any host may join or leave a multicast group by sending IGMP (Internet Group Management Protocol) messages to their local router. Even worse, multicast accentuates certain security threats: Especially active assaults such as masquerading or denial-of-service attacks may succeed rapidly, because of missing access control mechanisms.

A secure group communication protocol is characterised by a multitude of properties originating from different domains. The type of application dictates number, physical distribution and dynamics of the group members. The network influences reliability, fault tolerance and delay during the session. One of the prerequisites for security services is the knowledge of the key(s) used in the cryptographic primitives. Secure multicast requires a multi-party key agreement mechanism between the members, not only a two-party key agreement as does peer-to-peer networking. Such mechanisms in their basic form are often error-prone and expensive in communication and computational costs[1].

There exist two properties for a secure multicast protocol that are considered in the literature (e.g. [2]) to be the most important to achieve: *scalability* and *perfect forward secrecy*. Therefore, we will first define these key properties in the context:

PROPERTY 1

*A secure multicast protocol is suffering from the **1 affects n scalability problem** if a membership change in the group has an impact on all n members in the group. It is said to be **scalable** if the overhead involved in key updates, data transmission and encryption and decryption is independent of the size of the multicast group.*

PROPERTY 2

***Perfect Forward Secrecy** is ensured by a secure multicast protocol if only current members of the group can decrypt the multicast traffic.*

Property 1 guarantees that the protocol scales to large and very large groups (up to thousands or millions of members) while still guaranteeing a good performance which respect to transmission, storage and computations and avoiding single points of failures. On the other hand, property 2 allows for dynamic groups that resist collusions, i.e. it is not possible for a subset of disbanded members to recreate any keying material.

In [3], we proposed Semsomm, a new generation protocol for *scalable, perfect forward secure group communication for a classical one-to-many scenario with one sender and a huge number of receivers*. We combine the idea of periodic re-keying with a new double encryption scheme involving internal nodes of the multicast distribution scheme. This paper describes extensions of Semsomm to *few-to-many scenarios*, i.e. to applications involving a huge number of receivers, but only few senders and shows how Semsomm[2] achieves both the properties of scalability and perfect forward secrecy. The remaining of the paper is organised as follows: First, Section 2 presents related work. Section 3 details the data distribution protocol design and the basic secure group management operations. Section 4 contains the theoretical evaluation and and Section 5 the results of the test scenarios. Section 6 concludes by reviewing our claims.

2 Related Work

A number of approaches have been proposed in the last decade to secure group communications. The first schemes

[1]The IETF working group on multicast security [1] is currently standardising key agreement protocols relying on one central trusted group controller.

[2]We use the name "Semsomm" in the remainder of this paper to designate the few-to-many variant. References to any other variants are explicitly mentioned.

addressed broadcast scenarios, then major efforts for multicast scenarios were undertaken. Early work can be separated into two categories:

The naive – also called **static** – solutions require the establishment of a new group to cope with membership changes. Gong et al. [4] encrypt the key separately for each member and distribute it via unicast to each member. RFC 2094 [5] is the first proposal for IP Multicast. It uses a group wide encryption key. All of these schemes lack a method to remove members of a group. None provides perfect forward secrecy for groups.

The **dynamic** solutions, on the other hand, can change the group key material on the fly. In order to prevent new joining members from understanding past traffic and old, expelled members from listening to future messages, the traffic encryption key is changed without rebuilding the whole group. Among the existing dynamic approaches, we distinguish between three categories: (1) **centralised hierarchical** approaches, (2) **semi-distributed** key management, and (3) **distributed** schemes.

Hierarchical approaches use a key graph – typically a tree – to distribute secret keys (e.g. [6]): Members of a secure multicast group are part of this tree and receive the secret keys according to their position in the tree. Typically the members are positioned in the leaf nodes. An important characteristic of these approaches is the existence of one single group controller or key distributor. The task of this designated, centralised controller is to take care of distributing and/or of updating keying material. All the centralised hierarchical approaches below share the inherent drawbacks: (1) single point of failure, (2) danger of setup implosion, and (3) relatively large database for the keys.

Semi-distributed approaches introduce a partial distribution of crucial system parts, but still sustain the centralised nature of the secure multicast by relying on a central unit. **Iolus** [7] deals with the scalability issues in highly dynamic large groups by decomposing large groups into subgroups. Thus, a group membership change can be handled in the respective subgroup without affecting any other subgroups. While improving scalability, the absence of a global group key requires the introduction of secure agents, one for each subgroup, to relay messages and perform key translation. In addition to requiring full trust into each subgroup agent, extra delays in message delivery must be accepted. **CFlat** – Centralised Flat key management [8] – is an intermediate solution, which copes better with the memory allocation for the key space than the centralised, tree-based approach, yet preserves the simplicity of the centralised approach. In **Kronos** [9], group re-keying in the tree is fulfilled at periodic intervals rather than at membership changes. Thereby, costs incurred by joins and leaves are optimised at the price of the perfect forward secrecy. Kronos is fault-tolerant because the subgroup controller can generate the new keys independently of the group controller. However, the group security may be badly compromised if the re-key interval is chosen to large. On the other hand, short intervals result in a overloaded system.

The main concerns with centralised approaches are the danger of implosion and the existence of a single point of failure. It is thus attractive to search for a distributed solution for the key management problem. **DFlat** – Distributed Flat key management [8] – does not know of a group manager or any other centralised controlling unit. Each member of the group holds the data encryption key and a set group management keys. The collaboration of multiple members is required to propagate key changes to the whole group. There is no dedicated group manager, instead, every member may perform admission control and other administrative functions. This scheme is highly resilient to network or node failures because of its inherent self-healing capability, but is also more vulnerable to inside attacks. On the other hand, the scheme trusts all key holders of the group. Consequently, it is vulnerable to attacks from with this set of key holders. Furthermore, it is difficult to exclude colluding members.

3 Design of Few-to-Many Semsomm

Secure few-to-many multicast scenarios require the data to be distributed only to a set of legitimate users or subscribers. We take as an example a news distribution scenario (see Figure 1): The data is distributed to the subscribers through two senders. The subscribers are grouped into two locations A and B for performance reasons. $Sender_1$ is responsible for R_1, R_4 and R_5 at location A (right-hand side of Figure 1) and $Sender_2$ for R_2 and R_3 at location B.

As the senders cannot control the membership in an IP multicast group, anybody may receive the multicasted data. The typical approach to solve this problem is to encrypt the multicasted data with a symmetric – for better performance – session key and to distribute this key to the legitimate receivers. The distribution of the session key should be secure for dynamic groups (Property 1), and scalable to a large number of members (Property 2).

In the remaining of this section, we will now first describe the general data distribution protocol (Section 3.1). Then, the join and leave protocols (Section 3.2) and the periodic re-keying protocol (Section 3.3) are presented. In the rest of this paper, we use the notation described in Table 1.

3.1 General Data Distribution Protocol

The general data protocol illustrated in Figure 1 shows how the multiple encryption scheme of Semsomm is applied.

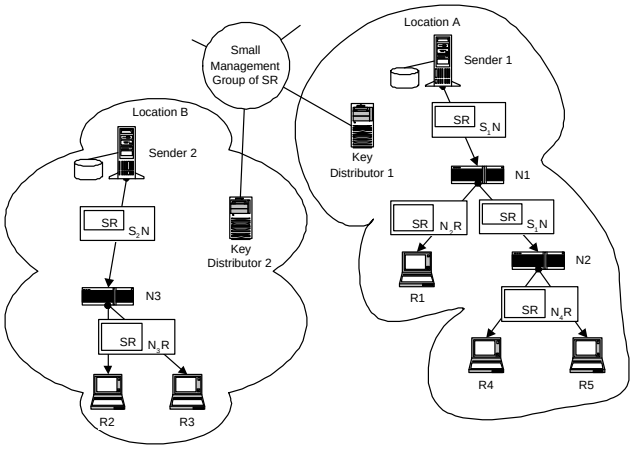


Figure 1. Key Usage in Few-to-Many Scenario.

R_i	: Receiver Number i
N_j	: Intermediate Node Number j
KD_k	: Key Distributer Number k
SR	: Encryption Key for multicast traffi c used between Sender and Receivers
S_kN	: Encryption Key used between Sender and intermediate Nodes at the same location
N_jR	: Encryption Key used between Node N_j and its adjacent Receivers
KD_kR_i	: Encryption Key used during the re-keying, known only to KD_k and R_i
f	: One-Way Hash Function known at least to all sender and intermediate nodes.

Table 1. Notation.

The data payload is first encrypted by the senders with the key known only by the senders, receivers and key distributors SR. Additionally, the senders also perform a second encryption with the keys S_1N and S_2N known only to them, the intermediate nodes and the key distributors. The intermediate nodes can be mapped on the multicast distribution tree. They are responsible for the distribution of the data to the receivers. Finally, the node N_x decrypts the packet for its receivers and forwards it to them after encrypting it with a key N_xR known to itself and its receivers.

Because the knowledge of the keys is distributed, only legitimate receivers knowing SR can decrypt the original message. The intermediate nodes are only trusted for forwarding and, if necessary, re-encrypting these encrypted messages. They cannot read the original message because they do not know SR.

3.2 Basic Operations

The basic operations – joining and leaving the group – are the most critical with respect to perfect forward secrecy and scalability of the key management scheme.

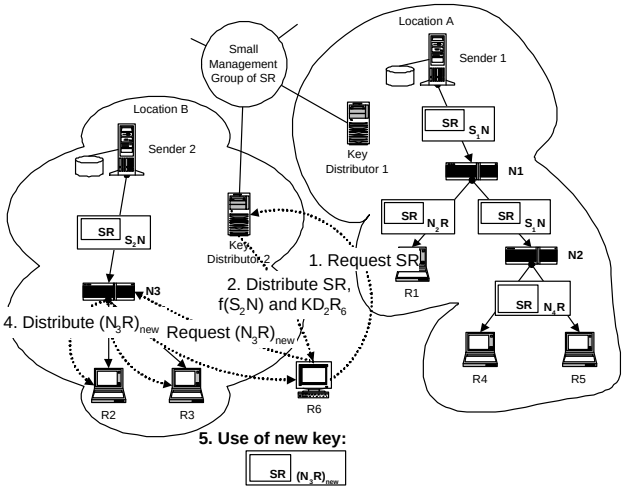


Figure 2. Join of R_6.

Join The join process of a new member is performed in five steps (Figure 2):

1: The new participant R_6 requests SR from the key distributor at the present location (no. 2) presenting its credentials – e.g. some kind of payment – to join the group.
2: The key distributor 2 verifies the credentials. In case of successful verification, it computes the result of $f(S_2N)$ and transmits both $f(S_2N)$ and SR together with an additional key KD_1R_6 through a secure channel to R_6.
3: R_6 asks N_3 for N_3R presenting $f(S_2N)$ as credential.
4: N_3 generates a new N_3R and distributes it to the subscribed receivers R_2, R_3 and R_6 upon successful verification of $f(S_2N)$.
5: From this point in time on, the $(N_3R)_{new}$ is used as encryption key by N_3.

Leave To illustrate the leave operation, we assume that R_3 leaves the group – voluntarily or forced. The leave operation is straightforward: We only need a new (N_3R_{new}) that is distributed by N_3 to the remaining receivers, in this case R_2 and R_6. Henceforth, this key is used by N_3. Note that there are only changes needed at location B. Additionally, N_3 needs to do bookkeeping of the membership changes for the later periodic re-keying.

3.3 Periodic Re-Keying

SR and S_xN are replaced by newly generated keys at regular time intervals. Thereby, (a) infinite lifetimes of the keys are avoided, and (b) the overhead involved is predictable, an important feature especially for applications running on limited resources. Contrary to the one-to-many Semsomm variant, we need an additional group to manage the re-keying between the key distributors. All key distributors belong to this group that can employ any of the fast, static multicast key agreement schemes discussed in Section 2. Thus, the periodic re-keying is a two step protocol: First, the respective key – here SR – is distributed to the key distributors, and they than multicast them to their receivers using the symmetric keys KD_kR_i to guarantee that only the legitimate receivers can decrypt the new SR. S_kN is updated in an analogue manner.

4 Evaluation

In this section, we evaluate the theoretical costs of Semsomm: We analyse its collusion resistance to show that Semsomm achieves perfect forward secrecy. On the other hand, we give the overhead metrics of interest: (1) the costs of communication, i.e. the number and type of messages sent, and (2) the computational costs per group operation, i.e. the number of exponentiations, signatures, verifications, encryptions, decryptions, hashes and key generations.

4.1 Collusion Analysis

The goal of a collusion analysis is to ensure that a coalition of non members of a secure multicast group cannot use their combined knowledge to gain more information than one individual member of the coalition already has. We distinguish between three kinds of collusions:

1. *Outsiders* that were *never members of the group*,

2. *Outsiders* that were at some point in time *members of the group*, and
3. *Intermediate nodes*.

The first attackers are no threat to the group as they do neither know SR nor S_kN nor any of the pairwise shared encryption keys.

The second group of adversaries knew at least one old SR. Furthermore, they know the local encryption keys used between their intermediate nodes and the node's adjacent receivers. From the combined knowledge, they can only follow the group communication until the last one of their coalition is expelled from the group. This is not a security leak since it is impossible to prevent that one of the members of the group forwards the received data in plaintext to outsiders.

For the last group of attackers, the intermediate nodes do not know the traffic encryption key at any time. So this attacker group is included in the first one described above.

4.2 Communication Overhead

We compare the complexity of the most frequent operations – namely the join and leave ones – with those of other approaches mentioned in Section 2: Table 2. If the number of members adjacent to the same intermediate node is smaller than the binary logarithm of the total number of members, then Semsomm's communication overhead is lower than the centralised approaches (Cent.Tree [6]), the semi-distributed Iolus approach [7] and the distributed DFlat one [8]. The probability for this event can rationally be assumed to be very high, because $m \rightarrow log(M)$ is equivalent to say that all group members tend to share the same intermediate node, i.e. they are co-located in the same local network environment. In this kind of arrangement broadcast encryption is the scheme of choice, because it can be done with a lower overhead in communication than $log(M)$. Iolus is less communication intensive than Semsomm on joins. On a leave, however, it is more communication intensive, because its subgroups are larger than m. So, $avg(l) > m$.

No. of Messages	Semsomm	Centr. Tree	Iolus	DFlat
Join	$o(m)$	$O(logM)$	$O(1)$	$O(logM)$
Leave	$o(m)$	$O(logM)$	$O(avg(l))$	$O(logM)$

l : No. Subgroups
m : No. Members Adjacent to the Same N_i
M : No. Members
$avg(l)$: Average Size of a Subgroup

Table 2. Communication Overhead.

4.3 Computational Overhead

Join The computational costs of a join are summed up over the five steps of the join protocol in Semsomm:

1: *The new joiner requests the traffic encryption key SR at the key distributor.* This step involves one Diffie-Hellman key exchange between the key distributor (KD) and and RSA signature generation and verification step for data authentication.
2: *The KD replies to the newcomer with the necessary information to permit him to join at the intermediate node.* The KD must generate one symmetric key[3], one SHA-1 hash of the encryption key used with the intermediate node by the KD, one encryption and an RSA signature. The newcomer must perform one decryption and one RSA signature verification.
3: *The newcomer requests the second traffic encryption key at its intermediate node.* Therefore, one needs again a Diffie-Hellman key exchange and one RSA signature-verification cycle.
4: *The intermediate nodes distributes the new second traffic encryption key.* So, one symmetric key must be newly generated, one SHA-1 hash must be computed and the result matched to the one presented by the newcomer, the new key must be encrypted for the new members and distributed to them in an authentic manner (one additional RSA signature and verification cycle).
5: *The new traffic encryption key is used.* No costs occur in this step.

The join operation is computationally intensive for the newcomer and to a smaller degree for the concerned intermediate node. These loads are rather moderate, and take less than 67 milliseconds for the newcomer and less than (36+0.005*m) milliseconds for the intermediate node (see Table 3).

Leave The leave operation is very inexpensive, because the concerned intermediate node must only generate one new key and distribute it to the remaining members. More details can be found in [11].

Memory consumption is very low for the members. They need to store four keys (two traffic encryption keys and the two symmetric communication keys shared with the key distributor and the adjacent intermediate node respectively.). An intermediate node must store the traffic encryption key shared with the other intermediate nodes, the shared key with its key distributor and the shared keys with its adjacent receivers and senders. Each key distributor finally has one key for each member and intermediate node it admitted to the group and the traffic keys in memory.

5 Results

5.1 Simulation Environment

Semsomm was implemented on top of the Spread group communication toolkit [12] for simulation purposes[4]. Spread can be used for local area and wide area networks and provides different kind of group services including unreliable and reliable packet delivery, FIFO, causal, and total ordering and stronger semantics as the Extended Virtual Synchrony (EVS) model and the View Synchrony (VS) model. Both EVS and VS guarantee that (1) all group members receive the same set of messages between two sequential group membership events, (2) the message order the sender requested is preserved. VS offers a stricter guarantee

[3]This symmetric key is later used in the re-key operation to distribute the new traffi c encryption keys securely to all group members

[4]The basic Spread group communication toolkit is available publicly as open source for non-commercial usage. The toolkit supports cross-platform applications and is available for several Unix and Windows platforms.

Function	Costs per Function in ms[a]	Key Distributor	Newcomer	Adjacent N_i	Adjacent M_i (to N_i)
DH Key Agreement	< 20	1	2	1	–
RSA Signature	< 10	1	2	1	–
RSA Verify	< 1	1	2	1	1
Key Generation	< 2	2	2	2	–
Hash (Alg.: SHA-1)	< 0.001	1	–	1	–
Encryption (Alg.: IDEA)	< 0.005	1	–	m	1
Decryption[a]	< 0.005	–	2	–	1
Total (in ms)		< 36	< 67	$< 36 + 0.005 * m$	< 2

[a] The costs per function were determined in the testbed using [10] (Section 5.2).

Table 3. Semsomm – Computational Costs of a Join.

on the messages delivered. In contrast to EVS all messages are guaranteed to be delivered to the same membership as viewed by the sender when it originally sent the message. Flush is an extension to Spread that adds the VS semantics to the the toolkit.

The system itself consists of a daemon and a library linked to the application. It is designed to support small to medium sized groups (up to 1000's members, with a large number of active members. The major advantage of this system over many others is that the price to pay for group membership changes is the lowest possible, so making it suitable for wide area settings: A simple join or leave translates into a single message. Conceptually, we use Spread as a practical realisation of the network component. The basic functionality of Semsomm is implemented as an event handling loop. Network events are generated by the Spread toolkit (that includes the Flush component for VS semantics[5]) and taken by Semsomm to be processed by the respective modules, e.g. the encryption module for bulk data encryption, currently implemented through the OpenSSL cryptographic library that provides a large selection of such algorithms [10].

5.2 Testbed

The experimental testbed consisted of three clusters each of six 500 MHz Pentium III PCs running Linux kernel version 2.2.19, and two single 800 MHz Pentium III PCs running Linux, kernel version 2.4.16. Group Members are distributed on all machines according to the simulated network topology (see test scenarios description below in Section 5.5 for details).

5.3 Costs of Cryptographic Operations

For the Diffie-Hellman key exchanges, we use $p = 1024$ and $q = 160$. Those values are believed to be a reasonable choice for key lengths [13].

[5] Flush is an extension to the Spread Wide Area Group Communication System that adds the View Synchrony semantics to the the toolkit.

Public Key signatures guarantee message origin and data authentication. We chose RSA as signature algorithm because Semsomm needs several signature verifications that should be completed as fast as possible. On the machines of our testbed 1024 bit-RSA (exponent 65557) signatures cost between 8 and 9 milliseconds and verifications between 0.4 and 0.6 milliseconds, whereas the figures for the comparable DSA are 4.5 milliseconds for a signature respectively 5.3 milliseconds for a verification.

SHA-1 is the message digest algorithm of our choice. The induced costs per hash in the order of 0.001 milliseconds are negligible in comparison to the other cryptographic operations. Similarly, the costs for symmetric encryptions and decryptions with IDEA could also be discarded as a typical data packet of size 500 Bytes can be encrypted or decrypted in 0.07 milliseconds.

Finally, we need to add the costs for a symmetric key generation that takes around 2 milliseconds on the PCs in our testbed. This operation can however be improved by hand optimised code and cryptographic hardware.

5.4 Costs of Spread

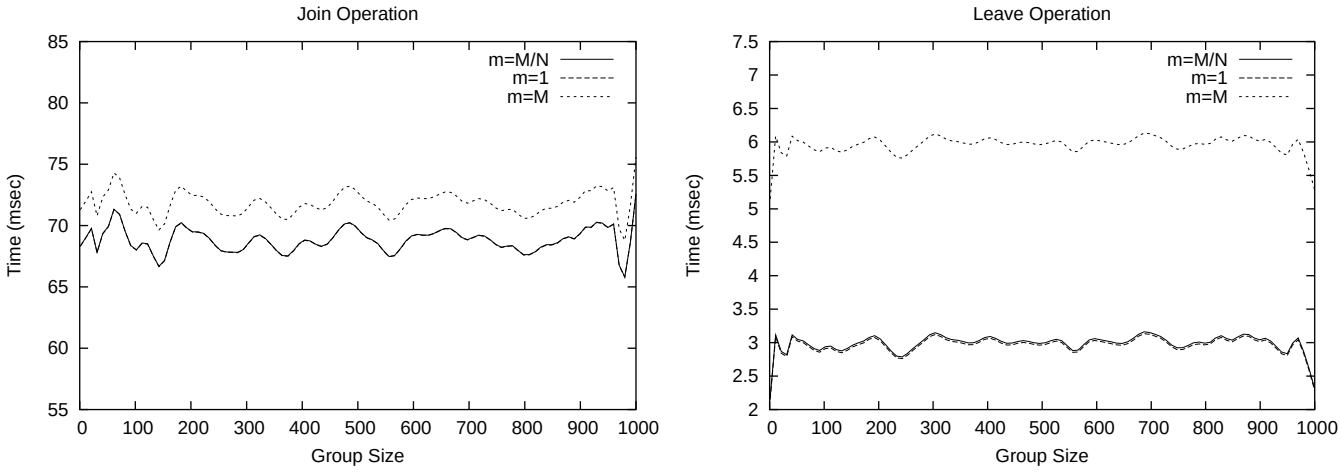
The tests performed in the testbed showed that the average cost of sending and delivering a multicast message is 0.85 milliseconds for a group of a thousand members.

The costs of the group membership service are linear in the number of members: for groups with only two members the overhead induced through Spread is 2 milliseconds, for groups with 100 members 18 milliseconds and for 1000 members 119 milliseconds.

5.5 Test Scenarios

In our simulations, we consider three cases:

1. the *optimal* case in which each intermediate node delivers exactly one member, i.e. $m = 1$,
2. the *average* case in which the members are distributed equally among the the intermediate nodes, i.e. the case where $m = M/N$,



(a) Join.

(b) Leave.

Figure 3. Performance of Basic Operations.

3. the *worst* case in which one single intermediate node delivers all the members, i.e. $m = M$.

The practical upper limit to the number of members was dictated by the group management framework that could not handle groups larger than 1000 participants[6].

Join We measure the delay for a member to join a group from the start of his first message to the key distributor till the point in time it gets the last encryption key from its intermediate node. Figure 3(a) depicts the join latency for a group of up to a hundred active users. The simulations confirm the theoretical assessment that a join operation completes in less than 70 milliseconds. The join latency induced by is thus confirmed to be independent of the group size.

The join latency increases however with the number of members per intermediate node, and there is the danger of implosion at this point.

Leave Figure 3(b) shows the average leave latency, i.e the average time for a group to establish secure group membership upon a member's leave event. In the worst case ($m = M$), the leave operation takes 6 milliseconds, in the average and best cases 3 milliseconds. Our simulations thus corroborate the theoretical evaluation: the leave latency is independent of the group size and inexpensive.

6 Conclusions

Few-to-many Semsomm is a new scalable, secure group communication solution for applications involving few senders and a large number of receivers. The approach proved to be collusion resistant. Furthermore, the computational and communication overheads depend only on the number of receivers per node. It fulfils the properties of scalability and perfect forward secrecy.

[6] We are currently designing an own simulator for larger groups to overcome these limitations.

Future work encompasses the use of our own simulator for very large groups in both the few-to-many and the many-to-many variants. On the other hand, we will deploy Semsomm to typical group application areas in self-organising networks such as service discovery protocols and multimedia streaming applications.

References

[1] IETF Working Group on Multicast Security (msec). `http://www.ietf.org/html.charters/msec-charter.html`.

[2] Canetti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M. and Pinkas, B. Multicast Security: A Taxonomy and Effi cient Constructions. In *Proceedings of INFOCOM'99*. New York, NY, USA, March 1999.

[3] Weiler, N. SEMSOMM - A Scalable Multiple Encryption Scheme for One-To-Many Multicast. In *Proceedings of IEEE WET ICE'01*. Boston, MA, USA, June 2001.

[4] Gong, L. and Shaham, N. Elements of Trusted Multicasting. In *Proceedings of IEEE International Conference on Network Protocols (ICNP'94)*. Boston, MA, USA, October 1994.

[5] Harney, H. and Muckenhirn, C. Group Key Management Protocol (GKMP) Architecture. RFC 2094, July 1997.

[6] Balenson, D. and amd Alan T. Sherman, D. M. Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization. draft-irtf-smug-groupkeymagmt-oft-00.txt, August 2000.

[7] Mittra, S. Iolus: A Framework for Scalable Secure Multicasting. In *Proceedings of ACM SIGCOMM '97*, pages 277–288. Cannes, France, September 1997.

[8] Caronni, G., Sun, D., Waldvogel, M., Weiler, N. and Plattner, B. The VersaKey Framework: Versatile Group Key Management. *IEEE JSAC*, September 1999.

[9] Setia, S., Loussih, S., Jajodia, S. and Harder, E. Kronos: A Scalable Group re-Keying Approach for Secure Multicast. In *Proceedings of IEEE S&P'00*. Oakland, CA, USA, May 2000.

[10] OpenSSL Project. `http://www.openssl.org`.

[11] Weiler, N. SEMSOMM. Technical Report 174, TIK, ETH Zürich, Switzerland, April 2003.

[12] Amir, Y. and Stanton, J. The Spread Wide Area Group Communication System. Tech. Rep. CNDS 98-4, 1998.

[13] Lenstra, A. K. and Verheul, E. R. Selecting Cryptographic Key Sizes. In *Public Key Cryptography*, pages 446–465. 2000.