ABSTRACT

Title of Thesis:	AUTHENTICATED KEY AGREEMENT IN DYNAMIC GROUPS
Degree candidate:	Arvind Mani
Degree and year:	Master of Science, 2001
Thesis directed by:	Professor John S. Baras Department of Electrical and Computer Engineering

Multicast security poses interesting challenges in the area of key management. Designing a good protocol for key agreement in dynamic multicast groups involves a thorough understanding of the trade-offs that exist among storage, communication and computation overhead.

The contribution of this thesis is a verifiable protocol for authenticated key agreement based on a distributed key generation scheme. The underlying key generation scheme has shown promise in being natural for collaborative group applications. To handle group membership changes in dynamic groups, an auxiliary key agreement protocol is introduced. The auxiliary protocol re-uses contributions to the key in the previous round, to form the new key. The key shares of the members contributing fresh values in the current round are more susceptible to discovery by colluding group members (not outsiders). The auxiliary protocol does not introduce any other security weakness. A protocol that starts from the scratch on membership change is going to be expensive, slow and unsuitable for most applications.

We use auxiliary key agreement protocol in conjunction with the well-known Logical Key Tree (LKT) to localize the effect of membership change. The protocol does not use time-stamps, and makes minimal use of public-key based computation.

AUTHENTICATED KEY AGREEMENT IN DYNAMIC GROUPS

by Arvind Mani

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2001

Advisory Committee:

Professor John S. Baras, Chair
Professor Virgil Gligor
Professor William Gasarch

ACKNOWLEDGEMENTS

I am extremely grateful to professor J. S. Baras for the technical, financial and moral support he provided throughout my study. He has opened my mind to research in the field of network security. I am also very grateful to the other members of the dissertation committee - professors V. Gligor, and W. Gasarch for their support and feedback. I had the opportunity to work with them and it has helped my research work immensely.

I would like to acknowledge R. Poovendran for his contribution to the thesis. I would like to thank A. Khalili for his comments. I would also like to thank V. Bharadwaj, K. Manousakis, H. Khurana, M. Rabi and S. Parthasarathy. I am also grateful to Ms. D. Hicks and Ms. A. Kirlew.

The research reported in this thesis was supported by the Center for Satellite and Hybrid Communication Networks (CSHCN), NASA cooperative agreement NCC3528; the Advanced Telecommunication Information Distribution (ATIRP) cooperative agreement QK9931; and Lockheed Martin Global Telecommunication (LMGT).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Group communications have recently become the focal point of research in the area of network security. Secure group communication is the key to success in diverse applications - flexible programming in satellite television to wireless sensor networks deployed in hostile environments. As part of the issues involved with securing multi-party communications, key management has received particular attention due to the vulnerabilities inherent in group communications that have no counterpart in the unicast case. Securing group communication, i.e., ensuring the integrity and confidentiality of the data communication among group members requires security services such as authentication of group members for access control, encryption of the data communication, and source authentication of the data. All these services require the use of symmetric or public-key encryption, that in turn require keys. In most existing group communication protocols, key management is carried out by the group controller. This is a major issue in certain applications, where a group controller cannot be naturally selected. Another challenging issue is the need for access control. This is a function that is hard to de-centralize without creating chaos. A trusted third-party is also required to implement many protocols that require the use of certificates. This trusted third party was required to be online at all times for verifying certificates piggybacked with the data. This thesis presents an authenticated key agreement protocol that takes the first step toward a fully distributed group key management protocol. The protocol requires the presence of a third-party for access control but it limits the use of certificates to the time when a group member is admitted into the group. This has efficiency considerations as well.
It is important to realize that important differences exist between the two-party and multi

party cases. Protocol efficiency is an important concern due to the number of participants. Also two-party communication can be viewed as a discrete phenomenon - it starts, lasts for a while and then ends. Group communication is more complicated - it starts, members join and leave and there may not be a well-defined end. See a discussion of these issues in [Poo99], [HT99], [JV96] and [BWM98].

This thesis describes an authenticated key agreement protocol suitable for dynamic groups. Examples include co-operating content providers (television programming), audio and video conferencing and more generally, collaborative applications of all kinds. Authenticated key agreement in dynamic groups, where many or all entities contribute to the session key (used later for data encryption), and do not reveal their contribution to the key is still considered an open problem. Source authentication for data communication (that occurs after the group key is established) is not addressed in this thesis. A scheme for source authentication of data (TESLA system) in group communications, is described in [PCST01]. We also do not deal with authorization issues, e.g., some applications have restrictions on which group members can send messages, others can only read messages. Authorization is handled by the Group Controller using Access Control Lists (ACLs), capability lists or certificates with attributes.

Authenticated key establishment protocols are designed to provide two or more parties communicating over an open network, with a shared secret key that may be subsequently used to achieve data confidentiality or data integrity. Key establishment protocols can be roughly classified into two categories: *key transport* protocols and *key agreement* protocols. In key transport protocols, a key is created by a central entity and securely transmitted to the other entities, while in key agreement protocols more than one party contributes information (**fractional keys**), that is used to derive the shared secret key. An important problem in certain applications is to find a key generation scheme that allows a set of members to jointly generate keys *without* having to expose their individual contributions. These issues are clearly explained in [HT99] and [CGI+99].

Most of the currently available key generation schemes are based on the generalization of the Diffie-Hellman (DH) key exchange protocol for group keying. The DH keys are secret keys. The generalized Diffie-Hellman scheme, that has been extensively used in recent group communication protocols, involves several exponentiation, and in the best schemes, the computations scale linearly as a function of group size. These methods rely on the assumed

cryptographic hard problem - performing discrete logarithm in finite fields. In contrast, the scheme proposed by Poovendran in [Poo99] does not depend on the computational difficulties of integer factoring or on discrete logarithm, instead it makes use of one-time pads. At the heart of this novel key agreement scheme is a way for the key generating members to **locally** compute one-time pads, and to compute a common group parameter. This group parameter will be used to remove the effect of the pads. Each group member masks his key-share with its one-time pad, and transmits this combined value to all other group members. This allows exchange of key-shares without exposing them. The padded shares are combined. The group parameter (called **binding parameter**) is a combination of all the pads, and can now be used to remove the *combined padding effect* and extract the common secret. If the key generation mechanism can provide uniformly distributed variables over an interval of interest, this scheme will be resistant to attacks from individual members or up to $(N-2)$ collaborating members. In his dissertation, Poovendran [Poo99] explicitly states that his key generation scheme has not addressed authentication issues. We assume enough trust in group members not to reveal the common secret. The objective is to prevent one or more group members from learning the contribution of another group member. Such a threat model is appropriate when we talk of applications such as delivering television programming, where competing content providers collaborate to provide common programming. Here, the threat to one content provider is to prevent a collaborator (also a content provider) from learning its contribution to the key. The motivation to keep the contribution secret arises from the fact that the content provider may want to re-use its contribution to form a secret key with another set of collaborators for delivering a different set of programs. The resulting keys would be different.
In this thesis we focus on key agreement with authentication, based on the key generation scheme proposed by Poovendran [Poo99]. First, we note the drawbacks of key transport (mentioned in [AST00]) in the context of group communication:

- The trusted third party that generates and distributes group keys is a single point of failure and is also a likely performance bottleneck. It also presents an attractive target.
- Group communications spanning multiple and independent administrative domains may not have a central party that they all trust (or do not want to) to generate group keys.
- Some groups (such as *ad hoc* networks) are highly dynamic and no group member can be

assumed present all the time. There is also no obvious trusted party that can be identified in ad hoc groups.

- Achieving *perfect forward secrecy* and *resistance to known-key attacks* in an efficient manner is difficult in a centralized key distribution setting. This is because a lot of keys have to be generated and distributed.

Although we argue in favor of contributory key agreement for group communications, we recognize the need for a central point of control for group membership operations such as adding and deleting members, i.e. centralized access control. This type of a role (group membership controller) serves only to prevent chaos. Access control is orthogonal to the issue of key establishment and is largely a matter of policy.

1.1 Security Goals and Definitions

We now reproduce formal definitions of terms taken from [AST00], [BWM98] and [LMQ+98]. These terms will appear in the text of this thesis.

Definition 1.1.1 *A* **key agreement protocol** *is a key establishment technique where a shared secret secret (possibly the key itself) is derived by two or more specified parties as a function of information contributed by each of these. No party can pre-determine the value of this secret.*

Definition 1.1.2 *A key agreement protocol is* **contributory** *if each party equally contributes to the key and guarantees its freshness.*

For example, according to this definition, the basic two-part Diffie-Hellman protocol is contributory. On the other hand, ElGamel one-pass protocol is not contributory, as only one of the parties contributes a fresh exponent.

Definition 1.1.3 *Let R be a $n-$ party key agreement protocol, M be the set of protocol parties and let S_n be a secret key jointly generated as a result of R. We say that R provides* **implicit key authentication** *if each $M_i \in M$ is assured that no party $M_q \notin M$ can learn the key S_n (unless aided by a dishonest $M_j \in M$). The property of implicit key authentication does not necessarily mean that M_i is assured that every $M_j \in M, i \neq j$ actually possesses the key.*

Definition 1.1.4 *A key agreement protocol that provides implicit key authentication to all parties is called an* **authenticated key agreement (AK)** *protocol.*

Definition 1.1.5 *A key agreement protocol is said to provide* **key confirmation** *if an entity is assured that all other parties actually have possession of a particular shared secret key.*

Definition 1.1.6 *If a key agreement protocol provides implicit key authentication and key confirmation, then it is said to provide* **explicit key authentication**.

Definition 1.1.7 *A key agreement protocol that provides explicit key authentication to all participating entities, is called a* **authenticated key agreement with confirmation (AKC)** *protocol.*

The authenticated key agreement protocol presented in this thesis does not provide Key confirmation.

Definition 1.1.8 *If each run of the key agreement protocol produces a unique secret key, called a session key, then the protocol is said to have* **known-key security**. *Session keys are desirable in order to limit the amount of data available for cryptanalytic attack, and to limit exposure in the event of session key compromise.*

Definition 1.1.9 *If long-term private keys of one or more participating entities are compromised, the secrecy of the previous session keys established by honest entities is not affected, then the protocol has* **perfect forward secrecy**.

Definition 1.1.10 *If an entity A's private key is compromised, an adversary that knows this value can impersonate A, since it is precisely this value that identifies A. If this does not however allow the adversary to impersonate other entities to A, then the protocol is immune to* **key-compromise impersonation**.

Definition 1.1.11 *If an entity B cannot be coerced into sharing a key with entity A without B's knowledge (when B believes the key is shared with some entity $C \neq A$), and A (correctly) believes the key is shared with B, then the protocol is resistant to* **unknown key share** *attack.*

Definition 1.1.12 *Let R be a n-party key agreement protocol and M be a set of protocol parties. We say that R is a* **complete group key authentication** *protocol*

if,$\forall i,j\ (0 < i \neq j \leq n)$ *, M_i and M_j compute the same key $S_{i,j}$* **only if** *$S_{i,j}$ has been contributed to by every $M_p \in \mathcal{M}$. (Assuming that M_i and M_j have the* **same view** *of the group membership.*

The following security goals and attributes are reproduced from [BWM98].

1.1.1 Fundamental Security Goals

- Implicit key authentication
- Explicit key authentication

1.1.2 Desirable Security Attributes

- Known-key security
- Perfect forward secrecy
- Key-compromise impersonation
- Unknown key share

All of these are necessary to achieve resistance to *active* attacks.

1.1.3 Desirable Performance Attributes

- Minimal number of passes - the number of messages exchanged
- Low communication overhead - total number of bits transmitted
- Low computation overhead - number and complexity of arithmetical operations required
- Possibility of pre-computation - minimize on-line computational overhead

1.1.4 Other Desirable Attributes

- Anonymity of the entities participating in a run of the protocol
- Role symmetry - the messages transmitted must have the same structure

- Non-interactiveness - the messages transmitted between two entities are independent of each other
- Non-reliance on time-stamping since it is difficult to implement securely in practice
- Non-reliance on encryption

1.2 Contributions

This thesis makes the following technical contributions:

1. It provides an authenticated key agreement protocol based on the key generation scheme proposed by Poovendran [Poo99]. The authentication protocol is then analyzed using BAN [BAN96] logic to verify the beliefs held by the group members at the end of the initialization phase and at the end of each re-key. It brings out the assumptions that are central to the protocol. Using BAN logic helps us avoid unnecessary computation and components in the protocol messages, thereby reducing computational and message overhead.

2. The scheme proposed by Poovendran [Poo99] in its original form is unsuitable for the functioning of dynamic groups. The protocol re-starts from the initialization phase in the event of a membership change such as a member joining the group. The initialization phase is expensive in terms of time and computation required, and is feasible only if it is performed once - during group initialization. Therefore, we add an auxiliary protocol (used in [AST00] and [STW00]) to be used during member join, leave or revocation. This protocol lends itself to easy implementation in the presence of an initiator or a Group Controller (GC). There should be a separate protocol or a standard way to choose the GC. This is already assumed in all current protocols and is not unreasonable.

3. Finally, the thesis addresses the issue of scalability in large dynamic groups by using the well-know logical key tree [WGL00] framework. We describe an of the protocol suitable for satellite networks.

1.3 Organization

In the second chapter, we present the basic Diffie-Hellman (DH) key exchange protocol. Then we review schemes that extend the basic DH to provide authentication in the multi-party case. These are collectively called Group Diffie-Hellman (GDH) protocols. We also present a short introduction to BAN logic. In the third chapter, the key generation scheme proposed by Poovendran [Poo99] is described. We study the scheme to understand the advantage over existing multi-party schemes, and also identify certain shortcomings. We address these issues in the fourth chapter by listing the requirements for an authenticated key agreement protocol based on Poovendran's key generation scheme. We then use BAN logic to analyze the complete protocol. In the fifth chapter, we construct an efficient protocol suitable in certain broadcast mediums (satellite communication, within a single LAN, wireless) that evolved from the authenticated key agreement protocol presented in the fourth chapter. We present our conclusions in the sixth chapter and discuss future research directions.

Chapter 2

Introduction to Key Agreement and Formal Analysis of Authentication

In this chapter, we review different key agreement protocols. We first describe the Diffie-Hellman two-party key exchange protocol that forms the basis for most of the reviewed work on contributory secret key agreement in group communication. We also review some group key generation schemes that do not provide key authentication. In this category, we cover Group Diffie-Hellman, Group ElGamal and Group RSA. The latter two are used to generate shared public keys. Then we briefly describe some authenticated multi-party key agreement protocols ([STW00] and [JV96]) and their notable features. Finally, we present the basics of BAN logic [BAN96] used to formally analyze our authenticated key agreement protocol.
First, we highlight the factors that influence the design of a key management scheme and the desirable properties of a multicast key management scheme. This enables us to evaluate the strengths and weaknesses of the protocols discussed. These factors govern the choice of the key management scheme for a given scenario. It is important to note that these features have more to do with performance and suitability of a scheme. The security attributes described in the first chapter are more fundamental and must be satisfied by all key management schemes. This list is reproduced from [Poo99].

2.1 Factors influencing the design of a Key Management Scheme

1. The nature of the application affects the possible type of encryption algorithm to be used, and the length of the key that can be supported by an end user.

2. The cost of setting up and initializing the entire system, such as selection of a Group Controller (GC), initial key distribution, membership change and group announcement.

3. Administrative policies, such as those defining members that are authorized to generate keys and key shares.

4. Required level of performance parameters, such as group dynamics, key generation rates and session sustainability.

5. Required additional external support mechanism, such as the availability of a Certificate Authority (CA) or a server to perform access control.

2.2 Desirable Properties of a Multicast Key Management Scheme

In additions to the factors mentioned above, a multicast key management scheme needs to exhibit the following desirable properties:

1. Ability to handle membership changes in a scalable manner. This is important since the whole group must share a single-session encryption key. The communication integrity in the presence of membership changes implies the ability of the group to update the session key and distribute it to the valid members with possible back traffic protection.

2. Ability to prevent user collusion. This is important since a subset of members or the deleted members should not be able to collaborate and construct the keys or key shares of other members or the future group keys.

3. Ability to handle inter-domain issues. This is important since some of the members may belong to more than one group and may need to communicate across the groups.

2.3 Diffie-Hellman

Diffie-Hellman gets its security from the difficulty of calculating discrete logarithms in a finite field, as compared with the ease of calculating exponentiation in the same field. Diffie-Hellman can be used for agreeing on a shared secret, that may later be used as a secret key, or the secret key could be the result of a cryptographic transformation on the shared secret. The protocol description is taken from [Sch96].

Alice and Bob agree on a large prime n, and g, such that g is primitive mod n. These two integers do not have to be secret; Alice and Bob can agree to them over some insecure channel. They can even be common among a group of users.

For example, lets say that Bob and Alice want to agree on a secret key over an insecure channel (assume passive adversaries only). Then the protocol proceeds as follows:

1. Alice chooses a random large integer x and sends Bob

$$X = g^x \text{mod } n$$

2. Bob chooses a random large integer y and sends Alice

$$Y = g^y \text{mod } n$$

3. Alice computes

$$k = Y^x \text{mod } n$$

4. Bob computes

$$K^{'} = X^y \text{mod } n$$

Both k and $K^{'}$ are equal to $g^{xy} \text{mod } n$. No one listening on the channel can compute that value; they only know n, g, X, and Y, and unless they can compute the discrete logarithm and recover

x or y, they do not solve the problem. So, k is the secret key that both Alice and Bob computed independently.

The choice of n can have a substantial impact on the security of this system. The number $(n-1)/2$ should also be prime. And most important, n should be large: The security of the system is based on the difficulty in factoring numbers the same size as n. As far as g is concerned, any g that is primitive mod n will do. g does not have to be primitive, it just has to generate a large subgroup of the multiplicative group mod n. Generally g is chosen to be a single- digit integer.

2.4 Generation of Group Keys

2.4.1 Group Diffie-Hellman

There are three versions of the multi-party group DH. We will describe the basic Generalized DH.1 (GDH.1). All three algorithms consists of two stages called *up-flow* and *down-flow*. In the up-flow stage, members collect the contributions from the other members and propagate to the next highest indexed member with modification in message sequence. The message exchange for the up-flow is given by:

$$M_i \rightarrow M_{i+1} : \{g^{\pi(a_l | l \in [1,j]) | j \in [1,i]}\}$$

For example, member M_5 receives:

$$\{g^{a_1}, g^{q_1 a_2}, g^{a_1 a_2 a_3}, g^{a_1 a_2 a_3 a_4}\}$$

and forwards

$$\{g^{a_1}, g^{a_1 a_2}, g^{a_1 a_2 a_3}, g^{a_1 a_2 a_3 a_4}, g^{a_1 a_2 a_3 a_4}\}$$

to member M_6. In the up-flow procedure, each member needs to perform one exponentiation. From the indices of the message, member M_i sends i messages to member M_{i+1}. The last member of the group, M_N computes the group key $K = g^{a_1 a_2 a_3 \ldots a_N}$.

At this stage, member M_N can broadcast the session key value to all the members. Instead of broadcasting the K to all the members, in order to provide the authentication part, the key

scheme has the down-flow part as follows:

$$M_{N-i} \rightarrow M_{N-i+1} : \{g^{\pi(a_l | l \notin [i,j]) | j \in [1,i]}\}$$

In the down-flow stage, i exponentiations are performed by M_i. One of these enables M_i to compute K, and the rest of the exponentiations ensure that the rest of the group members will eventually receive appropriate shares. In order to illustrate this case, we assume that the group size $N = 6$. In this example, the last member M_6 sends M_5 the message

$$\{g^{a_6}, g^{a_1 a_6}, g^{a_1 a_2 a_6}, \ldots, g^{a_1 a_2 a_3 a_4 a_6}\}$$

Using $g^{a_1 a_2 a_3 a_4 a_6}$, it computes $(g^{a_1 a_2 a_3 a_4 a_6})^{a_5} = g^{a_1 a_2 a_3 a_4 a_5 a_6}$. Member M_5 raises the rest of the terms to the power a_5 and sends it to M_4. This process is carried out by each member $M_i (1 \leq i \leq N)$ with appropriate modifications until M_1 computes the session key. There are $\mathcal{O}(\mathcal{N}^{\in})$ messages and exponentiations for such a process.

2.4.2 Generating Group ElGamal Keys

Each key generating member is associated with an individual ElGamal public key. The private keys of all the members were added to generate the group private key. The group public key is then the product of the individual public keys. Computational steps are summarized below.

1. M_i randomly chooses $1 \leq a_i \leq q-1$ (private key) and computes the public key g_i^a.

2. M_i sends a_i to other members as

$$M_i \rightarrow M_j (1 \leq j \leq N; j \neq i) : a_i.$$

3. Each M_i computes the group private key as

$$a = \sum_{i=1}^{N} a_i \; mod \; p$$

4. The group key is the product of the individual public keys modulo p. If we denote the group public key by K, it is given by

$$K = \pi_{i=1}^{N} g^{a_i} = g^{\sum_{i=1}^{N} a_i}$$

Although this method has less computation, it exposes the individual private keys of the generating members.

2.4.3 Efficient Generation of Shared RSA Keys

This section describes efficient protocols for a number of parties to generate an RSA modulus $N = pq$ where p, q are prime. At the end of the computation the parties are convinced that N is indeed a product of two large primes. However, none of the parties know the factorization of N. The parties can then proceed to compute a public exponent e and shares of the corresponding private exponent. The techniques include a distributed primality test. The test enables two (or more) parties to verify that a random integer N is a product of two large primes without revealing the primes themselves. Threshold cryptography is a concrete example where shared generation of RSA keys is very useful.
We present a high-level overview of the protocol. For details refer to [BF97]. Let k parties wish to generate a shared RSA key. In other words, they want to generate an RSA modulus $N = pq$ and a public/private key pair of exponents e, d where $e \cdot d = 1 \; mod \; \varphi(N)$. The factors p and q should be at least n bits each. At the end of the computation N and e are public, and d is shared between the k parties in a way that enables threshold decryption. All parties should be convinced that N is indeed a product of two primes, but no coalition of at most $t = \lfloor \frac{k-1}{2} \rfloor$ parties should have any information about the factors of N.

1. **pick candidates**. The following two steps are repeated twice.

   (a) **secret choice:** Each party i picks a secret $n-$bit integer p_i and keeps it secret.

   (b) **trial division:** Using a private distributed computation the k parties determine that $p = p_1 + \ldots + p_k$ is not divisible by any prime less than some bound B_1. If this fails repeat previous step.

Denote the secret values picked at the first iteration by $p_1, \ldots, p_k$, and at the second iteration by $q_1, \ldots, q_k$.

2. **compute N:** Using a private distributed computation the k parties compute

$$N = (p_1 + \ldots + p_k) \cdot (q_1 + \ldots + q_k)$$

Other than the value of N, this steps reveals no further information about the secret values $p_1, \ldots, p_k$ and $q_1, \ldots, q_k$. Now that N is public, the k parties can perform further trial divisions and test that N is not divisible by small primes in the range $[B_1, B_2]$ for some bound B_2.

3. **primality test:** The k parties engage in private distributed computation to test that N is indeed the product of two primes. If the test fails, then the protocol is restarted from step 1. The primality test protocol is $k-1$ private and applies whenever two (or more) parties are involved.

4. **key generation** Given a public encryption exponent e, the parties engage in a private distributed computation to generate a shared secret decryption exponent d.

2.5 Authenticated Multi-party Key Agreement Protocols

2.5.1 Gene Tsudik, Michael Steiner, et al.

The authors develop a multi-party extension to the basic two-party Diffie-Hellman key agreement protocol that provides key authentication. They do not use additional cryptographic tools (e.g., symmetric encryption or signatures) other than those necessary for plain DH key agreement. The key concept is that it should be possible to base all the security properties of a given protocol on a *single* hard problem such as the Discrete Diffie-Hellman (DDH) problem in prime-order subgroups. Please refer to the actual protocol description in [STW00] and [AST00].

Let $\{M_1, \ldots, M_n\}$ be the set of group members that wish to share a key S. The protocol proceeds in two stages like all Group Diffie Hellman protocols. In the first stage (n - 1 rounds), contributions are collected from individual group members, and, then in the last round, group

keying material is broadcast. For example, in the above case, in the first stage, each M_i selects a random number r_i; it sends to the next member in the group M_{i+1}, a set of $i+1$ values - $\{\alpha^{\frac{r_1 \cdots r_i}{r_j}} | j \in [1,i]\}$, $\alpha^{r_1 \cdots r_i}$. In the last round, M_n selects its random number r_n, and broadcasts, $\{\alpha^{\frac{r_1 \cdots r_n}{r_i}} | i \in [1,n[\}$. The shared secret key $S = \alpha^{r_1 \cdots r_n}$. The reason why S is not broadcast in the last round, is to prevent outsiders from learning the key.

This basic protocol can be easily amended to provide implicit key authentication in an efficient manner. This variation called A-GDH.2 differs from GDH.2 only in the last round. A basic assumption is that M_n shares or is able to share with each M_i a distinct secret K_{in}. For example, $K_{in} = F(\alpha^{x_i \cdot x_n} mod\ p)$ with $i \in [1, n-1]$, where x_i is a secret long term exponent selected by every M_i $(1 \leq x_i \leq q-1)$ and $\alpha^{x_i} mod\ p$ is the corresponding long-term public key of M_i.

The authors consider the above authentication as relatively *weak* since the key is not directly authenticated between an arbitrary M_i and M_j $(i \neq j)$. Also no one can be sure of another member's participation. They go on to describe variants of the protocol that provide strong authentication and another that minimizes communication overhead.

The authors further make another important observation. We require *key independence* and therefore need to compute a new (authenticated and contributory) key in the face of membership changes. This problem can be solved in two ways - start from the scratch or use previous information to save computation. They note that the first approach is expensive, unscalable and utterly unsuitable for environments with frequent membership changes. We have this observation in mind, when we design our auxiliary protocol.

2.5.2 Mike Just and Serge Vaudenay

This paper deals with key agreement protocols based on Diffie-Hellman that use public key techniques. It does not require the presence of an on-line trusted third-party. They first present a Diffie-Hellman based three-pass protocol that provides key authentication, key confirmation and forward secrecy. Then they extend the two party case to obtain a multi-party key agreement model. The multi-party protocol created by using this model and their specific two-party protocol reduces the amount of communication required between participants. For details refer to [JV96].

Below is their construction of a multi-party key agreement protocol MP from two-party key

agreement protocol P. Assume all users $u_1, u_2, \ldots, u_t$ are arranged on a ring and we will consider indices of u_i to be taken between 1 to t modulo t. Each pair (u_i, u_{i+1}) processes protocol P to obtain a session key K_i. Each u_i computes $W_i \equiv \frac{K_i}{K_{i-1}}$. Upon receiving the broadcast from other users, u_i computes the key $K \equiv K_{i-1}^t W_i^{t-1} W_{i+1}^{t-2} \ldots W_{i-2} \equiv K_1 K_2 \ldots K_t$. Equivalently, we can use $W_i = K_i - K_{i-1}$ and $K = K_1 + \ldots + K_t$. This is much cheaper than multiplication and has some practical value. We do not show the three-pass two-party Diffie-Hellman protocol for space considerations.

2.6 Formal Analysis of Authentication Protocols

We present a section on BAN logic taken from [BAN96]. Formalism enables us to express protocol assumptions and the steps with a precision, not otherwise possible. Authentication would be straight- forward in a sufficiently benign environment. Such cannot usually be assumed, and it is particularly necessary to take precautions against confusion caused by re-issue of old messages. Specifically, one must ensure that a replay cannot force the use of an old and possibly compromised secret. The logic helps us to explain the protocol step by step, with all initial assumptions made explicit and with final states clearly set out.
Using BAN we can analyze our protocol formally to see if it meets our objectives, the assumptions it makes, and if we are including components that are not necessary to strengthen the protocol, or if we are encrypting] components that could be sent in the clear without weakening the protocol.
In a later section, we show how the logic is able to help us validate the authenticated key agreement protocol. It is important to note that certain aspects of authentication protocols are ignored in this treatment. It does not consider errors introduced by concrete implementations of a protocol, such as deadlocks or inappropriate use of cryptosystems. Furthermore, while it allows the possibility of hostile intruders, there is no attempt to deal with the authentication of an untrustworthy principal, nor to detect weaknesses of encryption schemes or unauthorized release of secrets. The study concentrates on the beliefs of trustworthy parties involved in the protocols and on the evolution these beliefs as a consequence of communication.

2.6.1 The Formalism

Authentication protocols are typically described by listing the messages sent between the principals, and by symbolically showing the source, the destination, and the contents of each message. This conventional notation is not convenient for manipulation in a logic, since we wish to attach exact meanings to each part of each message and these meanings are not always apparent from the data contained in the messages. In order to introduce a more useful notation whilst preserving correspondence with the original description of the protocols, we transform each message into a logical formula according to the notation explained in [BAN96]. Then we annotate the idealized protocol with assertions as illustrated in the paper. An assertion usually describes beliefs held by the principals at the point in the protocol where the assertion is inserted.

2.6.2 Basic Notation

A brief overview of the notation used in the original paper, is presented. Typically, the symbols A, B, and S denote specific principals; K_a, K_b, and K_s denote specific public keys, and K_a^{-1}, K_b^{-1}, and K_s^{-1} denote the corresponding secret keys; N_a, N_b, and N_c denote specific statements. The symbols P, Q, and R range over principals; X and Y range over statements; K ranges over encryption keys. The following constructs are used:

Table 2.1: Notation used in BAN logic

$P \mid\equiv X$	P *believes* X, or P would be entitled to believe X. Principal P may act as though X is true.
$P \triangleleft X$	P *sees* X. Someone has sent a message containing X to P, who can read and repeat X.
$P \mid\sim X$	P *once said* X. The principal P at some time sent a message including the statement X. It is not known whether the message was sent long ago or during the current run of the protocol, but it is known that P believed X when he sent the message.

$P \mid\Rightarrow X$	P has *jurisdiction* over X. The principal P is an authority on X and should be trusted on this matter. This construct is used when a principal has delegated authority over some statement. For example, encryption keys need to be generated with care, and in some protocols certain servers are trusted to do this properly. This may be expressed by the assumption that the principals believe that the server has jurisdiction over the statements about the quality of keys.
$\#X$	The formula X is *fresh*, that is X has not been sent in a message at any time before the current run of the protocol. This is usually true for *nonces*, that is expressions generated for the purpose of being fresh. Nonces commonly include a time-stamp or a number that is used only once, such as a sequence number.
$P \overset{K}{\longleftrightarrow} Q$	P and Q may use the *shared key* K to communicate. The key K is good, in that it will never be discovered by any principal except P or Q, or a principal trusted by either P or Q.
$\overset{K}{\mapsto} P$	P has K as a *public key*. The matching *secret key* (the inverse of K, denoted K^{-1}) will never be discovered by any principal except P, or a principal trusted by P.
$P \rightleftharpoons Q$	The formula X is a *secret* known only to P and Q, and possibly to principals trusted by them. Only P and Q may use X to prove their identities to one another. Often, X is fresh as well as secret. An example of a shared secret is a password.
$\{X\}_K$	This represents the formula X encrypted under the key K. Formally, $\{X\}_K$ is an abbreviation for an expression of the form $\{X\}_K$ *from* P. We make the realistic assumption that each principal is able to recognize and ignore his own messages; the originator of each message is mentioned for this purpose.
$\langle X\rangle_Y$	This represents X combined with the formula Y; it is intended that Y be a secret, and that its presence prove the identity of whoever utters $< X >_Y$. In implementations, X is simply concatenated with the password Y; our notation highlights that Y plays a special role, as proof of origin for X. The notation is intentionally reminiscent of that for encryption, which also guarantees the identity of the source of a message through knowledge of a certain kind of a secret.

Logical postulates are presented below:

- The *message-meaning* rules concern the interpretation of messages. Two of the three concern the interpretation of messages with secrets. They all explain how to derive beliefs about the origin of messages.

  For shared keys:

$$\frac{P \mid\equiv\ Q\ \stackrel{K}{\longleftrightarrow}\ P,\ P\ \triangleleft\ \{X\}_K}{P \mid\equiv\ Q \mid\sim\ X}$$

  That is, if P believes that the key K is shared with Q and sees a message X encrypted under K, then P believes that Q once said X.

  Similarly for public keys,

$$\frac{P \mid\equiv \stackrel{K}{\mapsto}\ Q,\ P\ \triangleleft\ \{X\}_{K^{-1}}}{P \mid\equiv\ Q \mid\sim\ X}$$

  That is, if P believes that the secret Y is shared with Q and sees $\{X\}_{K^{-1}}$, then P believes that Q once said X.

  In real life the decryption of a message to yield a content says, only that the content was produced at some time in the past; we have no idea whether it is new or the result of a replay.

- The *nonce-verification* rule expresses the check that a message is recent, and hence that the sender still believes in it:

$$\frac{P \mid\equiv\ \#X,\ P \mid\equiv\ Q \mid\sim\ X}{P \mid\equiv\ Q \mid\equiv\ X}$$

  That is, if P believes that X could have been uttered only recently and that Q once said X, then P believes that Q believes X. For the sake of simplicity, X must be "clear text." that is, it should not include any sub-formula of the form $\{Y\}_K$. (When this restriction is not met, we can conclude only that Q has recently said X.)

- The *jurisdiction* rule states that if P believes that Q has jurisdiction over X then P trusts Q on the truth of X:

$$\frac{P \mid\equiv\ Q \mid\Rightarrow\ X,\ P \mid\equiv\ Q \mid\equiv\ X}{P \mid\equiv\ X}$$

- A necessary property of the belief operator is that P believes a set of statements if and

only if P believes each individual statement separately.

$$\frac{P \mid\equiv\ X, P \mid\equiv\ Y}{P \mid\equiv\ (X,Y)} \quad \frac{P \mid\equiv\ (X,Y)}{P \mid\equiv\ X} \quad \frac{P \mid\equiv\ Q \mid\equiv\ (X,Y)}{P \mid\equiv\ Q \mid\equiv\ X}$$

- A similar rule applies to the operator $\mid\sim$:

$$\frac{P \mid\equiv Q \mid\sim (X,Y)}{P \mid\equiv Q \mid\sim X}$$

  Note that if $P \mid\equiv Q \mid\sim X$ and $P \mid\equiv Q \mid\equiv Y$, it does *not* follow that $P \mid\equiv Q \mid\sim (X,Y)$, since this would imply that the two parts X and Y were uttered at the same time.

- If a principal sees a formula then he also sees its components, provided he knows the necessary keys:

$$\frac{P \triangleleft (X,Y)}{P \triangleleft X} \quad \frac{P \triangleleft \langle X \rangle_Y}{P \triangleleft X} \quad \frac{P \mid\equiv Q \overset{K}{\longleftrightarrow} P,\ P \triangleleft \{X\}_K}{P \triangleleft X} \quad \frac{P \mid\equiv \overset{K}{\mapsto} P, P \triangleleft \{X\}_{K^{-1}}}{P \triangleleft X}$$

- If one part of a formula is known to be fresh, then the entire formula must also be fresh:

$$\frac{P \mid\equiv \#X}{P \mid\equiv \#X, Y}$$

  Similar rules can be written, for instance to show that if X is fresh, then $\{X\}_K$ is fresh.

- The same key is used between a pair of principals in either direction.

$$\frac{P \mid\equiv R \overset{K}{\longleftrightarrow} R'}{P \mid\equiv R' \overset{K}{\longleftrightarrow} R}$$

- Similarly, a secret can be used between a pair of principals in either direction.

$$\frac{P \mid\equiv R \overset{X}{\rightleftharpoons} R'}{P \mid\equiv R' \overset{X}{\rightleftharpoons} R}$$

2.6.3 Delegation

Delegation statements typically mention one or more variables. For example, principal A may let server S generate an arbitrary key for A and B. We can express this as

$$A \mid\equiv S \mid\Rightarrow A \overset{K}{\rightleftharpoons} B$$

Here the key K is universally quantified, and we can make explicit this quantification by writing

$$A \mid\equiv \forall K.\ (S \mid\Rightarrow A \overset{K}{\rightleftharpoons} B)$$

2.6.4 Idealized Protocols

Each protocol step is typically written in the form

$$P \rightarrow Q : message$$

This denotes that the principal P sends the message and that the principal Q receives it. The message is presented in an informal notation designed to suggest the bit-string that a strong implementation would use. Unfortunately, this presentation is often ambiguous and obscure in its meaning, and is not an appropriate basis for formal analysis.
Therefore, we transform each protocol step into an idealized form. A message in the idealized protocol is a formula. For instance, in the literature we may find the protocol step

$$A \rightarrow B : \{A, K_{ab}\}_{K_{bs}}$$

This may tell B, who knows the key K_{bs} that K_{ab} is a key to communicate with A. This step should then be idealized as

$$A \rightarrow B : \{A \overset{K_{ab}}{\rightleftharpoons} B\}_{K_{bs}}$$

When the message is sent to B, we may deduce that the formula

$$B \triangleleft \{A \overset{K_{ab}}{\rightleftharpoons} B\}_{K_{bs}}$$

holds, indicating that the receiving principal becomes aware of the message and can act upon it.

In this idealized form, we omit parts of the message that do not contribute to the beliefs of the recipient. In particular, we remove hints that are added to an implementation to allow it to proceed in a timely fashion, but whose presence would not affect the result of the protocol if each host were to act spontaneously. For instance, we may omit a message used as a hint that communication is to be initiated.

The idealized protocol does not include the clear-text message parts; idealized messages of the form $\{X_1\}_{K_1}, \ldots, \{X_n\}_{K_n}$, where each encrypted part is treated separately. We have omitted clear-text communication simply because it can be forged, and so its contribution to an authentication protocol is mostly one of providing hints as to what it might be placed in encrypted messages.

2.6.5 Protocol Analysis

From a practical viewpoint, the analysis of a protocol is performed as follows:

- The idealized protocol is derived from the original one.
- Assumptions about the initial state are written.
- Logical formulas are attached to the statements of the protocol, as assertions about the state of the system after each statement.
- The logical postulates are applied to the assumptions and the assertions, in order to discover beliefs held by the parties in the protocol.

This procedure may be repeated as new assumptions are found to be necessary and the idealized protocol is defined.

Chapter 3

Distributed Shared Key Generation Using Fractional Keys

In this chapter we describe a distributed key generation and recovery algorithm suitable for group communication systems, where the group membership must be tightly controlled. The key generation scheme was proposed by Poovendran in [Poo99]. The key generation approach allows entities that may have only partial trust in each other to jointly generate a shared key, without exposing their contribution to the key. The group collectively generates and maintains a dynamic group parameter and the shared key could be generated using a pseudo-random function using this parameter as the seed. These requirements are realized through the use of fractional keys - a distributed technique recently developed to enhance the security of distributed systems in a non-cryptographic manner.

3.1 Basic Key Generation Protocol

3.1.1 Assumptions identified in original paper

The following is a list of the underlying assumptions of the above scheme:

- There exists binary operation on the set S of elements generating the secret such that $S \oplus S :\rightarrow S$.
- The shared keys are generated by a fixed number of participants n.
- A mechanism exists for certifying the members participating in the key generation

Table 3.1: Notation used in distributed key generation scheme

$\alpha_{i,j}$	The one-time pad of the i^{th} member at the j^{th} key update iteration. M_i uses $\alpha_{i,j}$ to hide its contribution to the key.
θ_j	The binding parameter at the j^{th} key update iteration. This is the shared secret that each M_i computes at the end of each round.
$\{K_i, K_i^{-1}\}$	Public/Private key pair of the member i. This pair is assumed to be updated appropriately to key the integrity and confidentiality of any communication transaction by and with member i.
$FK_{i,j}$	The Fractional Key (FK) of the i^{th} member at the j^{th} key update iteration. This is the contribution of M_i in the j^{th} round.
$HFK_{i,j}$	The *hidden FK* (HFK) of the i^{th} member at the j^{th} key update iteration
SK_j	The group session key (SK) at the j^{th} key update instance. It is a function of θ_j.
$A \rightarrow B : \mathcal{X} : \oplus$	Principal A sends principal B a message $\mathcal{X}$ $\oplus$ is the binary operation over the set of valid keys. In the sections below, $\oplus$ is addition modulo p, where p is a power of two. This simplifies calculation on computers.

procedure, for securely exchanging the quantities required in the algorithm and for authenticating the source of these quantities.

- Every member can generate uniformly distributed, independent random numbers in the given range.

3.1.2 Message Format

The message format is $\{\{T_i, M, j, Msg\}_{K_s^{-1}}\}K_R$, where:

- Ti: a real-valued, wall clock time stamp nonce generated by member i.
- M: denotes the *mode* of operation with "I" for Initialization mode, "G" for key Generation mode and "R" for key Recovery mode.
- j: integer-valued, denotes current iteration number.
- Msg: the message to be sent.
- K_s^{-1}: Denotes the private key of the sender S.
- K_R: Public key of the receiver.

The following properties are desirable for a multi-party key generation scheme:

- A FK contributed by a participating member should have the same level security as the group SK.
- A single participating member, without valid permissions, should not be able to obtain the FK of another member.
- If a FK-generating member has physically failed, been compromised or removed, the remaining FK-generating members should be able to jointly recover the FK of the failed member (this requires not majority voting but total participation).

The first property states that the distributed key generation scheme has to be such that each FK space has at least the same size as the final SK space. Hence, each member may generate FK of different size but, when combined, they lead to a fixed length SK.
The second property has to do with the need for protection of individual FKs that is desired due to the absence of a centralized key generation scheme. In the current scheme, every member performs an operation to *hide* its FK such that, when all the *hidden* FKs (HFK) and the group parameter are combined, the net result is a new SK. Note that even if a HFK is known, the problem of obtaining the actual FK or the SK needs further computation.
If a contributing member fails, becomes compromised, or has to leave the multicast group, then it becomes necessary to replace the existing member with a new member. Hence the newly-elected member should be able to securely recover the FK generated by the replaced member. However, to ensure the integrity of the scheme, this recovery is possible only if all the remaining contributing members cooperate.

3.1.3 Key Management Scheme

The key management scheme consists of three major parts:

1. Initialization - consisting of member selection, secure initial pad and binding parameter generation and distribution
2. Key Generation - an iterative process consisting of fractional, hidden and shared-key generation
3. Key Retrieval - required only in the case of a member node failure or compromise

3.1.4 Initialization Algorithm

A Group Initiator (GI) first selects a set of n FK generating members, and the GI may be one of these members. The GI then does **one of the following:**

- Contacts a Security Manager (SM) - a third party who is *not* a FK-generating member - who generates the initial pads and the binding parameter and distributes them to the members
- Initiates a distributed procedure among the group members to create these quantities without the aid of a third party.

SM-Based Initialization

At the time of initialization, n members are selected and the initial pads and binding parameter are distributed to each member i, for $i = 1, \ldots, n$, as

$$SM \rightarrow M_i : \{\{T_{SM}, I, 1, \alpha_{i,1}, \theta_1\} K_{SM}^{-1}\}_{K_i}$$

where $\alpha_{i,1}$ - its initial one time pad - is computed such that

$$\alpha_{1,1} \oplus \alpha_{2,1} \oplus \ldots \oplus \alpha_{n,1} = \theta_1$$

Distributed Initialization

The GI (assumed to be a member usually M_1) can perform the following steps to generate the initial parameters of the group:

1. Generate two uniformly-distributed random quantities γ and $\nu_{1,1}$ of bit length L, operate on these two quantities as

   $$\gamma \oplus \nu_{1,1} = \delta_1$$

   and send the result to M_2 (the next number in the group).

   $$M_1 \rightarrow M_2 : \{\{T_1, I, 1, \delta_1\}_{K_1^{-1}}\} K_2$$

2. The following steps are repeated for $i = 2, \ldots, n-1$:

   (a) M_i generates a uniform random variable $\nu_{i,1}$ of bit length L

   (b) M_i then operates on the quantity received from M_{i-1} as

$$\delta_{i-1} \oplus \nu_{i,1} = \delta_i$$

   (c) M_i then sends the result to M_{i+1} as

$$M_i \rightarrow M_{i+1} : \{\{T_i, I, 1, \delta_i\}_{K_i{}^{-1}}\} K_{i+1}$$

3. Eventually, M_n receives δ_{n-1} and then generates a uniformly-distributed random quantity $\nu_{n,1}$ of bit length L, performs

$$\delta_{n-1} \oplus \nu_{n,1}$$

   and then securely sends it the initiating M_1 as

$$M_n \rightarrow M_1 : \{\{T_n, I, 1, \delta_n\} K_n^{-1}\} K_1$$

4. The initiator (M_1) decrypts it and performs

$$\gamma \oplus \delta_n = \theta_1$$

   and then sends θ_1 to each M_i, for $i = 2, \ldots, n$ as

$$M_1 \rightarrow M_i : \{\{T_1, I, 1, \theta_1\} K_1^{-1}\} K_i$$

5. Each M_i privately computes

$$\alpha_{i,1} = \theta_1 \oplus \nu_{i,1}$$

and uses $\alpha_{i,1}$ as its initial pad.

The distributed initialization approach assumes that members are indexed and all members know the left or right neighbor. If there is no member collaboration, this approach prevents a member from knowing the individual secret of any other member. The computations scale as $\mathcal{O}(N)$ with N being the group size.

Note that the two approaches of initialization - security manager-controlled and distributed - are not equivalent unless additional security assumptions are made. For example, in the case of distributed initialization within the group, we point out that using the following attack is feasible - assume that M_{i-1} and M_{i+1} conspire to obtain the secret of M_i, where the numerical ordering corresponds to the order of message passing in the distributed algorithm.

1. M_{i-1} sends δ_{i-1} to M_i as per algorithm, and also to M_{i+1} without $M_i's$ knowledge.
2. M_i, who is unaware of the conspiracy between M_{i-1} and M_{i+1}, computes $\delta_i = \delta_{i-1} \oplus \nu_{i,1}$ and sends it to M_{i+1} securely.
3. M_{i+1} can now compute

$$\nu_{i,1} = \delta_{i-1} \oplus \delta_i$$

   and obtain the secret $\nu_{i,1}$ of M_i.

However the secret $\nu_{i,1}$ generated by M_i becomes part of the pads (i.e. the $\alpha's$) of M_{i-1} and M_{i+1}. Hence, application of this initialization assumes that the parties are benign.

Finally, in both the distributed and the SM-based initialization scheme, the n members who contribute to the key can be a subset of the group members with special privileges. The resulting shared secret key has to be distributed to the remaining members using other cryptographic mechanisms.

3.1.5 Key Generation Algorithm

The key generation algorithm is an iterative process. Each iteration j requires as input a set of one-time pads

$$\alpha_{i,j}, i = 1, \ldots, n$$

and the binding parameter θ_j, which are obtained from the initialization algorithm for iteration $j = 1$, and from the preceding iteration for $j > 1$.

The key generation algorithm consists of the following steps:

1. For $i = 1, \ldots, n$, M_i generates an L bit cryptographically-secure random number $FK_{i,j}$.

2. For $i = 1, \ldots, n$, M_i generates a quantity

$$HFK_{i,j} = \alpha_{i,j} \oplus FK_{i,j},$$

and all members securely exchange the $HFK's$ as

$$\forall 1 \leq l, m \leq n, l \neq m, M_l \rightarrow M_m : \{\{T_l, G, j, HFK_{l,j}\}_{K_l^{-1}}\}_{K_m}$$

3. Once the exchange is complete, M_i locally computes

$$\begin{aligned} \sum_{i=1}^{n} \oplus HFK_{i,j} &= \sum_{i=1}^{n} \oplus(\alpha_{i,j} \oplus FK_{i,j}) \\ &= \lambda_j \theta_j \oplus \sum_{i=1}^{n} \oplus FK_{i,j} \end{aligned}$$

4. Every member then computes the new value of the group shared secret θ_{j+1} by removing the effect of the initial shared secret value

$$\theta_{j+1} = \lambda_j \theta_j \oplus \sum_{i=1}^{n} \oplus FK_{1,j} \oplus \oplus \mu_j \theta_j$$

where μ_j is the appropriate *inverse* of λ_j. In this case, $\mu_j = p - \lambda_j$. If the resulting group parameter θ_{j+1} is *cryptographically-insecure* for a particular application, all members can repeat the above steps, creating a new high quality group parameter θ_{j+1}

5. M_i locally computes its new pad as

$$\alpha_{i,j+1} = \theta_{j+1} \oplus FK_{i,j+1}$$

essentially removing the effect of its own share.

6. The shared key $SK_j = f(\theta_{j+1})$ where $f(\cdot)$ is a pseudo-random function.

This summarizes the computational steps for generating the keys at each update. Note that the quantity $\alpha_{i,j+1}$ is computed such that, for an outsider, obtaining $\alpha_{i,J+1}$ is very hard even if the

actual key SK_j is compromised at any key update time interval $(j, j+1)$. Knowing the group key does not reveal the group parameter and, hence, the tight binding of the members will not be broken by the loss of the shared key.

Note the following additional features

- Although all members have each $HFK_{i,j}$, obtaining the $FK_{i,j}$ or $\alpha_{i,j+1}$ of another member involves search in the $L-$dimensional space, and obtaining their correct combination involves search in the $(n-2)$ of L dimensional (Key length) space. Hence, even if a fellow member becomes an attacker, that rogue member faces nearly the same computational burden in obtaining the set of n FKs as an outside cryptoanalyst; i.e. trust is *not unconditional*.
- For such an outside attacker, breaking the system requires, either a search in the L dimensional space to get θ, or n dimensional searches to break individual secrets of all the members. Access to all n $HFKs$ is alone insufficient to permit an attacker to determine the SK; for that, the attacker must also possess the current binding parameter θ which is time-varying and never transmitted. If a SK is known to be compromised (perhaps due to traffic analysis), since $f(\cdot)$ is a pseudo-random function, information regarding θ is not obtained.

3.2 Retrieval of the Fractional Key and Pad of a Failed Node

The following steps are involved in recovery of the $FK_{i,j}$ and $\alpha_{\hat{i},j}$ of the node failed $\hat{i}$, where j represents the iteration number in which the node was compromised or failed.

1. Any one $FK-$generating member - called the Recovery Initiator (RI) - must initiate recovery and give the HFK of the failed node $\hat{i}$ to the newly-elected M_i as

$$RI \rightarrow M_i : \{\{T_{RI}, R, j, SK_j\}_{K_{RI}^{-1}}\}_{K_i}.$$

2. The RI must also give the newly-elected M_i the current θ_j as

$$RI \rightarrow M_i : \{\{T_{RI}, R, j, \theta_j\}_{FK^{-1}_{RI,j}}\}_{FK_{i,j}}.$$

3. Using the same algorithm as is used for distributed initialization, with the following replacements:

   (a) θ by ξ

   (b) $\alpha_{l,j}$ by $\beta_{l,j}$.

   Except for the changes in the notation and the number of members participating, the algorithm for pad generation is same as for distributed initialization. Hence, at the end of this distributed pad generation, each member l has $\beta_{l,j}$ as its pad for key recovery process, and all these pads are bound with the parameter ξ.

4. For $l = 1, \ldots, n-1$, each M_l then computes a *modified* hidden fractional key

$$\widehat{HFK}_{l,j} = \beta_{l,j} \oplus FK_{l,j}$$

   and hands it to the newly-elected M_i as

$$M_l \rightarrow M_i : \{\{T_l, R, j, \widehat{HFK}_{l,j}\}_{K_l^{-1}}\}_{K_i}$$

5. M_i then combines all of the modified $HFKs$ and recovers the fractional key

$$FK_{\hat{i},j} = \lambda\xi \oplus \widehat{HFK}_{1,j} \oplus \ldots \oplus \widehat{HFK}_{n-1,j} \oplus \theta_{j+1}$$

6. M_i then extracts the pad $\alpha_{\hat{i},j}$ using the operation

$$\alpha_{\hat{i},j} = HFK_{\hat{i},j} \oplus FK_{\hat{i},j}.$$

Note that the recovered values of $FK_{i,j}$ and $\alpha_{i,j}$ are unique. Once the new node recovers the fractional key of the compromised node, it can inform the other contributing members to update the iteration number j to $j+1$, and then all members can execute the key generation algorithm. Note that even though the newly-elected member recovers the compromised fractional key and pad, the next key generation operation of the new node does not use the compromised key or pad. Hence, even if the attacker possesses the fractional key or pad at iteration j, it does not allow the attacker to obtain the future fractional keys or pads without computation.

3.3 Improvements to the Key Agreement Protocol

We analyze the protocol to simplify certain steps and this reduces message size and number of computations without affecting the security of the scheme. We also describe an Auxiliary Key Agreement (AKA) protocol.

- As indicated earlier, the $HFKs$ need not be encrypted because confidentiality of the hidden fractional keys is not required for the key agreement protocol to be secure. However integrity of the $HFKs$ must be ensured. Therefore the HFKs can be transmitted in the clear. A signature or a Message Authentication Code (MAC) will cover the whole message, thereby ensuring the integrity of the whole message, and trivially, each of its components.

- We do not use time-stamps to indicate freshness. Using time-stamps pose well-known security risks. Not using time-stamps requires additional per member storage at the SM or GI.

- In case of key recovery, the RI need not send SK to the replacement node. Only the current binding parameter needs to be transmitted. Moreover failure can be treated more simply as a group leave by the compromised member, and a group join by the replacement node.

- Most important of all, the key agreement scheme does not handle membership change well. We therefore develop an Auxiliary Key Agreement (AKA) Protocol to handle these two cases. The AKA is only partially contributory. This reduces computation and

message overhead, resulting in faster response times without weakening the security of the scheme.

- We do not base the security of the authentication scheme on the difficulty of factoring large numbers or discrete logarithms in a finite field. Consequently, we do not need a prime number p to perform modular arithmetic. However all computation when performed on a computer is based on finite number of bits. All our computation is modulo p, where p is a power of two.

- Finally, we present an authenticated key agreement protocol. A key observation while comparing this scheme to others, is that we do not use exponentiation. Our scheme involves less computation. The previously discussed schemes need to use some MAC to provide message integrity. Those that don't use a MAC can't provide key integrity and that was explicitly stated by Tsudik [STW00] in his work. By using a signature scheme, we provide both Key Integrity (KI) and Key Authentication (KA).

Chapter 4

Distributed Shared Key Generation Using Fractional Keys

In this chapter we describe a distributed authenticated key agreement protocol suitable for group communication, where the group membership must be tightly controlled. The key generation scheme was described in the previous chapter. Our protocol requires the presence of a public key infrastructure. We can reduce the message overhead and computation overhead by using message authentication codes and secret key encryption. First, we formally discuss our threat model and the requirements of such an authenticated key agreement protocol. To address concerns raised at the end of the chapter, we also list the requirements for an auxiliary key agreement scheme that can be used when group membership changes. As we list the requirements for the auxiliary key agreement scheme, it will become evident that the protocol becomes more susceptible to user collusion in the SM based scheme when the auxiliary scheme is incorporated. This is because the auxiliary key agreement scheme reduces overhead by making the key agreement partially contributory and re-using the contributions of the other members. However, the distributed initialization scheme is already vulnerable to user collusion, and incorporating the auxiliary protocol does not introduce a new vulnerability. It must be stressed that the only risk is knowledge of contribution of a member by colluding members, and not of non-members, not even users whose membership has been revoked. Then we describe a concrete protocol that satisfies these requirements and incorporates the auxiliary key agreement protocol. Finally we analyze this protocol using BAN logic.

4.1 Requirements for Authenticated Key Agreement Protocol

4.1.1 Requirements in Initialization Phase

Requirements in SM-Based Initialization

- SM and all the members have a common CA or have a certification path to a common CA.
- SM authenticates itself with every other member. SM authenticates all the members in the group.
- For every M_i, SM generates a sequence of random numbers - $CKY_{i,k} = R_i, \ldots, CKY_{i,0} = f^k(R_i)$, where f is a strong one-way hash function similar to the SKEY scheme described in [Sch96].

  SM sends $CKY_{i,0} = f^k(R_i)$ to M_i as part of the protocol for mutual authentication. The cookie can be sent in clear text form. The rest of the values are stored at the SM for use in subsequent messages from SM to M_i.
- SM generates n L bit random numbers - $\alpha_{i,1}, \ldots, \alpha_{n,1}$ - to act as initial one-time pads for each member M_i of the group. It also computes the binding parameter θ_1 as:

$$\alpha_{i,1} \oplus \ldots \oplus \alpha_{n,1} = \theta_1$$

  SM does not store any of these values, except for re-transmission on communication errors.
- For every member M_i, SM encrypts the binding parameter θ_1 and the initial one-time pad $\alpha_{i,1}$. It appends the cookie $CKY_{i,1} = f^{k-1}(R_i)$ (in clear-text form) to the message. SM signs the whole message and sends it to M_i. M_i deletes $CKY_{i,1}$ from its storage.
- Every M_i checks that $f(CKY_{i,1})$ equals $CKY_{i,0}$, to verify that the message is fresh.

  M_i verifies the signature of SM on the message. M_i decrypts and extracts the binding parameter θ_1 and its initial one-time pad $\alpha_{i,1}$.

- Every M_i then updates its secret to $CKY_{i,1}$.

Requirements in Distributed Initialization

- All members have a common CA or a certification path to a common CA.
- One of the members, say M_1 acts as the Group Initiator (GI).
- The members are indexed from $1, \ldots, n$, and every M_i (except M_1 and M_n) authenticates itself with M_{i+1} and M_{i-1}. M_1 authenticates itself with M_2 and M_n. M_n authenticates itself with M_1 and M_{n-1}.
- Every member M_i (except M_n), establishes a secret - $CKY_{i,i+1}$ - with member M_{i+1}. M_n establishes a secret with M_1. This will take place as part of mutual authentication of neighbors.
- The GI (M_1) authenticates itself to every member M_i (except M_n and M_2 who have already authenticated M_1). Every M_i, (except M_2 and M_n) is authenticated by GI.
- To establish a secret with each member M_i, the GI generates a sequence of Random numbers for each member M_i, $CKY'_{i,k} = R_i, \ldots, CKY'_{i,0} = f^k(R_i)$.

  The GI sends $CKY'_{i,0}$ to M_i as part of the mutual authentication protocol. The cookie can be sent in clear-text form.
- Every member M_i generates its own L bit secret random integer, γ_i.
- M_1 (GI) generates a random number α and operates on it as:

$$\delta_1 = \alpha \oplus \gamma_1$$

  where, γ_1 is its secret. It encrypts δ_1, and includes the cookie it shares with M_2, CKY_2. M_1 signs the whole message and sends this to M_2.
- For $i = 2, \ldots, n-1$, every member M_i, receives the message from member M_{i-1}. Member M_i, checks the cookie $CKY_{i-1,i}$ to verify the freshness of the message from M_{i-1}. M_i then verifies the signature on the message as belonging to M_{i-1}. M_i decrypts the message to get δ_{i-1}.

- Member M_i adds its secret γ_i to δ_{i-1} as:

$$\begin{aligned} \delta_i &= \delta_{i-1} \oplus \gamma_i \\ &= \alpha \oplus \sum_{j=1}^{i} \gamma_j \end{aligned}$$

  M_i encrypts δ_i, appends the secret it shares with M_{i+1}, CKY_{i+1} to the message. M_i signs the whole message and sends it to M_{i+1}.

  M_i stores δ_{i-1} it receives from M_{i-1} till the next time a new value is received.

- Member M_n receives the message from M_{n-1}. M_n verifies that $CKY_{n-1,n}$ is part of the message. M_n verifies the signature on the message as belonging to M_{n-1}. M_n decrypts δ_{n-1}. It computes δ_n as:

$$\delta_{n-1} \oplus \gamma_n = \delta_n$$

  M_n encrypts δ_n, it appends the secret it shares with M_1, $CKY_{n,1}$, to the message. It then signs the message and sends it to M_1.

- The GI (M_1) receives the message from M_n. It checks the message is fresh by verifying the presence of CKY_n. M_1 verifies the signature on the message as belonging to M_n. M_1 then decrypts the message to obtain δ_n.

  Member M_1 then computes the common secret θ_1 as:

$$\theta = \sum_{j=1}^{n} \gamma_j$$

- For every member M_i, M_1 encrypts θ_1. It includes $CKY_{i,1} = f^{k-1}(R_i)$ to guarantee the freshness of the message. M_1 signs the message and sends it to every other member M_i. It also deletes $CKY_{i,1}$ from its storage.

- Every member M_i receives the message and checks that $f(CKY_{i,1}^{'} = CKY_{i,0}^{'}$. M_i verifies the signature on the message and decrypts the message to obtain θ_1.

- Every member M_i updates its secret to the cookie from the message. It then computes:

$$\alpha_{i,1} = \theta_1 - \nu_{i,1}$$

4.1.2 Requirement in Key Generation Phase

1. Every member M_i, generates its contribution to the shared key - a random number $FK_{i,1}$, and hides its contribution by performing the following computation with its one-time pad $\alpha_{i,1}$ as:

$$FK_{i,1} \oplus \alpha_{i,1} = HFK_{i,1}$$

   Every member stores its contribution to the key till the next re-key.

2. Every member, M_i creates a message with its hidden fractional key $HFK_{i,1}$ in clear-text. M_i signs the message and the current binding parameter θ_1 of the group, and sends it to all other members. Note that the binding parameter θ_1 does not appear in the message itself. It is part of the signature and is used to guarantee freshness of the message (instead of creating a whole new set of shared secret values between members of the group.

3. Every member M_i receives the hidden fractional key $HFK_{j,1}$, from all other members M_j; $j \neq i$. M_i verifies the signature of M_j.

4. Once, M_i receives the hidden fractional keys from all other members, it computes:

$$\begin{aligned} \sum_{i=1}^{n} \oplus HFK_{i,1} &= \sum_{i=1}^{n} \oplus (\alpha_{i,1} \oplus FK_{i,1}) \\ &= \lambda_1\theta_1 \oplus \theta_2 \end{aligned}$$

   with $\theta_2 = \sum_{i=1}^{n} \oplus FK_{i,1}$ and $\lambda_1\theta_1$ is the result of operation $\oplus$ performed on θ_1, λ times.

5. Every member then locally computes the new value of the group shared secret θ_2 by removing the effect of the initial shared secret value:

$$\theta_2 = \lambda_1\theta_1 \oplus \theta_2 - \lambda_1\theta_1$$

6. Every member i locally computes its new pad as:

$$\alpha_{i,2} = \theta_2 - \gamma_2 FK_{i,2}$$

   essentially removing the effect of its own share.

7. At the share update step j, the procedure is:

- generate new individual shares $FK_{i,j}$
- combine it with the individual dynamic pad $\alpha_{i,j}$ to generate $HFK_{i,j}$
- exchange HFK's of all members securely
- compute the new shared secret $\theta_{j+1} = \lambda_j \theta_j \oplus \theta_{j+1} - \lambda_j \theta_j$.
- compute the new shared individual pad $\alpha_{i,j+1} = \theta_{j+1} \oplus \gamma_{j+1} FK_{i,j+1}$.

4.1.3 Requirements for Auxiliary Key Agreement Protocol

Requirements in SM-Based scheme

Member Join

Let the new member be denoted as M_{n+1}.

1. SM and M_{n+1} should have a common CA or a certification path to a common CA.
2. SM must authenticate M_{n+1}. M_{n+1} must authenticate SM.
3. SM generates a sequence of random numbers $CKY_{n+1,k} = R_{n+1}, \ldots, CKYn+1, 0 = f^K(R_{n+1})$. SM sends $f^k(R_{n+1})$ securely to M_{n+1} during the mutual authentication protocol.
4. If we denote the set of group members as $\mathcal{M}$, the SM chooses a random subgroup of members denoted by M_{sub}, with $M_{sub} \subset \mathcal{M}$. Let the members of this subgroup be denoted as $M_1, \ldots, M_r$ with the indices having no particular significance.

   The SM generates $r+1$ L bit random numbers $\alpha_{1,1}, \ldots \alpha_{r,1}$ and $\alpha_{n+1,1}$, to serve as the initial one-time pads of the members in the sub-group and the new member. The SM computes the new binding parameter $\theta_1^{'}$ using these values in addition to the contribution of other M_i, $M_i \notin M_r$ in the previous round.
5. For every member M_i, $M_i \notin M_r$, SM encrypts $\theta_1^{'}$. SM appends the cookie $CKY_{i,l}$ $(= f^{k-l}(R_i))$ (assume l^{th} round) to the message. SM computes the signature on the whole message.
6. For the the members M_i, $M_i \subset M_r$, SM encrypts $\theta_1^{'}$ and the initial one-time pad $\alpha_{i,1}$. It adds the cookie $CKY_{i,l}$ (l^{th} round) and signs the whole message. The SM sends a signed message containing $CKY_{n+1,1}$ and $\alpha_{n+1,1}$ to M_{n+1}.

7. Each M_i receives the message from SM. It checks that the message is fresh based on the cookie. It then verifies the signature on the message as belonging to SM. It decrypts the new binding parameter.

8. The remaining steps are the same as in the key generation phase.

Member Leave

Let the revoked or leaving member be denoted as M_l. The requirement are simpler in this case. SM picks a random subset of members denoted as $M_1, \ldots, M_r$ to distribute their new initial-one time pads. SM calculates the new binding parameter by using these new values, in addition to the old values. It need not remove the one-time pad of member l from the binding parameter. The remaining steps are same as above.

Requirements in Distributed Initialization scheme

Member Join

1. M_{n+1} should have a common CA with all other members or a certification path to a common CA.

2. M_{n+1} authenticates itself with the GI. GI authenticates itself with M_{n+1}. The GI generates a sequence of random numbers $CKY^{'}_{n+1,k} = R_{n+1} \ldots, CKY^{'}_{n+1,0} = f^k(R_{n+1}$. It sends $CKY^{'}_{n+1,0}$ securely to M_{n+1} during mutual authentication.

3. M_n authenticates itself to M_{n+1}. M_{n+1} authenticates itself with M_n. They establish a new shared secret $CKY_{n,n+1}$ during mutual authentication.

4. M_{n+1} and the GI establish a secret $CKY_{n+1,1}$.

5. The GI sends a request to M_n to restart initialization.

6. M_n receives the message. It verifies that the message is fresh by the presence of the cookie. It verifies the signature on the request as belonging to the GI. M_n generates a new random secret number $\gamma^{'}_n$. It computes $\delta^{'}_n$ as:

$$\delta_{n-1} \oplus \gamma^{'}_n = \delta^{'}_n$$

7. M_n encrypts $\delta_n^{'}$. It adds CKY_n to the message and signs the while message. It sends the message to M_{n+1}.

8. M_{n+1} generates a its random secret γ_{n+1}. It computes δ_{n+1} and encrypts this. It adds the cookie it shares with the GI CKY_{n+1} to the message. M_{n+1} signs the message and sends it to M_1.

9. The remaining steps are similar.

Member Leave

Member leave is simpler. Let M_i be the member who is leaving or whose membership has been revoked.

1. GI requests M_{i-1} to restart initialization and informs the group that M_i has left.

2. M_{i-1} has to authenticate itself with M_{i+1} and vice-versa. The establish a new shared secret between themselves.

3. The remaining steps are similar.

4.2 Threat Model

4.2.1 Assumptions

- Public Key Infrastructure (PKI) is present.
- Group members follow the protocol correctly.
- Group members do not disclose the binding parameter or the shared key.
- Collusion among group members is limited. In particular if there are n members, $n-1$ members can collude and obtain the fractional key of the other member. Using the auxiliary key agreement protocol makes the scheme more vulnerable to collusion among group members while improving performance.
- External attackers have substantial resources but cannot invert one-way hash functions, break public key algorithms or compromise the public key infrastructure.

4.2.2 Nature and Type of Threats

- External attackers can listen to messages, drop, re-send, modify or even concoct protocol messages.

4.2.3 Guarantees

Under these two assumptions, we can make the following statements:

- External attacker cannot lead group members into forming an incorrect key.In other words the protocol provides key authentication.
- External attackers cannot cause denial of service in a broadcast medium assuming that he can't control the link layer.

4.2.4 Protocol Design Considerations

- No time-stamps.
- Minimize possibility of using Denial of Service (DoS) attacks.
- Avoid unnecessary encryptions.
- Minimize use of Public-key based computation.

4.2.5 Summary of Security Attributes of AKA

- **External Attackers**
  - The external attacker(s) can drop, re-send or modify messages. The protocol is secure in the sense that such actions cannot lead the group members into forming a key that is known to the attacker.
  - The initialization phase of the protocol uses shared secrets that are sent in clear text, as part of the message. This helps a member decide whether a message is fresh without any computation, thus providing resistance to denial of service attacks. However, the key generation phase incorporates the shared secret (binding parameter) in signature, forcing members to verify the signature before they can

Table 4.1: Additional notation used in AKA protocol

R	Round number. Helps to synchronize events among group members
$H(M)$	Cryptographic hash of M using a basic algorithm such as MD4
$Sig_K(M)$	Signature of M. M is first hashed and the signature is calculated on the hash $E_{K^{-1}}\{H(M)\}$
$CKY_{i,R}$	random number incorporated in the message by SM or GI to indicate freshness of message. Prevents replay attacks
M_i	Member index. Helps to index into Certificate store to recover public key
G/I	G indicates Key generation phase and I indicates Initialization phase. MA indicates re-key and the member index in this case gives the identity of the new member.
$X\|Y$	Y is concatenated with X
Msg	The message - one of HFK, or the encrypted θ_j or δ_i

verify the freshness of the message. This makes it more susceptible to denial of service attacks.

- An external attacker cannot guess the binding parameter or obtain it without being able to invert a strong one-way strong function or break a public key. This is not transmitted in any other form during the key generation phase. Knowing all the HFKs does not allow an attacker to guess the FKs. Knowledge of a few of the FKs does not allow the attacker knowledge of the shared key. Even if the session key is compromised, after the next re-key, the attacker has to start from the scratch. Finally, if the binding parameter is compromised, when the new binding parameter is computed, the attacker compute it.

- **User Collusion:** The SM-based protocol prevents less than $n-1$ members from colluding and obtaining the secret contribution of the other member. The Distributed initialization scheme is susceptible to user collusion. The Auxiliary key agreement protocol introduces the collusion problem in the SM-Based scheme. We can however, make it more collusion-secure by using a k out of n scheme to distribute the new initial one-time pads.

4.3 Protocol AKA

4.3.1 Message Format

The general message format is:

$$M_i \rightarrow M_{i+1} : \{CKY_{i,R}, G/I, R, M_i, Msg, Sig_{K_i}(CKY_{i,R}|G/I|R|M_i|Msg)\}$$

The θ_j's have to be protected by encryption against non-members. The HFK's can be sent in the clear. δ_i should be seen only by member $i+1$ and is encrypted using the public key of $i+1$. In all the cases integrity is provided by the use of a signature.

4.3.2 Mutual Authentication

Mutual authentication and shared secret establishment that happens as part of the protocol use the following well-known scheme:

$$\begin{aligned} M_A &= \{R_A, I_A\} \\ A \rightarrow B : & \ Cert_A, Sig_{K_A}(M_A) \\ M_B &= \{R_B, R_A, I_B, d\} \\ B \rightarrow A : & \ Cert_B, Sig_{K_B}(M_B) \end{aligned}$$

4.3.3 Initial Key Agreement

SM-Based Initialization

As above, at the time of initialization, n members are selected and the initial pads and binding parameter are distributed to each member i, for $i = 1, \ldots, n$, as

$$SM \rightarrow i : \{CKY_{i,1}, I, 1, SM, E_{K_i}(\alpha_{i,1}|\theta_1), Sig_{K_{sm}}(CKY_{i,1}|I|1|SM|E_{K_i}(\alpha_{i,1}|\theta_1))\}$$

Note that the initial one-time pad $\alpha_{i,1}$ and θ_1 are both encrypted with the public key of i^{th} member. This is not strictly necessary. Only the initial-one time pad must remain secret from

other members. Here, $\alpha_{i,1}$ - its initial one time pad - is computed such that

$$\alpha_{1,1} \oplus \alpha_{2,1} \oplus \ldots \oplus \alpha_{n,1} = \theta_1$$

Distributed Initialization

1. Generate two uniformly-distributed random quantities γ and $\nu_{1,1}$ of bit length L, operate on these two quantities as

   $$\gamma \oplus \nu_{1,1} = \delta_1,$$

   and send the result to member 2 (the next member in the group) as

   $$M_1 \rightarrow M_2 : \{CKY_1, I, 1, GI, E_{K_2}(\delta_1), SIG_{K_1}(CKY_1|I|1|GI|E_{K_2}(\delta_1))\}$$

2. The following steps are repeated for $i = 2, \ldots, n-1$:

   (a) M_i generates uniform random variable $\nu_{i,1}$ of bit length L.

   (b) M_i then operates on the quantity it received from M_{i-1} as $\delta_{i-1} \oplus \nu_{i,1} = \delta_i$

   (c) M_i then sends the result to M_{i+1} as

   $$M_i \rightarrow M_{i+1} : \{CKY_i, I, 1, i, E_{K_{i+1}}(\delta_i), Sig_{K_i}(CKY_i|I|1|i|E_{K_{i+1}}(\delta_1))\}$$

3. Eventually, the group M_n receives δ_{n-1} and then generates a uniformly-distributed random quantity $\nu_{n,1}$ of bit length L, performs

   $$\delta_{n-1} \oplus \nu_{n,1}$$

   and then securely sends it the M_1 (GI) as

   $$M_n \rightarrow M_1 : \{CKY^{'}_{i,1}, I, 1, n, E_{K_i}(\delta_n), Sig_{K_n}(CKY^{'}_{i,1}|I|1|n|E_{K_i}(\delta_n)\}$$

4. M_1 (GI) decrypts it and performs

   $$\gamma - \delta_n = \theta_1$$

and then sends θ_1 to each M_i, for $i = 2, \ldots, n$ as

$$M_1 \rightarrow M_i : \{CKY_i, I, 1, 1, E_{K_i}(\theta_1), Sig_{K_1}(CKY_i, I, 1, 1, E_{K_i}(\theta_1)\}$$

5. Each M_i privately computes

$$\alpha_{i,1} = \theta_1 - \nu_{i,1}$$

and uses $\alpha_{i,1}$ as its initial pad.

The use of SKEY-based nonces (CKYs) eliminates the storage problems associated with nonces (alternative to time-stamps). Moreover, since the CKYs are sent in the clear, the possibility of DoS attacks by sending spurious messages is eliminated. To successfully disrupt group operation, an attacker must be able to intercept the original message before being delivered, and then modify it (Note that signature prevents him from affecting protocol correctness but such an adversary can disrupt protocol operation). In some environments such threats can be discounted.

Note that in the message the GI communicates to every other member, if there were another shared secret key, then computations at the GI are reduced - the signature needs to be computed only once, because the θ_1 is encrypted with the same key. The number of messages in a broadcast scenario will reduce from $n-1$ to 1.

4.3.4 Key Generation Phase

The iterative key generation phase remains the same. Only the format of the messages exchanging the HFKs changes.

$$\forall 1 \leq l, m \leq n, l \neq m, l \rightarrow m : \{G, j, l, HFK_{l,j}, Sig_{K_l}(\theta_l | G | j | l | HFK_{l,j})\}$$

First note that the HFKs can be sent in the clear. The FKs are masked with a random quantity to produce the HFKs. Also note that the binding parameter θ in the signature proves the message is fresh. We do not need time-stamps or nonces.

4.3.5 Auxiliary Key Agreement

This protocol helps the group recover quickly from membership changes and establish the new shared secret key.

Let the key contributing members be numbered $1, \ldots, n$. After a series of membership changes, let the set of key contributing members be numbered $1, \ldots, k$. A new binding parameter needs to be established among the members.

Note that it is advantageous to aggregate membership changes, however this is not possible in all applications. This affects only performance and not the correctness of the protocol. Also note that there is always centralized access control. This role can be assumed by the GI in the distributed initialization case or by the third-party SM in case of the centralized initialization. In the former case, there has to be a standard way of choosing the GI from a set of key contributing members and this protocol should take into account that the GI may leave the group. These are secondary to our level abstraction. We assume that such a protocol and access control mechanism exist.

It is clear that this entity needs to keep track of the members joining the group and leaving the group, whenever a re-key has to be performed.

Let us first illustrate the cases of single member join and single member leave and make the protocol concrete. We require that each M_i store the last message they received in the initialization phase (The message from M_{i-1} in distributed initialization phase or the initial-one time pad from the SM in case of SM-based initialization).

Member Join

In this case let the set of old members is $1, \ldots, n$, and let the new member be $n+1$.

Centralized Initialization: The SM chooses two members at random - let the selected members be j and k - generates initial pads for j, k and $n+1$, and distributes the new binding parameter to each member as:

$$SM \rightarrow i, i = 1, \ldots, n;\ i \neq j, k\{CKY_{i,l+1}, I, l{+}1, E_{K_i}(\theta'_{l+1}), Sig_{K_{sm}}(CKY_{i,l+1}|I|l{+}1|E_{K_i}(\theta'_{l+1}))\}$$

$$SM \rightarrow i,\ i = j, k, n{+}1\{CKY_{i,l+1}, I, l{+}1, E_{K_i}(\theta'_{l+1}|\alpha'_{i,l+1}), Sig_{K_{sm}}(CKY_{i,l+1}|I|l{+}1|E_{K_i}(\theta'_{l+1}|\alpha'_{i,l+1}))\}$$

where α's - initial one time pads - are computed such that

$$\alpha_{1,l+1} \oplus \ldots \oplus \alpha^{'}_{j,l+1} \oplus \ldots \oplus \alpha^{'}_{k,l+1} \ldots \oplus \alpha_{n,l+1} \oplus \alpha_{n+1,l+1} = \theta^{'}_{1+1}$$

$$\alpha_{i,l+1} = \alpha_{i,l}; i \neq= j, k, n+1.$$

Note that we have chosen to change the one-time pad of two members - j, and k -apart from assigning an initial one-time pad to member $n+1$. This is important. If we had only assigned a new one-time pad to $n+1$, all other members would be able to calculate member $n+1$'s one-time pad once they know the new binding parameter. If we changed the one-time pad of just one member, say j, then $n+1$ and j would be able to compute each other's one-time pad. By sending it to two other members, we make it impossible to compute the one-time pad of other members without collusion. The centralized initialization scheme was resistant to collusion of less than $n-1$ members. By introducing this scheme, we have made it less collusion-free. We can make it more secure by choosing k out of n members. This is a trade-off with performance. Knowing the one-time pad of another member, helps us to compute his contribution to the key, i.e., his fractional key. This does not help us calculate the fractional key of other members unless the membership is trivial - i.e., two members, etc. For such small groups, re-key is not a big challenge.

Distributed Initialization: Let M_1 be the GI. The GI must involve M_n in the membership join using a explicit message. M_n has δ_{n-1}, where l is the previous initialization round. M_n generates a uniform random variable $\nu^{'}_{n,l+1}$ of bit length L. M_n then operates on this quantity as:

$$\delta_{n-1} \oplus \nu^{'}_{n,1+1} = \delta^{'}_{n}$$

M_n then sends the result to M_{n+1} as

$$M_n \rightarrow M_{n+1} : \{CKY_n, I, l+1, n, E_{K_{n+1}}(\delta^{'}_{n}), Sig_{K_n}(CKY_n|I|l+1|n|E_{K_{n+1}}(\delta^{'}_{n}))\}$$

M_{n+1} obtains $\delta^{'}_{n}$ from M_n. M_{n+1} then generates a uniform random variable $\nu_{n+1,l+1}$ of bit length L.

M_{n+1} then operates on this quantity as

$$\delta_n^{'} \oplus \nu_{N+1,l+1} = \delta_{n+1}$$

M_{n+1} then sends the result to M_1 as

$$M_{n+1} \rightarrow M_1 : \{CKY_{n+1}, I, l+1, n+1, E_{K_1}(\delta_{n+1}), Sig_{K_{n+1}}(CKY_{n+1}|I|l+1|n+1|E_{K_1}(\delta_{n+1}))\}$$

The GI decrypts it and performs

$$\delta_{n+1} - \gamma = \theta_1^{'}$$

and then sends $\theta_1^{'}$ to each M_i, for $i = 2, \ldots, n+1$ as

$$M_1 \rightarrow M_i : \{CKY_{i,l+1}^{'}, I, 1+1, 1, E_{K_i}(\theta_1^{'}), Sig_{K_1}(CKY_{i,l+1}^{'}, I, l+1, 1, E_{K_i}(\theta_1^{'}))\}$$

Each M_i privately computes

$$i = 1, \ldots, n-1, n+1 : \alpha_{i,l+1} = \theta_1^{'} - \nu_{i,1+1} i = n : \alpha_{n.l+1} = \theta_1^{'} - \nu_{n,l+1}^{'}$$

and uses $\alpha i, 1$ as its initial pad.

Member Leave/Revocation

Centralized Initialization: The procedure is same as above, with the following modifications:

- The SM must choose at least three members at random, to distribute their new one-time pad. This is done to avoid the security pitfalls highlighted in the previous case.
- We can remove the contribution of revoked member (or leaving member). This does not appear to be necessary.

Distributed Initialization: Member revocation is not as efficient, except when the highest-index member leaves, assuming GI is member 1.
Let M_i leave, or his membership revoked. Then the GI has to initiate re-key by sending a message to M_{i-1}.

More message are required because, in effect, the initialization phase starts from M_{i-1}, and not M_n as in the previous case. However, on an average, it is still better than starting from M_1. The format of the message is the same as above.

4.3.6 Some Observations

Once the new binding parameter is established, the new members go into key generation phase to compute the shared secret key.

It is clear that total participation in the auxiliary key agreement protocol is impractical. The distributed scheme and the SM-based scheme try to minimize the computation required and the message overhead by re-using contributions to the key in the previous iterations. This does not weaken the security of the protocol, as can be seen by the formal analysis of the Initial Key Agreement and the Auxiliary Key Agreement protocols. It is to be noted that this protocol is a not a Contributory Key Agreement protocol (according to definitions presented in chapter 1.)

While aggregating membership changes, it is necessary to track members who want to leave or who would like to join, and make a decision as to how many members need to participate in the initialization phase, distributed or otherwise. This is not difficult to perform and is not an unreasonable demand. The procedure followed would be a combination of the above two instances.

4.3.7 Key Recovery

Unless the nature of the application is such, a failed node replaced with a new node can be treated as membership revocation of the failed node followed by membership join by the replacement node. The whole process should be clear from our discussion above. This is extremely efficient especially in the centralized approach.

In case this does not suffice, we follow the same procedure as highlighted in the original scheme.

The Recovery Initiator (assume M_1 is the RI, let the failed node be denoted as M_i, and let round number be j) should send the following message to the replacement member

$$RI \rightarrow M_i : \{CKY'_{i,l}, R, j, 1, HFK_{i,j}, E_{K_i}(\theta_j), Sig_{K_1}(CKY'_{i,l}, R, j, 1, HFK_{i,j}, E_{K_i}(\theta_j))\}$$

Follow the remaining steps to recover the compromised fractional key and one time pad. The message format to exchange the new HFKs is different from the one presented in previous section. The recovered values are not used, and the initialization phase starts again to establish the new binding parameter. Since the recovered values are not used, it is not clear why they should be recovered.

4.4 Formal Analysis of Authenticated Key Agreement Protocol

We analyze the Initial Key Agreement and Key Generation phase. Analysis of the Key recovery and Auxiliary Key Agreement Protocols is similar.

4.4.1 Initialization Phase

SM-Based Initialization

Assumptions

$\forall i$, SM $\mid\equiv \overset{K_i}{\mapsto} M_i$

$\forall i$, M_i $\mid\equiv \overset{K_{SM}}{\mapsto} SM$

$\forall i$, M_i $\mid\equiv$ SM $\mid\Rightarrow \alpha_{i,1}$

$\forall i$, M_i $\mid\equiv$ SM $\mid\Rightarrow \#\alpha_{i,1}$

$\forall i$, M_i $\mid\equiv$ SM $\mid\Rightarrow \theta_1$

$\forall i$, M_i $\mid\equiv$ SM $\mid\Rightarrow \#\theta_1$

$\forall i, j$, M_i $\mid\equiv$ SM $\mid\Rightarrow \#CKY_{i,j} = f^{k-j}(R_i)$

R is a random number. At this stage only SM knows $CKY_{i,k} = R_i, \ldots, CKY_{i,1} = f^{k-1}(R_i)$. M_i knows $f^k(R_i)$. This in conjunction with the belief that R_i is fresh, will help M_i verify that the source of the message is the SM.

Protocol Message

$$SM \rightarrow M_i : \{CKY_{i,1}, I, 1, SM, E_{K_i}(\alpha_{i,1}|\theta_1), Sig_{K_1}(CKY_{i,1}, I, 1, SM, E_{K_i}(\alpha_{i,1}|\theta_1))\}$$

Idealized Message

$$SM \rightarrow M_i : \{\{\alpha_{i,1}|\forall i,j\ , M_i \overset{\theta_1}{\rightleftharpoons} M_j\}_{K_i}, \{\langle H\{\alpha_i|\theta_1\}\rangle_{CKY_{i,1}=f^{k-1}(R)}\}_{K_{SM}^{-1}}\}$$

Analysis

$M_i \ \triangleleft \{\langle H\{\{\alpha_{i,j}|\theta_1\}_{K_i}\}\rangle_{CKY_{i,1}}\}_{K_{SM}^{-1}}$

$M_i \ |\equiv \overset{K_{SM}}{\mapsto} SM$

$M_i \ |\equiv \text{SM} \ |\sim \{\langle H\{\{\alpha_{i,j}|\theta_1\}_{K_i}\}\rangle_{CKY_{i,1}}\}$

Only SM could have said $CKY_{i,1}$ and since this takes place soon after mutual authentication, the message must be fresh as well.

$M_i \ |\equiv \#CKY_{i,1}$

$M_i \ |\equiv \#H\{\{\alpha_{i,j}|\theta_1\}_{K_i}\}$

$M_i \ |\equiv \text{SM} \ |\equiv H\{\{\alpha_{i,j}|\theta_1\}_{K_i}\}$

$M_i \ \triangleleft \{\alpha_{i,1}|\theta_1\}_{K_i}$

$M_i \ |\equiv \text{SM} \ |\equiv\{\alpha_{i,1}|\theta_1\}$

$M_i \ |\equiv \text{SM} \ |\equiv \alpha_{i,1}$

$M_i \ |\equiv \text{SM} \ |\equiv \theta_1$

Distributed Initialization

Assumptions

$\forall i, M_i \ |\equiv \overset{K_1}{\mapsto} M_1$

$\forall i, M_i \ |\equiv \overset{K_{i-1}}{\mapsto} M_{i-1}$

$\forall i, M_i \ |\equiv M_i \overset{CKY_{i,i+1}}{\rightleftharpoons} M_{i+1}$

$\forall i, M_i \ |\equiv M_{i-1} \ |\Rightarrow CKY_{i-1}$

$\forall i, M_i \ |\equiv M_{i-1} \ |\Rightarrow \#CKY_{i-1,i}$

$\forall i, M_i \ |\equiv M_{i-1} \ |\Rightarrow \delta_{i-1}$

$\forall i, M_i \ |\equiv M_{i-1} \ |\Rightarrow \#\delta_{i-1}$

R is a random number. At this stage only GI (M_1) knows $R, \ldots, CKY_i^{'} = f^{k-1}(R)$. M_i knows $f^k(R)$. This in conjunction with the belief that R is fresh, will help the member verify that the source of the message is the GI.

Protocol Message

$$M_i \rightarrow M_{i+1} : \{CKY_i, I, 1, i, E_{K_{i+1}}(\delta_i), Sig_{K_i}(CKY_i|I|1|i|E_{K_{i+1}}(\delta_1))\}$$

$$M_1 \rightarrow M_i : \{CKY_i, I, 1, 1, E_{K_i}(\theta_1), Sig_{K_1}(CKY_i, I, 1, 1, E_{K_i}(\theta_1)\}$$

Idealized Message

1. $$M_i \rightarrow M_{i+1} : \{\{\delta_i\}_{K_{i+1}}, \{\langle H\{\delta_i\}\rangle_{CKY_i}\}_{K_i^{-1}}\}$$

2. $$M_1 \rightarrow M_i : \{\{\forall i, j\ M_i \overset{\theta_1}{\rightleftharpoons} M_j\}_{K_i}, \{\langle H\{\theta_1\}\rangle_{CKY_i}\}_{K_1^{-1}}\}$$

Analysis

1. $M_{i+1} \ \triangleleft \{\langle H\{\delta_i\}\rangle_{CKY_{i,i+1}}\}_{K_1^{-1}}$
   $M_{i+1} \ |\equiv \overset{K_i}{\mapsto} \ M_i$
   $M_{i+1} \ |\equiv M_1 \ |\sim \langle H\{\delta_i\}\rangle_{CKY_{i,i+1}}$
   $M_{i+1} \ |\equiv M_{i+1} \overset{CKY_i}{\rightleftharpoons} M_i$
   $M_{i+1} \ |\equiv \#CKY_i$
   $M_{i+1} \ |\equiv \#H\{\delta_i\}$
   $M_{i+1} \ |\equiv M_i \ |\equiv H\{\delta_i\}$
   $M_{i+1} \ \triangleleft \{\delta_i\}_{K_{i+1}}$
   $M_{i+1} \ \triangleleft \delta_i$
   $M_{i+1} \ |\equiv M_i \ |\equiv \delta_i$

2. $M_i \ \triangleleft \{\langle H\{\theta_1\}\rangle_{CKY_i^{'}}\}_{K_1^{-1}}$
   $M_i \ |\equiv \overset{K_1}{\mapsto} \ M_1$
   $M_i \ |\equiv M_1 \ |\sim \langle H\{\theta_1\}\rangle_{CKY_i^{'}}$
   Only GI could have said $CKY_i^{'}$ and since this takes place soon after mutual authentication, the message must be fresh as well. $M_i \ |\equiv \#CKY_i^{'}$
   $M_i \ |\equiv \#H\{\theta_1\}$
   $M_i \ |\equiv M_1 \ |\equiv H\{\theta_1\}$
   $M_i \ \triangleleft \{\theta_1\}_{K_i}$

$M_i \ \triangleleft \theta_1$

$M_i \ |\equiv M_1 \ |\equiv \theta_1$

4.4.2 Key Generation Phase

Assumptions

$\forall i, j, \ M_i \ |\equiv \overset{K_j}{\mapsto} \ M_j$

$\forall i, j, \ M_i \ |\equiv M_i \overset{\theta}{\rightleftharpoons} M_j$

$\forall l, m :, \ M_l \ |\equiv M_m \ |\Rightarrow HFK_{m,j}$

$\forall l, m :, \ M_l \ |\equiv M_m \ |\Rightarrow \#HFK_{m,j}$

$M_m \ |\equiv \#\theta_1$

Protocol Message

$$\forall 1 \leq l, m \leq n, l \neq m, M_l \rightarrow M_m : \{G, j, l, HFK_{l,j}, Sig_{K_l}(\theta_l | G | j | l | HFK_{l,j})\}$$

Idealized Message

$$M_l \rightarrow M_m : \{\{\langle H\{HFK_{l,j}\}\rangle_{\theta_j}\}_{K_l^{-1}}\}$$

Analysis

$M_m \ \triangleleft \{\langle H\{HFK_{l,j}\}\rangle_{\theta_j}\}_{K_l^{-1}}\}$

$M_m \ |\equiv \overset{K_l}{\mapsto} \ M_l$

$M_m \ |\equiv \mathrm{GI} \ |\sim \langle H\{HFK_{l,j}\}\rangle_{\theta_j}$

$M_m \ \triangleleft \langle H\{HFK_{l,j}\}\rangle_{\theta_j}$

$M_m \ |\equiv M_m \overset{\theta_1}{\rightleftharpoons} M_l$

$M_m \ |\equiv \#\theta_1$

$M_m \ |\equiv M_m \ |\equiv H\{HFK_{l,j}\} \ M_m \ \triangleleft HFK_{l,j}$

$M_m \ |\equiv M_l \ |\equiv HFK_{l,j}$

4.4.3 Unresolved Problems

- Extensive use of public key based computation. Increases number of messages and precludes pre-computation savings.

- A way to localize group membership changes.

This completes the formal analysis of the Authenticated Key Agreement protocol. In the next chapter we will present an optimized version (in terms of computation and messages) of the key agreement protocol that is suitable for broadcast mediums with no storage constraints.

Chapter 5

Efficient Authenticated Key Agreement Protocol

5.1 Motivation

As pointed out in the first chapter, all authentication protocols must satisfy the required security attributes. However the needs of a particular application will dictate the choice of the authentication scheme. The trade-offs are discussed in [CMN99] and [CGI+99]. As far as the protocol presented in the previous chapter is concerned, it satisfies the required and desired security attributes. We can tailor the protocol to individual applications. We have listed some of the overheads:

- Number and size of protocol messages.
- Number of computations - encryptions, signatures, MACs, etc.
- Nature of computation - symmetric *vs* public key encryption, signature *vs* MACs, etc
- Storage requirements. Using MAC for source authentication and using secret keys instead of public keys requires more storage.

In the next section we will present a protocol that has been optimized for satellite communication (no storage constraints; and also in a single LAN segment, or wireless). We therefore minimize the number of messages sent and the computation. This makes the protocol well suited for large dynamic groups. We note that the storage requirement at the SM for this optimized protocol is less than the base version of the AKA because we do not have to maintain a hash sequence of nonces per member. This is explained below.

5.2 Overview of Optimized AKA protocol (AKA-SAT)

It is clear that the protocol presented in the previous chapter suffers from significant message and computation overhead. The computation overhead is due to the use of signatures and public key encryption in the protocol messages. The message overhead follows as a consequence. The protocol messages in the SM-based initialization and in the key generation phase are shown below to highlight this -

Initialization Phase

$$SM \rightarrow M_i : \{CKY_{i,1}, I, 1, SM, E_{K_i}(\alpha_{i,1}|\theta_1)\}$$

The initial one time pad has to be securely sent each member. The binding parameter has to be protected only from non-group members. Using the public key to encrypt both solves the problem albeit inefficiently. To reduce the computation overhead, symmetric encryption can be used in place of a public key encryption. The binding parameter can be encrypted with a common key-encrypting key, while the one-time pads have to individually encrypted.
Note that we no longer need a signature. Since we assume that a message cannot be modified while in transit, the SKEY-based nonce provides authentication. A signature to ensure message integrity is no longer needed.

Key Generation Phase

$$\forall 1 \leq l, m \leq n, l \neq m, M_l \rightarrow M_m : \{G, j, l, HFK_{l,j}, Sig_{K_l}(\theta_l|G|j|l|HFK_{l,j})\}$$

The HFKs can be sent in the clear. A signature computed over the HFK and the binding parameter, ensures message freshness and provides source authentication. There is only one message sent. There is only computation overhead. To preserve message integrity, a signature is applied to the message. Message Authentication Codes (MACs), i.e., key based hash functions can be used instead of signatures. Another solution is to use the SKEY based nonces as in the initialization phase. However we will pursue the use of MACs.
There are two problems - MACs and encryption need keys and while MACs can be used for message integrity, MACs do not without modification to the scheme provide source authentication. For more on this subject please refer to [BCK97] and [PCST01].

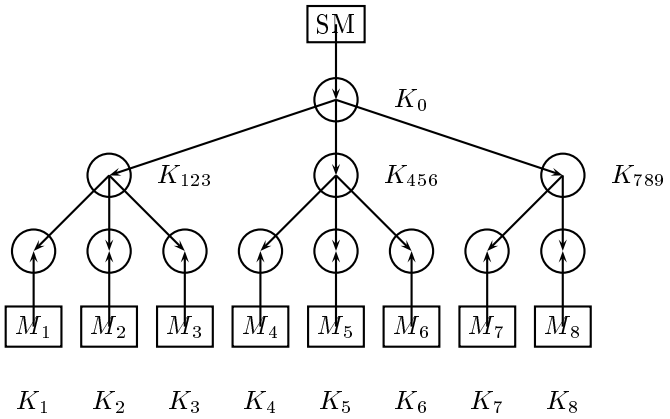

Figure 5.1: LKT structure for keys used in symmetric encryption

To solve the problem of maintaining keys efficiently for use in symmetric encryption we use a Logical Key Tree (LKT) structure (describe in [WGL00]). The keys that form the nodes of the logical key tree can be mathematically related resulting in more efficient schemes. These are beyond the scope of the thesis. Please refer to [MS98] and [MP00] for such schemes.
We need a tree structure to efficiently update these message encrypting keys ($K_{MEK}s$).
The binding parameter can be encrypted with K_{MEK_0} because it is known to all members.
The $K_{MEK}s$ corresponding to the leaf nodes are shared between the SM and that member.
The SM uses these keys during the auxiliary key agreement protocol to encrypt the initial one-time pad. SM also uses these keys in all cases when a member leaves the group, to encrypt the new key values for key-oriented-rekey in that sub-tree. We need only a star structure for the MAC keys. One key that the SM uses to compute the MAC on the protocol messages it broadcasts during the initialization phase, and is known to all group members. The other keys are used by each member to compute the MAC over its hidden contribution to the shared session key. Now these $K_{MAC}s$ should be distributed to other members. At the same time they should be revealed only at the time of sending the message, to provide source authentication. Therefore each member computes a chain of hashes to be used as keys in the MAC, in order, starting from the last. Since the keys used to compute the MAC are in a chain, it enables each group member to verify that the message received apparently from another member is in fact from that member. The hashed values of the $K_{MAC}s$ for all members - $h_1, \ldots, h_n$ - are stored at the SM for distribution to new members.
The keys used for computing MACs can be used for just one message, after which both sender

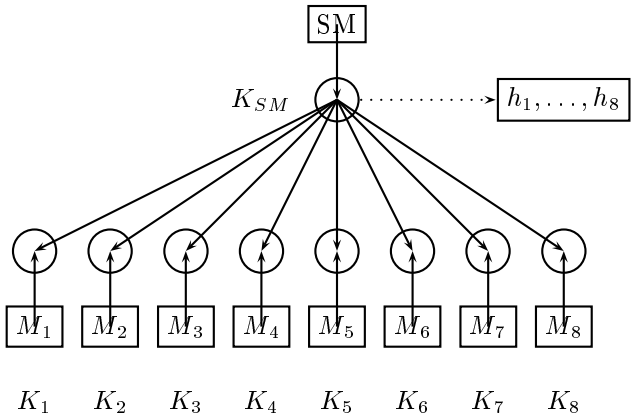

Figure 5.2: Star structure for keys used to compute MAC

and receiver know the key and hence the MAC based on this key can't be used to distinguish between the two. To account for this, the keys in the MAC-tree belong to a hash sequence similar to the one used to guarantee freshness in the protocol described in the previous chapter. We shall present the complete protocol right from the mutual authentication stage. This is because the mutual authentication step will also involve the transfer of the one-time pad, the relevant keys in the MEK-tree, the set of hashed keys from the MAC-tree and the nonce that will guarantee the freshness of messages sent by the SM during later protocol runs. The new user will also send his hashed K_{MAC} to the SM. We have made a small change to the protocol in that the SM does not maintain one nonce sequence per user but just one for the whole group. We could not do this in the previous case because there were n messages sent by the SM, one for each member. If we shared the nonce among members, the first member to receive the message could use this nonce to send messages to other members, thereby attempting to spoof the SM. The other members would not be able to verify the signature of the SM on the message and would reject his message but this has created a wonderful opportunity for a denial of service. It is precisely to prevent his attack that the nonce was sent in the clear. Then a group member has to just compute a hash to verify the freshness of the message. The argument against denial of service in such a scenario is possible only in a broadcast scenario where it is difficult to modify/drop messages sent by other buts easier to re-send or attempt to spoof another member.

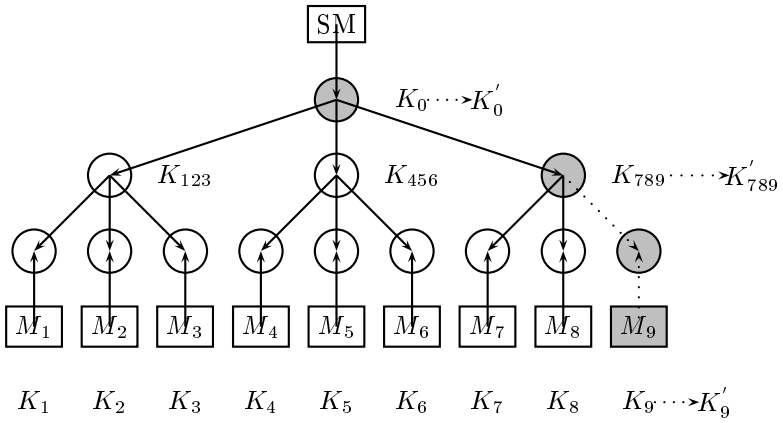

Figure 5.3: Protocol AKA-SAT - initial group

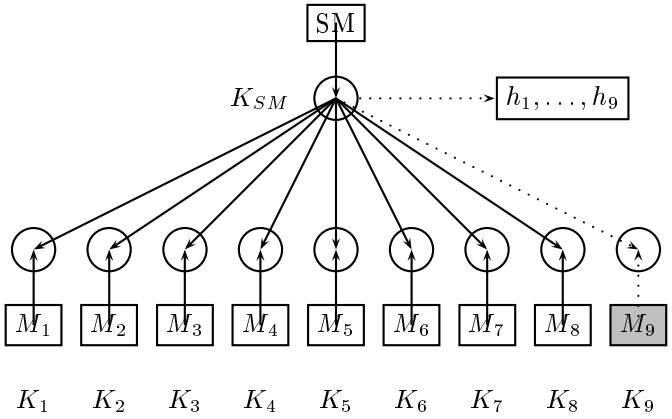

Figure 5.4: Protocol AKA-SAT - member join

5.3 Protocol AKA-SAT

We describe the protocol by listing protocol messages when a new member joins the group. Protocol messages for other scenarios are easy to extrapolate. We show below a group originally consisting of eight members $M_1, \ldots, M_8$. The new member is M_9.
The shaded nodes represent the pair of key that have to be changed when M_9 joins the group. The list of messages now are:

1. **Mutual Authentication**

   Assume that the SM has participated in m rounds.

   $M = \{R_9, I_9\}$

   $M_9 \rightarrow SM : \{M, Cert_9, Sig_{K_9}(M)\}$

   $M' = \{R_{SM}, R_9, I_{SM}, CKY =$

$f^{k-m}(R), E_{K_9}(\alpha_{9,1}, K^{'}_{MEK_0}, K^{'}_{MEK_789}, K_{MEK_9}), h_{SM}, h_1, \ldots, h_8\}$

$SM \rightarrow M_9 : \{M^{'}, Cert_{SM}, Sig_{K_{SM}}(M^{'})\}$

$M = \{R_{SM}, E_{K_{SM}}(h_9)\}$

$M_9 \rightarrow$ SM: $M, Sig_{K_9}(M)$

We make the following observations:

- The binding parameter θ_1 could have been sent to the new member during mutual authentication but this is inefficient in a broadcast medium where message can be encrypted with a shared secret key, when the new binding parameter has to be anyway sent to other members.
- The SM has to store all the $K_{MEK}s$ and the current value of the $K_{MAC}s$ of each member. The SM can pick the $K_{MAC}s$ values from the protocol messages exchanged among the group members.
- The SM does not maintain a per member nonce chain.
- Only the Message Encrypting Keys ($K_{MEK}s$) need to be changed.

2. **Initialization**

   Note that we are using the auxiliary key agreement protocol during membership change. This means that except for a random and small subset of members, the rest do not receive new initial one-time pads. We show the broadcast message to these members below:

   $M = \{CKY = f^{k-m-1}(R), E_{K^{'}_{MEK_0}}(\theta_1 | K_{MAC_{SM}}), E_{K_{MEK_0}}(K^{'}_{MEK_0},$

   $E_{K_{MEK_{789}}}(K^{'}_{MEK_{789}}), h_9\}$

   SM broadcasts: $\{M, MAC_{K_{MAC_{SM}}}(M)\}$

   The presence of CKY helps the member verify the freshness of the message. Members $M_1, \ldots M_6$ will be able to decode only some parts of the message. Appropriate headers in the packet or a fixed packet format is needed.

3. **Key Generation**

   $M = \{G, i, j, HFK_{i,j}, E_{K_{MAC_0}}(K_{MAC_i})\}$ M_i broadcasts: $\{M, MAC_{K_{MAC_i}}(M)\}$

If we assume that an active attacker cannot modify messages in transit in a broadcast medium, then we need not encrypt the key used to compute the MAC, in the message.

Thus we have developed a protocol suitable for large dynamic groups in a broadcast medium. The gain comes at the cost of extra storage at each member node and a tremendous increase in complexity.

Chapter 6

Conclusion

This thesis addresses authenticated key agreement in dynamic multicast groups. The protocol enables group members to compute a joint secret key that can subsequently be used for securing data communication. The protocol provides key authenticity and the underlying key generation scheme provides perfect forward secrecy. The key generation scheme is quite general and is suitable for single sender-multiple receiver as well as multiple sender-multiple receiver models. We however are not able to remove the dependence on the assumption of a public key infrastructure and the presence of centralized access control. However the use of public keys can be restricted to the initialization phase by using symmetric keys for encrypting the protocol messages. The keys used for encrypting the protocol messages and ensuring the integrity of the communication during the protocol run are quite distinct from the shared secret key the protocol helps the group members compute. Public key encryption naturally provides source authentication. In order to provide source authentication for protocol messages (a very necessary requirement), we need to use a different key for each protocol run. Also the source must commit to the key that he is going to use to encrypt a particular message without revealing it, apriori. Since we use these schemes only during the initialization phase, we do not expect serious synchronization problems. We have not addressed the issue of source authentication for data communication. This is an orthogonal issue and we can fit in a suitable scheme in future. This leads to a more a complex protocol. We have not analyzed it formally for weaknesses. It is also clear from this section that we can fine-tune the protocol for different networks and different applications.

Efficiency of multicast re-keying schemes is measured by several parameters - communication

complexity, computation, user storage, center storage and time complexity. In this paper we have developed a general protocol and then traded user and center storage for lesser computation and time complexity. This is suitable for the application we have in mind but will no longer hold, when we change the network or application. We believe that one direction for future research is to investigate these issues deeper and develop optimal schemes for different networks and different applications. Source authentication for data communication is still an open issue and to deal with it in a heterogeneous network is another interesting line to pursue. Also measures need to be incorporated in the protocol to make it robust in the presence of network failures and communication errors.

Multicast security is an interesting and challenging area that is gaining prominence because of killer applications. We have attempted to provide a solution for one specific problem. The matter is not closed an there is scope for a lot if interesting work.

Bibliography

[AST00] Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4), 2000.

[BAN96] Burrows, Abadi, and Needham. A logic of authentication, from proceedings of the royal society, volume 426, number 1871, 1989. In *William Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Society Press, 1996.* 1996.

[BCK97] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. HMAC: Keyed-hashing for message authentication. Technical Report 2104, 1997.

[BF97] D. Boneh and M. Franklin. Efficient generation of shared RSA keys. *Lecture Notes in Computer Science*, 1294:425–439, 1997.

[BWM98] Simon Blake-Wilson and Alfred Menezes. Authenticated diffie-hellman key agreement protocols. In *Selected Areas in Cryptography*, pages 339–361, 1998.

[CGI+99] Canetti, Garay, Itkis, Micciancio, Naor, and Pinkas. Multicast security: A taxonomy and some efficient constructions. In *INFOCOM: The Conference on Computer Communications, joint conference of the IEEE Computer and Communications Societies*, 1999.

[CMN99] Ran Canetti, Tal Malkin, and Kobbi Nissim. Efficient communication-storage tradeoffs for multicast encryption. In *Theory and Application of Cryptographic Techniques*, pages 459–474, 1999.

[HT99] T. Hardjono and G. Tsudik. Ip multicast security: Issues and directions, 1999.

[JV96] Just and Vaudenay. Authenticated multi-party key agreement. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology*. LNCS, Springer-Verlag, 1996.

[LMQ+98] Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, and Scott Vanstone. An efficient protocol for authenticated key agreement. Technical Report CORR 98-05, 1998.

[MP00] Refik Molva and Alain Pannetrat. Scalable multicast security with dynamic recipient groups. *ACM Transactions on Information and System Security*, 3(3):3, 2000.

[MS98] David A. McGrew and Alan T. Sherman. Key establishment in large dynamic groups using one-way function trees. Manuscript, 1998.

[PCST01] Adrian Perrig, Ran Canetti, Dawn Song, and J.D. Tygar. Efficient and secure source authentication for multicast. In *Network and Distributed System Security Symposium*, 2001.

[Poo99] Radha Poovendran. *Key Management for Secure Multicast Communication*. PhD thesis, University of Maryland, College Park, 1999.

[Sch96] Bruce Scheneir. *Applied Cryptography*. John Wiley and Sons, New York, second edition, 1996.

[STW00] M. Steiner, G. Tsudik, and M. Waidner. Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769–783, 2000.

[WGL00] Wong, Gouda, and Lam. Secure group communications using key graphs. *IEEETNWKG: IEEE/ACM Transactions on NetworkingIEEE Communications Society, IEEE Computer Society and the ACM with its Special Interest Group on Data Communication (SIGCOMM), ACM Press*, 8, 2000.