Key Agreement in Dynamic Peer Groups

Michael Steiner, Gene Tsudik, *Member*, *IEEE Computer Society*, and
Michael Waidner, *Member*, *IEEE Computer Society*

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 videoconferencing. 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 stepping 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 multiparty extensions of the well-known Diffie-Hellman key exchange method. The protocols are efficient and provably secure against passive adversaries.

Index Terms—Collaborative work, communication system security, cryptography, decision Diffie-Hellman problem, dynamic peer groups, key establishment/agreement protocols, multiparty computation.

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 videoconferencing. 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. First, 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 the following, we specifically focus on the requirements of **Dynamic Peer Groups (DPGs)**. DPGs are common in many layers of the network protocol stack and many application areas of modern computing. Examples of DPGs include replicated servers (such as database, web, time), audio and video conferencing, and more generally, collaborative applications of all kinds. In contrast to large multicast groups, DPGs tend to be relatively small in size, on the order of a hundred members. (Larger groups are harder to control on a peer basis and are typically organized in a hierarchy of some sort.) DPGs typically assume a many-to-many communication pattern rather than one-to-many commonly found in larger, hierarchical groups.

In this paper, we concentrate on secure and efficient group key agreement. We start in Section 2 by discussing contributory key agreement and requirements in supporting the dynamics of groups. In Section 3, we define a class of protocols that we call "natural" extensions of the two-party Diffie-Hellman key exchange [1] and prove the security of all protocols in this class against passive adversaries, provided the two-party **Decisional Diffie-Hellman (DDH)** 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 in Section 4, we present two new protocols, each optimal with respect to certain aspects of protocol efficiency. Subsequently in Section 5, 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 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 nonrepudiation of group membership. Protocols and mechanisms in support of these are treated in another paper [2]. In Section 6, we compare our work with related work and conclude in Section 7.

- *M. Steiner is with the Universität des Saarlandes, 66123 Saarbrücken, Germany. E-mail: steiner@acm.org.*
- *G. Tsudik is with the Department of Information and Computer Science, University of California, Irvine, CA 92697-3425. E-mail: gts@ics.uci.edu.*
- *M. Waidner is with the IBM Research Division, Zürich Research Laboratory, CH-8803 Rüschlikon, Switzerland. E-mail: wmi@zurich.ibm.com.*

Manuscript received 29 Jan. 1999; revised 18 Feb. 2000; accepted 23 Apr. 2000.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 109078.

2 Dimensions of Key Agreement

All of our protocols are based on **contributory key agreement**. This means that a group key K is generated as $f(N_1, \ldots, N_n)$, where $f()$ is some one-way function and N_i is an input (or key share) randomly chosen by the ith 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 N_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 [3], [4], [5], [6], [7], [8], [9]; however, none have been widely used. In practice, group key agreement is typically done in a *centralized* manner [10], [11], [12]: one dedicated party (typically, a group leader) chooses the group key and distributes it to all group members. This actually translates into **key transport** or **key distribution**, 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 (nonhierarchical) groups with dynamically changing membership.

A permanently fixed group leader is a potential performance bottleneck and a single point of failure. Some DPG environments (such as ad hoc wireless networks) are highly dynamic and no group member can be assumed to be present all the time. This is also the case in wired networks when high availability is required. Therefore, our view is that fault tolerance (such as handling network partitions and other events) is best achieved by treating all parties as peers. This is supported by the state-of-the-art in reliable group communication (see, for example, [13].)

To achieve fault tolerance of our protocols, we use reliable group communication means. In the following, we assume an underlying group communication system resistant to fail-stop failures. This system should provide a consistent membership view to all group members and reliable and causally ordered multicasts. However, note that secure group key agreement protocols such as CLIQUES depend on such group communication systems only to guarantee *liveness* (e.g., to prevent trivial denial of service) but *not to ensure safety*. Nevertheless, the integration of group key agreement and reliable group communication to form a secure group communication system raises a number of issues such as efficient handling of various cascading failures. Owing to the built-in flexibility of CLIQUES protocols, these issues can be resolved in an efficient and modular manner without interfering with the security properties discussed in this paper. For further information we refer the reader to some recent work [14] which reports on the integration of CLIQUES with the SPREAD [15] reliable group communication system.

There is no inherent reason to require a single group leader to make the decisions as to whom to add to, or exclude from, a group.[1] Ideally, decisions regarding *who* can add a new member or delete an old one should be taken according to some local group policy. 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 approach supports it quite elegantly and efficiently: any party can initiate all membership change protocols.

Although we argue in favor of distributed, contributory key agreement for DPGs, we also recognize the need for a central point of control for group membership operations, such as adding and deleting members. This type of a role (group controller) serves only to synchronize the membership operations and prevent chaos. However, the existence and assignment of this role is orthogonal to key establishment, can be changed at any time, and is largely a matter of policy.

A further advantage of contributory schemes is that they automatically provide freshness and assure randomness of new keys: each party i can check whether N_i was included in K, hence, whether K is fresh and random. Furthermore, our protocols can be easily extended to *authenticated* group key agreement providing **perfect forward secrecy (PFS)** [16], [17] and resistance to active **known-key attacks (KKA)** [18], [19], [17], as shown in [2]. This is necessary for robust protocols withstanding strong *active* attacks. We note that most key transport protocols fail to provide at least one of PFS and KKA resistance.

In the following, 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.

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 prerequisite 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 two-party Diffie-Hellman key agreement—means resistance to passive attacks. Equivalently, this means the inability to recover any information of 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

1. One obvious issue with having a fixed group leader is how to handle its expulsion from the group.

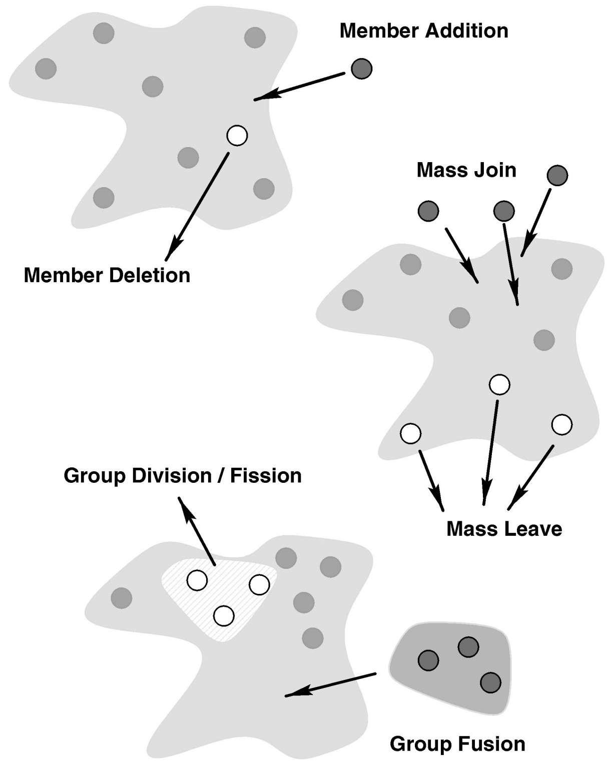

Fig. 1. AKA "Operations."

be possible to coincide (or interleave) with the IKA other security services such as authentication, access control and nonrepudiation. This is something to keep in mind for the follow-on work.

2.2 Auxiliary Key Agreement (AKA) 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 (see Fig. 1). In this section, we discuss other auxiliary group key operations and the attendant security issues (see also Fig. 2).

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 KKA. 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 [19]. Since this paper (and our protocol model) is concerned with unauthenticated key agreement, 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.) However, note that our protocols provide an ideal basis to achieve PFS in authenticated group key agreement protocols [2].

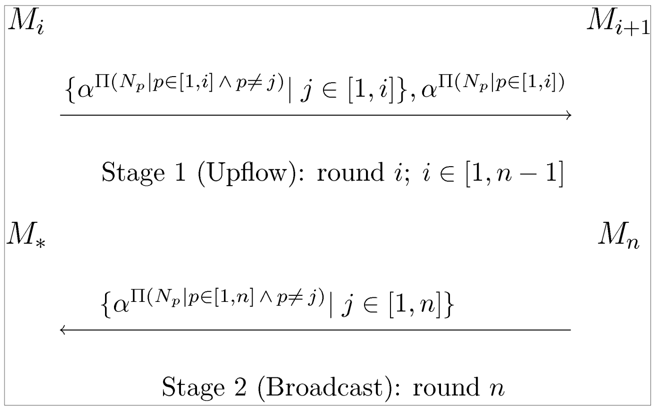
M_i
M_{i+1}
$\{\alpha^{\Pi(N_p|p\in[1,i] \wedge p\neq j)} \mid j\in[1,i]\}, \alpha^{\Pi(N_p|p\in[1,i])}$
Stage 1 (Upflow): round i; $i\in[1,n-1]$
M_*
M_n
$\{\alpha^{\Pi(N_p|p\in[1,n] \wedge p\neq j)} \mid j\in[1,n]\}$
Stage 2 (Broadcast): round n

Fig. 2. Group key agreement: IKA.1.

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 some of the intended participants running late. Upon their arrival, it might be best not to change the current group key so as to allow the tardy participants to catch up.[2] Although the new member has not contributed to the group key, it can do so later by initiating a key refresh. 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 multilateral 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,

2. Adding a new member without changing a key is trivial: the controller sends the new member the current key (using, for example, El Gamal encryption with a one-time key chosen by the new member).

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.[3]

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 from key updates due to membership changes, we will refer to this operation as **key refresh**.

3 Generic n-Party Diffie-Hellman Key Agreement

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

n	number of protocol participants rsp. group members
i, j, h, p, d, c	indices of group members
M_i	i-th group member; $i \in [1, n]$
M_*	all group member
G	cyclic algebraic group of prime order
q	order of the algebraic group, prime
α	exponentiation base; generator in the algebraic group G delimited by q
N_i	secret exponent $\in_{\mathcal{R}} \mathbb{Z}_q$ generated by M_i
$\mathcal{S}$	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 two-party Diffie-Hellman key exchange: Like in the two-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, chooses randomly a value $N_i \in \mathbb{Z}q$. The group key will be $K = \alpha^{N_1 \cdots N_n}$.

In the two-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 two-party case is directly based on the two-party DDH problem: Given $(\alpha, \alpha^{N_1}, \alpha^{N_2}, \alpha^X)$, 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**[4]: 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 two-party DDH problem is hard, then the n-party DDH problem is hard, as well. This proves the security of the 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 polynomial time with k and n as inputs.

For concreteness, we consider a specific class of groups G: On input k, algorithm *gen* chooses at random a pair (q, α) where q is a k-bit value, q and $q' = tq + 1$ 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 [22] and DSS [23]. It is commonly assumed that the two-party DDH problem is hard for these groups, i.e., for all polynomial time attackers A, for all polynomials $Q(k)$, for $X_0 := N_1 N_2$ and $X_1 := N_3$ with $N_1, N_2, N_3 \in_{\mathcal{R}} \mathbb{Z}q$ uniformly chosen, and for a random bit b, the probability that $A(1^k, G, \alpha, \alpha^{N_1}, \alpha^{N_2}, \alpha^{X_b}) = b$ is smaller than $(\frac{1}{2} + \frac{1}{Q(k)})$.

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 two-party Diffie-Hellman key is considered, e.g., in [24]. The notion of polynomial indistinguishability is related to polynomial time statistical test as defined in [25], [17]. 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 than $\frac{1}{2}$. 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

3. 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.

4. Note that this problem is also known in the literature as the decisional version of the Generalized Diffie-Hellman (GDH).

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:

$$\Big(view(n-1,(N_1,X)), K(n-1,(N_1,X)), \\ view(n-1,(N_2,X)), K(n-1,(N_2,X)), \\ view(n-1,(N_1N_2,X)) \Big)$$

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. 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 $y \in G$ at random.

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

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 specific 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 [7].)

4.1 IKA.1

The first IKA protocol (IKA.1), depicted in Fig. 2 is simple and straightforward. It consists of upflow and downflow stages.

The purpose of the upflow stage is to collect contributions from all group members, one per round. In round i ($i \in [1, n-1]$), M_i unicasts M_{i+1} a collection of i values. Of these, $i-1$ are intermediate, and one is *cardinal*. The cardinal value CRD_i is simply the generator raised to all secret exponents generated so far:

$$\mathrm{CRD}_i := \alpha^{\Pi(N_p | p \in [1,i])}.$$

Let $\mathrm{INT}_{i,j}$ denote the jth intermediate value in round i. It is always of the following form (i.e., CRD_i with the jth exponent missing):

$$\mathrm{INT}_{i,j} := \alpha^{\Pi(N_p | p \in [1,i] \wedge p \neq j)} \qquad \text{for } j \in [1,i].$$

M_i's computations can now be described as follows:

1. generate private exponent N_i
2. set $\mathrm{INT}_{i,j} = (\mathrm{INT}_{i-1,j})^{N_i}$ for all $j \in [1, i-1]$
3. set $\mathrm{INT}_{i,i} = \mathrm{CRD}_{i-1}$
4. set $\mathrm{CRD}_i = (\mathrm{CRD}_{i-1})^{N_i}$.

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:

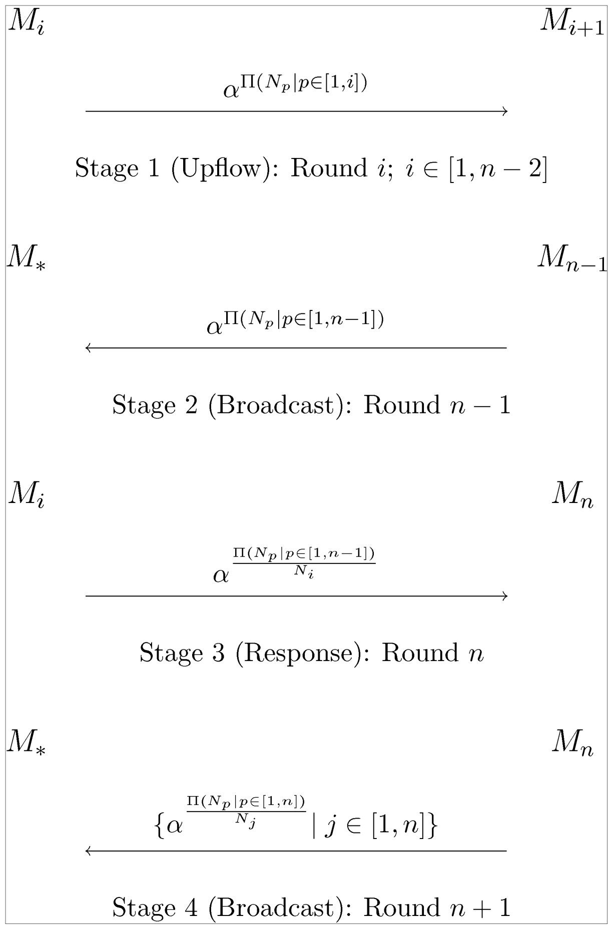
M_i
M_{i+1}
$\alpha^{\Pi(N_p|p\in[1,i])}$
Stage 1 (Upflow): Round i; $i \in [1, n-2]$
M_*
M_{n-1}
$\alpha^{\Pi(N_p|p\in[1,n-1])}$
Stage 2 (Broadcast): Round $n-1$
M_i
M_n
$\alpha^{\frac{\Pi(N_p|p\in[1,n-1])}{N_i}}$
Stage 3 (Response): Round n
M_*
M_n
$\{\alpha^{\frac{\Pi(N_p|p\in[1,n])}{N_j}} \mid j \in [1,n]\}$
Stage 4 (Broadcast): Round $n+1$

Fig. 3. Group key agreement: IKA.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. You might also wonder why we broadcast the last flow and don't unicast the $n-1$ shares individually to save some bandwidth. The reason for this will become apparent later in Section 5, when we talk about AKA operations: This allows us to achieve policy independence on group controllership.

4.2 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 smartcards 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 Fig. 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 (nth round), $n-1$ unicast messages are sent to M_n. This might lead to congestion at M_n.

5 Cliques: Auxiliary Key Agreement

Both IKA protocols operate in two phases: a gathering phase whereby M_n collects contributions from all participants to compute $\{\alpha^{\frac{N_1\cdots N_n}{N_i}}|i\in[1,n]\}$ 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 redistributed to the new incarnation of the group. In particular, any member who caches the most recent message of the final broadcast round can initiate an AKA operation. Any member can take over the role of group controller at no cost and whenever the situation requires it, e.g., when the former group controller abruptly disappears due to a crash or network partition. This way, our protocols achieve complete policy independence.

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 Fig. 4. As mentioned above, we assumed that the current group controller M_c $(c \in [1,n])$ remembers the contents of the

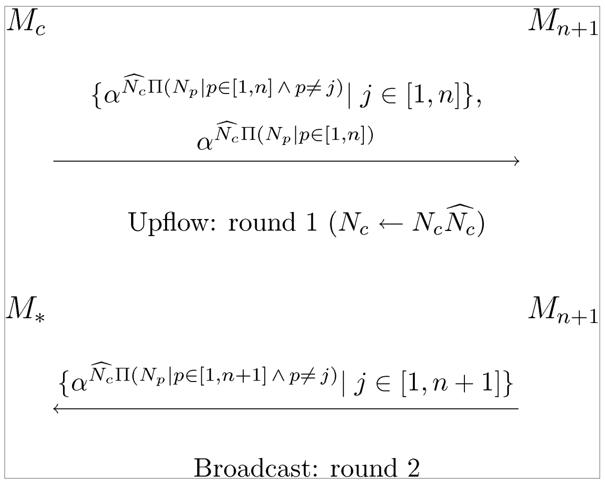
M_c
M_{n+1}
$\{\alpha^{\widehat{N_c}\Pi(N_p|p\in[1,n]\wedge p\neq j)} | j\in[1,n]\},$
$\alpha^{\widehat{N_c}\Pi(N_p|p\in[1,n])}$
Upflow: round 1 ($N_c \leftarrow N_c\widehat{N_c}$)
M_*
M_{n+1}
$\{\alpha^{\widehat{N_c}\Pi(N_p|p\in[1,n+1]\wedge p\neq j)} | j\in[1,n+1]\}$
Broadcast: round 2

Fig. 4. Member addition.

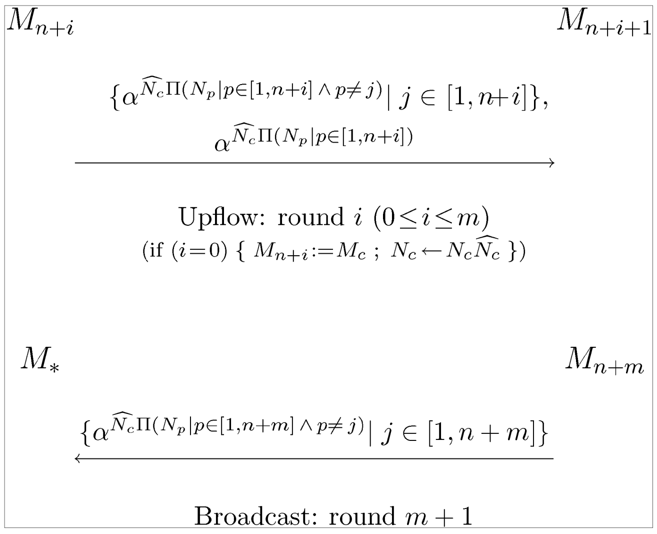
M_{n+i}
M_{n+i+1}
$\{\alpha^{\widehat{N_c}\Pi(N_p|p\in[1,n+i]\wedge p\neq j)} | j\in[1,n+i]\},$
$\alpha^{\widehat{N_c}\Pi(N_p|p\in[1,n+i])}$
Upflow: round i ($0\leq i\leq m$)
(if ($i=0$) { $M_{n+i}:=M_c$; $N_c\leftarrow N_c\widehat{N_c}$ })
M_*
M_{n+m}
$\{\alpha^{\widehat{N_c}\Pi(N_p|p\in[1,n+m]\wedge p\neq j)} | j\in[1,n+m]\}$
Broadcast: round $m+1$

Fig. 5. Mass join.

broadcast message that was sent in the last round in the IKA protocol of Fig. 2.[5]

In effect, M_c extends Stage 1 of the IKA protocol by one round: it generates a new exponent $\widehat{N_c}$ and creates a new upflow message. $\widehat{N_c}N_c$ is used in place of N_c to prevent the new member and outsiders from learning the old group key. Additionally, it replaces N_c by $\widehat{N_c}N_c$ as its own contribution for further AKA operations.

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, new members are disparate (i.e., have no prior common association) and need to be added in a hurry. Alternatively, 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:

- $2m$ rounds.
- 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 Fig. 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.

5. This is only the case for the very first member addition; subsequent member additions as well as other AKA operations require the current controller to save the most recent broadcast message from the preceding AKA operation.

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 preexisting 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 Fig. 5, or
2 Creation of a new super-group via IKA of Fig. 2.

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 toward 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 Becker et al. in [9]. 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 $K_{12}=\alpha^{K_1K_2}$. The actual exchange can be undertaken by the group controllers. Note that this type of fusion is very fast, since in principle it can 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 Fig. 5.

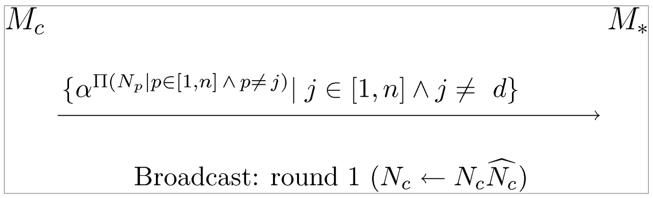
M_c
M_*
$\{\alpha^{\Pi(N_p|p\in[1,n]\wedge p\neq j)}|\ j\in[1,n]\wedge j\neq\ d\}$
Broadcast: round 1 $(N_c \leftarrow N_c\widehat{N_c})$

Fig. 6. Member exclusion.

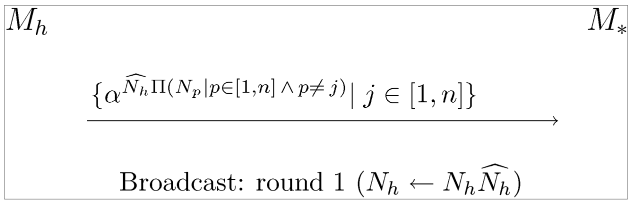
M_h
M_*
$\{\alpha^{\widehat{N_h}\Pi(N_p|p\in[1,n]\wedge p\neq j)}|\ j\in[1,n]\}$
Broadcast: round 1 $(N_h \leftarrow N_h\widehat{N_h})$

Fig. 7. Key refresh.

5.4 Member Exclusion

The member exclusion protocol is illustrated in Fig. 6. In it, M_c effectively "reruns" the last round of the IKA: as in member addition, it generates a new exponent $\widehat{N_c}$ and constructs a new broadcast message—but with $\widehat{N_c}N_c$ instead of N_c—using the most recently received broadcast message. (Note that the last broadcast message can be from an IKA or any AKA, depending on which was the latest to take place.) M_c then broadcasts the message to the rest of the group. The private exponents of the other group members remain unchanged.

Let M_d be the member to be excluded from the group. We assume, for the moment, that $d \neq c$. Since the following subkey:

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

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

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

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_c 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 Fig. 6. The only difference is that the group controller computes and broadcasts fewer subkeys; only those which correspond to the remaining members.

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 subkeys 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 available to crypt–analysis for a given group session key.

This 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 the disclosure of all the session keys to which the member has contributed with this share. Therefore, not only session keys, but also the individual key shares must be refreshed periodically.

This leads to the following key refresh protocol: The member M_h, which is the least recent to have refreshed its key share[6], generates a new share (exponent) $\widehat{N_h}$ and "reruns" the broadcast round, as shown in Fig. 7.

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-backed 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 intragroup communication.

5.7 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.
- $\mathcal{P} = \{M_{c+1}, \ldots, M_p\}$ denotes all *past* (excluded before) group members.
- $\mathcal{F} = \{M_{p+1}, \ldots, M_f\}$ 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

$$K = \alpha^{N_1\cdots N_c N_{c+1}\cdots N_p}.$$

To simplify our discussion, we collapse all members $\mathcal{P}$ and $\mathcal{F}$ into a single powerful adversary (Eve). (This is especially fitting, since $\mathcal{P}$ and $\mathcal{F}$ are not necessarily disjoint.) The result is that $\mathrm{Eve} = \mathcal{P} \cup \mathcal{F}$ and she possesses $\{N_j\ |M_j \in\ \mathrm{Eve}\}$. Further on and without loss of generality, we assume that M_c was group controller for both the operation leading to the current and to the following state.

We can thus rewrite the key as:

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

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

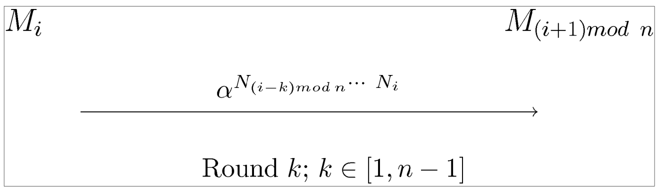
M_i
$M_{(i+1) mod\ n}$
$\alpha^{N_{(i-k) mod\, n} \cdots N_i}$
Round k; $k \in [1, n-1]$

Fig. 8. ING Protocol.

where B is a *constant* known to Eve, $\mathcal{E} = \{N_1, \dots, N_{c-1}, N_c\}$ are the secret exponents (contributions) of current group members. Note that the group controller's current exponent N_c is independent from both its past exponent $N_c' = N_c/\widehat{N_c}'$ and its future exponent $N_c'' = N_c * \widehat{N_c}''$ as the blinding factors $\widehat{N_c}'$ and $\widehat{N_c}''$ were both chosen randomly.

In Eve's view, the only expressions containing N_c are in the last broadcast round of either member addition or member exclusion protocols:

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

We can further assume that Eve also knows all:

$$\{\alpha^{\Pi(\mathcal{S})} |\ S \subset \mathcal{E}\}.$$

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:

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

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.

6 Related Work

6.1 Contributory Key Agreement

The earliest attempt to provide contributory key agreement and to extend DH to groups is due to Ingemarsson et al. [3] The protocol in Fig. 8 (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.

We note that this protocol falls into the class of *natural* DH extensions as defined in [7]. It is, thus, suitable for use as an IKA protocol. However, because of its symmetry,[7] (no natural group leader) it is difficult to use it as a foundation for auxiliary key agreement protocols.

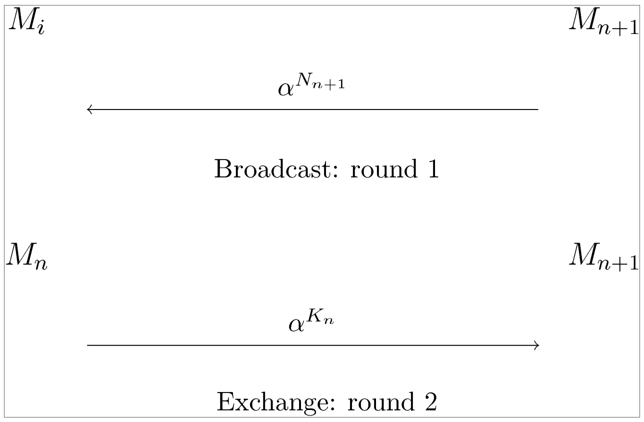

Fig. 9. Member Addition in STR.

Another DH extension geared towards teleconferencing was proposed by Steer et al. in [4]. 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:

$$K_n \ = \ \alpha^{N_n \alpha^{N_{n-1} \alpha^{\dots N_3 \alpha^{N1N2}}}}.$$

Interestingly, STR is well-suited for adding new members; see Fig. 9. As in IKA.1, it takes only two rounds to add a new member. Moreover, this protocol is computationally more efficient than IKA.1 member addition, since fewer exponentiations take place.[8] 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, recomputing a common key (when M_i leaves) is straightforward for all M_j, $j < i$. Meanwhile, all M_j, $j > i$ need to receive input from lower-numbered members.

One notable result is due to Burmester and Desmedt [5]. They construct a very efficient protocol (BD) which executes in only three rounds:

1. Each M_i generates its random exponent N_i and broadcasts $z_i = \alpha^{N_i}$.
2. Each M_i computes and broadcasts $X_i = (z_{i+1}/z_{i-1})^{N_i}$.

7. It is also not very efficient.
8. Note that, for a reasonable degree of security, STR requires a bijective mapping f from $\mathbb{Z}p^*$ to $\mathbb{Z}q^*$. However, $f(x) := xq$, as implicitly defined by STR, is not bijective. While there is an efficient mapping for safe primes [26] it is not clear if such efficient mappings exist also for other prime order groups such as the ones proposed in Section 3.2. Hence, the exponentiations in the CLIQUES protocols would be considerably faster than in a secure version of STR.

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 of each M_i to receive $n-1$ messages in a single round, and
3. the ability of the system to handle n simultaneous broadcasts.

While the BD (IKA) protocol is efficient and secure, we claim that it is not well-suited for dynamic groups. While addition looks trivial at first sight, closer inspection reveals that all group members have to refresh their share to prevent leaking too much information or serve as a exponentiation oracles. This means that, in fact, AKA operation gets as expensive in terms of communication and computation as the BD IKA. So the cost savings of BD IKA when compared to IKA.1 and IKA.2 are very quickly amortized and exceeded by the costs of their much less efficient (but much more frequent) AKA operations, e.g., BD IKA/AKA cost three rounds with n simultaneous broadcasts each, compared to two rounds and a single broadcast for CLIQUES AKA operations. Note that in practice, DPGs tend to start with only a very small number of initial members (if not even a single one) and grow mostly through AKA operations. Therefore, IKA operation is far less relevant than AKA operations.

In the most recent work, Becker and Wille [9] systematically analyze the communication complexity of contributory group key agreement protocols. They prove lower bounds for the number of messages, exchanges, simple and synchronous rounds and, e.g., confirmed that IKA.1 is optimal in respect to the number of messages and exchanges. Additionally, they also describe a novel protocol, 2^d-octopus, which reaches the lower bound for simple rounds ($d = \lceil \log_2 n \rceil$). Their main idea is to arrange the parties on a d-dimensional hypercube, i.e., each party is connect to d other parties. The protocol proceeds through d rounds, $1 \ldots d$. In the jth round, each player performs a two-party DH with its peer on the ith dimension, using the key of the $j-1$-th round as its secret exponent. The exponents of the 0th rounds are chosen at random by each party. For illustrative purposes, we show the resulting key for a group of eight parties:

$$K_8 = \alpha^{(\alpha^{(\alpha^{(N_1N_2)}\alpha^{(N_3N_4)})}\alpha^{(\alpha^{(N_5N_6)}\alpha^{(N_7N_8)})})}.$$

While adding new members and in particular groups is easy with 2^d-octopus, it fails completely in terms of member exclusion. Splitting the group on the dth dimension into two halves seems the only efficient exclusion procedure.

9. All indexes are modulo n.

6.2 Key Transport

The focus on our work was on contributory key agreement, not key transport. As discussed in Section 2, contributory key agreement has a number of advantages over (centralized) key transport. However, there is one main drawback with contributory schemes. Due to the contributory nature and perfect key independence, the protocols inevitably require exponentiations linear in the number of participants for AKA operations; of course, this doesn't scale well to very large groups. This is not a fundamental problem for DPGs as they tend to be reasonably small (< 100). However, in situations where the security, fault tolerance and flexibility requirements are less stringent and scalability and computation efficiency is the main issue, key distribution protocols might be more favorable.

Early key transport proposals [10], [27] were all based on a fixed group controller and did not address scalability or dynamics in group membership to a large extent. Subsequent work [28], [29] addressed scalability by splitting up the group into a hierarchy of subgroups controlled by subgroup controllers. These protocols improve overall efficiency, but their support for the dynamics of group is either limited or has costly side effects (e.g., Iolus [29] requires intermediary subgroup controllers to relay all messages and perform key translation).

Tree-based group rekeying systems, independently proposed by Wallner et al. [12] and Wong et al. [11], achieve all AKA operations in two rounds and bring the communication and storage costs down to $O(\log(n))$. Optimized variants [30], [31] reduce the communication overhead by half and their security can be proven using standard cryptographic assumptions. Due to their communication and computation efficiency, these protocols scale very well to large groups. Their main drawback is their reliance on a fixed group controller. Caronni et al. [32] overcome this by distributing the role of group controller over all members. Unfortunately, their protocols are vulnerable to collusions by excluded members. Another approach to increase safety of the tree-based group rekeying schemes is described in Rodeh et al. [33].

6.3 Other

We can find further related work in the context of distributed and fault-tolerant computing [13], [34]. Protocol suites and toolkits, such as Rampart [35], [36], aim at achieving high fault tolerance, even in the presence of malicious (i.e., byzantine) faults inside a group. This level of fault tolerance and the underlying model of virtual synchronous process groups might be required for securely and reliably replicating services [37] of great importance. However, these protocols are very expensive, as they rely on reliable and atomic multicasts secure against byzantine faults, see [38], [39] for some protocols.

7 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 agreement in the context of dynamic peer groups. Such groups are relatively small and nonhierarchical. In large groups, it is unclear that key agreement is appropriate since collecting contributions from all members can become very costly. Instead, key transport mechanisms can be used. This subject (key transport in large and dynamic groups) is an active research area; for example, [30], [31], [32].

Our emphasis has been on *bare* key agreement resistance 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 [2].

ACKNOWLEDGMENTS

The authors thank N. Asokan, G. Ateniese, V. Shoup, U. Wille, and the anonymous reviewers for comments on the drafts of this paper. This research was supported in part by the US Defense Advanced Research Project Agency, Information Technology Office (DARPA-ITO), under contract DABT63-97-C-0031.

REFERENCES

[1] W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Trans. Information Theory,* vol. 22, no. 6, pp. 644–654, Nov. 1976.
[2] G, Ateniese, M. Steiner, and G. Tsudik, "New Multiparty Authentication Services and Key Agreement Protocols," *IEEE J. Selected Areas in Comm.,* vol. 18, no. 4, Apr. 2000.
[3] I. Ingemarsson, D. Tang, and C. Wong, "A Conference Key Distribution System," *IEEE Trans. Information Theory,* vol. 28, no. 5, pp. 714–720, Sept. 1982.
[4] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener, "A Secure Audio Teleconference System," *Proc. Advances in Cryptology—CRYPTO '88,* pp. 520–528, 1988.
[5] M. Burmester and Y. Desmedt, "A Secure and Efficient Conference Key Distribution System," *Proc. Advances in Cryptology—EUROCRYPT '94,* 1995.
[6] M.K. Just, "Methods of Multiparty Cryptographic Key Establishment," MS thesis, Computer Science Dept., Carleton Univ., Ottawa, Ontario, Aug. 1994.
[7] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-Hellman Key Distribution Extended to Groups," *Third ACM Conf. Computer and Comm. Security,* pp. 31–37, Mar. 1996.
[8] M. Just and S. Vaudenay, "Authenticated Multi-party Key Agreement," *Proc. Advances in Cryptology—EUROCRYPT '96,* 1996.
[9] K. Becker and U. Wille, "Communication Complexity of Group Key Distribution," *Proc. Fifth ACM Conf. Computer and Comm. Security,* pp. 1–6, 1998.
[10] H. Harney and C. Muckenhirn, "Group Key Management Protocol (GKMP) Architecture," Internet Request for Comment RFC 2094, Internet Engineering Task Force, July 1997.
[11] C.K. Wong, M.G. Gouda, and S.S. Lam, "Secure Group Communications Using Key Graphs," *Proc. ACM SIGCOMM '98 Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm.,* pp. 68–79, 1998.
[12] D.M. Wallner, E.G. Harder, and R.C. Agee, "Key Management for Multicast: Issues and Architecture," Internet-Draft, Draft-Wallner-Key-Arch-00.txt, June 1997.
[13] K. Birman, *Building Secure and Reliable Network Applications.* Manning Publications, 1996.
[14] Y. Amir, G. Ateniese, D. Hasse, Y. Kim, C. Nita-Rotaru, T. Schlossnagle, J. Schultz, J. Stanton, and G. Tsudik, "Secure Group Communication in Asynchronous Networks with Failures: Integration and Experiments," *Proc. 20th IEEE Int'l Conf. Distributed Computing Systems (ICDCS),* Apr. 2000.
[15] Y. Amir and J. Stanton, "The Spread Wide Area Group Communication System," Technical Report CNDS 98-4, The Center for Networking and Distributed Systems, John Hopkins Univ., 1998.
[16] C.G. Günther, "An Identity-Based Key-Exchange Protocol," *Advances in Cryptology—EUROCRYPT '89,* pp. 29–37, Apr. 1990.
[17] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography.* CRC Press, 1997.
[18] Y. Yacobi and Z. Shmuely, "On Key Distribution Systems," *Advances in Cryptology—CRYPTO '89,* pp. 344–355, Aug. 1990.
[19] M. Burmester, "On the Risk of Opening Distributed Keys," *Proc. Advances in Cryptology—CRYPTO '94,* pp. 308–317, 1994.
[20] M. Naor and O. Reingold, "Number-Theoretic Constructions of Efficient Pseudo-Random Functions," *Proc. 38th Symp. Foundations of Computer Science (FOCS),* pp. 458–467, 1997.
[21] E. Biham, D. Boneh, and O. Reingold, "Breaking Generalized Diffie-Hellman Modulo: A Composite Is No Easier than Factoring," *Information Processing Letters,* vol. 70, pp. 83–87, 1999.
[22] C.P. Schnorr, "Efficient Signature Generation by Smart Cards," *J. Cryptology,* vol. 4, no. 3, pp. 161–174, 1991.
[23] U.S. National Institute of Standards and Technology (NIST}, "The Digital Signature Standard," FIPS PUB 186, May 1994.
[24] S. Brands, "An Efficient Off-Line Electronic Cash System Based on the Representation Problem," Technical Report CS-R9323, Centrum voor Wiskunde en Informatica, Mar. 1993.
[25] D.R. Stinson, *Cryptography: Theory and Practice.* Boca Raton, Fla.: CRC Press, 1995.
[26] D. Chaum, "Zero-Knowledge Undeniable Signatures," *Advances in Cryptology—EUROCRYPT '90,* pp. 458–464, May 1991.
[27] L. Gong, "Enclaves: Enabling Secure Collaboration over the Internet," *IEEE J. Selected Areas in Comm.,* pp. 567–575, 1997.
[28] A. Ballardie, "Scalable Multicast Key Distribution," Internet Request for Comment RFC 1949, Internet Engineering Task Force, May 1996.
[29] S. Mittra, "Iolus: A Framework for Scalable Secure Multicasting," *Proc. ACM SIGCOMM '97,* Sept. 1997.
[30] D.A. McGrew and A.T. Sherman, "Key Establishment in Large Dynamic Groups Using One-Way Function Trees," manuscript, May 1998.
[31] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, "Multicast Security: A Taxonomy and Some Efficient Constructions," *Proc. INFOCOMM '99,* Mar. 1999.
[32] G. Caronni, M. Waldvogel, D. Sun, N. Weiler, and B. Plattner, "The VersaKey Framework: Versatile Group Key Management," *IEEE J. Selected Areas in Comm.,* vol. 17, no. 9, Sept. 1999.
[33] O. Rodeh, K. Birman, and D. Dolev, "Optimized Rekey for Group Communication Systems," *Proc. Symp. Network and Distributed Systems Security (NDSS '00),* pp. 37–48, Feb. 2000.
[34] M. Reiter, K. Birman, and R. van Renesse, "A Security Architecture for Fault-Tolerant Systems," *ACM Trans. Computer Systems,* vol. 12, no. 4, pp. 340–371, Nov. 1994.
[35] M.K. Reiter, "Distributing Trust with the Rampart Toolkit," *Comm. ACM,* vol. 39, no. 4, pp. 71–74, Apr. 1996.
[36] M.K. Reiter, "A Secure Group Membership Protocol," *Proc. IEEE Symp. Research in Security and Privacy,* May 1994.
[37] M.K. Reiter and K.P. Birman, "How to Securely Replicate Services," *ACM Trans. Programming Languages and Systems,* vol. 16, no. 3, pp. 986–1,009, May 1994.
[38] D. Malkhi, M. Merrit, and O. Rodeh, "Secure Reliable Multicast Protocols in a WAN," *Int'l Conf. Distributed Computing Systems (ICDCS '97),* pp. 87–94, 1997.
[39] D. Malkhi and M. Reiter, "A High-Throughput Secure Reliable Multicast Protocol," *J. Computer Security,* vol. 5, pp. 113–127, 1997.

Michael Steiner received a Diploma in computer science from the Swiss Federal Institute of Technology (ETH) in 1992 and is working toward the PhD degree in computer science from the Universität des Saarlandes, Saarbrücken. He is a research scientist at the Department of Computer Science, Universität des Saarlandes, Saarbrücken, and in the network security research group at the IBM Zürich Research Laboratory. His interests include secure and reliable systems, as well as cryptography.

Gene Tsudik received the PhD degree in computer science from the University of Southern California in 1991. Since January 2000, he is an associate professor in the Department of Information and Computer Science at the University of California (USC), Irvine. Between 1996 and 2000, he was a project leader at USC/ISI, and a research associate professor in the Computer Science Department at USC. His research interests include network security, secure e-commerce, applied cryptography, and routing in wireless networks. He is a member of the IEEE Computer Society.

Michael Waidner received the diploma and doctorate degrees in computer science from the University of Karlsruhe, Germany. He is the manager of the network security research group at the IBM Zürich Research Laboratory. His research interests include cryptography, security, and all aspects of dependability in distributed systems. He has coauthored numerous publications in these fields, and is a member of the IEEE Computer Society.