CLIQUES: A New Approach to Group Key Agreement

Michael Steiner
IBM Zürich 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 Zürich Research Laboratory
CH-8803 Rüschlikon, Switzerland
wmi@zurich.ibm.com

Abstract

This paper considers the problem of key agreement in a group setting with highly-dynamic group member population. A protocol suite, called CLIQUES, is developed by extending the well-known Diffie-Hellman key agreement method to support dynamic group operations. Constituent protocols are provably secure and efficient.

1 Introduction

Popularity of group-oriented applications and protocols is currently on the increase and, as a result, 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 typically necessary to provide communication privacy and integrity.

While peer-to-peer security is a mature and well-developed field, secure group communication remains comparatively unexplored. Contrary to a common initial impression, secure group communication is not a simple extension of secure two-party communication. The greatest 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, in contrast, 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 – most importantly, key agreement.

Key distribution (or key agreement in this context) is the cornerstone of secure communication irrespective of the application domain. In this paper, we develop a protocol suite, called CLIQUES, for key agreement in dynamic groups. However, this paper does not consider other security services such as key integrity, entity authentication, non-repudiation and access control.

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

2 Dimensions of Key Agreement

We distinguish among Initial Key Agreement (IKA) and Auxiliary Key Agreement (AKA) operations. IKA refers to the initial group key agreement, a sort of a *group genesis*. AKA encompasses all subsequent key agreement operations.

We also consider two *types* of group key agreement:

- Centralized: entire key generation is performed by a single entity; typically, a group leader. A special case is the scenario where the key is generated by some trusted third party (TTP) which, itself, is not a group member. (This actually translates into *key distribution*, not *key agreement*.)
- Contributory: each group member makes an independent contribution to the group key. We make a further distinction among two slightly different flavors of contributory key agreement:

  Partially Contributory: some operations result in contributory and others, in centralized, key agreement.
  Fully Contributory: all key agreement operations are contributed to by each group member.

Centralized key agreement is the most intuitive and the most natural. It has been used in a number of past and current mechanisms and its use is commensurate with important advantages as well as certain drawbacks. One such drawback is the overall reliance on a single party.

In the domain of group communication, contributory key agreement, has been, for the most part, restricted to the cryptographic literature [5, 11, 3, 6, 12, 7] and has remained of largely theoretical interest. However, the aesthetics of symmetry, the intrinsic guarantee of key freshness and suitability to group-wide mutual authentication makes it worthwhile to explore.

3 Design Rationale

In this section we provide justification for certain choices made in the development of CLIQUES protocols.

- Why Diffie-Hellman?
  Several factors motivate our choice of using Diffie-Hellman (DH) key agreement as a basic building block. First, since it was proposed in 1976, DH has been scrutinized by many able cryptographers, implemented in many crypto libraries and software packages and deployed in a

variety of settings. Thus far, it withstood the test of time rather well. Second, DH is simple, requiring no complex arithmetic outside of exponentiation in finite fields. Third, DH is *democratic*, i.e., both parties contribute equally to the shared secret.

- Why partially contributory model?
From the choice of DH key agreement it follows that centralized model is essentially ruled out. On the other hand, fully contributory model, while desirable, is impractical since it implies that all AKA operations would have to mimic IKA, i.e., there would be little (if any) distinction between IKA and AKA.[1] Partially contributory model represents a workable compromise.

- Why have group controllers?
The main reason for having group controllers is, once again, to support efficient AKA operations. As will be seen in section 8, some IKA protocols support leaderless (or symmetric) operation. However, we argue that **group controller is necessary** if only for the purposes of adding and excluding members, i.e., managing group membership. The alternative appears rather complex and unattractive.

4 Initial Key Agreement

As defined in section 2, IKA takes place at the time of group *genesis*. It is the time when protocol overhead must be minimized since key agreement must naturally precede any kind of secure group communication. On the other hand, for 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. Or, equivalently, the inability to recover the key by mere eavesdropping.

Naturally, IKA requires contacting every group member-to-be. Contributory key agreement also calls for a key share to be solicited 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 infrequent, no AKA protocols may be necessary. The exception might be key refresh but this can be mimicked (though expensivly) with IKA.

5 Auxiliary Key Agreement

As mentioned above, the initial full-blown 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 the other, auxiliary group key operations and the attendant security issues. (See also figure 1.)

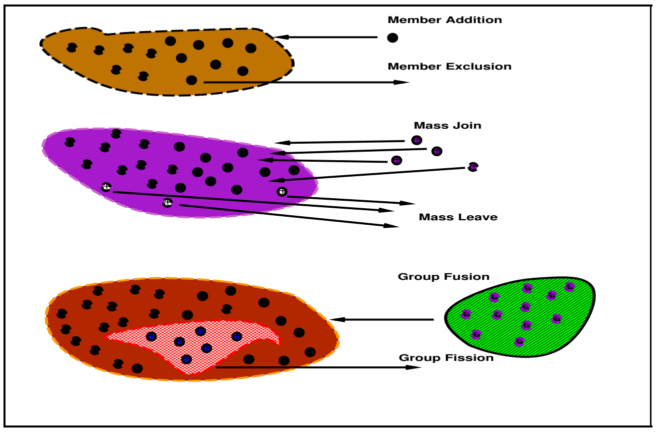

Figure 1: AKA Operations

5.1 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.

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), i.e., 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) [8, 2]. 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 [2]. 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 *perfect forward secrecy* (PFS) [10, 8] since no long-term secrets are assumed in this context. (Recall that PFS is premised on the possibility of compromise of long-term secrets.)

To be more precise, our communication model assumes all communication to be **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,

[1] All types of group key changes would be equally expensive; more on this below.

an adversary in the IKA/AKA protocols can be an outsider or a quasi-insider. An outsider is a passive adversary not party to 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 his arrival, it might be best not to change the current group key so as to allow the tardy participant to catch up. However, this decision should be determined by local policy.

5.2 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.

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

5.3 Group Key Refresh

For a variety of reasons it is often necessary to perform a routine key change/refresh 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 from key changes resulting from membership changes, we will refer to this operation as *key refresh*.

6 CLIQUES Protocol Suite

We now turn to the actual protocols employed in the CLIQUES suite. Their initial derivation can be found in [12]. Also included in [12] is a blanket proof for an entire protocol class collectively referred to as the *generic* n-party Diffie-Hellman (DH) key agreement.[2] In brief, it is shown that:

> *If a 2-party DH key is indistinguishable from a random value then an n-party DH key is also indistinguishable from a random value.*

[2] Of course, where $n > 2$.

In the rest of this section we walk through the entire protocol suite and, in the process, discuss and evaluate the proposed mechanisms.

n	number of protocol participants (group members)
i, j, k	indices of group members
M_i	i-th group member; $i \in [1, n]$
q	order of the algebraic group
g	exponentiation base; generator in the algebraic group delimited by q
N_i	random (secret) exponent 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

Table 1: Notation used in the remainder of the paper.

6.1 Initial Key Agreement

The cornerstone of the CLIQUES protocol suite is the IKA protocol called GDH.2 depicted in figures 2. (The name GDH.2 is kept for historical reasons; see [12].) This protocol executes in n rounds. In the first stage ($n-1$ rounds) contributions are collected from group members and in the second stage (round n) the group keying material is broadcast.

$$M_i \xrightarrow{\{\mathrm{g}^{\frac{N_1 \cdots N_i}{N_k}} \mid k \in [1,i]\},\ \mathrm{g}^{N_1 \cdots N_i}} M_{i+1}$$

Stage 1 (Upflow): round i; $i \in [1, n-1]$

$$M_i \xleftarrow{\{\mathrm{g}^{\frac{N_1 \cdots N_n}{N_i}} \mid i \in [1,n]\}} M_n$$

Stage 2 (Broadcast): round n

Figure 2: Group Initial Key Agreement: GDH.2

In more detail, the i-th round of stage 1 is as follows:

1. M_i ($0 < i \leq 1$) receives a sequence of $(i-1)$ intermediate key values and one *cardinal* value. (M_1 can be thought of as implicitly receiving an *empty set* of intermediate values and a cardinal value of $K_0 = \mathrm{g}$.)

   For each subset of $i-2$ lower-indexed members (and there are $i-1$ of those) an intermediate value is the group key for that subset.

   The cardinal value is essentially the group key for all $(i-1)$ preceding group members, i.e., $K_{i-1} = \mathrm{g}^{N_1 \ldots N_{i-1}}$.
2. M_i generates its own contribution N_i.
3. Raises each received intermediate value to the power of N_i thus producing a set of new intermediate values.

4. The old cardinal value is added to the set of new intermediate values.

5. M_i computes the new cardinal value $K_i = (K_{i-1})^{N_i}$.

6. If $i < n$, M_i sends K_i and the new intermediate values to M_{i+1}.

The highest-indexed group member M_n plays a special role of a group controller; it is required to complete stage 1 and initiate stage 2. However, despite the communication asymmetry, GDH.2 is computationally symmetric, i.e., even M_n performs the sequence of n exponentiations in the same order as described above.

The last round is M_n broadcasting $n-1$ intermediate values to the entire group. (Of course, the last cardinal value cannot be broadcast as it is the actual group key.) Each receiving M_i identifies its intermediate value (i.e., a group key corresponding to the other $n-1$ members) and exponentiates it with N_i thus computing the final group key.

It is important to note that stage 2 does not have to be a true broadcast. If broadcast services are either unavailable or the size of the last message is an issue, M_n may choose to perform $(n-1)$ unicasts instead. The unicasts can be simultaneous or serial depending on the underlying network technology.

6.2 Member Addition

The member addition protocol is shown in figure 3. 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 Upflow message that it received in round $n-1$ in the IKA protocol of figure 2.[3]

In effect, M_n extends Stage 1 of the IKA protocol by one round: it generates a new exponent $\widehat{N_n}$ and creates a new upflow message (with $\widehat{N_n}$ instead of N_n) using the contents of the previously received Upflow message. It then forwards the message to the new member which, in turn, takes the same sequence of steps as M_n in the IKA protocol.

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 $n+1$.

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

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

M_n $\qquad\qquad$ M_{n+1}

$$\xrightarrow{\{g^{\frac{N_1 \ldots \widehat{N_n}}{N_k}} \mid k \in [1,n]\},\, g^{N_1 \ldots \widehat{N_n}}}$$

Upflow: round 1

M_i $\qquad\qquad$ M_{n+1}

$$\xleftarrow{\{g^{\frac{N_1 \ldots \widehat{N_n} N_{n+1}}{N_i}} \mid i \in [1,n]\}}$$

Broadcast: round 2

Figure 3: Member Addition: floating group controller

The resultant protocol is shown in figure 4. The first message is a duplicate of either the upflow message in round $(n-1)$ of the original IKA protocol (only if this is the first member addition) or the upflow message in round 2 of the last member addition protocol.

M_n – new member $\qquad\qquad$ M_{n+1} – formerly M_n

M_n $\qquad\qquad$ M_{n+1}

$$\xleftarrow{\{g^{\frac{N_1 \ldots N_{n-1}}{N_i}} \mid i \in [1, n-1]\},\, g^{N_1 \ldots N_{n-1}}}$$

$$\xrightarrow{\{g^{\frac{N_1 \ldots N_n}{N_i}} \mid i \in [1,n]\},\, g^{N_1 \ldots N_n}}$$

Simulated Upflow : rounds 1 &2
(N_n - new member's contribution)

M_i $\qquad\qquad$ M_{n+1}

$$\xleftarrow{\{g^{\frac{N_1 \ldots N_{n+1}}{N_i}} \mid i \in [1,n]\}}$$

Broadcast: round 3

Figure 4: Member Addition: fixed group controller

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.

6.3 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 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 treated 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* soon thereafter. To be more specific, if m new members were to be added in this fashion, the cost would be:

- $3m$ ($2m$) rounds with fixed (floating) controller
- 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 5. 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 5 shows only the floating controller model. A chained fixed controller model can be trivially and similarly constructed from the protocol in figure 4.

M_{n+j} $\qquad\qquad$ M_{n+j+1}

$$\xrightarrow{\{g^{\frac{N_1\cdots N_{n+j}}{N_k}} \mid k \in [1, n+j]\} \quad g^{N_1\cdots\widehat{N_n}N_{n+1}\cdots N_{n+j}}}$$

Upflow: round j $1 \leq j \leq m$

M_i $\qquad\qquad$ M_{n+m}

$$\xleftarrow{\{g^{\frac{N_1\cdots N_{n+m}}{N_i}} \mid i \in [1, n+m]\}}$$

Broadcast: round $m+1$

Figure 5: Mass Join (floating controller)

6.4 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: 1) special case of mass join as in figure 5 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 [11]. 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: g^{K_1} and g^{K_2} and compute a new super-group key $K_{12} = g^{K_1 K_2}$. 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.

6.5 Member Exclusion

The member exclusion protocol is illustrated in figure 6. The chief assumption here is that the only entity having the authority to exclude group members is the current group controller.

In the present protocol, 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 $\widehat{N_n}$ instead of N_n–using the most recently received Upflow message. (Note that the last Upflow message can be from an IKA or member addition, 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.

M_n $\qquad\qquad$ M_i

$$\xrightarrow{\{g^{\frac{N_1\cdots\widehat{N_n}}{N_i}} \mid i \in [1, n-1] \wedge i \neq p\}}$$

Broadcast: round 1

Figure 6: Member Exclusion

Let M_p be the member to be excluded from the group. We assume, for the moment, that $p \neq n$. (Excluding a group controller is a special case addressed below.) Since the following sub-key:

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

$$g^{N_1 \dots N_{p-1} N_{p+1} \dots N_{n-1} \widehat{N_n}}$$
is conspicuously **absent** from the set of broadcasted sub-keys, the newly excluded M_p is unable to compute the new group key:
$$K_{new} = g^{N_1 \dots N_p \dots N_{n-1} \widehat{N_n}}$$
A notable side-effect is that the excluded member's contribution N_p 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, M_{n-1} must assume its role. Barring a complete group rekey (i.e., re-running the IKA or somehow soliciting fresh contributions from other group members) only M_{n-1} has the material sufficient to perform a fast key update and *depose* M_n. This material is the last Upflow message seen by M_{n-1}. The protocol itself is identical to the one in figure 6.

Conceptually, requiring M_{n-1} to assume the group controller's role is reasonable in the floating controller model since M_{n-1} is always the most recent controller. It is less clear in the fixed controller model. Perhaps, the original indexing of group members should take into account their abilities to function as group controllers. In other words, higher numbered members are more trusted, better equipped and more likely to be, group controllers.

6.6 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 6. 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 first exclude the future members of the smaller group via the mass leave protocol and, then, to create a new group made up of those excluded members. In contrast to its counterpart (group fusion), we argue that group fission does not warrant special treatment, i.e., a mechanism distinct from those illustrated thus far. The chief reason is that, in this case, the obvious solution works well.

The remaining case is that of group fission: splitting a group that was formed as a result of group fusion. Ideally, the group controllers of all previously fused groups are still *alive and well* at the time of fission. If so, the entire task can be reduced to each former group controller broadcasting a key update much as in the mass leave or member exclusion scenarios. If a former controller is unavailable for some reason, its role must be assumed by the next highest indexed (within the original group) member. This has some implications for state retention, i.e., group members must keep state pertaining to the group before fusion.

7 Security Considerations

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:

- $\mathcal{C} = \{M_1, \dots, M_p\}$ denotes all **current** group members and M_p is the group controller.
- $\mathcal{P} = \{M_{p+1}, \dots, M_q\}$ denotes all **past** (excluded before) group members.
- $\mathcal{F} = \{M_{q+1}, \dots, M_n\}$ 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.
$$K = g^{N_1 \dots \widehat{N_p} N_{p+1} \dots N_q}$$
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:
$$K = g^{B(\Pi\mathcal{E})}$$
where B is a *constant* known to Eve, $\mathcal{E} = \{N_1, \dots, N_{p-1}, \widehat{N_p}\}$ are the secret exponents (contributions) of current group members and $\widehat{N_p}$ is the group controller's exponent. In Eve's view, the only expressions containing $\widehat{N_p}$ are in the last Broadcast round of either member addition or member exclusion protocols:
$$\{g^{\frac{N_1 \dots N_{p-1} \widehat{N_p}}{N_i}} \,|\, M_i \in \mathcal{C}\}$$
We can further assume that Eve also knows all:
$$\{g^{\Pi S} \,|\, S \subset \mathcal{E}\}$$
However, Eve's knowledge is a subset of what in [12] is called $view(p, \mathcal{E})$. The following theorem is proven in [12]:

Theorem: For each constant n, $A_2 \approx_{\text{poly}} D_2$ implies $A_n \approx_{\text{poly}} D_n$.

where:

- "$\approx_{\text{poly}}$" 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 $g^{N_{i_1} \cdots N_{i_m}}$ for all proper subsets $\{i_1, \dots, i_m\}$ of $\{1, \dots, n\}$,
- $K(n, X) := g^{N_1 \cdots N_n}$.
- $X = \{N_1, \dots, N_n\}$

If we substitute n with p, 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 [12] and benefit from the same security properties.

8 Related Work

The earliest attempt to extend DH to groups is due to Ingemarsson et al. [5] The protocol in figure 7 (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.

$$
\begin{array}{lcl}
M_i & & M_{(i+1) mod\ n} \\
 & \xrightarrow{g^{(\Pi\{N_j \mid j \in [(i-k) mod\ n, i]\})}} &
\end{array}
$$

Figure 7: 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 [12]. It is, thus, suitable for use as an IKA protocol. However, because of its symmetry[6] (no 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 [11]. This approach (STR) requires all members to have broadcasting facilities and takes n rounds to complete. In some ways, STR is similar to GDH.2 IKA. 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:

$$K_n = g^{N_n g^{N_{n-1} g^{\cdots N_3 g^{N_1 N_2}}}}$$

Interestingly, STR is well-suited for adding new members; see figure 8. It takes only two rounds to add a new member just like its counterpart in GDH.2 (with floating controller). Moreover, this protocol is computationally more efficient than GDH.2 member addition since fewer exponentiations take place. are based on a structure where they assume Member exclusion, on

$$
\begin{array}{lcl}
M_i & & M_{n+1} \\
 & \xleftarrow{g^{N_{n+1}}} & \\
M_n & & M_{n+1} \\
 & \xrightarrow{g^{K_n}} &
\end{array}
$$

Figure 8: Member Addition in STR.

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_p, $p > i$ need to receive input from lower-numbered members.

One notable recent work is due to Burmester and Desmedt [3]. They designed a much more efficient protocol (BD) which executes in only three rounds:

1. Each M_i generates its random exponent N_i and broadcasts $z_i = g^{N_i}$.

2. Each M_i computes and broadcasts $X_i = (z_{i+1}/z_{i-1})^{N_i}$

3. Each M_i can now compute[7] 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 the previous protocols, namely $K_n = g^{N_1N_2+N_2N_3+\cdots+N_nN_1}$. Nevertheless, the protocol is proven secure provided the DH problem is intractable.

Some important assumptions underlying this protocol are:

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 as suitable as GDH.2 for dynamic groups. To demonstrate this claim, we briefly consider what is takes to add a new member in BD. Note that, like in GDH.2, 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[8] 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 GDH.2. This is a definite benefit especially whenever low-power hardware is used.

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, $\widehat{z_n} = g^{\widehat{N_i}}$, to M_2 and M_{n-1}. Then:

[6] It is also not very efficient.

[7] All indexes are modulo n.

[8] 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_n computes $\widehat{X_n} = (z_2/z_{n-1})^{\widehat{N_i}}$
- 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.

9 On-going and Future Work

In summary, this paper represents only an initial attempt to analyze the requirements for, and specify, key agreement protocols for dynamic groups. There remain a number of issues and topics for future work:

- Better group fusion
  Group fusion has not yet been addressed in a satisfactory manner. More efficient and elegant solutions need to be investigated.

- Key integrity provisions
  IKA and AKA protocols discussed above provide nothing but key agreement in the spirit of *raw* DH. To be truly useful, key integrity must be offered, i.e., the protocols need to achieve *authenticated* key agreement.

- Member authentication
  It is certainly possible to treat entity (member) authentication as being orthogonal to key agreement. However, if only for efficiency's sake, it pays to combine member authentication with authenticated key agreement.

- Scaling issues
  This paper carefully side-stepped the issue of scale. Clearly, CLIQUES protocols can become prohibitively expensive as group sizes grow and so does the rate of membership change. The traditional answer to scale has been the imposition of some kind of a hierarchy [9, 4, 1]. However, it is unclear how to construct a key agreement hierarchy without unpleasant consequences, such as having to do key translation.

- API definition
  One of the long-term goals of the CLIQUES project is the development of a generic toolkit for group-oriented security services. (Key agreement forms the foundation thereof.) Service primitives within the toolkit must be accessible via a well-defined and flexible API.

10 Acknowledgements

We thank Giuseppe Ateniese and the anonymous referees for many useful comments.

References

[1] A. Ballardie. Scalable multicast key distribution. In *INTERNET Request for Comments: RFC 1949*, May 1996.

[2] M. Burmester. On the risk of opening distributed keys. In *Advances in Cryptology – CRYPTO'94*, pages 308–317, 1994.

[3] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology – EUROCRYPT'94*, May 1994.

[4] H. Harney, C. Muckenhirn, and T. Rivers. Group key management protocol (gkmp) architecture. In *INTERNET-DRAFT, work in progress*, Version 1.0 1996.

[5] I. Ingemarsson, D. Tang, and C. Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, September 1982.

[6] M. Just. Methods of multi-party cryptographic key establishment. Master's thesis, Carleton University, Computer Science Department, August 1994.

[7] M. Just and S. Vaudenay. Authenticated multiparty key agreement. In *Advances in Cryptology – EUROCRYPT'96*, May 1996.

[8] 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.

[9] S. Mittra. Iolus: A framework for scalable secure multicasting. In *ACM SIGCOMM'97*, September 1997.

[10] H. Orman. The oakley key determination protocol. In *INTERNET-DRAFT, work in progress*, Version 1.0 1996.

[11] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener. A secure audio teleconference system. In *Advances in Cryptology – CRYPTO'88*, August 1990.

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