Exploring Robustness in Group Key Agreement *

Yair Amir † Yongdae Kim ‡ Cristina Nita-Rotaru † John Schultz † Jonathan Stanton †
Gene Tsudik §

Abstract

Secure group communication is crucial for building distributed applications that work in dynamic environments and communicate over unsecured networks (e.g. the Internet). Key agreement is a critical part of providing security services for group communication systems. Most of the current contributory key agreement protocols are not designed to tolerate failures and membership changes during execution. In particular, nested or cascaded group membership events (such as partitions) are not accommodated.

In this paper we present the first robust contributory key agreement protocols resilient to any sequence of events while preserving the group communication membership and ordering guarantees.

1 Introduction

The explosive growth of the Internet has increased both the number and the popularity of applications that require a reliable group communication infrastructure, such as voice- and video-conferencing, white-boards, distributed simulations, and replicated servers of all types.

Secure group communication is crucial for building distributed applications that work in dynamic network environments and communicate over insecure networks such as the global Internet. Key management is the base for providing common security services (data secrecy, authentication and integrity) for group communication. There are several approaches to group key management.

One approach relies on a single, centralized entity, to generate keys and distribute them to the group. In this case, a so-called key server maintains long-term shared keys with each group member in order to enable secure two-party communication for the actual key distribution. A specific form of this solution uses a fixed trusted third party (TTP) as the key server. This approach has two problems: 1) the TTP must be constantly available and 2) a TTP must exist in every possible subset of a group in order to support continued operation in the event of network partitions. The first problem can be addressed with fault-tolerance and replication techniques. The second, however, is impossible to solve in a scalable and efficient manner. We note, however, that centralized approaches work well in a one-to-many multicast scenario since a TTP (or a set thereof) placed at, or very near, the source of communication can support continued operation within an arbitrary partition as long as it includes the source. (Typically, one-to-many settings only aim to offer continued operation within a single partition that includes the source; whereas, many-to-many environments must offer the same in an arbitrary number of partitions.)

Another key management approach involves dynamically selecting – in some deterministic manner – a group member charged with the task of generating keys and distributing them to other group members. This approach is robust and more amenable to many-to-many type of group communication since any partition can continue operation by electing a temporary key server. The drawback here is that, as in the TTP case, a key server must establish long-term pairwise secure channels with all current group members in order to distribute group keys. Consequently, each time a new key server comes into play, significant costs must be incurred to set up these channels. Another disadvantage, again as in the TTP case, is the reliance on a single entity to generate good (i.e., cryptographically strong, random) keys.

In contrast to the above, contributory key management asks each group member to contribute an equal share to the common group key (computed as a function of all members' contributions). This approach avoids the problems

*This work was supported in part by a grant from the National Security Agency under the LUCITE program and by grant F30602-00-2-0526 from The Defense Advanced Research Projects Agency.

†Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218, USA. Email: {yairamir, crisn, jschultz, jonathan}@cs.jhu.edu

‡Computer Networks Division, USC Information Sciences Institute, Marina Del Ray, CA 90292-6695, USA. Email: yongdaek@isi.edu

§Information and Computer Science Department, University of California, Irvine Irvine, CA 92697-3425, USA. Email: gts@ics.uci.edu

with the single points of trust and failure. Moreover, some contributory methods do not require the establishment of pairwise secret channels among group members. However, current contributory key agreement[1] protocols are not designed to tolerate failures and group membership changes during execution. In particular, nested (or cascaded) failures, partitions and other group events are not accommodated. This is not surprising since most multi-round cryptographic protocols do not offer built-in robustness with the notable exception of protocols for fair exchange [1].

The main goal of this paper is to demonstrate how provably secure, multi-round group key agreement protocols can be combined with reliable group communication services to obtain provably **fault-tolerant** group key agreement solutions. More precisely, we present two robust contributory key agreement protocols which are resilient to any sequence (even cascaded) of events while preserving group communications membership and ordering guarantees. Both protocols are based on Cliques GDH contributory key agreement that generalizes on the two-party Diffie-Hellman [2] key exchange. Our first protocol utilizes membership information provided by the group communication system in order to appropriately re-start Cliques GDH key agreement in an agreed-upon manner every time the group changes. The second protocol optimizes the performance of common cases at the cost of a more sophisticated protocol state machine.

The rest of the paper is organized as follows. The remainder of this section focuses on our motivation in pursuing this work and overviews related work. We then present Secure Spread, a secure group communication system which utilizes our key agreement protocols. The two subsequent sections present two robust key agreement protocols. Finally, we summarize our work and discuss some future directions.

1.1 Motivation

As mentioned earlier, a prominent challenge encountered in securing group communication is in developing robust, reliable and fault-tolerant group key management mechanisms that perform well in practice. While the motivation for security services (key management, in particular) in a tightly-coupled group communication setting is fairly intuitive, the need for reliable group communication services by the group key management is less obvious. We claim that reliable and sequenced message delivery is important (and even crucial) for cryptographic group protocols. Asynchronous network behavior must be handled by the underlying group communication layer, which prompts the need for a highly reliable group communication service.

[1] We use the term "agreement," as opposed to "distribution", to emphasize the contributory nature of the key management.

This dependence is both natural and mutual. It is natural since secure dynamic peer groups always require certain communication guarantees. (Best-effort datagram service is not usually a viable option, whereas, it may suffice for one-to-many type groups encountered in Internet multicast settings.) It is mutual since reliable group communication systems are of limited utility in open networks without strong security services and guarantees. Thus, we have interdependence among reliable group communication and group key management protocols.

Cryptographic protocol designers are primarily concerned with security and typically assume that protocol robustness is handled by the particular application or by the underlying communication layer. This is reasonable in two-party protocols where communication failures are relatively easy to handle and recover from. The picture changes dramatically in group protocols where the behavior model is richer.

Multi-round group key management protocols cannot be expected to run to completion without being possibly interrupted by various group membership events: joins, leaves, disconnects, partitions, merges or any combination thereof.

Our previous work [3] focused on the performance evaluation in the scenario with no network faults or cascaded events and provided a good insight of the overall cost of high security in a group communication environment. The present work goes into the details of a complete solution that handles every possible combination of group membership events. The contribution of this paper, therefore, is the design, and the proof of correctness of, a robust contributory key agreement algorithm.

1.2 Related Work

In this section we consider related work in two areas: group key management and reliable group communication.

1.2.1 Group Key Management

Cryptographic techniques for securing all types of multicast- or group-based protocols require all parties to share a common key. This requires a Group Key Management (GKM) protocol to provide methods for generating new group keys and updating existing keys. GKM protocols generally fall into two classes:

- Protocols designed for large-scale (e.g., IP Multicast) applications with a one-to-many communication paradigm and relatively weak security requirements.
- Protocols designed to support tightly-coupled dynamic peer groups with modest scalability requirements, a many-to-many communication paradigm and strong security requirements.

A number of GKM protocols supporting abstract peer groups have been developed in the last decade [4], [5], [6], [7], [8], [9]. All, except [9], extend the well-known Diffie-Hellman key exchange [2] method to group of n parties. These protocols vary in degrees of protection from hostile attacks and in their performance characteristics. (For an in-depth comparison, see [8].) In this paper, we make use of the CLIQUES toolkit which implements – among other methods – a suite of protocols, called generic Group Diffie-Hellman (GDH). GDH offers contributory authenticated group key agreement and handles dynamic membership changes [7, 8]. The entire protocol suite has been proven secure with respect to both passive and active attacks.

1.2.2 Reliable Group Communication

Reliable group communication in LAN environments have a well-developed history beginning with ISIS [10], and more recent systems such as Transis [11], Horus [12], Totem [13], and RMP [14]. These systems explored several different models of Group Communication such as Virtual Synchrony [15] and Extended Virtual Synchrony [16]. More recent work in this area focuses on scaling group membership to wide-area networks [17], [18].

Research in securing group communication is fairly new. The only actual implementations of group communication systems that focus on security (in addition to ours), are SecureRing [19] project at UCSB, and the Horus/Ensemble work at Cornell [20]. The SecureRing system protects a low-level ring protocol by using cryptographic techniques to authenticate each transmission of the token and each data message received. The Ensemble security work is the state-of-the-art in secure reliable group communication and addresses problems as group keys and re-keying. It also allows application-dependent trust models and optimizes certain aspects of group key generation and distribution protocols. In comparison with our approach, Ensemble uses a different group key structure that is not contributory and provides a different set of security guarantees.

Recent research on Bimodal-Multicast, Gossip-based protocols [21] and the Spinglass system has largely focused on increasing the scalability and stability of reliable group communication services in more hostile environments such as wide-area and lossy networks by providing probabilistic guarantees about delivery, reliability, and membership.

2 A Secure Group Communication Environment

The work discussed in this paper has involved integrating the Spread wide-area group communication system with the group key agreement protocols in the Cliques GDH protocol suite. In this section we overview both the Spread and Cliques toolkits.

2.1 Spread Toolkit

Spread [22], [23] is a group communication system for wide and local area networks. It provides all the services of traditional group communication systems, including: unreliable/reliable delivery, FIFO, causal, total ordering, and membership services with strong semantics.

Spread creates an overlay network that can impose an arbitrary network configuration (such as point-to-multi-point, tree, ring, tree-with-subgroups or any combination thereof) to adapt the system to different network environments. The Spread architecture allows multiple protocols to be used on links both between and within sites. The Spread toolkit is very useful for applications that need traditional group communication services (such as causal and total ordering, membership and delivery guarantees) but also need to operate over wide-area networks.

The system consists of a long-running daemon and a library linked with the application.

Spread scales well with the number of groups used by the application without imposing any overhead on network routers. Group naming and addressing is not a shared resource (as in IP multicast addressing) but rather a large space of strings which is unique to a collaboration session.

The toolkit can support a large number of different collaboration sessions, each of which spans the Internet but has only a moderate number of participants. This is achieved by using unicast messages over the wide-area network, routing them between Spread nodes on the overlay network.

The Spread system provides two different semantics: Extended Virtual Synchrony [16, 24] and View Synchrony [25]. In this paper, and for our implementation we only use the View Synchrony semantics of Spread.

The Spread toolkit is available publicly and is being used by several organizations for both research and practical projects. The toolkit supports cross-platform applications and has been ported to several Unix platforms as well as Windows and Java environments.

2.2 Cliques Toolkit

Cliques [8, 7, 26] is a cryptographic toolkit providing key management services for dynamic peer groups. Cliques includes several protocol suites:

- GDH: based on group extensions of the 2-party Diffie-Hellman key exchange [7, 8]; provides fully contributory authenticated key agreement. GDH is fairly computation-intensive requiring $O(n)$ cryptographic operations upon each key change. It is, however, bandwidth-efficient.

- CKD: centralized key distribution with the key server dynamically chosen from among the group members. A key server uses pairwise Diffie-Hellman key exchange to distribute keys. CKD is comparable to GDH in terms of both computation and bandwidth costs.
- TGDH: tree-based group Diffie-Hellman [26]; TGDH is more efficient than the above in terms of computation as most operations require $O(\log n)$ cryptographic operations. (The security of TGDH is slightly weaker and it lacks several other features not germane in this context.)
- BD: a protocol based on Burmester-Desmedt [5] variation of group Diffie-Hellman. BD is computation-efficient requiring constant number of exponentiations upon any key change. However, communication costs are significant with two rounds of n-to-n broadcasts.

All Cliques protocol suites offer key independence, perfect forward secrecy and resistance to known key attacks. (See [27, 8] for precise definitions of these properties.)

In this paper, we focus only on the GDH protocol suite within the Cliques toolkit. As mentioned earlier, our specific goal is to take a provably secure, multi-round group key agreement protocol (GDH) and, by combining it with the reliable group communication service (Spread), obtain a provably **fault-tolerant** group key agreement solution.

Cliques GDH API [28] is the implementation of the GDH protocol suite. It contains GDH cryptographic primitives while assuming the existence of a reliable communication platform for transporting protocol messages. GDH assigns a special role to the last member to join a group. This role, referred to as the group controller, floats as group membership changes. A group controller is charged with initiating key updates following membership changes.[2] The following operations trigger a key update: 1) join – add a single new member to the group (handled as a special case of merge); 2) merge – add multiple members to the group; 3) leave: one member voluntarily leaves the group (handled as a special case of partition); 4) partition: multiple members leave the group due to expulsion or a network event.

3 System Model

In this section we specify the failure and the group communication models used in this paper.

3.1 Failure Model

We consider a *distributed system*, a group of processes executing on one or more computers and coordinating actions by exchanging messages. The message exchange is achieved via asynchronous multicast and unicast messages. Messages can be lost.

The system is subject to process crashes and recoveries. A crash of any component of the process such as the key-agreement layer, the Cliques library, or the group communication system is considered a process crash. It is assumed that the crash of one of these components is detected by all the other components and is treated as a process crash.

Also, the system is prone to partitions which may result a network being split into disconnected subnetworks. When such a partition is fixed, the disconnected components merge into a larger connected component. While processes are in separate disconnected components they cannot exchange messages.

We assume that message corruption is masked by a lower layer. Byzantine failures are not considered.

Our intruder model takes into account only outside intruders, both passive and active. An outsider is anyone who is not a current group member. (Of course, any former and future member, is an outsider according to this definition.) We do not consider insider attacks since our threat model concentrates on the secrecy of group keys and the integrity of the group membership (i.e., the inability to spoof authenticated membership). Consequently, insider attacks are not relevant because a malicious insider can always reveal the group key and/or its own private key thus allowing for fraudulent membership authentication.

Passive outsider attacks involve eavesdropping with the aim of discovering the group key(s). This attack type has been proven to be computationally infeasible in [7]. Active outsider attacks involve injecting, deleting, delaying and modifying protocol messages. Some of these attacks aim to cause denial of service; we do not address these denial of service attacks. Attacks with the goal of impersonating a group member are prevented by the use of public key-based signatures. (All protocol messages are signed by the sender and verified by all receivers.)

3.2 Group Communication Model

A group communication system usually provides fundamental services such as membership as well as dissemination, reliability and ordering of messages. The membership service notifies the upper-level application with a list of group members each time the group changes. This notification-of-membership service is called a *view*.

Several different sets of membership properties have been defined in the literature. Each provides a different set of semantic guarantees to the application, and are usually called Virtual Synchrony semantics or some variant on the name. The many variations of virtual synchrony are all based on the property that processes moving together from one view to another deliver the same set of messages in the

[2] GDH API also allows a key refresh operation which may be initiated only by the current controller.

former membership view.

Some group communication systems have been built [12], [14], [18] that approximate the virtual synchrony model along with some related properties. However, each system does not provide the exact same set of properties, and to the best of our knowledge a canonical "Virtual Synchrony model" of an entire system has not been defined in the literature. A good survey describing many of the variations of different properties for virtual synchrony semantics can be found in [29].

Virtual synchrony strengthens the shared state of the system by delivering messages in the same membership as they were sent in. This enables the use of a shared key to encrypt data, since the receiver is guaranteed to have the same membership view as the sender and therefore the same key (ignoring for now some constraints on rekeying).

This work assumes that the group communication system supports virtual synchrony semantics as they are defined below. The description of the properties is largely based on the survey [29] and the description of the Extended Virtual Synchrony semantics [16].

Note that we define that some event occurred in view v at process p if the most recent view installed by process p was v.

1. *Self Inclusion*
   If process p installs a view v then p is a member of v.
2. *Local Monotonicity*
   If process p installs a view v after installing a view v' then the identifier id of v is greater than the identifier id' of v'.
3. *Sending View Delivery*
   A message is delivered in the view that it was sent in.
4. *Delivery Integrity*
   If process p delivers a message m in a view v, then there exists a process q that sent m in v causally before p delivered m.
5. *No Duplication*
   A message is not sent twice. A message is not delivered twice to the same process.
6. *Self Delivery:*
   If process p sends a message m, then p delivers m unless it crashes.
7. *Transitional Set*
   1) If two processes p and q install the same view, and q is included in p's transitional set for this view then p's previous view was identical to q's previous view.
   2) If two processes p and q install the same view, and q is included in p's transitional set for this view then p is included in q's transitional set for this view.
8. *Virtual Synchrony*
   Two processes that move together[3] through two consecutive views deliver the same set of messages in the former.
9. *Causal Delivery*
   If message m causally precedes message m', and both are sent in the same view, then any process q that delivers m' delivers m before m'.
10. *Agreed Delivery*
    1) Agreed delivery maintains causal delivery guarantees.
    2) If agreed messages m and m' are delivered at process p in this order, and m and m' are delivered by process q, then m' is delivered by q after it delivers m.
    3) If agreed messages m and m' are delivered by process p in view v in this order, and m' is delivered by process q in v before a transitional signal, then q delivers m. If messages m and m' are delivered by process p in view v in this order, and m' is delivered by process q in v after a transitional signal, then q delivers m if r, the sender of m, belongs to q's transitional set.
11. *Safe Delivery*
    1) Safe delivery maintains agreed delivery guarantees.
    2) If process p delivers a safe message m in view v before the transitional signal, then every process q of view v delivers m unless it crashes. If process p delivers a safe message m in view v after the transitional signal, then every process q that belongs to p's transitional set delivers m after the transitional signal unless it crashes.

[3] If process p installs a view v with process q in its transitional set and process q installs v as well, then p and q are said to move together.

4 A Basic Robust Algorithm

This section discusses the details of a basic robust key agreement algorithm. Throughout the remainder of the paper, we mean by the group communication system (GCS), a group communication system providing the virtual synchrony semantics. Our basic algorithm is based on the Cliques GDH IKA.2 protocol. Briefly, this protocol works as follows (see [7] for a complete description):

When an additive group view change happens (a join or a merge) the current group controller generates a new key token by refreshing its contribution to the group key and passes the token to one of the new members. When that new member receives this token, it adds its own contribution and passes the token to the next new member[4]. Eventually, the token reaches the last new member. This new member, who is slated to become the new group controller, broadcasts the token to the group without adding its contribution. Upon receiving the broadcast token, each group member (old and new) factors out its contribution

[4] The new member list and its ordering is decided by the underlying group communication system; Spread in our case. The actual order is irrelevant to Cliques.

and unicasts the result (called a factor-out token) to the new controller. The new controller collects all the factor-out tokens, adds its own contribution to each of them, builds a list of partial keys and broadcasts the list to the group. Every member can then obtain the group key by factoring in its contribution. (This is actually performed with modular exponentiation.)

When some members leave the group, the group controller (who, at all times, is the most recent group member) removes their corresponding partial keys from the list of partial keys, refreshes each partial key in the list and broadcasts the list to the group. Each remaining member can then compute the shared key.

The algorithm described above is secure and correct. Security is preserved independently of any sequence of membership events, while correctness holds only as long as no additional group view change takes place before the protocol terminates.

To elaborate on this claim, consider what happens if a subtractive (leave or partition) group membership event occurs while the above protocol is in progress, for example, while the group controller is waiting for individual unicasts from all group members. Since the Cliques protocol is unaware of the membership change (which is "visible" only to the group communication system), the group controller will not proceed until all factor-out tokens (including those from former members) are collected. Therefore, the system will block. Similar scenarios are also possible, e.g., if one of the new members crashes while adding its contribution to a group key. In this case, the token will never reach the new group controller and the protocol will, once again, simply block.

If the nested event is additive (join or merge), the protocol operates correctly. In other words, it runs to completion and the nested event is handled serially. (We note, however, that this is not optimal since, ideally, multiple additive events can be "chained" effectively reducing broadcasts and factor-out token implosions.)

As the above examples illustrate, the protocol does not function correctly in the face of cascaded subtractive membership events. This behavior is not acceptable for reliable group communication systems that aim to provide a high degree of robustness and fault-tolerance.

A natural and correct solution to this problem is as follows: every time a group view change occurs, the group deterministically chooses a member (say, the oldest) and runs the Cliques GDH protocol with the chosen member initializing it. Note that this approach costs twice in computation and $O(n)$ more in the number of messages for the common case with no cascading membership events. This will be rectified in the second protocol described in Section 5.

When the key-agreement protocol is integrated with a group communication system and virtual synchrony se-

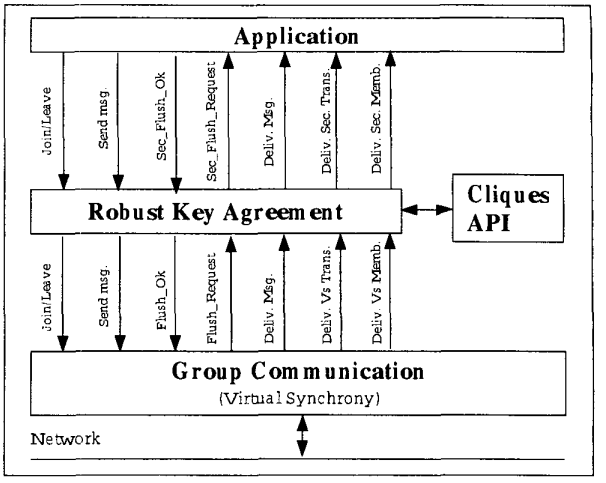


Figure 1. Secure group communication model

mantics must be preserved, extra care must be taken in order to provide all its guarantees to the application, including delivery of the correct views, transitional signal and transitional sets. We will elaborate on these issues later. Figure 1 presents the architecture of a secure group communication system. The system uses the following types of messages: Cliques messages (final_token_msg, partial_token_msg, key_list_msg, fact_out_msg), which are specific to the key agreement protocol (see [28]); membership notification messages (memb_msg); transitional signal messages (trans_signal_msg); application messages (data_msg); flush mechanism messages (flush_request_msg, flush_ok_msg).

To satisfy *Sending View Delivery* without discarding messages from live and connected members, a group communication system must block the sending of messages before the new membership is installed. In order to implement *Sending View Delivery* the group communication system sends a message (flush_request_msg) to the client asking for permission to install a new membership before actually creating the membership. The application responds with an acknowledgement message (flush_ok_msg) which follows all the messages sent by the application in the old view. After sending the acknowledgement message, the application is not allowed to send any messages until the new view is delivered. In Figure 1, the key-agreement algorithm (KAA) interacts with both the application and GCS. KAA implements the blocking mechanism transparently. When a flush_request_msg message is received from GCS, it is delivered to the user application. When the application acknowledgement message is received it is sent down to GCS.

A process starts executing the algorithm by invoking the *join* primitive of the key-agreement module which translates into a group communication join call. In any state of the algorithm a process can voluntarily leave by invok-

ing the *leave* primitive of the key-agreement module which translates it into a group communication leave call.

The specification of the algorithm is defined in terms of the following received events which are associated with a specific group:

- Partial_Token: a partial token message (partial_token_msg) was received by the KAA from the GCS.
- Final_Token: a final token message (final_token_msg) was received by the KAA from the GCS.
- Fact_Out: a factor out message (factor_out_msg) was received by the KAA from the GCS.
- Key_List: a key list message (key_list_msg) was received by the KAA from the GCS.
- User_Message: a data application message (data_msg) was received by the KAA from the application. The user can send messages using broadcast or unicast services.
- Data_Message: a data application message (data_msg) was received by the KAA from the GCS.
- Transitional_Signal: a transitional signal message (trans_signal_msg) was received by the KAA from the GCS.
- Membership: a membership message (memb_msg) was received by the KAA from the GCS.
- Flush_Request: a flush request message (flush_request_msg) was received by the KAA from the GCS.
- Secure_Flush_Request: a flush request message (flush_request_msg) was received by the application from the KAA.
- Secure_Flush_Ok: a flush acknowledge message (flush_ok_msg) was received by the KAA from the application.

Note that the same type of message can be associated with different events, depending on the source of the message. For example, both Flush_Request and Secure_Flush_Request events are associated with a flush_request_msg message, but in the first case the message is received by the KAA from the application, while in the second case the message is received by the application from the KAA.

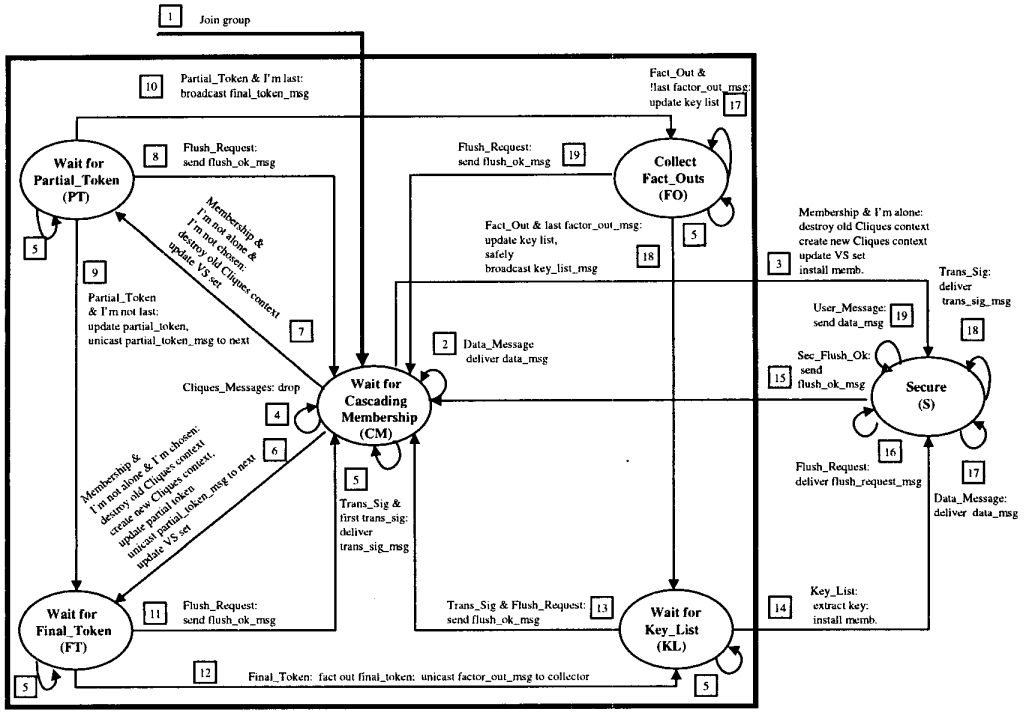
The algorithm consists of a state machine having the following states :

- SECURE (S): in this state the secure group is functional, all of the members have the group key and can communicate securely; the possible events are Data_Message, User_Message, Secure_Flush_Ok, Flush_Request, and Transitional_Signal; getting a Secure_Flush_Ok without receiving a Flush_Request is illegal; all other events are not possible.
- WAIT_FOR_PARTIAL_TOKEN (PT): in this state the process is waiting for a partial_token_msg message; the possible events are Partial_Token, Flush_Request and Transitional_Signal; User_Message and Secure_Flush_Ok are illegal; all other events are not possible.
- WAIT_FOR_FINAL_TOKEN (FT): in this state the process is waiting for a final_token_msg message; the possible events are Final_Token, Flush_Request and Transitional_Signal; User_Message and Secure_Flush_Ok are illegal; all other events are not possible.
- COLLECT_FACT_OUTS (FO): in this state the process is waiting for $N - 1$ fact_out_msg messages (where N is the size of the group); the only possible events are Fact_Out, Flush_Request, and Transitional_Signal; User_Message and Secure_Flush_Ok are illegal; all other events are not possible.
- WAIT_FOR_KEY_LIST (KL): in this state the process is waiting for a key_list_msg message; the possible events are Key_List, Flush_Request and Transitional_Signal; User_Message and Secure_Flush_Ok are illegal; all other events are not possible.
- WAIT_FOR_CASCADING_MEMBERSHIP (CM): in this state the process is waiting for are membership and transitional signal messages (memb_msg and trans_signal_msg); the possible events are Membership, Transitional_Signal, Data_Message (possible only the first time the process gets in this state), Partial_Token, Final_Token, Fact_Out and Key_List (they correspond to Cliques messages from a previous instance of the key agreement protocol when cascaded events happen); User_Message and Secure_Flush_Ok are illegal; all other events are not possible.

For an illegal event, an error message will be returned to the user. A process handles an event by performing two types of actions. The first type of action is a group communication operation and can be either a message delivery, or a message send such as unicast, broadcast, or send_flush_ok. The second type of action is a key agreement specific action. This translates into either computation or access to Cliques state information

As the state machine in Figure 2 shows, the Cliques GDH protocol remains intact, i.e., all of its protocol messages are sent and delivered in the same order as specified in [7]. Therefore, the basic robust key agreement algorithm provides the same security guarantees as the Cliques GDH protocol.

The complete proof that the algorithm presented above preserves virtual synchrony semantics as described in Section 3.2 as well as the detailed pseudocode were omitted because of space constraints, but they are included in the extended version of this paper [30].



Notes: VS_set is delivered as part of the membership
: All Cliques messages but key_list_msg are sent FIFO
: A process can leave the group in any state

Figure 2. Basic algorithm

5 An Optimized Robust Algorithm

In this section we show how the algorithm presented in the previous section can be optimized, such that the price paid for handling common, non-cascaded events is lower, while preserving the same set of group communication semantics and security guarantees.

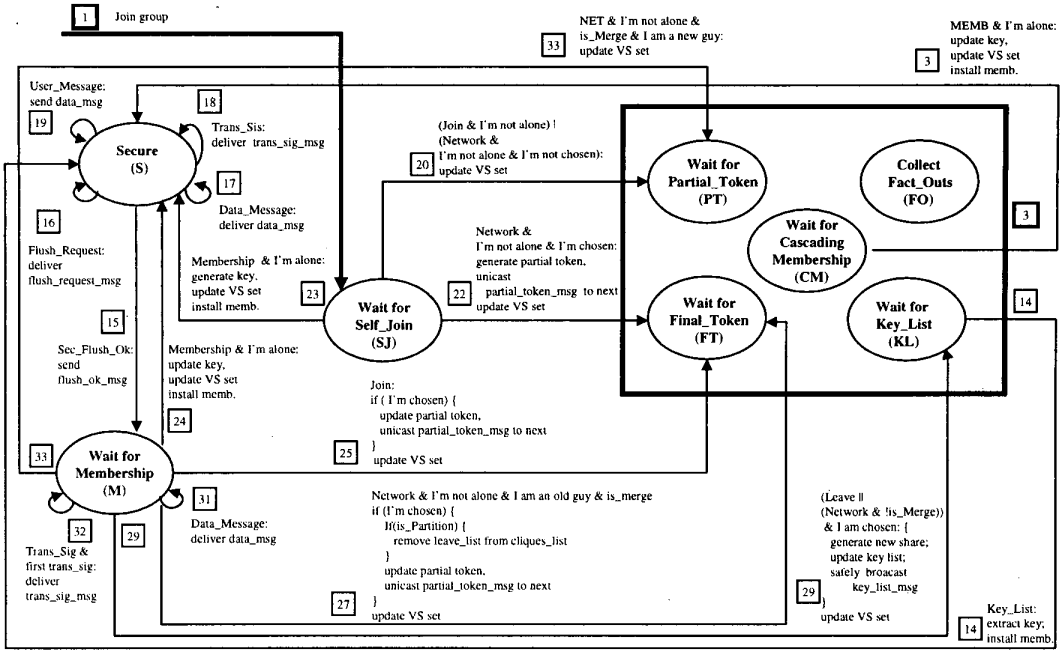
The basic algorithm presented in Section 4 is robust even when cascaded group events occur. Every time a membership notification is delivered from the group communication system, the algorithm ignores all the previous key agreement information and starts the merge protocol choosing a member from the new group to initialize it. Therefore, this algorithm pays more than necessary for computing a group key in a regular case, because it does not distinguish between a membership that finished without being interrupted and a cascaded membership.

The algorithm described above can be optimized so that it distinguishes between these two cases. Every time the group view changes, the algorithm detects the cause of the group change (join, leave, partition, merge or a combination of partition and merge) and invokes the Cliques GDH specific protocol. For example, in the case where a leave occurred, the leave protocol is invoked. Computing a new key in the case that a leave or partition occurred, requires only one broadcast. Thus, leave events can be handled immediately with a lower communication and computation cost than the basic algorithm required.

In the optimized key-agreement algorithm the process still starts executing the state machine by invoking the *Join* primitive. Also, at any moment, a process can voluntarily leave the algorithm by invoking the *Leave* primitive.

The optimized algorithm utilizes the following two states in addition to those of the basic algorithm:

- WAIT_FOR_SELF_JOIN (SJ): this is the initial state in which a process that joined a group enters the state machine; the process is waiting for the membership message that notifies the group about its joining. In case a network event happens between the join request and the membership notification delivery, the GCS will report the cause of the group change as being a network event and the transitional set will contain only the joining member. The only possible event is a Membership. User_Message and Secure_Flush_Ok events are illegal. All other events are not possible.
- WAIT_FOR_MEMBERSHIP (M): in this state the process is waiting for a membership notification. The possible events are: Transitional_Signal, Data_Message and Membership. The membership notification can be caused by voluntarily events such as join or leave, or network events. User_Message and Secure_Flush_Ok events are illegal. All other events are not possible.



Notes: VS_set is delivered as part of the membership
: All Cliques messages but key_list_msg are sent FIFO; key_list_msg is sent as a safe message.
: A process can leave the group in any state

Figure 3. Optimized algorithm

While a process starts the basic algorithm in the CM state, in the optimized algorithm a process starts the algorithm in state SJ. From the stable state (S state) if the group changed the process moves to the M state instead of moving to the CM state as in the basic algorithm. From here, depending on the cause of the group change, the merge or the leave Cliques GDH protocols are invoked. Also, a combined network event which includes both joins and leaves simultaneously can be handled by a modified version of the Cliques GDH merge protocol. If another group change happens before a key is computed, the process will move to the CM state and execute the basic algorithm.

A diagram showing the state machine of the algorithm is presented in Figure 3. The corresponding pseudo-code along with the proof that the optimized algorithm presented above preserves virtual synchrony semantics described in Section 3.2 is omitted for space reasons but can be found in an extended version of this paper [30].

5.1 Handling Bundled Events

Most group events are homogeneous in nature: leave (partition) or join (merge) of one or more members. However, a group communication system can decide to bundle several such events if they occur in close proximity, i.e., within a very short time interval. The main incentive for doing so is to reduce communication costs and limit the impact and overhead on the application.

Cliques provides two separate protocols that handle leave and merge events. Each of these protocols can trivially handle bundled events of the same type, i.e., the Cliques merge protocol can accommodate any combination of bundled merges and the Cliques leave protocol can do the same for any combination of leaves and partitions. A more interesting scenario occurs when a single membership event bundles merges/joins with leaves/partitions. One obvious way to handle this type of event is to first invoke Cliques leave to process all leaves/partitions and then invoke Cliques merge to process joins/merges. However, this is inefficient since the group would essentially perform two separate key agreement protocols where only one is truly needed. We can take advantage of the fact that both protocols in Cliques are initiated by the group controller. After processing all leaves/partitions, the group controller can suppress the usual broadcast of new partial keys and, instead, forward the resulting set to the first merging/joining member thereby initiating a merge protocol. This saves an extra round of broadcast and at least one cryptographic operation for each member.

6 Conclusions

This work provides two robust key agreement algorithms. We prove that by integrating them with a group

communication systems supporting Virtual Synchrony, the group communication membership and ordering guarantees are preserved.

We intend to explore and experiment with robustness and recovery techniques for a spectrum of other group key management mechanisms, such as the centralized approach and the Burmester-Desmedt protocol.

Finally, several necessary services for a secure group communication could lead to interesting future work. They include services such as group member certification, intra-group authentication, private communication within a group and private communication between members and non-members of the group.

References

[1] N. Asokan, V. Schoup, and M. Waidner, "Optimistic fair exchange of digital signatures," *IEEE Journal on Selected Area in Communications*, 2000.

[2] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 644–654, Nov. 1976.

[3] 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," in *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems*, (Taipei, Taiwan), pp. 330–343, April 2000.

[4] A. Fiat and M. Naor, "Broadcast encryption," *Advances in Cryptology - CRYPTO'93*, August 1993.

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

[6] M. Just and S. Vaudenay, "Authenticated multiparty key agreement," *Advances in Cryptology - EUROCRYPT'96*, May 1996.

[7] M. Steiner, G. Tsudik, and M. Waidner, "Key agreement in dynamic peer groups," *IEEE Transactions on Parallel and Distributed Systems*, August 2000.

[8] G. Ateniese, M. Steiner, and G. Tsudik, "New multi-party authentication services and key agreement protocols," *IEEE Journal of Selected Areas in Communication*, vol. 18, March 2000.

[9] R. Poovendran, S. Corson, and J. Baras, "A shared key generation procedure using fractional keys," *IEEE Milcom 98*, October 1998.

[10] K. P. Birman and R. V. Renesse, *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, March 1994.

[11] Y. Amir, D. Dolev, S. Kramer, and D. Malki, "Transis: A communication sub-system for high availability," *Digest of Papers, The 22nd International Symposium on FaultTolerant Computing Systems*, pp. 76–84, 1992.

[12] R. V. Renesse, K.Birman, and S. Maffeis, "Horus: A flexible group communication system," *Communications of the ACM*, vol. 39, pp. 76–83, April 1996.

[13] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. Agarwal, and P. Ciarfella, "The totem single-ring ordering and membership protocol," *ACM Transactions on Computer Systems*, vol. 13, pp. 311–342, November 1995.

[14] B. Whetten, T. Montgomery, and S. Kaplan, "A high performance totally ordered multicast protocol," in *Theory and Practice in Distributed Systems, International Workshop*, Lecture Notes in Computer Science, p. 938, September 1994.

[15] K. P. Birman and T. Joseph, "Exploiting virtual synchrony in distributed systems," in *11th Annual Symposium on Operating Systems Principles*, pp. 123–138, November 1987.

[16] L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal, "Extended virtual synchrony," in *Proceedings of the IEEE 14th International Conference on Distributed Computing Systems*, pp. 56–65, IEEE Computer Society Press, Los Alamitos, CA, June 1994.

[17] T. Anker, G. V. Chockler, D. Dolev, and I. Keidar, "Scalable group membership services for novel applications," in *Proceedings of the workshop on Networks in Distributed Computing*, 1998.

[18] I. Keidar, K. Marzullo, J. Sussman, and D. Dolev, "A client-server oriented algorithm for virtually synchronous group membership in wans," Tech. Rep. CS99-623, Univ. of California, San Diego Tech Report, June 1999.

[19] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, "The securering protocols for securing group communication," in *Proceedings of the IEEE 31st Hawaii International Conference on System Sciences*, vol. 3, (Kona, Hawaii), pp. 317–326, January 1998.

[20] O. Rodeh, K. Birman, M. Hayden, Z. Xiao, and D. Dolev, "Ensemble security," Tech. Rep. TR98-1703, Cornell University, Department of Computer Science, September 1998.

[21] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," Tech. Rep. TR99-1745, Department of Computer Science, Cornell University, May 1999.

[22] Y. Amir and J. Stanton, "The spread wide area group communication system," Tech. Rep. 98-4, Johns Hopkins University Department of Computer Science, 1998.

[23] Y. Amir, C. Danilov, and J. Stanton, "A low latency, loss tolerant architecture and protocol for wide area group communication," in *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 327–336, June 2000.

[24] Y. Amir, *Replication using Group Communication over a Partitioned Network*. PhD thesis, Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel, 1995.

[25] A. Fekete, N. Lynch, and A. Shvartsman, "Specifying and using a partionable group communication service," in *Proceedings of the 16th annual ACM Symposium on Principles of Distributed Computing*, (Santa Barbara, CA), pp. 53–62, August 1997.

[26] Y. Kim, A. Perring, and G. Tsudik, "Simple and fault-tolerant key agreement for dynamic collaborative groups," in *7th ACM Conference on Computer and Communications Security*, pp. 235–244, ACM Press, November 2000.

[27] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[28] G. Ateniese, O. Chevassut, D. Hasse, Y. Kim, and G. Tsudik, "Design of a group key agreement api," in *DARPA Information Security Conference and Exposition (DISCEX 2000)*, January 2000.

[29] R. Vitenberg, I. Keidar, G. V. Chockler, and D. Dolev, "Group communication specifications: A comprehensive study," Tech. Rep. CS0964, Computer Science Department, the Technion, Haifa, Israel; Tech. Rep. MIT-LCS-TR-790,Massachusetts Institute of Technology, Laboratory for Computer Science; Tech. Rep. CS99-31, Institute of Computer Science, The Hebrew University of Jerusalem., 1999.

[30] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Schultz, J. Stanton, and G. Tsudik, "Exploring robustness in group key agreement," Tech. Rep. CNDS 2000-4, Department of Computer Science, Johns Hopkins University, 2000. http://www.cnds.jhu.edu/publications/.