Key Agreement in Dynamic Peer Groups*

Michael Steiner
IBM Research Division, Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
sti@zurich.ibm.com

Gene Tsudik †‡
USC Information Sciences Institute
Marina del Rey, CA 90292
gts@isi.edu

Michael Waidner
IBM Research Division, Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
wmi@zurich.ibm.com

January 29, 1999

Abstract

As a result of the increased popularity of group-oriented applications and protocols, group communication occurs in many different settings: from network multicasting to application layer tele- and video-conferencing. Regardless of the application environment, security services are necessary to provide communication privacy and integrity.

This paper considers the problem of key agreement in dynamic peer groups. (Key agreement, especially in a group setting, is the steeping stone for all other security services.) Dynamic peer groups require not only initial key agreement (IKA) but also auxiliary key agreement (AKA) operations such as member addition, member deletion and group fusion. We discuss all group key agreement operations and present a concrete protocol suite, CLIQUES, which offers complete key agreement services. CLIQUES is based on multi-party extensions of the well-known Diffie-Hellman key exchange method. The protocols are efficient and provably secure against passive adversaries.

Keywords: Key Agreement, Secure Group Communication, Cryptography, Multi-Party Computation, Dynamic Peer Groups.

*Some of the material in this paper has been adapted from [20] and [21].

†Research supported by the Defense Advanced Research Project Agency, Information Technology Office (DARPA-ITO), under contract DABT63-97-C-0031.

‡Contact Author.

1 Introduction

As a result of the increased popularity of group-oriented applications and protocols, group communication occurs in many different settings: from network layer multicasting to application layer tele- and video-conferencing. Regardless of the underlying environment, security services are necessary to provide communication privacy and integrity.

While peer-to-peer security is a mature and well-developed field, secure group communication remains relatively unexplored. Contrary to a common initial impression, secure group communication is not a simple extension of secure two-party communication. There are two important differences. Firstly, protocol efficiency is of greater concern due to the number of participants and distances among them. The second difference is due to group dynamics. Two-party communication can be viewed as a discrete phenomenon: it starts, lasts for a while and ends. Group communication is more complicated: it starts, the group mutates (members leave and join) and there might not be a well-defined end. This complicates attendant security services among which key agreement is the most important.

In this paper, we concentrate on secure and efficient group key agreement. We start by defining a class of protocols that we call "natural" extensions of the 2-party Diffie-Hellman key exchange and prove the security of all protocols in this class against passive adversaries, provided the 2-party Diffie-Hellman decision problem is hard. This result allows us to craft a number of efficient protocols without having to be concerned about their individual security. In particular, we present two new protocols, each optimal with respect to certain aspects of protocol efficiency.

Subsequently, we consider a number of different scenarios of group membership changes and introduce protocols which enable addition and exclusion of group members as well as refreshing of the keys. Altogether, the protocols described below form a complete key management suite suited specifically for Dynamic Peer Groups (DPGs). However, it should be noted from the outset, that many other group security properties and services are not treated in this paper. These include: key authentication/integrity, entity authentication, key confirmation, group signatures and non-repudiation of group membership. Protocols and mechanisms in support of these are treated in another paper [1].

2 Dimensions of Key Agreement

All our protocols are based on **contributory key agreement**. This means that a group key K is generated as $f(N_1, ..., N_n)$, where $f()$ is some one-way function and N_i is an input (or key share) hosen by the i-th party. The method of computing group keys must guarantee that:

- each party contributing one N_i can calculate K;
- no information about K can be extracted from a protocol run; without knowledge of at least one of the N_i
- all inputs N_i are kept secret, i.e., if party i is honest then even a collusion of all other parties cannot extract any information about N_i from their combined view of the protocol.

The first two requirements are obviously needed. The last property ensures that the inputs x_i can be reused for subsequent key agreements. This is essential for DPGs, as will be seen below.

Several contributory schemes key agreement have been proposed in the literature [8, 18, 4, 11, 20, 9], however, none have been widely used. In practice, group key agreement is typically done in a *centralized* manner: one dedicated party (typically, a group leader) chooses the group key and distributes it to all group members. This actually translates into *key transport*, not *key agreement.*

While the centralized approach works reasonably well for static groups or very large groups, it turns out that contributory key agreement is superior for DPGs, i.e, flat (non-hierarchical) groups with dynamically changing membership.

In this paper we distinguish between **Initial Key Agreement** (IKA), a kind of group genesis, and **Auxiliary Key Agreement** (AKA). AKA encompasses all operations that modify group membership, such as member

addition and deletion. The central security requirement on AKA is key independence, i.e., each AKA operation should result in a new group key that is independent of all previous keys.

Ideally, the decision regarding *who* can add a new member, or delete on old one, should be taken according to local policy. There is no inherent reason to require the a single group leader to make these decisions. (One problem is with this setting is the exclusion of such a leader.) For instance, in some applications, each peer must be allowed to add new members and delete members that it previously added. This policy independence cannot be easily implemented in centralized schemes, while our schemes support it quite elegantly and efficiently.

Another advantage of contributory schemes in general is that they automatically provide freshness of new keys: each party i can check whether x_i was considered in K, hence, whether K is fresh. Furthermore, our protocols can be easily extended to *authenticated* group key agreement providing *perfect forward secrecy* (PFS) [15, 16, 12, 19], as shown in [1]. This is necessary for robust protocols withstanding *active* attacks.

2.1 Initial Key Agreement (IKA)

IKA takes place at the time of group *genesis*. This is the time when protocol overhead should be minimized since key agreement is a pre-requisite for secure group communication. On the other hand, for highly dynamic groups, certain allowances can be made: for example, extra IKA overhead can be tolerated in exchange for lower AKA (subsequent key agreement operations) costs.

Note that it is the *security* of the IKA, not its overhead costs, that is the overriding concern. In this context, security–as in the original 2-party Diffie-Hellman key agreement–means resistance to passive attacks. Equivalently, this means the inability to recover the group key by mere eavesdropping.

Naturally, IKA requires contacting every prospective group member. Contributory key agreement also calls for a key share to be obtained from each member. Hence, it may be possible to coincide (or interleave) with the IKA other security services such as authentication, access control and non-repudiation. This is something to keep in mind for the follow-on work.

We also note that, in some environments, IKA alone is sufficient. For example, if group membership is static or changes are very infrequent, AKA protocols may be unnecessary. The exception might be key refresh which can be mimicked (though expensively) with IKA.

2.2 Auxiliary Key Agreement Operations

As mentioned above, initial group key agreement is only a part, albeit a major one, of the protocol suite needed to support secure communication in dynamic groups. In this section we discuss other auxiliary group key operations and the attendant security issues. (See also Figure 1.)

The security property crucial to all AKA operations is **key independence**. Informally, it encompasses the following two requirements:

- Old, previously used group keys must not be discovered by new group member(s). In other words, a group member must not have knowledge of keys used before it joined the group.
- New keys must remain out of reach of former group members.

A related term found in the security literature is resistance to *known key attacks* (KKA) [15, 3]. A protocol is said to be *KKA-resistant* if knowledge of one or more past session (short-term) keys cannot be used to compute a current session key or a long-term secret. Generally, a known-key attack can be passive or active. The latter is addressed in detail by Burmester [3]. Since this paper (and our protocol model) is concerned with key agreement without any related services (e.g., implicit key authentication) we only consider passive known-key attacks on short-term session keys.

Along the same lines, we are not considering PFS since no long-term keys are assumed in this context. (Recall that PFS is premised on the possibility of compromise of long-term secrets.)

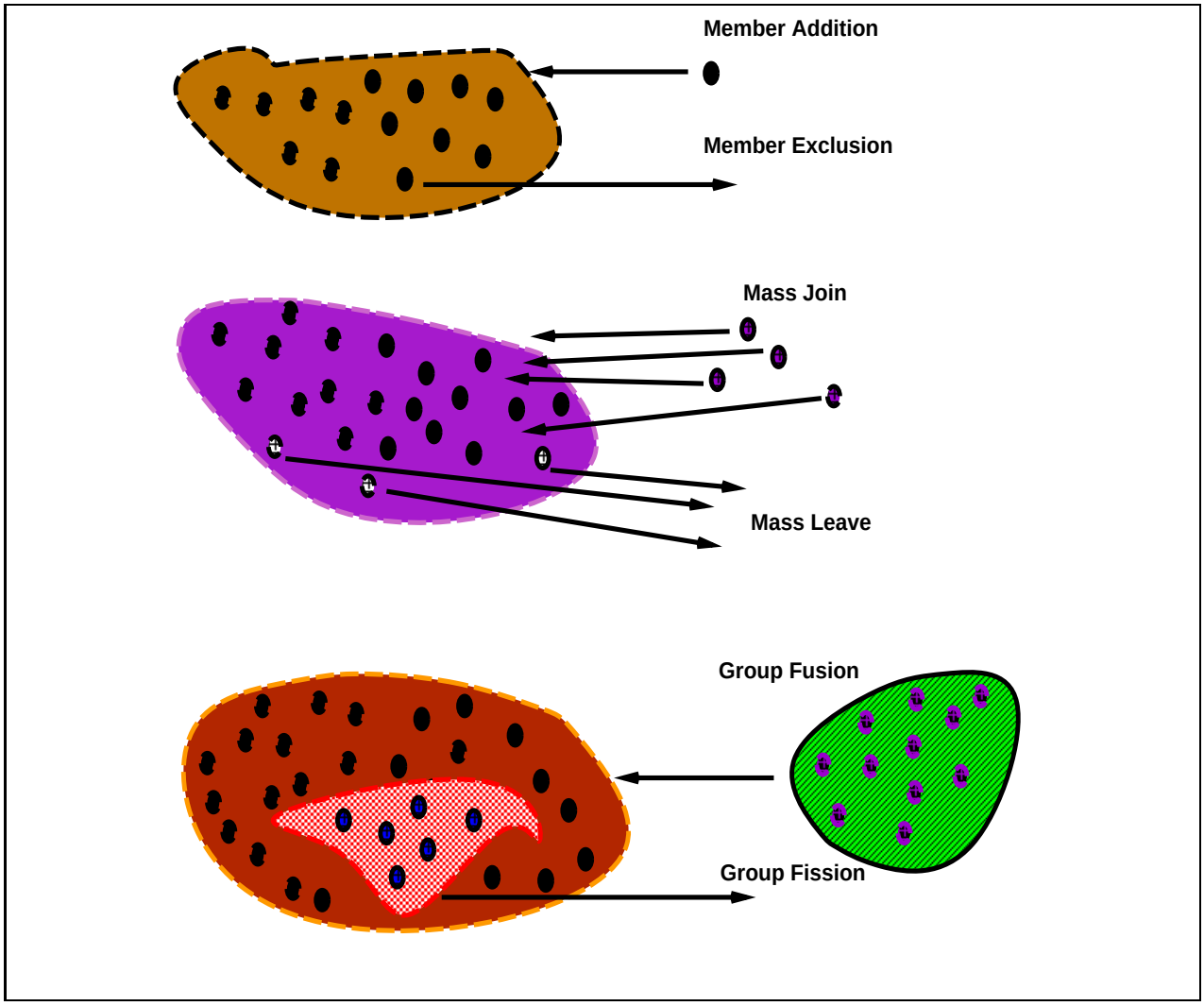

Figure 1: AKA Operations

More precisely, our communication model assumes that all communication is **authentic** but *not private*. An adversary is assumed to be strictly passive, i.e., it may eavesdrop on arbitrary communication but may not, in any way, interfere with it. Furthermore, an adversary in the IKA/AKA protocols can be an outsider or a quasi-insider. An outsider is a passive adversary not participating in the protocols. A quasi-insider is a one-time group member who wants to (passively) discover group session keys used **outside of its membership interval.**

While the requirement for key independence is fairly intuitive, we need to keep in mind that, in practice, it may be undesirable under certain circumstances. For example, a group conference can commence despite one of the intended participants running late. Upon its arrival, it might be best not to change the current group key so as to allow the tardy participant to catch up. In any case, this decision should be determined by local policy.

2.2.1 Single Member Operations

The AKA operations involving single group members are member addition and member exclusion. The former is a seemingly simple procedure of admitting a new member to an existing group. We can assume that member addition is always multi-lateral or, at least, bilateral (i.e., it takes at least the group leader's

and the new member's consent to take place.) Member exclusion is also relatively simple with the exception that it can be performed either unilaterally (by expulsion) or by mutual consent. In either case, the security implications of member exclusion are the same.

2.2.2 Subgroup Operations

Subgroup operations are group addition and group exclusion. Group addition, in turn, has two variants:

- Mass join: the case of multiple new members who have to be brought into an existing group and, moreover, these new members do not already form a group of their own.
- Group fusion: the case of two groups merging to form a super-group; perhaps only temporarily.

Similarly, subgroup exclusion can also be thought of as having multiple flavors:

- Mass leave: multiple new members must be excluded at the same time.
- Group division: monolithic group needs to be broken up in smaller groups.
- Group fission: previously merged group must be split apart.[1]

Although the actual protocols for handling all subgroup operations may differ from those on single members, the salient security requirements (key independence) remain the same.

2.2.3 Group Key Refresh

For a variety of reasons it is often necessary to perform a routine key change operation. This may include, for example, local policy that restricts the usage of a single key by time or by the amount of data that this key is used to encrypt or sign. To distinguish it key updates precipated by membership changes, we will refer to this operation as *key refresh*.

2.3 Who controls a group?

All AKA operations will be started by a single party. It might be natural to assume that this party is a *fixed* group controller, and this controller is the only one deciding to add or delete members. On the other hand, this is just one possible policy. Another reasonable policy would be to allow everybody to add new members, but only a dedicated controller to delete members. Alternatively, member deletion can be performed only by the party who orignally added them.

In general, AKA protocols should be designed in a way that can support (ideally) all kinds of policies. The CLIQUES protocols actually allow any party to run any AKA protocol, thus, CLIQUES can serve as a basis for a wide range of policies.

3 Generic n-Party Diffie-Hellman Key Agreement

The following notation is used throughout the remainder of this paper:

[1] Arguably, group fission is only relevant in special scenarios and in most cases it might not be worth the bookkeeping effort of keeping track of subgroups.

n	number of protocol participants (group members)
i, j, h, p, d, c	indices of group members
M_i	i-th group member; $i \in [1, n]$
G	cyclic algebraic group
q	order of the algebraic group
α	exponentiation base; generator in the algebraic group G delimited by q
N_i	random (secret) exponent $\in \mathbb{Z}_q$ generated by M_i
$\mathcal{S}, \mathcal{T}$	subsets of $\{N_1, \ldots, N_n\}$
$\Pi(\mathcal{S})$	product of all elements in set $\mathcal{S}$
K_n	group key shared among n members

3.1 Security proof outline

All our key agreement protocols belong to a family of protocols that we refer to as "natural" extensions of the 2-party Diffie-Hellman key exchange [5]: Like in the 2-party case, all participants $M_1, \ldots, M_n$ agree a priori on a cyclic group, G. Let α be a generator of G. For each key exchange, each member, M_i, choses randomly a value $N_i \in \mathbb{Z}_q$. The group key will be $K = \alpha^{N_1 \cdots N_n}$.

In the 2-party case, K is computed by exchanging α^{N_1} and α^{N_2}, and computing $K = (\alpha^{N_1})^{N_2} = (\alpha^{N_2})^{N_1}$.

To solve the n-party case, a certain subset of $\{\alpha^{\Pi(\mathcal{S})} |\ \mathcal{S} \subset \{N_1, \ldots, N_n\}\}$ is exchanged between the players. This set includes all values $\alpha^{N_1 \cdots N_{i-1} N_{i+1} \cdots N_n}$ for all i. Obviously, if M_i gets $\alpha^{N_1 \cdots N_{i-1} N_{i+1} \cdots N_n}$ it can easily compute K.

The security of the 2-party case is directly based on the 2-party decision Diffie-Hellman (DDH) problem: Given $(\alpha^{N_1}, \alpha^{N_2}, \alpha^X)$ decide whether $X = N_1 N_2$ (i.e., the secret key K) or some randomly chosen exponent.

This can be easily generalized to what we call the *n-party DDH problem:* Given $\{\alpha^{\Pi(\mathcal{S})} |\ \mathcal{S} \subset \{N_1, \ldots, N_n\}\}$ and α^X, decide whether $X = (N_1 \cdots N_n)$ or some random value.

In the following section we prove that if the 2-party DDH problem is hard, then the n-party DDH problem is hard as well. This proves the security of all natural Diffie-Hellman extension at once.

3.2 Security of all natural extensions

Let k be a security parameter. All algorithms run in probabilistic polynimial time with k and n as inputs.

For concreteness, we consider a specific class of groups G for which it is commonly assumed that the 2-party Diffie-Hellman decision problem is hard:

> On input k, algorithm gen chooses at random a pair (q, α) where q is a k-bit value, q and $q' = 2q + 1$ are both prime, and α is a generator of the unique subgroup G of $\mathbb{Z}_{q'}^*$ of order q. Groups of this type are used, e.g., in Schnorr signatures [17] and DSS [6].

For $(q, \alpha) \leftarrow gen(k)$, $n \in \mathbb{N}$, and $X = (N_1, \ldots, N_n)$ for $N_i \in \mathbb{Z}_q$, let:

- $view(q, \alpha, n, X) :=$ the ordered set of all $\alpha^{N_{i_1} \cdots N_{i_m}}$ for all proper subsets $\{i_1, \ldots, i_m\}$ of $\{1, \ldots, n\}$,
- $K(q, \alpha, n, X) := \alpha^{N_1 \cdots N_n}$.

If (q, α) are obvious from the context, we omit them in $view()$ and $K()$. Note that $view(n, X)$ is exactly the view of an adversary in the generic n-party DH-protocol, where the final secret key is $K(n, X)$. Let the following two random variables be defined by generating $(q, \alpha) \leftarrow gen(k)$ and choosing X randomly from $(\mathbb{Z}_q)^n$:

- $A_n := (view(n, X), y)$, for a randomly chosen $y \in G$,
- $D_n := (view(n, X), K(n, X))$.

Let the operator "$\approx_{\text{poly}}$" denote *polynomial indistinguishability.*

Remark: Polynomial indistinguishability of the 2-party Diffie-Helman key is considered, e.g., in [2]. The notion of polynomial indistinguishability is related to polynomial time statistical test as defined in [22, 14]. In this context, it means that no polynomial-time algorithm can distinguish between a Diffie-Hellman key and a random value with probability significantly greater that $\frac{1}{2}$. More specifically, let K and R be l-bit strings such that R is random and K is a Diffie-Hellman key. We say that K and R are **polynomially indistinguishable** if, for all polynomial time distinguishers, A, the probability of distinguishing K and R is smaller than $(\frac{1}{2} + \frac{1}{Q(l)})$ for all polynomials $Q(l)$.

Theorem 1 *For any $n > 2$, $A_2 \approx_{\text{poly}} D_2$ implies $A_n \approx_{\text{poly}} D_n$.*

Proof (by induction on n): Assume that $A_2 \approx_{\text{poly}} D_2$ and $A_{n-1} \approx_{\text{poly}} D_{n-1}$. Thus, we have to show $A_n \approx_{\text{poly}} D_n$. We do this by defining random variables B_n, C_n, and showing $A_n \approx_{\text{poly}} B_n \approx_{\text{poly}} C_n \approx_{\text{poly}} D_n$, which immediately yields: $A_n \approx_{\text{poly}} D_n$.

We can rewrite $view(n, (N_1, N_2, X))$ with $X = (N_3, \ldots, N_n)$ as a permutation of:

$$\begin{pmatrix} view(n-1,(N_1,X)), K(n-1,(N_1,X)), \\ view(n-1,(N_2,X)), K(n-1,(N_2,X)), \\ view(n-1,(N_1N_2,X)) \end{pmatrix}$$

and $K(n, (N_1, N_2, X))$ as $K(n-1, (N_1N_2, X))$.

We use this to redefine A_n and D_n. All in all, we consider the following four distributions. All of them are defined by $(q, \alpha) \leftarrow gen(k)$, choosing $c, N_1, N_2 \in \mathbb{Z}_q$ and $X \in (\mathbb{Z}_q)^{n-2}$ and $y \in G$ randomly.

- $A_n := (view(n-1,(N_1,X)), K(n-1,(N_1,X)),$
  $view(n-1,(N_2,X))), K(n-1,(N_2,X)),$
  $view(n-1,(N_1N_2,X)), y)$
- $B_n := (view(n-1,(N_1,X)), K(n-1,(N_1,X)),$
  $view(n-1,(N_2,X))), K(n-1,(N_2,X)),$
  $view(n-1,(c,X)), y)$
- $C_n := (view(n-1,(N_1,X)), K(n-1,(N_1,X)),$
  $view(n-1,(N_2,X))), K(n-1,(N_2,X)),$
  $view(n-1,(c,X)), K(n-1,(c,X)))$
- $D_n := (view(n-1,(N_1,X)), K(n-1,(N_1,X)),$
  $view(n-1,(N_2,X))), K(n-1,(N_2,X)),$
  $view(n-1,(N_1N_2,X)), K(n-1,(N_1N_2,X)))$

Note that only the last two components vary.

$A_n \approx_{\text{poly}} B_n$ follows from $A_2 \approx_{\text{poly}} D_2$:
Assume that adv distinguishes A_n and B_n, and let (u, v, w) be an instance of $A_2 \approx_{\text{poly}} D_2$. We produce an instance for adv by using u for α^{N_1}, v for α^{N_2}, and w for $\alpha^{N_1N_2}$ (or α^c), and choosing X and y randomly. If (u, v, w) belongs to A_2 (D_2), this new distribution belongs to B_n (A_n).

$B_n \approx_{\text{poly}} C_n$ follows from $A_{n-1} \approx_{\text{poly}} D_{n-1}$:
Assume that adv distinguishes B_n and C_n, and (ignoring a necessary permutation in order) let: $(view(n-1,(c,X)), y)$ be an instance for $A_{n-1} \approx_{\text{poly}} D_{n-1}$ (i.e., the problem is to decide whether $y = K(n-1,(c,X))$.) We produce an instance for adv by choosing N_1, N_2 randomly, and computing $(view(n-1,(N_i,X)), K(n-1,(N_i,X)))$ based on those values in $view(n-1,(c,X))$ that do not contain c as an exponent. The rest follows as in the last case.

$C_n \approx_{\text{poly}} D_n$ follows from $A_2 \approx_{\text{poly}} D_2$, almost exactly like the first statement. The only difference is that we do not choose y randomly, but as $K(n-1,(w,X))$.
□

Hereafter, the above result allows us to construct a number of protocols belonging to the natural DH extensions family without worrying about their individual security.

4 CLIQUES: Initial Key Agreement

The cornerstone of the CLIQUES protocol suite is formed by two IKA protocols called IKA.1 and IKA.2. (They were referred to as GDH.2 and GDH3, respectively, in [20].)

4.1 IKA.1

The first IKA protocol (IKA.1) depicted in Figure 2 is simple and straight-forward. It consists of an upflow and downflow stages. The purpose of the upflow stage is to collect contributions from all group members, one per round.

In round $i-1$ $(1 < i < n)$, M_i receives a collection of i values. Of these, $i-1$ are intermediate and one is *cardinal*. Let INT_j^{i-1} denote the j-th intermediate value in round $i-1$. It is always of the form:

$$INT_j^{i-1} \;=\; \alpha^{\frac{N_1 \cdots N_{i-1}}{N_j}} \quad \text{for } 1 < j < i$$

whereas a cardinal value is simply:

$$CRD_{i-1} \;=\; \alpha^{N_1 \cdots N_{i-1}}$$

M_i's actions are as follows:

1. generate private exponent N_i
2. set $INT(i,j) = (\ INT(i-1,j)\ )^{N_i}$
3. set $INT(i,i) = CRD_{i-1}$
4. set $CRD_i = (CRD_{i-1})^{N_i}$

In total, M_i composes i intermediate values (each with $(i-1)$ exponents.) and a cardinal value containing i exponents. For example, M_4 receives a set:

$$\{\alpha^{N_1N_2N_3}, \alpha^{N_1N_2}, \alpha^{N_1N_3}, \alpha^{N_3N_2}\}$$

and outputs a set:

$$\{\alpha^{N_1N_2N_3N_4}, \alpha^{N_1N_2N_3}, \alpha^{N_1N_2N_4}, \alpha^{N_1N_3N_4}, \alpha^{N_3N_2N_4}\}$$

In round $(n-1)$, when the upflow reaches M_n, the cardinal value becomes $\alpha^{N_1 \cdots N_{n-1}}$. M_n is thus the first group member to compute the key K_n. Also, as the final part of the upflow stage, M_n computes the last batch of intermediate values. In the second stage M_n broadcasts the intermediate values to all group members.

IKA.1 has the following characteristics:[2]

rounds	n
messages	n
combined message size	$(n-1)(n/2+2)-1$
exponentiations per M_i	$(i+1)$ for $i < n$, n for M_n
total exponentiations	$\frac{(n+3)n}{2} - 1$

The highest-indexed group member M_n plays a special role by having to broadcast the last round of intermediate values. (However, this special role *does not* afford M_n any added rights or privileges.)

[2] Assuming atomic, one-message broadcast.

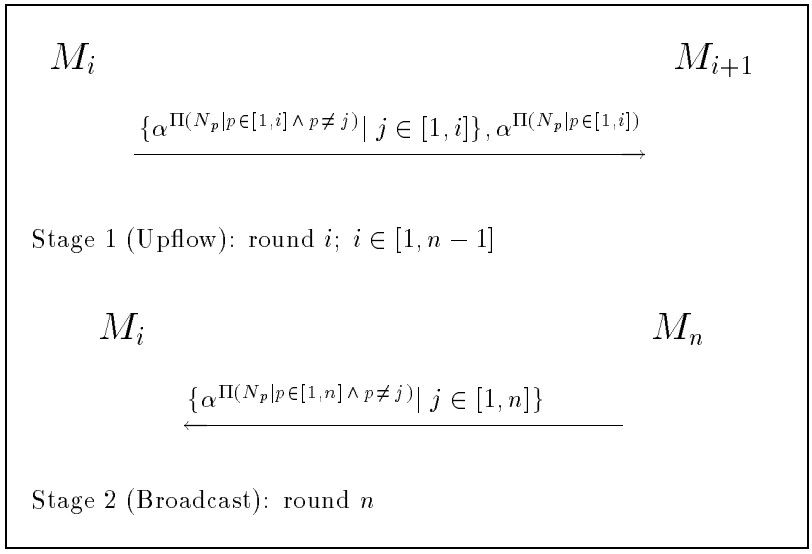

Figure 2: Group Key Agreement: IKA.1

4.2 Group Initial Key Agreement: IKA.2

In certain environments, it is desirable to minimize the amount of computation performed by each group member. This is particularly the case in large groups or groups involving low-power entities such as smart-cards or PDAs. Since IKA.1 requires a total of $(i+1)$ exponentiations of every M_i, the computational burden increases as the group size grows. The same is true for message sizes.

In order to address these concerns we construct a very different protocol, IKA.2 (see Figure 3). IKA.2 consists of four stages. In the first stage we collect contributions from all group members similar to the upflow stage in IKA.1. After processing the upflow message M_{n-1} obtains $\alpha^{\Pi\{N_p|p\in[1,n-1]\}}$ and broadcasts this value in the second stage to all other participants. At this time, every M_i $(i \neq n)$ factors out (divides by) its own exponent and forwards the result to M_n. (Note that factoring out N_i requires computing its inverse – N_i^{-1}. This is always possible if we choose the group q as a group of prime order). In the final stage, M_n collects all inputs from the previous stage, raises every one of them to the power of N_n and broadcasts the resulting $n-1$ values to the rest of the group. Every M_i now has a value of the form $\alpha^{\Pi\{N_p|p\in[1,n]\wedge p\neq i\}}$ and can easily generate the intended group key K_n.

IKA.2 has two appealing features:

- Constant message sizes
- Constant (and small) number of exponentiations for each M_i (except for M_n with n exponentiations required)

Its properties are summarized in the following table:

rounds	$n+1$
messages	$2n-1$
combined message size	$3(n-1)$
exponentiations per M_i	4 for $i<(n-1)$,
	2 for M_{n-1}, n for M_n
total exponentiations	$5n-6$

One notable drawback of IKA.2 is that, in Stage 3 (n-th round), $n-1$ unicast messages are sent to M_n. This might lead to congestion at M_n.

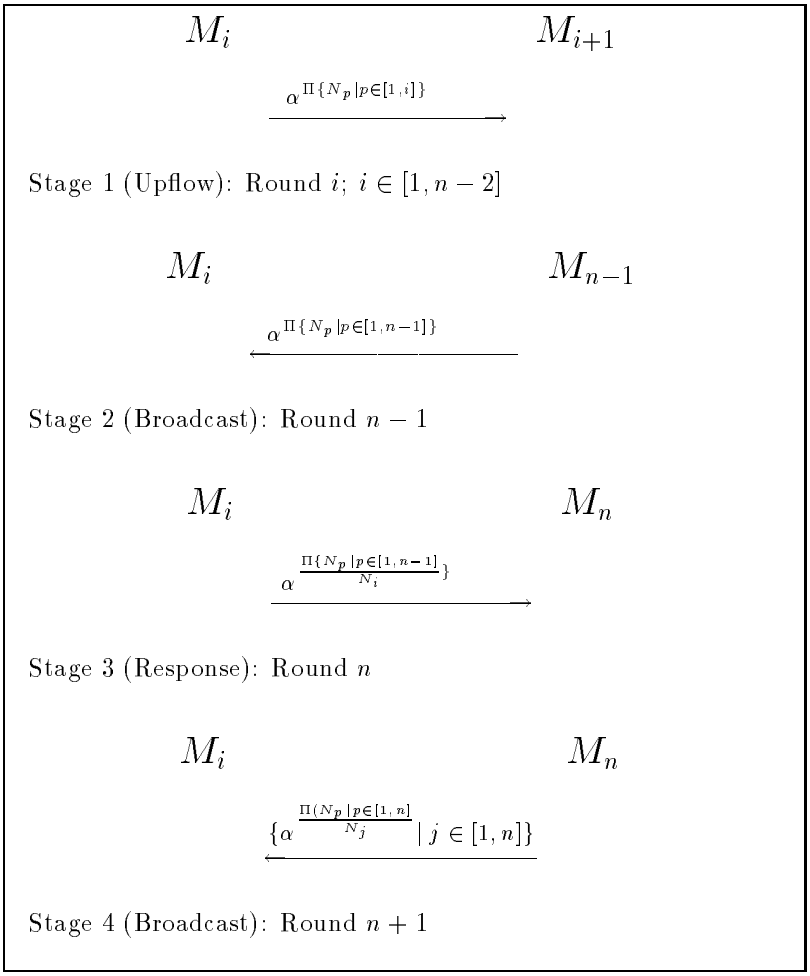

Figure 3: Group Key Agreement: IKA.2

5 CLIQUES: Auxiliary Key Agreement

Both IKA protocols operate in two phases: a gathering phase whereby M_n collects all $\{\alpha^{\frac{N_1 \cdots N_n}{N_i}} | i \in [1,n]\}$ and a final broadcast phase. Our AKA operations take advantage of the keying information (i.e., partial keys) collected in the gathering phase of the most recent IKA protocol run. This information is incrementally updated and re-distributed to the new incarnation of the group.

Since the final broadcast phase is exactly the same for both IKA.1 and IKA.2 we also note that the AKA operations described below work with both IKA protocols. This results in flexibility to choose an IKA protocol that suits a particular DPG setting.

5.1 Member Addition

The member addition protocol is shown in Figure 4. The protocol's main premise is that the new member M_{n+1} becomes the new group controller. It is assumed that the "old" controller M_n saves the contents of the last Broadcast message that was sent in the last round in the IKA protocol of Figure 2.[3]

[3] This is only the case for the very first member addition; subsequent member additions require the current controller to save the most recent Broadcast message from the preceding member addition protocol.

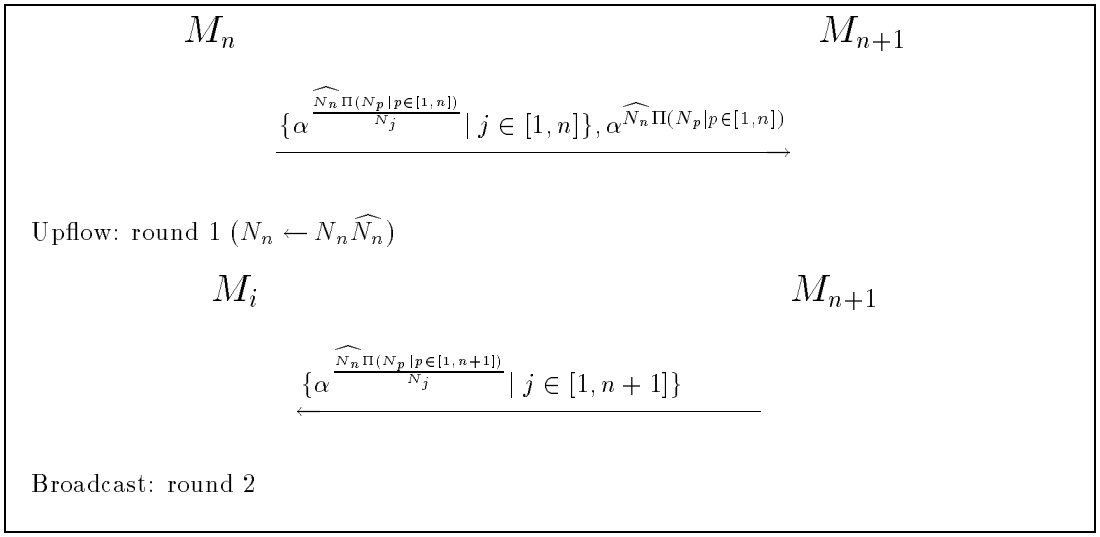

Figure 4: Member Addition: floating group controller

In effect, M_n extends Stage 1 of the IKA protocol by one round: it generates a new exponent $\widehat{N_n}N_n$ and creates a new upflow message (where $\widehat{N_n}N_n$ is used in place of N_n.)

It might seem more natural to replace N_n by $\widehat{N_n}$ instead of combining the two. However, this way not just the group controller but any member who stores the latest broadcast message can initiate member addition. This feature may prove useful in some environments. Unfortunately, this potential benefit becomes a drawback in the context of member exclusion. (If any member can exclude any other member, anarchy will reign!)

The role of the group controller is thus passed on to the newest group member. Although this protocol fits in nicely with the IKA, its basic assumption of a *floating* group controller might be unrealistic in some environments. For example, the new member may, in fact, be the one least trusted by the rest of the group.[4] In order to address this concern, we modify the present protocol to support a fixed group controller. For the sake of clarity, we assume that, while the controller stays fixed, its index keeps growing. In other words, the new member becomes M_n and the group controller assumes the index $n+1$.

The resultant protocol is shown in figure 5. The first message is a duplicate of either:

- Upflow message in round $(n-1)$ of the original IKA protocol
  (only if this is the first member addition)

  OR

- Upflow message in round 2 of the last member addition protocol

One interesting and useful feature of the two member addition protocols is their ability to co-exist within a group. Consequently, a group may start out with a fixed group controller and, later, switch over to the floating controller mode or viceversa.

5.2 Mass Join

Distinct from both member and group addition is the issue of *mass* join. When is mass join necessary? In cases when multiple new members need to be brought into an existing group. In most cases, the new members

[4] On the other hand, it can also be argued that this approach is fair since it off-loads the bulk of the computation to the newcomer.

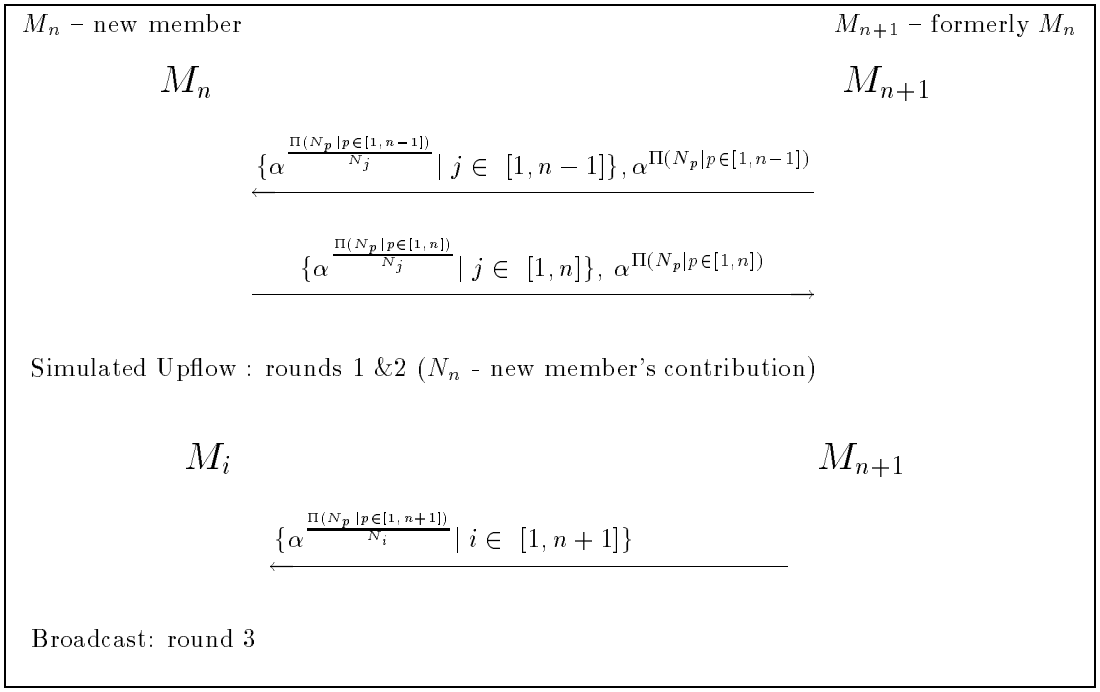

Figure 5: Member Addition: fixed group controller

are disparate (i.e., have no prior common association) and need to be added in a hurry. Alternatively, the new members may already form a subgroup but policy might dictate that they should be admitted individually.

It is, of course, always possible to add multiple members by consecutive runs of a single-member addition protocol. However, this would be inefficient since, for each new member, every existing member would have to compute a new group key only to *throw it away* thereafter. To be more specific, if m new members were to be added in this fashion, the cost would be:

- $3m$ rounds with fixed group controller and $2m$ – with floating.
- Included in the above are m rounds of broadcast
- m exponentiations by every "old" group member

The overhead is clearly very high.

A better approach is to *chain* the member addition protocol as shown in Figure 6. The idea is to capitalize on the fact that multiple, but disparate, new members need to join the group and chain a sequence of Upflow messages to traverse all new members in a certain order. This allows us to incur only one broadcast round and postpone it until the very last step, i.e., the last new member being *mass-joined* performs the broadcast. The savings, compared with the naive approach, amount to $m-1$ broadcast rounds.

For brevity's sake Figure 6 shows only the floating controller model. A chained fixed controller model can be trivially and similarly constructed from the protocol in Figure 5.

5.3 Group Fusion

Group fusion, as defined above, occurs whenever two groups merge to form a super-group. The only real difference with respect to mass join is that group fusion assumes pre-existing relationships within both groups. Thus, it is important to recognize from the outset that the most expedient way to address group fusion is to treat it as either:

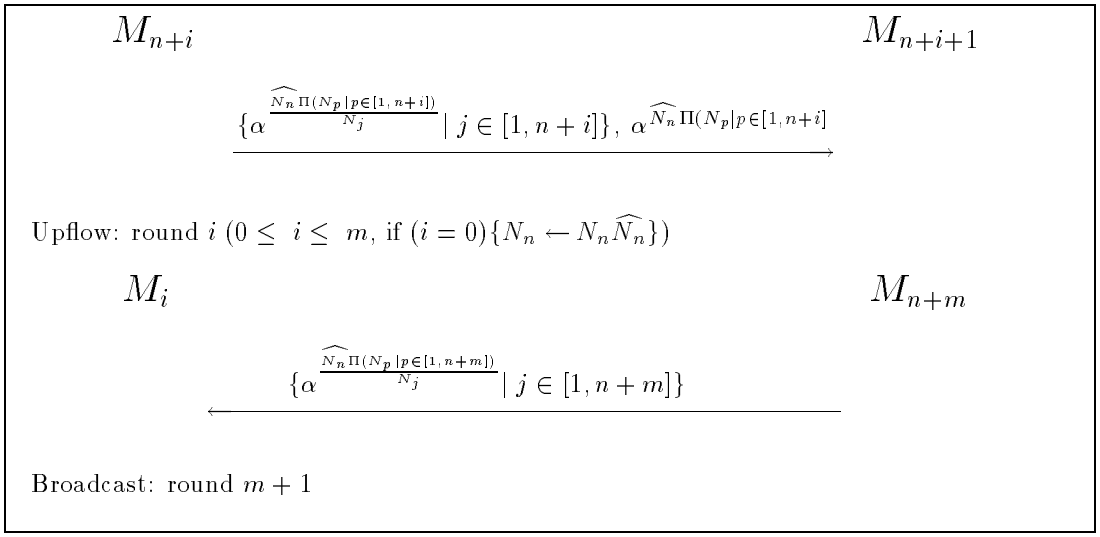

Figure 6: Mass Join (floating controller)

1) Special case of mass join as in Figure 6

or

2) Creation of a new super-group via IKA of Figure 2

The first choice is appropriate if one of the groups is small. (Recall that mass join takes $m+1$ rounds where m is the smaller group's size.) On the other hand, creating a new group from scratch may be more secure[5] and not too expensive if both groups are relatively small. Another reason can be the need to re-assign the group controller's role.

It is certainly possible to end the discussion of group fusion at this point. The outcome would be a heuristic- or policy-driven decision to use (1) or (2) on a case-by-case basis. However, if only for purely academic reasons, it might be worthwhile to investigate more efficient, or at least more elegant, solutions geared specifically towards group fusion. Although this remains a subject for future work, we briefly sketch one possible solution below.

One promising approach to group fusion is a technique fashioned after the one developed by Steer et al. in [18]. In brief, suppose that two groups G_1 and G_2 currently using group keys K_1 and K_2, respectively, would like to form a super-group. To do so, the two groups exchange their respective key residues: α^{K_1} and α^{K_2} and compute a new super-group key $K_{12} = \alpha^{K_1 K_2}$. The actual exchange can be undertaken by the group controllers. Note that this type of fusion is very fast since it can in principle be accomplished in one round of broadcast. Furthermore, reverting to the original group structure is easy since each group can simply fall back to using K_1 and K_2 at any time thus effectively reversing the fusion but any other group split seems to require two complete and inefficient IKA operations. So unless one only has groups which only grow or only split into previously existing groups it seems easier to use the Mass Join protocol in Figure 6.

5.4 Member Exclusion

The member exclusion protocol is illustrated in Figure 7. In it, M_n effectively "re-runs" the last round of the IKA: as in member addition, it generates a new exponent $\widehat{N_n}$ and constructs a new Broadcast message–but with $\widehat{N_n}N_n$ instead of N_n–using the most recently received Broadcast message. (Note that the last Broadcast message can be from an IKA or any AKA, depending which was the latest to take place.) M_n then broadcasts the message to the rest of the group. The private exponents of the other group members remain unchanged.

[5] Because re-running an IKA involves a *liveness* test of all group members.

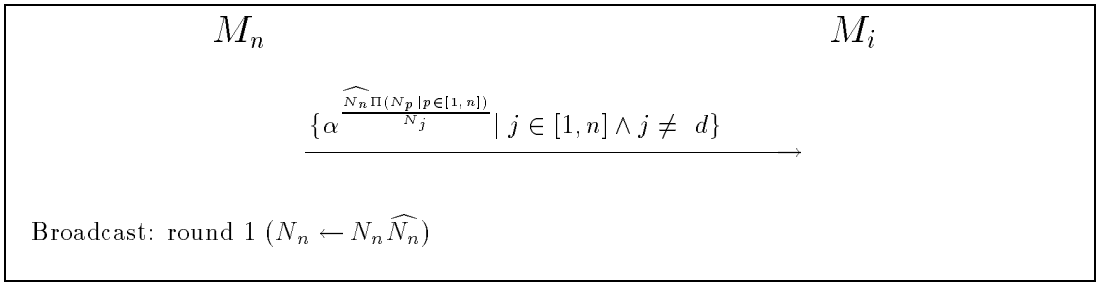

Figure 7: Member Exclusion

Let M_d be the member to be excluded from the group. We assume, for the moment, that $d \neq n$. Since the following sub-key:

$$\alpha^{\widehat{N_n}\,\Pi(N_p|p\in[1,n]\wedge p\neq d)}$$

is conspicuously **absent** from the set of broadcasted sub-keys, the newly excluded M_d is unable to compute the new group key:

$$K_{new} = \alpha^{\widehat{N_n}\,\Pi(N_p|p\in[1,n])}$$

A notable side-effect is that the excluded member's contribution N_d is still factored into the new key. Nonetheless, this in no way undermines the new key's secrecy.

In the event that the current group controller M_n has to be excluded, any other M_i can assume its role, assuming it stored the last Broadcast message.

5.5 Subgroup Exclusion

In most cases, subgroup exclusion is even simpler than single member exclusion. The protocol for mass leave is almost identical to that in figure 7. The only difference is the group controller having to compute and send fewer sub-keys in the final broadcast message. (Only those sub-keys corresponding to remaining members are computed and broadcast.)

A slightly different scenario is that of group division when a monolithic group needs to be split into two or more smaller groups. The obvious way of addressing this is to select for each of the subgroups a subgroup controller which runs the group exclusion protocol within its subgroup by broadcasting only those sub-keys corresponding to subgroup members.

5.6 Key Refresh

There are two main reasons for the group key refresh operation:

- limit exposure due to loss of group session keys
- limit the amount of ciphertext generated with the same key.[6]

Whereas the second reason does not pose any special requirements for the key refresh protocol, the first makes it important for the key refresh protocol not to violate key independence. (For example, this rules out using a straight-forward method of generating a new key as a result of applying a one-way hash function to the old key.) Additionally, we note that loss of a member's key share (N_i) can result in disclosure of all session keys contributed to by this member with this share. Therefore, not only should the session keys, but also individual key shares must be periodically refreshed.

[6] It is easier to perform cryptanalysis with more plaintext/ciphertext pairs.

This leads to following key refresh protocol: The member M_h which is the least recent to have refreshed its key share[7] generates a new share (exponent) $\widehat{N_h}$ and "re-runs" the broadcast round as shown in figure 8

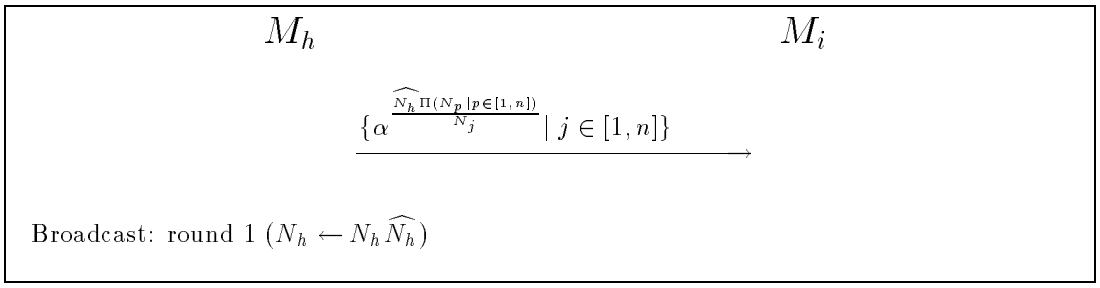

Figure 8: Key Refresh

This procedure guarantees key independence between session keys and limits the damage of leaked key share to at most n epochs. We also note that this one-round protocol can be piggy-packed easily and at almost no cost on a group broadcast which is a likely operation assuming that the established group key is used to protect intra group communication.

6 Security Considerations for AKA Operations

In order to demonstrate security of the AKA protocols, we need to consider a snapshot in a life of a group, i.e., the lifespan and security of a particular short-term key.

The following sets are defined:

- $\mathcal{C} = \{M_1, \ldots, M_c\}$ denotes all **current** group members and M_c is the group controller.
- $\mathcal{P} = \{M_{c+1}, \ldots, M_q\}$ denotes all **past** (excluded before) group members.
- $\mathcal{F} = \{M_{q+1}, \ldots, M_n\}$ denotes all **future** (subsequently added) group members.

Note that the term *future* is used relative to the specific session key. The issue at hand is the ability of all past and future members to compute the current key.

$$K = \alpha^{N_1 \cdots \widehat{N_c} N_{c+1} \cdots N_q}$$

To simplify our discussion we collapse all members of P and F into a single powerful adversary (Eve). (This is especially fitting since P and F are not necessarily disjoint.) The result is that $Eve = \mathcal{P} \cup \mathcal{F}$ and she possesses $\{N_j,\ |M_j \in\ (\mathcal{P}, \mathcal{F})\}$.

We can thus rewrite the key as:

$$K = \alpha^{B(\Pi\mathcal{E})}$$

where B is a *constant* known to Eve, $\mathcal{E} = \{N_1, \ldots, N_{c-1}, \widehat{N_c}\}$ are the secret exponents (contributions) of current group members and $\widehat{N_c}$ is the group controller's exponent. In Eve's view, the only expressions containing $\widehat{N_c}$ are in the last Broadcast round of either member addition or member exclusion protocols:

$$\{\alpha^{\frac{N_1 \cdots N_{c-1}\widehat{N_c}}{N_i}} \mid M_i \in \mathcal{C}\}$$

We can further assume that Eve also knows all:

$$\{\alpha^{\Pi\mathcal{S}} \mid S \subset \mathcal{E}\}$$

However, Eve's knowledge is a subset of what we previously referred to as $view(c, \mathcal{E})$. Recall that in Section 3.2 we have shown that for any n:

$$A_2 \approx_{\text{poly}} D_2 \text{ implies } A_n \approx_{\text{poly}} D_n$$

where:

[7] Other policies, e.g., taking into account the vulnerability of individual members to subversion attacks, are also possible.

- "$\approx_{poly}$" denotes polynomial indistinguishability
- $A_n := (view(n, X), y)$, for a randomly chosen $y \in G$,
- $D_n := (view(n, X), K(n, X))$.
- $view(n, X) :=$ ordered set of all $\alpha^{N_{i_1} \cdots N_{i_m}}$ for all proper subsets $\{i_1, \ldots, i_m\}$ of $\{1, \ldots, n\}$,
- $K(n, X) := \alpha^{N_1 \cdots N_n}$.
- $X = \{N_1, \ldots, N_n\}$

If we substitute n with c, X with $\mathcal{E}$, and $K(n, X)$ with K, it follows that K is polynomially indistinguishable from a random value. □

Consequently, all AKA protocols presented above fall into the class of "natural" DH extensions defined in Section 3.2 and benefit from the same security properties.

7 Related Work

The earliest attempt to extend DH to groups is due to Ingemarsson et al. [8] The protocol in figure 9 (called ING) requires synchronous startup and executes in $(n-1)$ rounds. The members must be arranged in a logical ring. In a given round, every participant raises the previously-received intermediate key value to the power of its own exponent and forwards the result to the next participant. After $(n-1)$ rounds everyone computes the same key K_n.

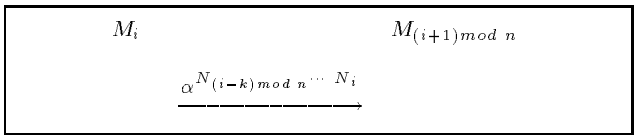

Figure 9: ING Protocol: Round k; $k \in [1, n-1]$

We note that this protocol falls into the class of *natural* DH extensions as defined in [20]. It is, thus, suitable for use as an IKA protocol. However, because of its symmetry[8] (no natural group leader) it is difficult to use it as a foundation for auxiliary key agreement protocols.

Another DH extension geared towards teleconferencing was proposed by Steer et al. in [18]. This protocol (referred to as STR) requires all members to have broadcasting facilities and takes n rounds to complete. In some ways, STR is similar to IKA.1. Both take the same number of rounds and involve asymmetric operation. Also, both accumulate keying material by traversing group members one per round. However, the group key in STR has a very different structure:

$$K_n = \alpha^{N_n \alpha^{N_{n-1} \alpha \cdots N_3 \alpha^{N1 N2}}}$$

Interestingly, STR is well-suited for adding new members; see figure 10. As in IKA.1, it takes only two rounds to add a new member (assuming floating controller). Moreover, this protocol is computationally more efficient than IKA.1 member addition since fewer exponentiations take place. Member exclusion, on the other hand, is difficult in STR since there is no natural group controller. For example, excluding M_1 or M_2 is problematic since their exponents are used in the innermost key computation. In general, re-computing a common key (when M_i leaves) is straight-forward for all M_j, $j < i$. While, all M_j, $j > i$ need to receive input from lower-numbered members.

One notable recent result is due to Burmester and Desmedt [4]. They construct a very efficient protocol (BD) which executes in only three rounds:

[8] It is also not very efficient.

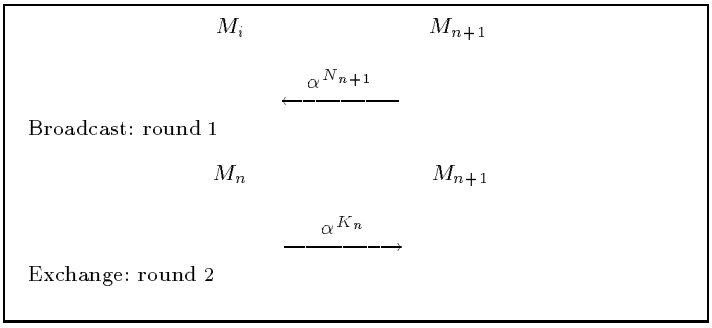

Figure 10: Member Addition in STR.

1. Each M_i generates its random exponent N_i and broadcasts $z_i = \alpha^{N_i}$.
2. Each M_i computes and broadcasts $X_i = (z_{i+1}/z_{i-1})^{N_i}$
3. Each M_i can now compute[9] the key $K_n = z_{i-1}^{nN_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \cdots X_{i-2} \bmod p$

The key defined by BD is different from all protocols discussed thus far, namely $K_n = \alpha^{N_1N_2+N_2N_3+\cdots+N_nN_1}$. Nonetheless, the protocol is proven secure provided the DH problem is intractable.

Some important assumptions underlying the BD protocol:

1. the ability of each M_i to broadcast to the rest of the group
2. the ability to of each M_i to receive $n-1$ messages in a single round
3. the ability of the system to handle n simultaneous broadcasts.

While the BD protocol is efficient and secure, we claim that it is not well-suited for dynamic groups. To demonstrate this claim, we briefly consider what is takes to add a new member in BD. Note that, like in IKA.1, at least one of the current members needs to generate a new exponent whenever a member is added. Assuming synchronized clocks among members, the addition protocol takes two rounds:

- first round to distribute individual contributions $\widehat{z_n}$ and z_{n+1} generated by M_n and M_{n+1} respectively.
- second round for each of: $M_1, M_{n+1}, M_n, M_{n-1}$ to generate and broadcast to the rest of the group: $\widehat{X_1}, \widehat{X_{n+1}}, \widehat{X_n}, \widehat{X_{n-1}}$, respectively. Finally, all group members compute a new key in the usual fashion.

Despite the small number of rounds, every group member[10] needs to receive four messages from four different sources in the second round. This translates into relatively high overhead. Another point of concern is the necessity for all members to keep transient state while the protocol executes, i.e., receiving the four messages in the second round is not an *atomic* operation.

On the other hand, member addition in BD is computationally lighter since no one member performs the bulk of the computation as in IKA.1. This is a definite benefit especially when low-power hardware is involved.

Member exclusion in BD is similar in spirit. As before, at least one remaining member (say, M_n) must generate a new exponent. Assuming that M_1 is to be excluded, a two-round protocol is executed:

- during the first round, M_n distributes its new contribution, $\widehat{z_n} = \alpha^{\widehat{N_i}}$, to M_2 and M_{n-1}. Then:
  - M_n computes $\widehat{X_n} = (z_2/z_{n-1})^{\widehat{N_i}}$

[9] All indexes are modulo n.

[10] Except $M_1, M_{n+1}, M_n, M_{n-1}$ each of which receives three in the second round but at least one in the first.

- M_2 computes $\widehat{X_2} = (z_3/\widehat{z_n})^{N_2}$
- M_{n-1} computes $\widehat{X_2} = (\widehat{z_n}/z_{n-2})^{N_{n-1}}$

- during the second round : M_2, M_{n-1}, and M_n each broadcast to the rest of the group: $\widehat{X_2}, \widehat{X_{n-1}}$, and $\widehat{X_n}$, respectively. Finally, all group members compute a new key in the usual fashion.

Although it is computationally efficient, this protocol requires each member to receive three messages from three sources and to keep transient state in the process.

8 Summary

In summary, this paper identified the requirements for IKA and AKA operations and developed corresponding CLIQUES protocols based on the Diffie-Hellman key exchange. The protocols presented above achieve secure and efficient key agrement in the context of dynamic peer groups. Such groups are relatively small and non-hierarchical. In large groups, it is unclear that key agreement is appropriate since collecting contributions from all members can become very costly. Instead, key distribution mechanisms can be used. This subject (key distribution in large and dynamic groups) is an active research area; for example, [7, 13, 10].

Our emphasis has been on *bare* key agreement resistant to passive attacks. In practice, one must contend with *active* attacks and intruders; to this end, authenticated key agreement must be employed. Related issues include key confirmation, group integrity and member authentication. These and other group security services are addressed in another paper [1].

9 Acknowledgements

The authors thank N. Asokan, G. Ateniese, V. Shoup and U. Wille for comments on the drafts of this paper.

References

[1] G. Ateniese, M. Steiner, and G. Tsudik. Authenticated group key agreement and friends. In *ACM Symposium on Computer and Communication Security*, November 1998.

[2] Stefan Brands. An efficient off-line electronic cash system based on the representation problem. Technical Report CS-R9323, CWI, March 1993.

[3] M. Burmester. On the risk of opening distributed keys. In *Advances in Cryptology – CRYPTO*, 1994.

[4] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology – EUROCRYPT*, 1994.

[5] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[6] The digital signature standard proposed by NIST. *CACM*, 35(7):36–40, Jul 1992.

[7] H. Harney, C. Muckenhirn, and T. Rivers. Group key management protocol (gkmp) architecture, July 1997. RFC 2094.

[8] Ingemar Ingemarsson, Donald Tang, and C. Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, 28(5):714–720, September 1982.

[9] Mike Just and Serge Vaudenay. Authenticated multi-party key agreement. In Ueli Maurer, editor, *Advances in Cryptology – EUROCRYPT 96*, number 1070 in Lecture Notes in Computer Science. Springer-Verlag, Berlin Germany, 1996.

[10] W. Kei, M. Gouda, and S. Lam. Secure group communications using key graphs. Technical report, UT Austin, CS Dept. TR-97-23, 1997.

[11] Michael K.Just. Methods of multi-party cryptographic key establishment. Master's thesis, Carleton University, Computer Science Department, Caleton University, Ottawa, Ontario, August 1994.

[12] Hugo Krawczyk. SKEME: A versatile secure key exchange mechanism for internet. In *Symposium on Network and Distributed Systems Security*, pages 114–127, San Diego, California, February 1996. Internet Society.

[13] D. McGrew and A. Sherman. One-way function trees. Technical report, DARFT, in submission, 1998.

[14] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of applied cryptography*. CRC Press series on discrete mathematics and its applications. CRC Press, 1996. ISBN 0-8493-8523-7.

[15] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of applied cryptography*. CRC Press series on discrete mathematics and its applications. CRC Press, 1996. ISBN 0-8493-8523-7.

[16] H. Orman. The oakley key determination protocol, Version 2.0 1997. INTERNET-DRAFT, work in progress.

[17] C. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[18] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener. A secure audio teleconference system. In S. Goldwasser, editor, *Advances in Cryptology – CRYPTO 88*, number 403 in Lecture Notes in Computer Science, pages 520–528, Santa Barbara, CA, USA, August 1990. Springer-Verlag, Berlin Germany.

[19] M. Steiner, G. Tsudik, and M. Waidner. Refinement and extension of *encrypted key exchange*. *ACM Operating Systems Review*, 29(3):22–30, July 1995.

[20] M. Steiner, G. Tsudik, and M. Waidner. Diffie-hellman key distribution extended to groups. In *ACM Symposium on Computer and Communication Security*, March 1996.

[21] M. Steiner, G. Tsudik, and M. Waidner. Cliques: A new approach to group key agreement. In *IEEE Conference on Distributed Computing Systems*, May 1998.

[22] Douglas R. Stinson. *Cryptography: theory and practice*. CRC Press Series on Discrete Mathematics and Its Applications, edited by Kenneth Rosen. CRC Press, Boca Raton, Florida, 1995.