Department of Computer Science
Rochester Institute of Technology

MS Project Final Report

Group Key Agreement Protocols with Implicit Key Authentication

Jisoo Kim
jsk4445@cs.rit.edu

July 13, 2005

Committee Members:

Chairman : *Stanisław P. Radziszowski*	Date
Reader : *Hans-Peter Bischof*	Date
Observer : *Alan Kaminsky*	Date

Abstract

There have been numerous studies performed on secure group communication over unsecured channels such as the Internet and ad-hoc network. Most of the results are focused on cryptographic methods to share secret keys within the group. In the real world, however, we cannot establish an application for group communication without considering authentication of each peer (group member) since the adversary could digitally disguise itself and intrude into the key sharing process without valid membership. Therefore, authentication is an inevitable component for any secure communication protocols as well as peer group communication.

In the classical design of group key protocols, each peer should be authenticated by a separate and centralized authentication server (e.g. Kerberos). Although many practical protocols present efficient ways for authentication, we are still facing the necessity of optimization between authentication and group key sharing. In that sense, implicit key authentication is an ideal property for group key protocols since, once it is possibly put into practice, we do not need any separate authentication procedure as a requisite.

There was an attempt to devise implicit key authentication service in conjunction with group key agreement protocol: Authenticated Group Diffie-Hellman (A-GDH) and its stronger version (SA-GDH). Unfortunately, both were proved to have some weakness from the man-in-the-middle attacks. In this project, practical fixes for A-GDH and SA-GDH using Message Authentication Code (MAC) schemes are proposed and performance evaluation is carried out from implementation and experimentation for each: A-GDH, SA-GDH, A-GDH with MAC, and SA-GDH with MAC. Finally, the policies how and where to deploy authenticated GDH protocols are discussed under various group communication scenarios.

Contents

1 Introduction

Since 2-party Diffie-Hellman key exchange was first proposed in 1976 [5], its contributory nature has attracted many cryptographers into trying to extend it to a group setting. Among those efforts, Group Diffie-Hellman (GDH) in [18] is thought as one of the successful extensions of Diffie-Hellman to the n-party case. There are several versions of GDH, among which GDH.2 and GDH.3 are considered as practical group protocols [18]. Nevertheless, GDH cannot stand alone, as other key sharing protocols, since authentication of each peer (group member) should precede the group key sharing procedure in the real-world applications. Although there are useful authentication techniques for group communication, most of them depend upon a centralized server, *trusted third party*, and require a separate authentication mechanism for every session. This not only increases communication costs but also deteriorates security of the protocol.

Implicit Key Authentication, which will be discussed in the next section, is an ideal property for secure group communication since it eliminates a separate authentication process ahead of key sharing. In [2] and [3], Ateniese *et al.* proposed authenticated versions of GDH, which support the property of implicit key authentication; Authenticated GDH (A-GDH.2) and Strong Authenticated GDH (SA-GDH.2). Several years after A-GDH.2 and SA-GDH.2 were proposed, however, Pereira and Quisquater [15] proved that the authenticated versions of GDH still have weakness from message forgery attacks.

This project proposes fixes for A-GDH and SA-GDH using Message Authentication Code (MAC) schemes, which will protect the protocols from an active attack to forge the messages. The Keyed-Hash Message Authentication Code (HMAC) is plugged in the protocols, associated with MD5 and SHA-1 hashing algorithms. Experimental performance evaluation for each protocol will be given with various parameters in Section 3.7.

The Group Diffie-Hellman (GDH) protocol supports CLIQUES, a dynamic peer group protocol suite, operations defined in [19]. CLIQUES defines various group operations such as a member joining/leaving and group merging/partitioning. It can be conjuncted with authenticated GDH protocols under some authentication policies. In this project, it is also studied what policy should be considered in a group communication application on the usage of authenticated GDH protocols and how we can possibly apply implicit key authentication property to CLIQUES.

2 Notations, Definitions, and Security Properties

2.1 Notations for authenticated GDH protocols

The notation throughout this project is mostly taken from [2] and [3], and all arithmetic operations are performed in a cyclic group G of prime order q which is a unique subgroup of $\mathbb{Z}_p^*$, the multiplicative group of integers modulo p, for a prime p such that $p = kq+1$ for some small $k \in \mathbb{N}$ (*e.g.* $k = 2$). For compactness, *modulo* p is omitted in all arithmetic expressions. However, in case of *modulo* q, it is specified as *mod* p. The notation used in this project is given in Table 1.

Since the unique subgroup G is a cyclic group of finite order q, *i.e.* a finite group

n	number of group members
i, j	indices of group members
M_i	i-th group member; $i \in [1, n]$
p, q	p, q prime, $q \mid \phi(p)$
G	unique subgroup of $\mathbb{Z}_p^*$ of order q
α	exponentiation base; generator of G
x_i	long-term secret key of M_i
r_i	M_i's secret exponent (nounce) $\in \mathbb{Z}_q^*$
S_n	group key shared among n members
$S_n(M_i)$	M_i's view on a group key
K_{ij}	long-term secret shared by M_i and M_j
$F()$	a function from G to $\mathbb{Z}_q$
I_{ij}	intermediate value computed by M_i for M_j
$MAC_K(text)$	MAC tag computed from *text* using the shared secret K between sender and receiver
$text_A \parallel text_B$	concatenation of two text $text_A$ and $text_B$

Table 1: Notation

with q elements that can be generated by a single element called a group generator, α, we have that $\alpha^q = 1 \pmod p$ and $\alpha^i \neq 1 \pmod p$ where $i \in [1, q-1]$. In this project, p has been chosen to be a safe prime, $p = 2q + 1$, for a large prime q. Therefore, it is true that $\phi(p) = p - 1 = 2q$, for the Euler's totient function $\phi(x)$ defined as the number of positive integers not greater than and relatively prime to x. This gives us that there exist two subgroups of $\mathbb{Z}_p^*$, orders of which are 2 and q. Only the subgroup G of order q will be used.

The long-term secret K_{ij} is defined as $F\Big(\alpha^{x_i x_j} \pmod p\Big)$, where $F \in \mathcal{F} : G \longrightarrow \mathbb{Z}_q$. Note that $K_{ij} = K_{ji}$ and that $\alpha^{x_i} \pmod p$ is the long-term public key of M_i for given long-term secret x_i. If M_j knows M_i's public key $\alpha^{x_i} \pmod p$, she/he can compute K_{ij} from the long-term secret x_j, for the given public parameters, p, q, α, and the function F.

2.2 Definitions and security properties

The underlying security of the protocols which will be discussed in this project is based on the hardness of the well-known Discrete Logarithm problem, which is presumed to be intractable [10].

Definition 1 *There are given a modulus number p and a base $\alpha \in \mathbb{Z}_p^*$. **Discrete Logarithm (DL)** is a problem of finding x for a given number $y \in \mathbb{Z}_p$ such that $\alpha^x = y \pmod p$.*

There is another well-known problem, Computational Diffie-Hellman, and its definition is given in Definition 2.

Definition 2 *There are given a prime p and a base $\alpha \in \mathbb{Z}_p^*$.* ***Computational Diffie-Hellman (CDH)*** *is a problem of finding the value of $\alpha^{ab} \pmod p$, for known $\alpha^a \pmod p$ and $\alpha^b \pmod p$,.*

CDH is Turing-reducible to DL [20] (*i.e.* CDH is at least as hard as DL.). Therefore, there is no practical way to solve a CDH problem as long as it is infeasible to solve DL.

Every protocol in this project is extended to group settings from the 2-party Diffie-Hellman (DH) protocol, which is defined between M_1 and M_2 with two public parameters, a prime number p and $\alpha \in \mathbb{Z}_p^*$, as following:

Step 1: M_1 generates a random secret $r_1 \in \mathbb{Z}_{p-1}^*$ and sends $\alpha^{r_1} \pmod p$ to M_2 via an unsecured channel.

Step 2: M_2 generates a random secret $r_2 \in \mathbb{Z}_{p-1}^*$ and sends $\alpha^{r_2} \pmod p$ to M_1 via an unsecured channel.

Step 3: Finally, M_1 and M_2 can compute the common secret $\alpha^{r_1 r_2} \pmod p$ without revealing their secrets, r_1 and r_2, respectively.

As one can see, breaking the 2-party Diffie-Hellman protocol is equivalent to solving the Computational Diffie-Hellman (CDH) problem so that the eavesdropper has no practical way to compute the shared secret $\alpha^{r_1 r_2}$ from α^{r_1} and α^{r_2}. DH is distinctive of its collaborative nature, which means that each party equally contributes to the key generation. In another well-known public key system, RSA, which can be also used to share a secret between two parties, either of participants should generate the secret and send it to the other in an encrypted message. In that sense, DH is more democratic and more suitable for the communication on an equal footing.

DH protocol without message authentication[1] is known to be vulnerable to an active attack, called the *man-in-the-middle attack*. An adversary M_{bad} can perform the man-in-the-middle attack by forging the messages between M_1 and M_2 as follows:

Step 1: M_{bad} generates a random number $x \in \mathbb{Z}_{p-1}^*$.

Step 2: M_{bad} captures the message $\alpha^{r_1} \pmod p$ sent by M_1 and sends a forged message $\alpha^x \pmod p$ to M_2, instead of $\alpha^{r_1} \pmod p$.

Step 3: M_{bad} captures the message $\alpha^{r_2} \pmod p$ sent by M_2 and sends a forged message $\alpha^x \pmod p$ to M_1, instead of $\alpha^{r_2} \pmod p$.

Step 4: M_1 and M_2 compute $\alpha^{xr_1} \pmod p$ and $\alpha^{xr_2} \pmod p$, respectively, as their shared secrets. Now, M_{bad} can compute both $\alpha^{xr_1} \pmod p$and $\alpha^{xr_2} \pmod p$ from the captured message $\alpha^{r_1} \pmod p$ and $\alpha^{r_2} \pmod p$.

Because of the vulnerability of 2-party DH, it is very important to find potential weakness under various assumptions so that the protocol developer can prevent any possible security holes. The commonly desired properties we have to consider to design a secure group communication protocol are:

[1]Message authentication provides two services. It provides a way to ensure message integrity and a way to verify who sent the message.

- Key Authentication
- Perfect Forward Secrecy (PFS)
- Resistance to Known-Key Attacks (RKKA)

The properties above are considered as necessary to achieve resistance to active attacks as well as passive attacks [3]. *Key authentication* is the problem of being certain that a public key is trully that of a designated member. This can be provided by Public Key Infrastructure (PKI) which will be discussed in Section 3.5. Most of all, PFS and RKKA are considered as important issues in security analysis on a cryptographic protocol and their definitions are given in the following.

Definition 3 *A protocol offers* ***perfect forward secrecy (PFS)*** *if compromise of a long-term key(s) cannot result in the compromise of past session keys.*

Definition 4 *A protocol offers* ***resistance to known-key attacks (RKKA)*** *if compromise of session key cannot result in 1) the compromise of other session keys by a passive adversary or 2) impersonation as one of the protocol parties by an active adversary.*

A-GDH and SA-GDH, which will be discussed in Section 3, have been designed to provide all the properties mentioned above (see [2] and [3] for their own proofs that the protocols satisfy those protperties). However, it has been proved by Pereira and Quisquater [15] that the original design of authenticated GDH does not provide all the secure properties.

Authenticated versions of GDH are distinguished from other group key distribution protocols by the following characteristics:

- Key Agreement Protocol
- Contributory Protocol
- Providing Implicit Key Authentication

The definitions for the above are also given below (these are adapted from [2] or [3]).

Definition 5 *A* ***key agreement protocol*** *is a key establishment technique whereby a shared secret key is derived by two or more specified parties as a function of information contributed by, or associated with, each of these, such that no party can predetermine the resulting value.*

Definition 6 *A key agreement protocol is* ***contributory*** *if each party equally contributes to the key and guarantees its freshness.*

Definition 7 *Let $\mathcal{R}$ be an n-party key agreement protocol, $\mathcal{M}$ be the set of protocol parties and let S_n be a secret key jointly generated as a result of $\mathcal{R}$. We say that $\mathcal{R}$ provides* ***implicit key authentication*** *if each $M_i \in \mathcal{M}$ is assured that no party $M_q \notin \mathcal{M}$ can learn the key S_n unless aided by a dishonest $M_j \in \mathcal{M}$.*

[2] and [3] present the proof that A-GDH and SA-GDH provide implicit key authentication. As mentioned above, this property can remove the need for a separate authentication mechanism during group key sharing.

3 Authenticated Group Diffie-Hellman Protocols

In this section, we will discuss A-GDH, SA-GDH, and their fixes by using MAC. The discussion will be mainly on *Initial Key Agreement* (IKA), the very first group key agreement at the time of group *genesis*. *Auxiliary Key Agreement* (AKA) refers to all subsequent key agreement operations through membership changes after the initial group key is established. (See the details of IKA and AKA in [19]). Note that authentication during member joining or group merging depends on the protocol policies, related to the problem of *who authenticate(s) whom*. The authentication policies under various cases will be discussed in Section 4.

Remind that every arithmetic expression is under the operation of *modulo* p but it is omitted for compactness.

3.1 A-GDH protocol

A-GDH in this project refers to A-GDH.2 proposed in [2] and [3]. The protocol's specification is given in Figure 1. Note that I_{ij} is the intermediate value that is updated by M_i and will be used for M_j to compute the group key (at broadcast step). The notation using I_{ij} looks very different from the original papers, [2] and [3]. However, Figure 1 illustrates the same protocol in the view of each intermediate value updated in sequence so that it is more intuitive when an indexed data structure (*e.g.* array type) for sequence of intermediate values is used to implement the protocol.

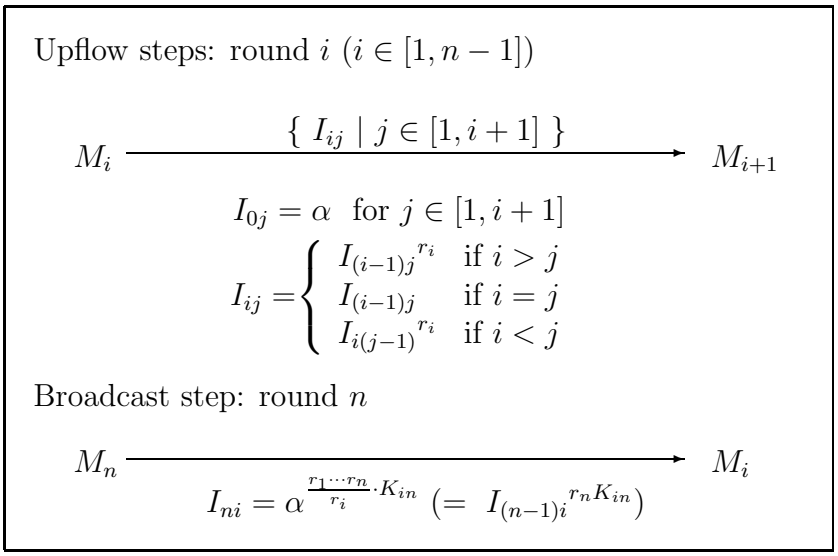

Figure 1: A-GDH protocol

As one can see, the upflow stage (*i.e.* the protocol rounds from 1 to $n-1$) is identical to GDH.2, a non-authenticated protocol, in [18] and [19]. The only difference is that, on the broadcast stage, the group controller (the last member M_n) sends the intermediate value I_{ni} to each member M_i ($i \in [1, n-1]$), hiding the value with the long-term secret key between M_i and M_n, $K_{in} = F(\alpha^{x_i x_n} (mod\ p))$. Since x_i and x_n are the long-term secrets of M_i and M_n, respectively, K_{in} can be computed only by M_i and M_n on the assumption

that long-term public keys α^{x_k} ($k \in [1, n]$) are already published. The problem to find α^{ab} from given α^a and α^b is the Computational Diffie-Hellman problem as defined in Section 2.2.

The last round of this protocol guarantees that no one without an appropriate long-term key could compute the same group key. For example, M_i receives $\alpha^{\frac{r_1 \cdots r_n}{r_i} \cdot K_{in}}$ from the group controller M_n. M_i can compute the group key $S_n = \alpha^{r_1 \cdots r_n}$ only if M_i can compute K_{in}^{-1} where $K_{in} \cdot K_{in}^{-1} = 1(mod\ q)$. If M_j cannot compute K_{in}, which means the member does not know the long-term secret x_i, K_{in}^{-1} cannot be calculated, neither. Therefore, $M_I \notin G$ disguised as M_i cannot compute the group key without knowing x_i. This not only satisfies the property of implicit key authentication but also provides resistance to known-key attacks (*i.e.* it prevents illegal impersonation).

However, A-GDH is a relatively weak form of implicit key authentication since the authentication is performed only between M_i and M_n, $i \in [1, n-1]$. An example of A-GDH with four parties is given in Figure 2. If we take a snapshot in the view of M_3, we can see that M_3 receives from M_2 the sequence of intermediate values $\{I_{21}, I_{22}, I_{23}\} = \{\alpha^{r_2}, \alpha^{r_1}, \alpha^{r_1 r_2}\}$ and sends $\{I_{31}, I_{32}, I_{33}, I_{34}\} = \{\alpha^{r_2 r_3}, \alpha^{r_1 r_3}, \alpha^{r_1 r_2}, \alpha^{r_1 r_2 r_3}\}$ to M_4. Note that there is no authentication mechanism among the members until the last member M_n broadcasts $I_{ni} = \alpha^{\frac{r_1 \cdots r_n}{r_i} \cdot K_{ni}}$ to each member M_i.

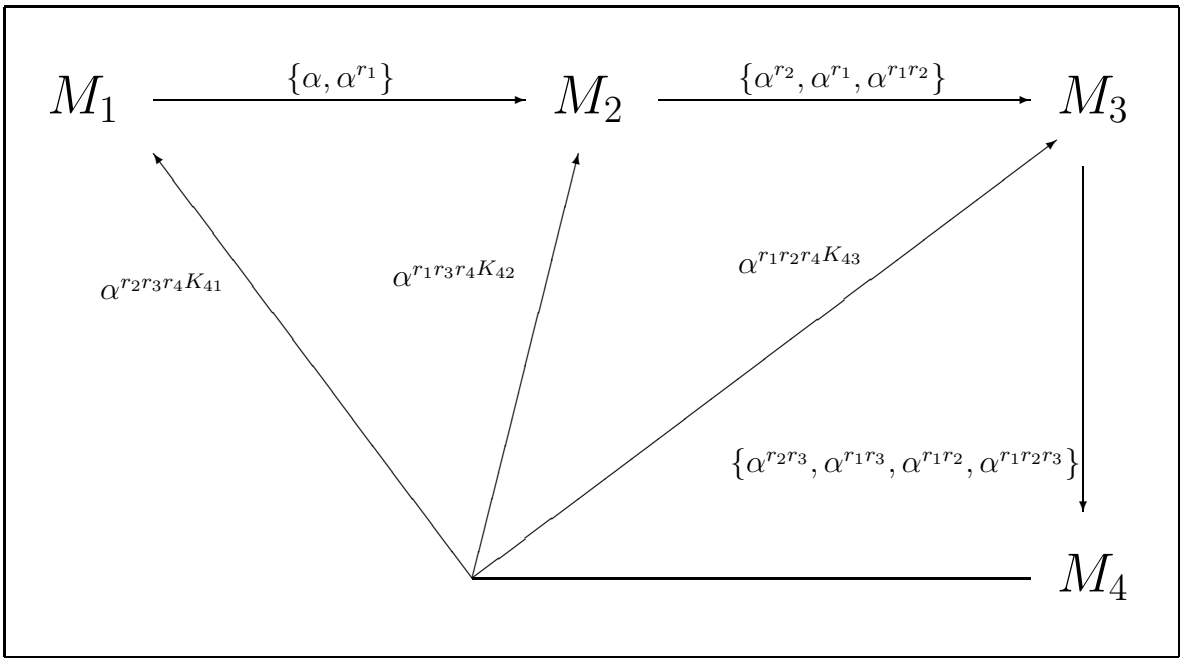

Figure 2: An example of A-GDH with 4 members

3.2 SA-GDH protocol

As mentioned above, A-GDH is not strong enough since contribution to authentication is centralized to the last member (group controller). For example, a dishonest group controller can invite a member M_I, disguised as an intended member M_i, by sending

$\alpha^{\frac{r_1 \cdots r_n}{r_I}}$ instead of $\alpha^{\frac{r_1 \cdots r_n}{r_I} \cdot K_{ni}}$ or by revealing the value of K_{ni} to M_I. In order to provide stronger authentication service, Ateniese *et al* also proposed SA-GDH, named SA-GDH.2 in [2] and [3], in which every member contributes to the authentication of the others. The protocol is illustrated in Figure 3.

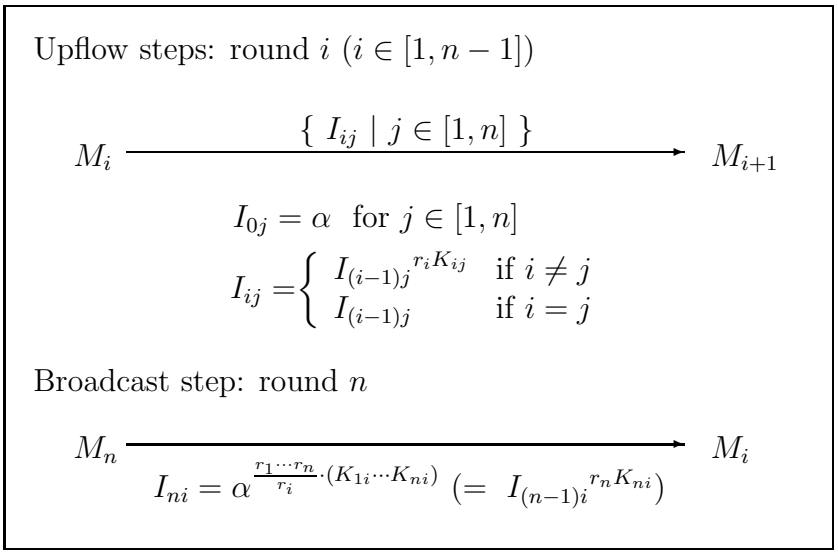

Figure 3: SA-GDH protocol

Unlike A-GDH, SA-GDH requires each intermediate value I_{ki} to be computed by using K_{ki} $(0 < i \neq k \leq n)$ in the upflow stage as well so that each member can not only equally contribute to authentication but also be explicitly aware of the exact group membership when the protocol is initiated. Another advantage of SA-GDH over A-GDH is that each member performs the same sequence of computational steps and the same number of exponentiations since static number of intermediate values computed by M_i are sent to the next member M_{i+1}. For example, M_{i+1} in SA-GDH receives from M_i a sequence of intermediate values $\{I_{i1}, I_{i2}, \ldots I_{in}\}$, instead of $\{I_{i1}, I_{i2}, \ldots I_{i(i+1)}\}$ in A-GDH. An example of SA-GDH with four members is given in Figure 4. Obviously, SA-GDH is more expensive in computation of intermediate values than A-GDH.

Although SA-GDH has higher cost of computation than A-GDH, if the group settings (*e.g.* group membership, communication order of each member, etc) are determined in advance of the protocol's beginning (this seems to be a common situation in group communication), the computational costs can be mostly saved by pre-computation. Let I_{ij} be the intermediate value for M_j updated by M_i, then M_i computes I_{ij} as (1).

$$I_{ij} = \begin{cases} {I_{(i-1)j}}^{E_{ij}} & \text{if } i \neq j \\ I_{(i-1)j} & \text{if } i = j \end{cases} \tag{1}$$

$$E_{ij} = r_i \cdot K_{ij} \pmod{q} \tag{2}$$

As one can see in (2), the exponent E_{ij} can be pre-computed at the preliminary stage of key agreement protocol. Therefore, we can minimize the additional computation on SA-GDH in a practical way.

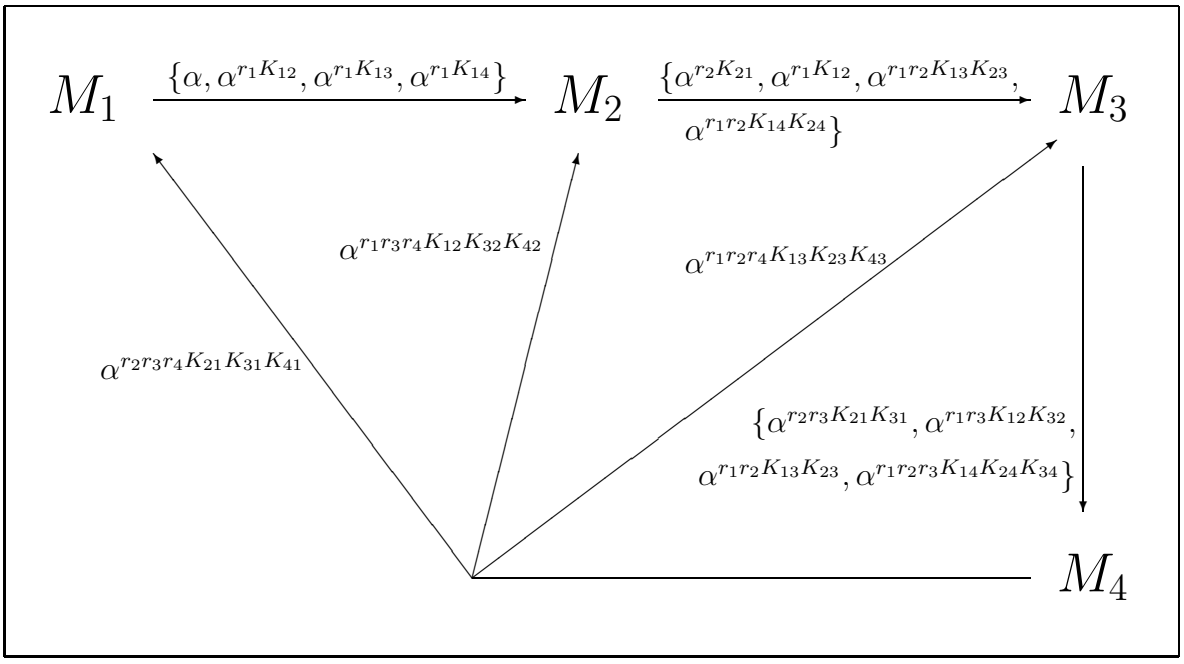

Figure 4: An example of SA-GDH with 4 members

3.3 Attacks on authenticated GDH protocols

Pereira and Quisquater [13] provided security analysis on authenticated GDH protocols. First, they tried to find attacking schemes on A-GDH in conjunction with the major secure properties: implicit key authentication, perfect forward secrecy, and resistance to known-keys attacks. For example, the active adversary can perform an attack on A-GDH against PFS, as illustrated in Figure 5, on the assumption that the long-term secret K_{41} is compromised. The adversary replaces $I_{31} = \alpha^{r_2r_3}$ with $I_{34} = \alpha^{r_1r_2r_3}$. By capturing $I_{41} = \alpha^{r_1r_2r_3r_4K_{41}}$ in the broadcast message, the adversary can finally compute the group key $\alpha^{r_1r_2r_3r_4}$ since K_{41} has been already compromised. M_1 is isolated from other members as intended by the adversary who impersonates M_1. In fact, this scenario is possible on compromise of any member's long-term secret.

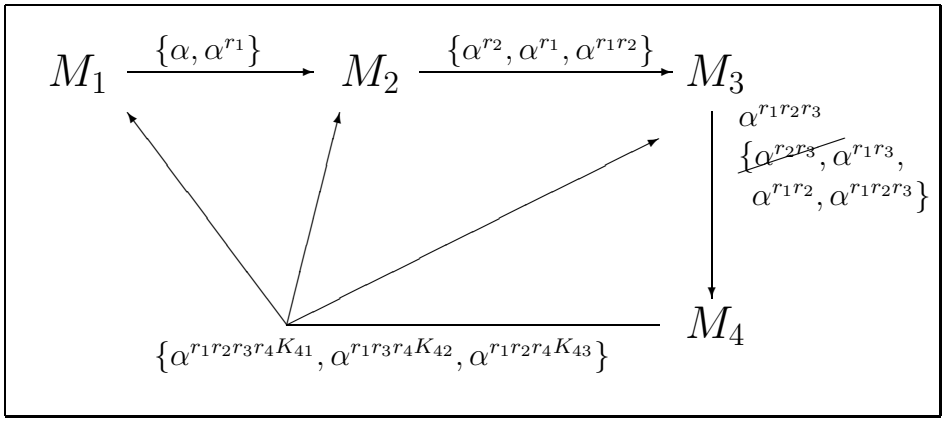

Figure 5: Attack against PFS in A-GDH

Two years later, Pereira and Quisquater published another paper [14] including some attacks on SA-GDH as well as A-GDH. The effect by attack on SA-GDH is more limited than on A-GDH so that it just results in isolating a single member who is fooled into getting a group key which is wrong, but known to the adversary. Finally, they proposed the proof on the generic insecurity of authenticated GDH protocols in 2004 [15]. They presented the proof that it is impossible to design a scalable authenticated group key agreement protocol based on the same building blocks as the authenticated GDH ones under some conditions.

3.4 Fixes for A-GDH and SA-GDH by using MAC

Attack techniques mentioned in Section 3.3 have some impractical requirements on the intruder's ability which include the following:

- The intruder can eavesdrop an intended message sent by any participant in the current group session.
- The intruder can forge an intended message sent by any participant without a noticeable delay.

Although both requirements above seem to be more infeasible as group size increases, this shows that A-GDH and SA-GDH cannot satisfy the recommended secure properties mentioned in Section 2.2. One of the solutions suggested by Pereira and Quisquater in the conclusion of [15] is the use of message authentication codes (MAC) which prevents the intruders from forging messages. They thought this separates the key generation part of the protocol from the authentication mechanisms. However, we should not necessarily separate those two parts since the authenticated GDH protocols have the long-term secret K_{ij} which can be implicitly plugged in a MAC scheme.

3.4.1 Keyed-hash message authentication code

The Message Authentication Code (MAC) has been proved to be a reliable technique against active adversaries who can forge messages in the network and is widely used in real-world applications. For example, Secure Socket Layer (SSL) protocol, a secure protocol supported by many web browsers, uses MAC in order to authenticate both the source of a message and its integrity when we shop at the on-line malls or do internet banking.

Among many MAC protocols being used in communication applications, the Keyed-Hash Message Authentication Code (HMAC) is a cryptographic standard adopted by National Institute of Standards and Technology (NIST) of United States [11] and proven to be secure as long as the underlying hash function (*e.g.* SHA-1 in [6], MD5 in [16], etc) has some reasonable cryptographic strengths. In addition, HMAC is included in many programming libraries such as OpenSSL in C++, javax.crypto in Java, and so on.

HMACs require two functionally distinct parameters, a message input and a secret key[2] known only to the message sender and intended recipient. Since the authenticated

[2]In order that HMAC should work, the message sender and the receiver must share a pre-arranged secret key. See the details in [11].

GDH protocols, both A-GDH and SA-GDH, have a secret K_{ij} shared between M_i and M_j, we can design a fix using HMAC, without any separate authentication mechanisms. Therefore, in this project, HMACs associated with the hash functions, SHA-1 and MD5, are used in the implementation and experimentation of the fixed protocols designed to prevent the active adversary from forging the message.

How reliably can HMAC prevent or detect message forgery? Assume that the intruder does not know the long-term secret, then a typical attack by forging a message is to invent different input messages which produces the same MAC tags. The intruder may choose a useful one from the invented messages. This is not an attack on the HMAC protocol itself, but on the underlying hash function. Therefore, as mentioned above, HMAC is considered as safe enough as long as the associated hash function is invulnerable.

There is a well-known attack on hash algorithms, called *Birthday Attack*, which can find a collision, a pair of message with the same output, in $2^{\frac{n}{2}}$ operations for an n-bit hash function [7]. Since MD5 produces a 128-bit hash output and SHA-1 a 160-bit one, a brute-force birthday attack can be performed in 2^{64} operations for MD5 and 2^{80} operations for SHA-1. According to [17], a new attacking method on SHA-1 was proposed in 2005 so that it is possible to find a collision in 2^{69} calculations[3] against SHA-1. Nevertheless, it still sounds very slow. With the attacking schemes of Pereira and Quisquater mentioned in Section 3.3, the intruder should forge the messages in a relatively tiny moment. In addition, birthday attack itself is not enough to perform the attacks on A-GDH and SA-GDH since just a collision does not work for appropriate forgery. As discussed in Section 3.3, the intruder needs specific forged message which produces the exactly same MAC as the original message. This is almost impossible. Besides, HMAC can be more strengthened by hash functions with larger results such as SHA-256, SHA-384, and SHA-512 specified in [12].

3.4.2 Fixes using MAC

In order to prevent messages in the protocol from being forged, a MAC tag can be generated from each intermediate value as shown in Figure 6 and used to verify integrity of the message. Note that the sequence of intermediate values in the up-flow stages (the rounds from 1 to $n-1$) is different between A-GDH and SA-GDH. In the case with four members, for example, the third member M_3 receives $\{I_{21}, I_{22}, I_{23}\}$ in A-GDH and $\{I_{21}, I_{22}, I_{23}, I_{24}\}$ in SA-GDH. If the MAC tags are not valid, the receiver may request the sender (the former member in the protocol sequence) to send the message again or declare the session is void.

The design on Figure 6 is inefficient since the MAC tags are generated for all the intermediate values, which requires higher computational costs and longer messages. Regardless of length of input text, HMAC provides the same security level as long as the length of input is less than $2^B - 8B$ [11], where B is the block size of the input to the approved hash functions. Since $B = 64$ for both MD5 and SHA-1, the maximum length of input text is $2^{64} - 8 \cdot 64$, which is large enough for practical uses. Indeed, the whole sequence of intermediate values can be considered as a single message input to HMAC

[3] About 2,000 times faster than by brute-force

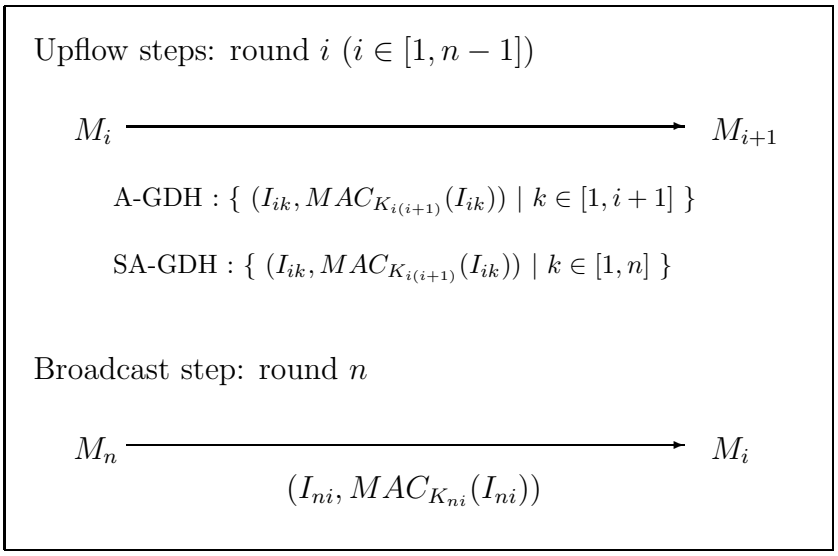

Figure 6: Using MAC in authenticated GDH protocols

as illustrated in Figure 7. However, in the broadcast step, the group controller should generate a MAC tag for each member so that each final intermediate value is protected separately.

As mentioned above, a MAC protocol is possibly vulnerable to the brute-force way to find a preimage, an invented input which produces the same MAC as the original message. However, it seems to be infeasible to attack on the protocols in Figure 6 and Figure 7, following the attacking schemes proposed in [13], [14], and [15], since the synchronous nature[4] of group key agreement protocols does not allow the intruder to have enough time to invent a valid preimage.

The fixes illustrated in Figures 6 and 7 can adopt any keyed MAC protocols. In this project, HMAC with MD5 and SHA-1 hash functions, discussed in Section 3.4.1, was chosen for the implementation and experimentation.

3.5 Public key infrastructure for authenticated GDH protocols

With a strict analysis of the A-GDH and SA-GDH protocols, the group members in the protocols cannot be truly *implicitly authenticated* since sharing of the long-term secret K_{ij} between M_i and M_j is also based on 2-party Diffie-Hellman which is vulnerable to the man-in-the-middle attack. If the intruder wants to break into the system by trying to forge the long-term public keys, then the disguise as an intended member will be very easy. Therefore, for the completed design of authenticated GDH protocols, we need a Public-Key Infrastructure (PKI) in order to authenticate the long-term public keys published.

[4]Authenticated GDH protocols also support asynchronous communication. In this case, however, the implicit key authentication service is not as useful as in the synchronous communication. Instead, the classical design (separate authentication mechanism) without implicit key authentication is more adequate.

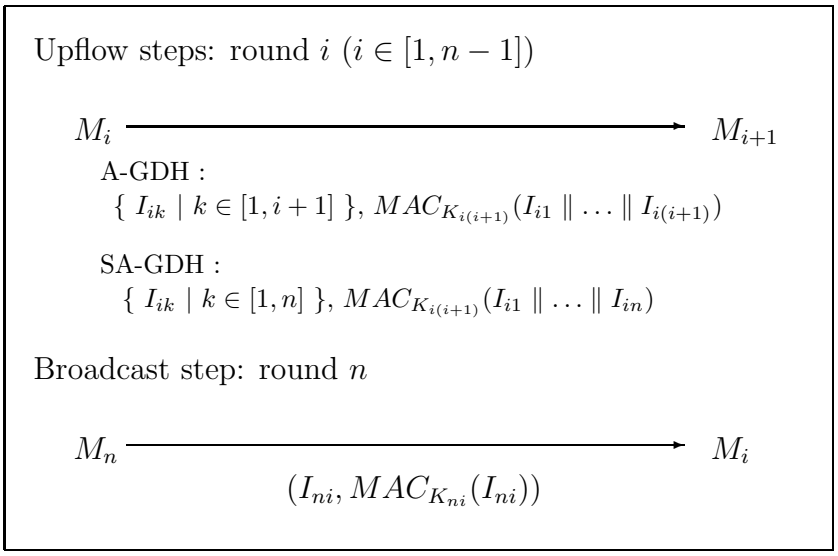

Figure 7: Optimized design for using MAC in authenticated GDH protocols

There is a typical scenario of PKI. When a new membership needs to be created, the new member should register itself to a Certificate Authority (CA), a trusted third party. Once a member is registered, the CA knows what public key belongs to the member. Each member carries and publishes its long-term public key along with the certificate signed [5] by CA so that everybody can verify its validity.

What is the benefit from implicit key authentication if we still need a CA? Once M_i verifies M_j's long-term public key and has a record of it, then M_i does not need the public key verifying step in another communication session with M_j as long as the certificate is valid. If the certificate issued by CA is expired, it should be refreshed periodically just as other PKI systems (*e.g.* the web servers' certificates in SSL). Therefore, in order to maximize the advantage of implicit key authentication, each member would better maintain the list of group candidates' public keys which are likely to be needed. If the group communication happens frequently among specific members, this saves the overhead from authentication of each member at every group session.

The instant messenger program can be a typical application for implicit key authentication protocols based on a PKI system. When a user signs up, the central server (we can consider this as a CA) issues a long-term public key[6] certified by its digital signature. The user can add a member, who has already signed up, to his or her friends list. If the other member agrees on being a *friend*, the long-term public keys are exchanged and verified with CA's digital signature. Once key exchange is successfully done, each user maintains the long-term keys of *friends*. This set-up for secure group communication using the instant messenger program is ready for implicit key authentication.

[5] A digital signature scheme is used for the certificate. Various digital signature schemes are discussed in [10].

[6] The long-term secret, of course, should be also known to the user.

3.6 Theoretical analysis on the protocol costs

The experimental evaluation of the protocols in this project was focused on the computational costs. The experiment for communication cost was excluded for lack of stability in the real-world networks, which are expected to produce widely floating results. Instead, the theoretical analysis will be given on both communication and computation costs.

There are six protocols analyzed and experimented in this project:

- Non-optimized A-GDH : authenticated group key agreement as specified in Figure 1. Long-term keys K_{in} are accumulated and computed as part of the protocol.
- Non-optimized SA-GDH : the stronger version of authenticated GDH as specified in Figure 3 without pre-computing the shared keys K_{ij}. The long-term public keys are assumed to have been shared before the protocol but long-term keys K_{in} are computed during the protocol.
- Optimized A-GDH : same as non-optimized A-GDH but long-term keys K_{in} and K_{in}^{-1} are pre-computed including multiplication operations, on the assumption that the long-term keys are already shared.
- Optimized SA-GDH : same as non-optimized SA-GDH but long-term keys K_{ij} between two members M_i and M_j and K_{ij}^{-1} are pre-computed including multiplication operations, on the assumption that the long-term keys are already shared.
- Optimized A-GDH with MAC : optimized version of A-GDH with HMAC in conjunction with either of MD5 or SHA-1.
- Optimized SA-GDH with MAC : optimized version of SA-GDH with HMAC in conjunction with either of MD5 or SHA-1.

Tables 2 and 3 show, respectively, the communication and computation cost of each protocol. Note that the computation costs for handling MAC with the fixes are not considered in Table 3.

	Non-optimized		Optimized			
	A-GDH	SA-GDH	Without MAC		With MAC	
			A-GDH	SA-GDH	A-GDH	SA-GDH
rounds	n	n	n	n	n	n
broadcasts	1	1	1	1	1	1
total messages	n	n	n	n	n	n
total bandwidth	n^2-1	n^2-1	$\frac{n(n-1)}{2}+2(n-1)$	n^2-1	$\frac{n(n-1)}{2}+4(n-1)$	n^2+2n-3

Table 2: Communication costs of Protocols

3.7 Experimental evaluation on computational costs

The six protocols listed in Section 3.6 were implemented for the experimental evaluation. This evaluation has been performed only on computational cost since the experimental

	Non-optimized		Optimized			
	A-GDH	SA-GDH	Without MAC		With MAC	
			A-GDH	SA-GDH	A-GDH	SA-GDH
exponentiations for M_i	$i+1$	n	$i+1$	n	$i+1$	n
exponentiations for M_n	n	n	n	n	n	n
total exponentiations	$\frac{n(n+3)}{2}-1$	n^2	$\frac{n(n+3)}{2}-1$	n^2	$\frac{n(n+3)}{2}-1$	n^2
inverses for M_i	1	$n-1$				
inverses for M_n		$n-1$				
total inverses	$n-1$	$n(n-1)$				
multiplications for M_i	1	$2(n-1)$				
multiplications for M_n	$n-1$	$2(n-1)$				
total multiplications	$2(n-1)$	$2n(n-1)$				

Table 3: Computation costs of Protocols

evaluation of communication cost is affected by too many environmental variables (*e.g.* LAN vs WAN, service quality provided by ISPs, and so on). However, since the size of each message is not so large, the results from these experiments can be considered to dominate when a group protocol with implicit key authentication is designed and implemented.

The implementation was done in Java which appears, from a simple experiment, to be more efficient in exponentiations of large numbers than C/C++ GMP library. Even though Java is much slower in loading the Java Virtual Machine, Java seems to be better-optimized than the C/C++ GMP library for modular exponentiations of large numbers. Another advantage of Java over C/C++ is that Java is machine-independent so that the experiments would be done on various platforms. The two testbeds in this project are shown in Table 4.

Categories	PC	Sun Microsystems
CPU	Pentium 4	4 X UltraSPARC-II
CPU Speed	2.0GHz	450MHz
System Clock	400MHz	113MHz
Memory	1.0GB	4096MB
OS	Windows XP Professional	Solaris 5.9
URL	*N/A*	holly.cs.rit.edu/hilly.cs.rit.edu

Table 4: Testbeds

3.7.1 Parameters and implementation of the experiments

Determination of the public parameters in the group protocols had to precede the experiments. Table 5 shows the public parameter settings in the experiments. The order of the unique subgroup, q, has been picked up in designated bit size: 512, 768, 1024, 1280, 1536, and 2048. Table 6 shows the bit lengths of parameters in each experiment. Table 7, adopted from [9], shows the recommended field sizes in each year and MIPS years[7] for

[7]A MIPS year is the number of instructions a processor which runs at 1 MIPS would complete in 1 years time. Roughly 1 MIPS Year equates to $3.1536 * 10^{15}$ instructions.

brute-force attack. Unnecessarily big group generators, α, have been experimented with since this can show the upper bound of the time for computations. Note that the length of generators does not affect the protocol security level.

p	$p = 2q + 1$, known as a safe prime
q	order of G, the unique subgroup of $\mathbb{Z}_p^*$
α	exponentiation base; generator of G
$F()$	$F(x) :=$ **if** $(x \leq q)$ **then** x **else** $p - x$ **fi**

Table 5: Public parameters

q	p	α
512	513	510
768	769	767
1024	1025	1024
1280	1281	1280
1536	1537	1534
2048	2049	2046

Table 6: Bit length of parameters in the experiments

Year	Size in bits	MIPS Yrs	Corresponding Yrs on 450 MHz P2 PC
2002	768	$2.06 \cdot 10^{10}$	$4.59 \cdot 10^{7}$
2005	864	$1.02 \cdot 10^{11}$	$2.26 \cdot 10^{8}$
2010	1056	$1.45 \cdot 10^{12}$	$3.22 \cdot 10^{9}$
2020	1472	$2.94 \cdot 10^{14}$	$6.54 \cdot 10^{11}$
2030	2016	$5.98 \cdot 10^{16}$	$1.33 \cdot 10^{14}$

Table 7: Recommended field sizes for subgroup discrete logarithm

In this project, six Java classes were implemented, each of which represents a member in the protocol specified in Section 3.6 and implements either of two interfaces: *GroupDiffieHellmanInterface* and *OptimizedGDHInterface*. The interfaces are including the following methods:

- upflow() : computes and updates the intermediate values sent from the previous member, in the view of current member, and returns the time (millisecond) elapsed to compute the intermediate values
- finish() : computes the common group key from the final intermediate values in the view of current member and returns the time (millisecond) elapsed to compute the group key

- precomputeK() : optimized protocols only. This method pre-computes the necessary values before the key agreement protocol begins.

A brief UML diagram of the implementation is given in Figure 8. The interface *OptimizedGDHInterface* is a subclass of the interface *GroupDiffieHellmanInterface* so that the optimized versions have three of the methods: *upflow()*, *finish()*, and *precomputeK()* while the non-optimized versions don't include the method *precomputeK()*. Note that every protocol class is extended from *MemberGDH*, which implements the non-authenticated Group Diffie-Hellman protocol.

The classes do not include the networking component which can exchange the intermediate values over the real-world network since this experiment is focused on the computational evaluation only. Instead, a global variable (array of *BigInteger* type has been used in the implementation) is used to share the intermediate values between each member object. However, each class does not only perform a simulation but also carry out appropriate computations as a program component once it is plugged in a complete group communication application. The executable class *SimulateGDH* (not shown on the UML given in Figure 8) simulates each protocol with specified parameters and accumulates the total amount of time required to compute intermediate values and group key. At the broadcast step, only the maximum time required to compute the group key is added to the total time since each computation is performed concurrently after the final intermediate values are broadcasted.

3.7.2 Results from the experiments

In order to show how practical the authenticated Group Diffie-Hellman protocols are, experimental evaluations have been performed with various parameters and group sizes. A typical result of the experiments is shown in Figure 9, with 1024-bit subgroup size. With a slightly conservative attitude based on Table 7, 1024-bit length is currently reliable in 2005.

As you see from the results in Figure 9, there are three groups that the protocols are forming. The first group, the slowest, includes only one, the non-optimized SA-GDH protocol. The second group, in the middle, consists of optimized SA-GDH and MAC SA-GDH with MD5 and SHA-1. The last group, the fastest, is composed of non-optimized and optimized A-GDH, and MAC A-GDH with MD5 and SHA-1.

The entire data from the experiments is also given in Tables 8 and 9, which contain results from UltraSPARC and Pentium 4, respectively. For each bit length experimented with, the protocols form three groups in the results as Figure 9.

3.7.3 Summary and discussion

As expected from the theoretical analysis in Section 3.6, the result of each protocol displays a quadratic behavior. The remarkable point is that the non-optimized A-GDH is in the same group as optimized A-GDH (see Figure 9). This means that the cost from multiplication and getting inverse in Java is negligible so that it does not affect the result much. Therefore, with a tiny overhead[8], we can take the advantage of non-optimized

[8]Certificate verification should be additionally considered in this case.

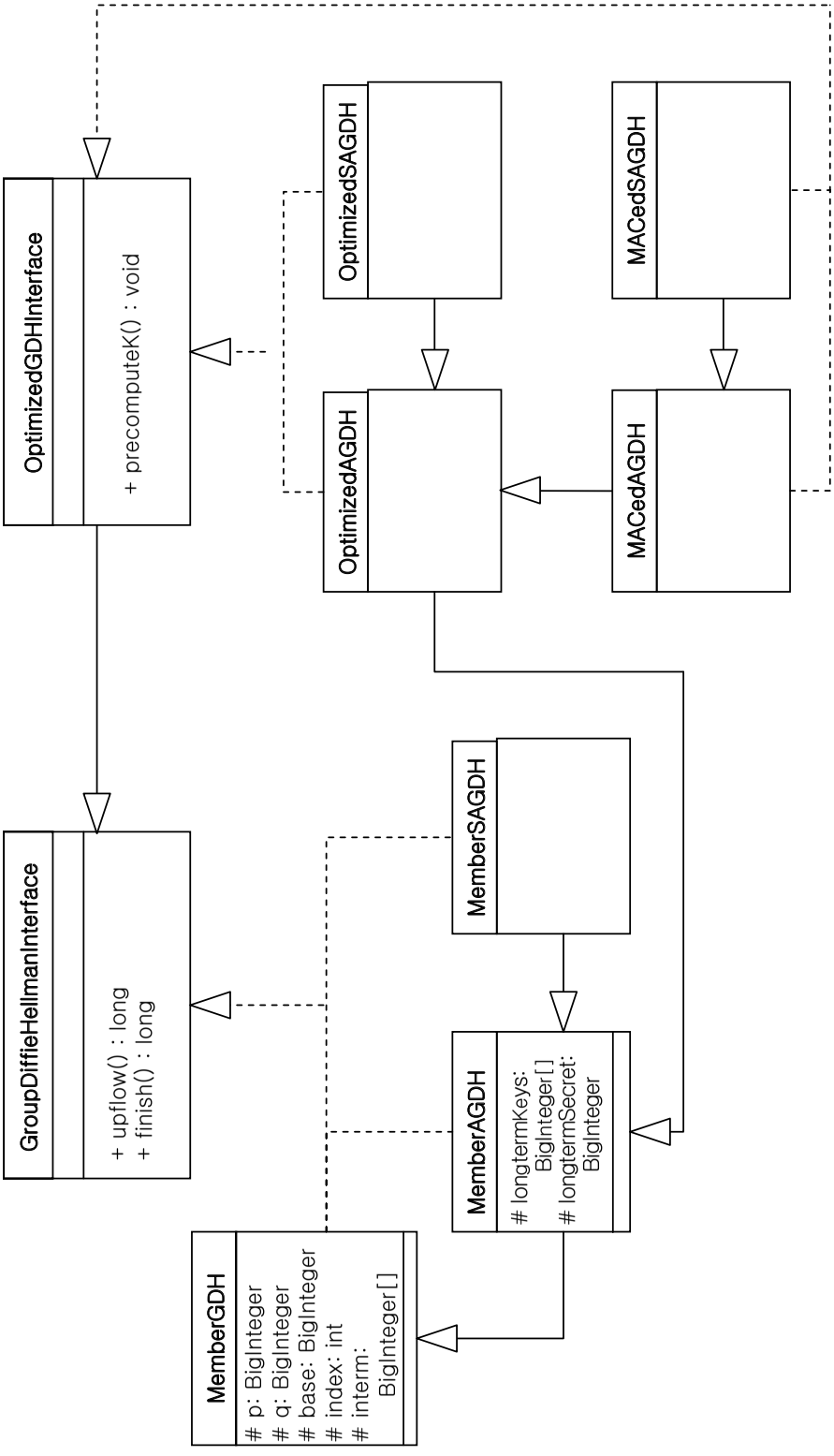

Figure 8: UML diagram for the protocol implementation

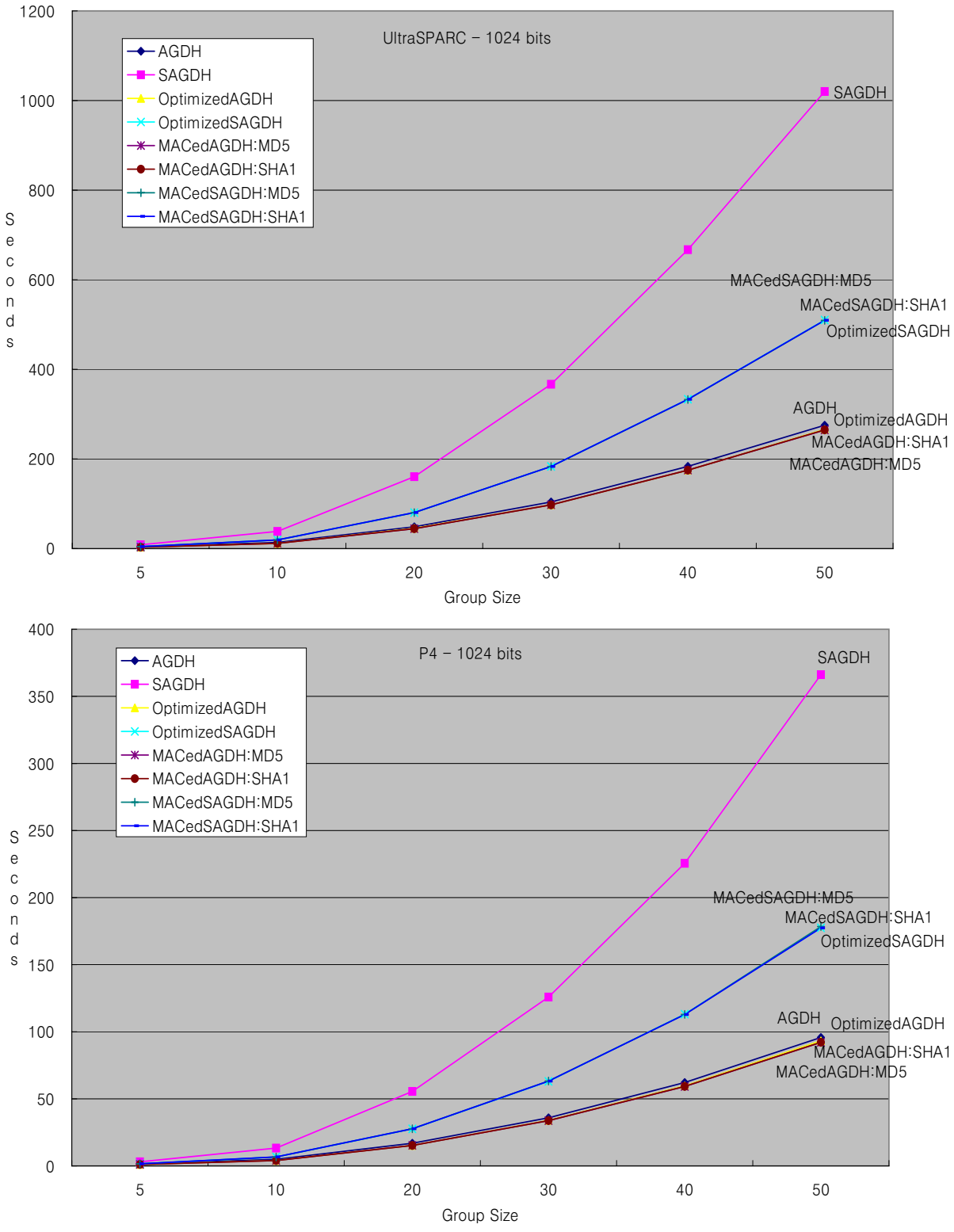

Figure 9: Results from 1024-bit field size in Solaris and Pentium4 machines

Size of Bits	Size of Group	Non-optimized		Optimized		AGDH with MAC		SAGDH with MAC	
		AGDH	SAGDH	AGDH	SAGDH	MD5	SHA1	MD5	SHA1
512	5	0.5875	1.1782	0.4266	0.5938	0.4322	0.4277	0.5950	0.5982
	10	1.9307	5.3413	1.6193	2.6884	1.6414	1.6238	2.6818	2.6763
	20	6.9290	22.8562	6.2834	11.4830	6.2989	6.3177	11.4343	11.4138
	30	13.0708	46.0871	12.2736	23.0424	12.3483	12.3219	22.9367	23.0077
	40	23.6492	85.5435	22.5356	42.7421	22.4708	22.4743	42.7515	42.7649
	50	39.8508	147.4582	38.4599	73.7792	38.3794	38.4113	73.8119	73.8923
768	5	1.8654	3.7991	1.3890	1.9543	1.3914	1.3860	1.9512	1.9587
	10	5.8997	16.2416	4.9307	8.1400	4.9208	4.9653	8.1395	8.1396
	20	20.9152	69.0506	19.0174	34.3875	18.9772	19.0577	34.4560	34.4888
	30	45.1789	158.7232	42.2594	79.2333	42.3546	42.3871	79.3961	79.3363
	40	75.5076	273.7468	71.8355	136.8588	72.0532	71.9246	136.8437	136.7838
	50	120.7569	446.2381	115.8866	222.7196	115.7052	115.9602	223.0605	223.2719
1024	5	4.1786	8.4895	3.1177	4.3606	3.1360	3.1290	4.3709	4.3346
	10	13.7005	38.2775	11.6165	19.2703	11.5902	11.6303	19.1983	19.2244
	20	48.4898	160.4789	45.3853	80.2493	44.3690	44.3519	80.3047	80.3348
	30	104.0498	366.7316	97.7982	183.2982	97.8497	97.4235	183.1805	183.1648
	40	183.5377	667.1602	174.9683	333.3218	175.1099	175.0146	332.7922	333.1905
	50	275.2802	1020.1579	265.9113	510.2101	264.4807	264.9492	509.2961	509.4264
1280	5	7.6797	15.5712	5.7396	7.9998	5.7533	5.7171	8.0271	8.0327
	10	26.0327	72.4897	21.9475	36.3202	22.0228	21.9699	36.3457	36.3807
	20	93.0170	308.5816	85.0665	154.1668	85.1841	85.0226	154.1447	154.2439
	30	198.3315	697.2335	186.3047	348.5784	185.3776	185.3461	347.2292	348.1092
	40	332.9987	1210.9859	316.9835	611.0856	322.3887	323.7804	613.7623	616.1379
	50	516.0360	1893.9845	491.8992	946.1488	492.2575	492.5814	947.4359	947.3458
1536	5	12.99465	26.5868	9.733	13.64185	9.7302	9.75765	13.5648	13.6117
	10	44.4109	123.5673	37.8871	62.12505	37.58535	37.59275	62.06325	62.29225
	20	158.1601	523.39555	144.1083	261.30525	144.16555	144.2242	261.28185	260.2881
	30	319.90145	1124.56805	300.24365	561.9851	299.94435	300.26615	563.0162	562.87615
	40	584.06045	2121.5768	559.3286	1059.97615	558.2832	562.11575	1061.85005	1072.9775
	50	867.03935	3148.33255	820.0548	1581.82175	820.06155	828.2776	1580.28685	1577.7117
2048	5	29.5616	60.8396	22.1434	31.1706	22.3436	22.2896	31.1524	31.1592
	10	91.4206	254.4478	77.431	127.9082	77.3964	77.4492	128.0578	128.0784
	20	332.9	1098.5524	303.3728	549.7058	303.3184	303.0992	550.2756	549.735
	30	717.997	2514.2484	671.198	1257.6404	671.2016	671.186	1257.5464	1258.5942
	40	1244.4006	4517.994	1185.8446	2258.3132	1187.1464	1185.5258	2258.6982	2258.8198
	50	*	*	*	*	*	*	*	*

* – not tested

Table 8: Times in seconds with UltraSPARC machine

A-GDH that there is no need to pre-share the long-term keys. Another conclusion from the results is that using MAC also has negligible cost. Some average values from the results even show that the fixes are faster. Note that the fixes using MAC belong to the same group as the protocols without MAC in the graphs shown in Figure 9.

However, the linear nature of the protocols makes it infeasible to finish the group key agreement in affordable time with big group size. Although the results do not reflect communication cost, the protocols with group size over 30 are slow enough in both testbeds. For general purpose, an application supporting large-group communications seems to be impractical for now.

Size of Bits	Size of Group	Non-optimized		Optimized		AGDH with MAC		SAGDH with MAC	
		AGDH	SAGDH	AGDH	SAGDH	MD5	SHA1	MD5	SHA1
512	5	0.2792	0.5192	0.1761	0.2457	0.1782	0.1826	0.2474	0.2529
	10	0.7467	2.0917	0.6114	1.0568	0.7846	0.7049	1.1607	1.1048
	20	2.6233	8.8992	2.3526	4.4104	2.3699	2.3066	4.3144	4.1832
	30	5.5616	20.2429	5.1908	9.7480	5.0966	5.0986	9.5082	9.5253
	40	9.5743	36.1477	9.0918	17.1277	8.8770	8.8544	16.9559	16.9996
	50	14.8388	55.6864	13.8200	26.3185	13.6684	13.6802	26.2865	26.3176
768	5	0.6750	1.3814	0.5005	0.6966	0.4971	0.5036	0.6995	0.6971
	10	2.1729	5.9327	1.8260	2.9610	1.8189	1.8198	2.9862	2.9941
	20	7.5298	24.6768	6.9506	12.5260	6.9299	6.9595	12.6212	12.7073
	30	16.3069	56.9057	15.1297	28.3407	15.2267	15.1939	28.3844	28.4186
	40	27.8326	100.9848	26.5197	50.7064	26.7189	27.5015	52.3360	52.1305
	50	44.1631	164.9042	43.4518	79.3072	41.1585	41.3672	79.1987	79.1682
1024	5	1.4986	3.0115	1.1057	1.5533	1.1080	1.1131	1.5496	1.5596
	10	4.9345	13.3173	4.0823	6.6443	4.0430	4.0154	6.6520	6.7420
	20	16.8871	55.4866	15.2884	27.6022	15.2696	15.2320	27.7210	27.6712
	30	35.8612	125.8783	33.8899	63.0218	33.6858	33.7929	63.2864	63.2265
	40	62.1643	225.5601	59.6804	113.0999	59.2051	59.2654	112.9318	112.8326
	50	95.7463	366.1307	93.1614	177.5114	92.0489	92.1231	178.3828	177.3412
1280	5	2.7823	5.6386	2.0794	2.9082	2.0590	2.0626	2.8629	2.8957
	10	8.9291	24.6538	7.4823	12.3479	7.5566	7.5388	12.4771	12.4292
	20	31.2599	103.6074	28.7056	52.0728	28.7809	28.9145	52.2714	52.3070
	30	67.6635	237.7794	63.9861	124.9063	63.9687	63.8465	119.9329	122.9855
	40	121.5002	425.2065	112.1629	214.1174	112.0048	112.0172	216.2370	211.3932
	50	179.1298	664.8380	174.0688	338.8018	173.9704	172.5760	332.0180	332.0754
1536	5	4.6465	9.4657	3.4637	4.8387	3.4580	3.4536	4.8410	4.8433
	10	14.9939	41.7384	12.6036	20.8802	12.6609	12.6959	20.9359	20.9331
	20	53.6417	175.7003	48.5463	88.3284	49.1121	48.6907	88.2301	88.2923
	30	114.7961	402.2575	107.3316	202.0779	107.7692	108.0066	202.3262	201.9401
	40	198.9776	721.4291	189.4822	360.2315	188.5309	188.1877	358.2780	357.1885
	50	304.5689	1143.2353	291.9433	562.1643	291.9606	292.5448	563.2057	561.6327
2048	5	10.6294	21.7314	8.0619	11.1651	7.9322	8.0347	11.0875	11.1212
	10	34.2759	95.3775	28.8584	47.9549	29.1112	29.1105	48.1194	48.0934
	20	120.5726	398.5519	109.6494	199.1691	109.8652	109.9151	200.5550	199.6619
	30	231.3244	800.8072	213.1270	402.4971	214.6009	214.2831	401.8572	401.8188
	40	452.4687	1641.4010	432.1300	825.0157	431.3907	431.2867	821.1570	821.8180
	50	696.4273	2574.7760	668.6770	1288.3437	671.7547	671.2027	1290.2197	1292.8280

Table 9: Times in seconds with Pentium 4 machine

4 Authentication Policies with Implicit Key Authentication

In this section, we will discuss the authentication policies with implicit key authentication in a Dynamic Peer Group (DPG). A DPG is defined as a dynamic[9] communication group where there is no hierarchy among members and all members have identical rights and duties. The examples of DPG include replicated servers (such as database, web, time), audio and video conferencing, collaborative workspaces, multi-user games, and so on. Therefore, the DPG models cover one-to-many as well as many-to-many communications.

Before discussing the authentication policies in DPGs, it is necessary to consider the admission control of members in the group communication. The admission control may determine *who is/are in charge of authentication of group members*. Therefore, authentication policies cannot be independent of admission policies.

[9]The word *dynamic* means the group membership is occasionally changed.

Yongdae *et al.* in [8] presented an admission control framework suitable for different flavors of peer groups that we have to consider in authentication policies. This project, however, is focused on the authentication policies that implicit key authentication property can be applied to. Therefore, it is enough to consider the cases of admission by member(s) in [8] as follows [10]:

Admission by a single member This is a typical group communication model one member (*e.g.* group host) controls admission of each member. This member should play the core role in authentication of the members.

Admission by fixed t-out-of-n threshold There is a subgroup of t members having higher authority in the group of size n. If the admission of a member is determined by those having higher authority, fixed t-out-of-n threshold scheme is also necessary in authentication process.

Admission by every member For some reasons, members cannot trust each other at all. There should be unanimity in admission of the new member(s).

In this section, the protocols will be discussed without considering MAC. However, each member can share the long-term secret which can be easily embedded in those protocols. Once a secret is shared between two parties, MAC schemes can be used associated with any kind of message.

4.1 Authentication on initial key agreement

Initial Key Agreement (IKA) is performed with static group membership at the time of group genesis. It is obvious that the participants of the communication are determined preceding the group genesis. Once each member is admitted to the communication group, authentication policy providing implicit key authentication follows one of followings:

Authentication by a single member As mentioned above, this is one of typical group communication models. Many applications, such as chatting, multi-user games, video conferencing, and etc, allow a user to be the host of communication. Each member should get an invitation or admission from the host. The underlying assumption is that every participant trusts the host during the communication. We can employ the A-GDH protocol discussed in Section 3.1 without additional mechanism for this model.

Authentication by fixed t-out-of-n threshold In some cases, a subgroup of the communication group has more authority than others. A possible case is video conferencing between hostile companies where the presidents from both companies can admit whoever they want to in a distributed manner. Since the two presidents do not trust each other and they have doubts that the other might invite unintended participant, both should be involved in the authentication mechanism. We can slightly modify A-GDH to provide implicit key authentication in conjunction with

[10] The case of admission by external entity or by voting among the members is not considered here.

fixed t-out-of-n threshold. An example of 2-out-of-5 is given in Figure 10. The doubled circles refer to the members with higher authorities. Note that the t members in doubled circles should be the last in the protocol.

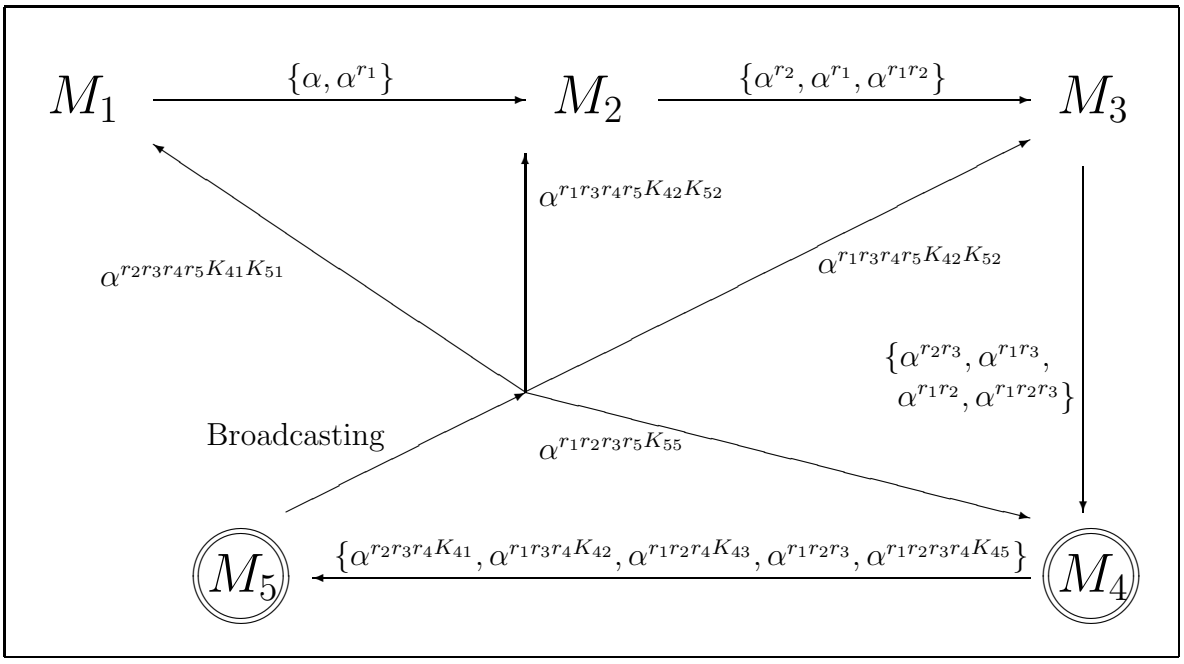

Figure 10: An example of 2-out-of-5 threshold using implicit key authentication

Authentication by every member If nobody can trust anybody in the group and the admissions should be obtained from unanimous agreement, then every member should be collaboratively involved in the authentication mechanism. This is exactly what the SA-GDH protocol requires members to do.

4.2 Authentication on auxiliary key agreement

Auxiliary Key Agreement (AKA) is performed when a change of membership in the group occurs during ongoing session. In [19], AKA operations are defined in detail. Without loss of generality, it is enough to consider the following four operations[11]:

- Member addition
- Member exclusion
- Group merge/fusion
- Group division

[11]Other operations are not considered here since those can be covered by these four operations. For example, Mass Join is basically the same as frequent Member Addition or Group Merge with some intervals.

Among the operations above, authentication is not required for member exclusion operations. In that case, only key refreshing is needed for resistance to the known key attacks, possibly performed by the excluded member. Member exclusion of M_p ($n \neq p$) in conjunction with the authenticated GDH protocols is illustrated in Figure 11, where the group controller generates a new secret r'_n and broadcast new intermediate values, on the assumption that M_n memorized the old intermediate values. The rest of the operations will be discussed below considering the authentication policies.

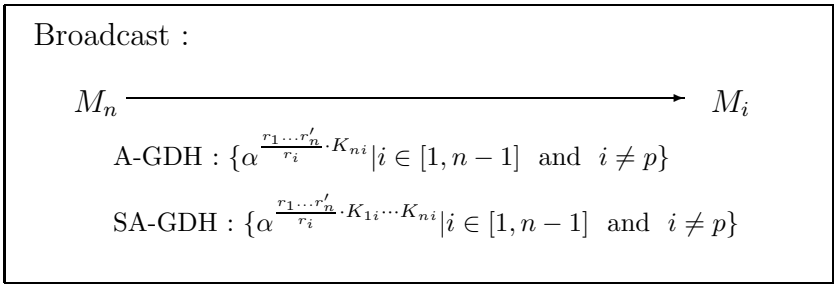

Figure 11: Member exclusion

4.2.1 Member addition

When a member joins the group, key refreshing is naturally required since the new member might have collected the old messages exchanged before he/she joins. The three authentication policies mentioned in Section 4.1 can be also applied here.

Authentication by a single member If a new member is admitted by the group host (*i.e.* group controller in the protocols), the authentication of the new member should be implicitly performed by the group controller. A member addition protocol for A-GDH, named A-GDH.2-MA, is given in [3]. In A-GDH.2-MA protocol, however, the authentication is implicitly done between the new member M_{n+1} and every other member M_i, $i \in [1, n]$. This is unnecessary under the policy of *authentication by a single member* since the members other than group controller do not need to contribute to authentication of the new member. Therefore, the simplified protocol illustrated in Figure 12 is more suitable for A-GDH when a new member is admitted. Note that r'_n is the new secret generated by M_n and broadcast to the old members via the secure channel which is already established. Finally, the new group key will be $\alpha^{r_1 \cdots r_n r'_n r_{n+1}}$.

Admission by t-out-of-n threshold Associated with the threshold authentication protocol illustrated as an example in Figure 10, the new member should be also implicitly authenticated by fixed t-out-of-n threshold. Figure 13 shows an example where a new member M_5 joins under 2-out-of-4 threshold ($M_1 \ldots M_4$ are the old members). Note that the new member M_5 should be able to compute K_{35}^{-1} and K_{45}^{-1} to be implicitly authenticated under 2-out-of-5 threshold policy. The new group key in the example will be $\alpha^{r_1 r_2 r_3 r'_4 r_5}$.

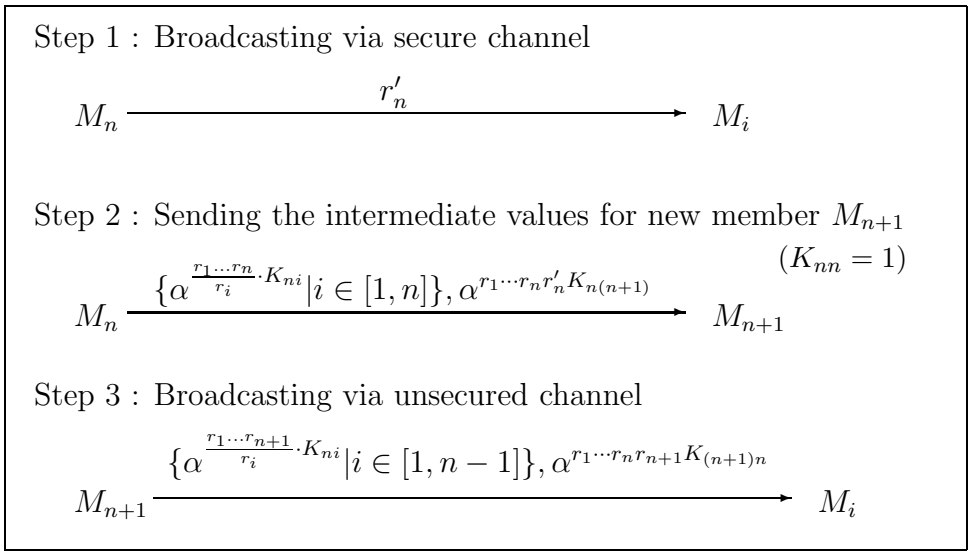

Figure 12: Member addition with authentication by a single member

Admission by every member As mentioned above, A-GDH.2-MA proposed in [3] satisfies this policy since the new member is implicitly authenticated by every other member. However, since this policy requires the SA-GDH protocol on IKA, we need a slight modification of the protocol as illustrated in Figure 14. Now the new member M_{n+1} cannot compute the group key without knowing the inverses of $K_{i(n+1)}$, $i \in [1, n]$.

4.2.2 Group merge

When a group is merged into another, there are various factors for authentication policies considering who contribute(s) to the authentication mechanism. If group B is merged into group A and M_b is the group controller in B, we have to determine whether or not authentication of M_b implies the authentication of the whole group B. Ultimately, only two cases need to be considered for authentication of the merged group:

A single member's authentication implies the whole group's authentication If a communication group is tightly bound by a single member (*e.g.* a company president and the employees, a military officer and his soldiers, etc), then authentication of a single member is enough since the proof of the leader's identity implies that of the whole group's identity. Let S_A and S_B be the current group keys established in groups A and B, respectively. If M_a and M_b are the group leader or host (usually the group controller in the protocols), the implicit key authentication of two groups can be performed between two leaders as shown in Figure 15, by simply exchanging $\alpha^{F(S_A)K_{ab}}$ and $\alpha^{F(S_B)K_{ab}}$. M_a and M_b can compute $\alpha^{F(S_A)F(S_B)}$ only when they can get the value of K_{ab}^{-1}, where K_{ab} is the long-term shared secret between M_a and M_b. Then, M_a and M_b can broadcast the new group secret $\alpha^{F(S_A)F(S_B)}$ via the secure group channel established before.

Every member should be authenticated Consider that Group B is merged into Group

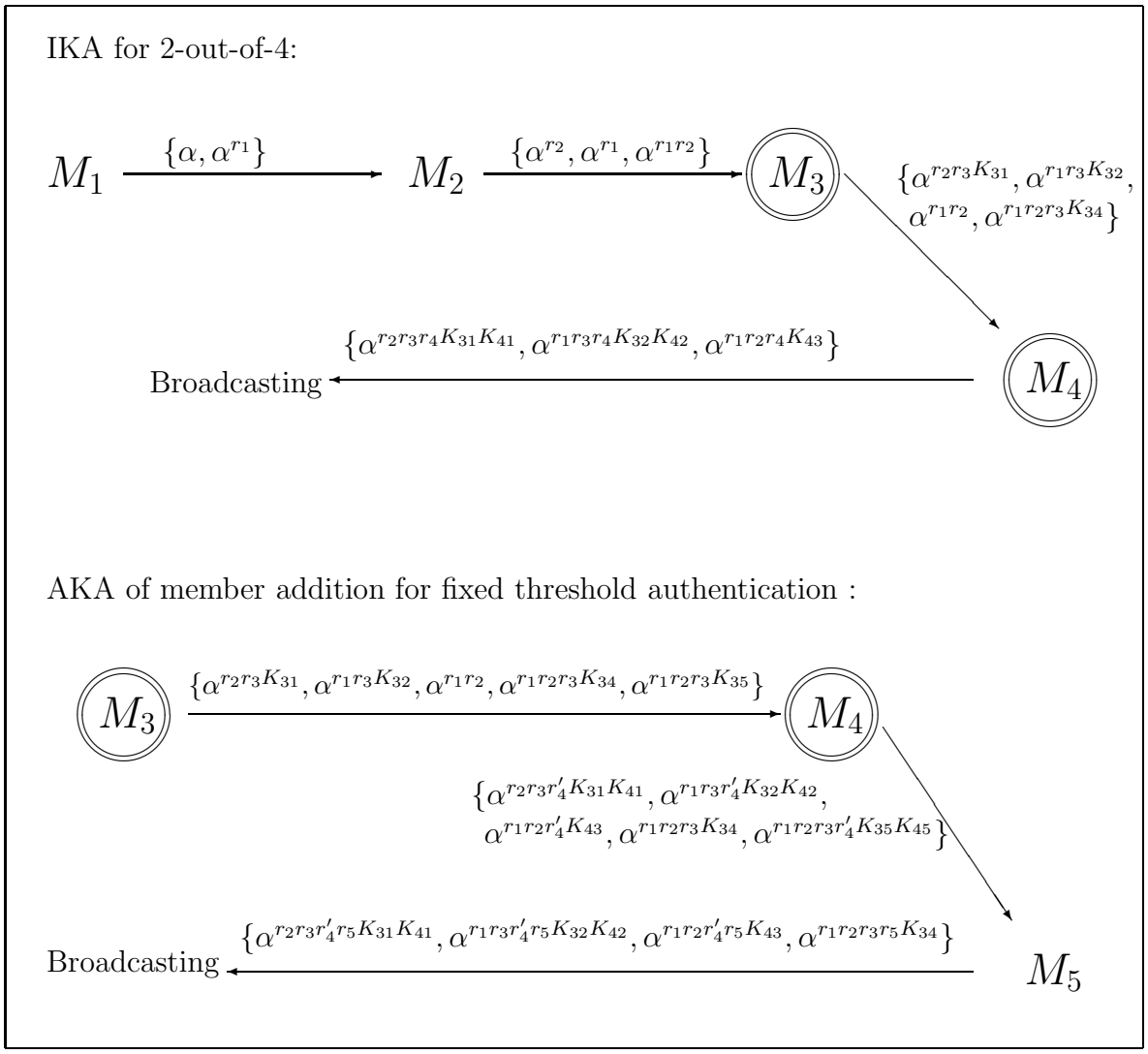

Figure 13: An example of 2-out-of-4 threshold with Member Addition

A. In the view of Group A, each member in Group B is possibly required to get authenticated by one or more members of Group A. If the group controller M_a of Group A is reliable enough [12], the authentication of Group B is simply performed as illustrated in Figure 16. Finally, the new secret is shared between both groups with implicit key authentication on every member in group B.

4.2.3 Group division

In group division operations, there are two different cases. One is that a group is split into two monolithic groups that had been merged before[13]. The other is that a group is split into two groups in which members are newly grouped. In the former case, the

[12]Otherwise, every member $M_i \in A$ can contribute to accumulation of the long-term shared secrets, *i.e.* $\alpha^{S_A K_{1k} \cdots K_{nk}}$ for $M_k \in B$.

[13]This case is called *group fission* in [19].

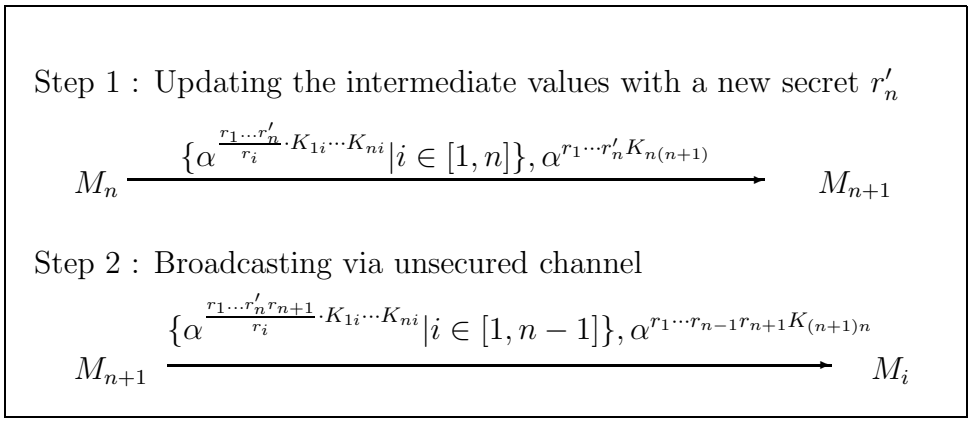

Figure 14: Member addition protocol for SA-GDH

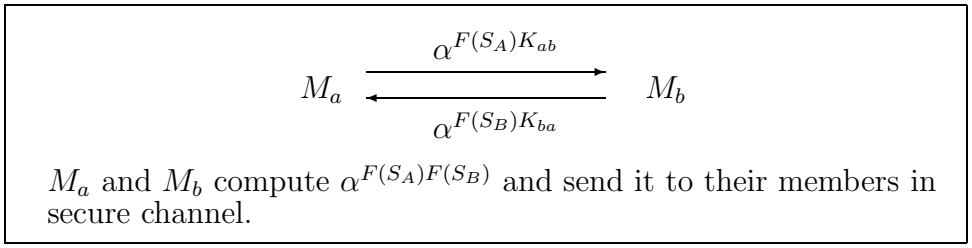

Figure 15: Group merge with single authentication

group controller memorizes the old intermediate values and broadcasts the final values, *i.e.* $\{\alpha^{\frac{r_1 \cdots r'_n}{r_i} \cdot K_{ni}} | i \in [1,n-1]\}$, with a new secret r'_n. Meanwhile, the latter case has no difference from forming a new group. Therefore, IKA should be performed again in each split group.

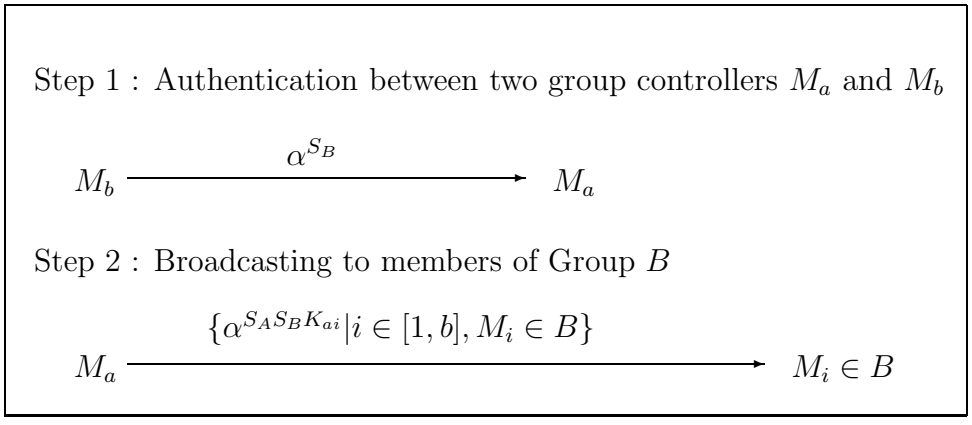

Figure 16: Group merge with every member authentication

5 Conclusion and Future Work

In order to build a secure group communication system based on the authenticated Group Diffie-Hellman (GDH) protocols, we have to consider 1) group size, and 2) authentication policy. As we can see from the empirical costs on computations in Section 3.7.2, group size is a major restriction of the protocols when we build a group communication system. With computational performance of the current computing environment in 2005, it looks impossible to deploy authenticated GDH protocols with group size over 100 in most applications. This is as a result of the linear structure of those protocols, which requires the exponentations (the most expensive operation in the protocol) to be done in sequence, from the first member to the group controller. Since the result in Section 3.7.2 reflects the computational costs only, the assumption that the network delay is high will produce remarkably worse performance. We can guess how badly those protocols degrade as network delay increases, based on [1] in which Amir *et al.* conclude the communication cost is the most important factor in high delay network among the group key agreement protocols. However, because of the linear structure, we can develop a group communication system with least infrastructure since the simplicity of the protocol does not require many external settings in order to manage the group communication. This attracts people into trying to plug in ad-hoc networks, where communication is achieved in peer-to-peer manners without any centralized control. Conclusively, A-GDH can be possibly adopted in the group communication with size 20∼30 while SA-GDH can be with the size less than 20, implemented with a key length safe enough.

What if group size is hundreds or more? Practically this is limited to one-to-many and few-to-many communications. Imagine that hundreds of netizens keep typing in a single chatting window, just as a lot of people keep talking in a room. Therefore, many-to-many is not necessarily considered in such a big group communication. Instead, one-to-many has more chances to be necessary for such a large group communication. In that case, we would better not consider an implicit key authentication protocol since one-way key distribution is more efficient on par with the same security level.

Regardless of the fact that the implicit key authentication protocols are practical in a group of small size, PKI is still needed, as discussed in Section 3.5. PKI will be always required in order to prevent an active attack as long as the public key systems exist. There is an unforgeable key distribution scheme, using Quantum Cryptography [4], but it has not been considered for group settings yet.

In Section 4, it was discussed how to employ the implicit key authentication property associated with the authentication policies, which can be determined by the admission policies. Implicit key authentication service can be provided in AKA operations as well as IKA. However, this is not proved to be practically robust in continuous member addition/exclusion, group merge/division, and so on. Alternatively, we may possibly exploit a hybrid method, which applies A-GDH or SA-GDH to IKA and a separate authentication protocol to AKA.

The GDH protocols suites including A-GDH and SA-GDH could be implemented associated with elliptic curve cryptographic schemes since the protocols are based on the discrete logarithm. It is known how to transform a discrete-logarithm-based protocol into an elliptic curve cryptographic system, including 2-party Diffie-Hellman, ElGamal

system, Elliptic Curve Digital Signature Standard(ECDSA), and so on [21]. Once GDH protocol suites are transformed into elliptic curves, we can also take the advantage of elliptic curve cryptography, which provides equivalent security with fewer bits so that cheaper and faster devices are available with the same level of security.

The fixes using MAC have shown almost the same performance as A-GDH and SA-GDH in computational cost. Therefore, the required security properties discussed in Section 2.2 can be protected by MAC without remarkable degrades of the performance.

The experiments in this project do not include the communication costs. It is possible to extend it to evaluating the performance of protocols including communication costs in various network environments. Another possible experimentation is on the AKA operations. Finally, the performance comparison between the protocols with and without implicit key authentication service needs to be done in order to show whether or not the property of implicit key authentication is practically more efficient than the separate authentication mechanism.

References

[1] Yair Amir, Yongdae Kim, Cristina Nita-Rotaru, and Gene Tsudik. *On the performance of group key agreement protocols*, in Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (Viena, Austria), June 2002.

[2] Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. *Authenticated group key agreement and friends*, in Proceedings of the 5th ACM Conference on Computer and Communications Security, pages 17-26, San Francisco, USA, 1998. ACM Press.

[3] Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. *New multi-party authentication services and key agreement protocols*, IEEE Journal on Selected Areas in Communication, 2000.

[4] Charles H. Bennett, Gilles Brassard, Seth Breidbart, and Stephen Wiesner. *Quantum cryptography, or unforgeable subway tokens*, Advances in Cryptology:Proceedings of Crypto 82, 267-275, 1983.

[5] Whitfield Diffie, and Martin E. Hellman. *New Directions In Cryptography*, IEEE Transactions on Information Theory, IT-22(6):644-654, November 1976.

[6] Donald E. Eastlake, 3rd, and Paul E. Jones. *US Secure Hash Algorithm 1 (SHA1)*, RFC 3174, September 2001. <`http://www.faqs.org/rfcs/rfc3174.html`>

[7] Niels Ferguson, and Bruce Schneier. *Practical Cryptography*, Wiley Publishing, Inc., 2003. ISBN 0-471-22357-3.

[8] Yongdae Kim, Daniele Mazzocchi, and Gene Tsudik. *Admission control in peer groups*, In IEEE International Symposium on Network Computing and Applications (NCA), Apr. 2003.

[9] Arjen K. Lenstra, and Eric R. Verheul. *Selecting cryptographic key sizes*, Shorter version of the report appeared in the Proceedings of the Public Key Cryptography Conference (PKC2000) and in the Autumn 99 PricewaterhouseCoopers CCE newsletter. Nov 1999.

[10] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*, The CRC Press Series on Discrete mathematics and its applications. CRC Press, 1997. ISBN 0-8493-8523-7.

[11] National Institute of Standards and Technology. *The Keyed-hash Message Authentication Code (HMAC)*, Federal Information Processing Standards Publication 198, 6 March 2002. <`http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf`>

[12] National Institute of Standards and Technology. *Secure Hash Standard*, Federal Information Processing Standards Publication 180-2, 1 Aug 2002. <`http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf`>

[13] Olivier Pereira and Jean-Jacques Quisquater. *A security analysis of the Cliques protocols suites*, in Proceedings of the 14th IEEE Computer security Foundations Workshop - CSFW'01, pages 73-81, Cap Breton, Canada, 2001. IEEE Computer Society Press.

[14] Olivier Pereira and Jean-Jacques Quisquater. *Some attacks upon authenticated group key agreement protocols*, Journal of Computer Security, 11(4):555-580, 2003.

[15] Olivier Pereira and Jean-Jacques Quisquater. *Generic insecurity of Cliques-type authenticated group key agreement protocols*, Computer Security Foundations Workshop, 2004. Proceedings. 17th IEEE, 2004.

[16] Ronald L. Rivest. *The MD5 Message-Digest Algorithm*, RFC 1321, April 1992.

[17] Bruce Schneier. *Cryptanalysis of SHA-1*, 18 Feb 2005. <`http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html`>.

[18] Michael Steiner, Gene Tsudik, and Michael Waidner. *Diffie-Hellman key distribution extended to groups*, in Third ACM Conference on Computer and Communications Security. Mar 1996, pp.31-37, ACM Press.

[19] Michael Steiner, Gene Tsudik, and Michael Waidner. *CLIQUES: A new approach to group key agreement*, In IEEE International Conference on Distributed Computing Systems, May 1998.

[20] Douglas R. Stinson. *Cryptography: Theory and Practice*, Second Edition, CRC Press, 2002. ISBN 1-58488-206-9.

[21] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*, CRC Press, 2003. ISBN 1-58488-365-0.