Secure Group Communication in Asynchronous Networks with Failures: Integration and Experiments

Yair Amir[*], Giuseppe Ateniese[†], Damian Hasse[‡], Yongdae Kim[‡], Cristina Nita-Rotaru[*], Theo Schlossnagle[*], John Schultz[*], Jonathan Stanton[*§], Gene Tsudik[‡]

August 22, 1999

Abstract

Increasing popularity and diversity of collaborative applications prompts the need for highly secure and reliable communication platforms for dynamic peer groups. Security mechanisms for such groups tend to be both expensive and complex and their integration with reliable group communication services presents a formidable challenge.

This paper discusses some important integration issues, reports on the implementation experience and provides experimental results. Our approach utilizes distributed group key management developed by the Cliques project. We enhance it to handle processor and network faults (under a fail-stop or crash-and-recover model) and asynchronous membership events (such as cascading joins, leaves, merges and network partitions). Our approach leverages the strong properties provided by the Spread group communication system, such as message ordering, clean failure semantics and a membership service. The result of this work is a secure group communications layer and an API which provide the application programmer both standard group services as well as flexible security services.

Keywords: secure group communication, reliable group communication, dynamic peer groups, group key management, security overhead.

1 Introduction

Fault-tolerant, scalable, and reliable communication services have become critical in modern computing. A major focus today is in taking traditional, centralized services such as file sharing, authentication, web, and mail services and distributing them across multiple systems and networks.

[*]Computer Science Department, Johns Hopkins University; {*yairamir,crisn,jesus,jschultz,jonathan*}*@cs.jhu.edu*
[†]IBM Research Laboratory, Rüeschlikon, Switzerland; *gat@zurich.ibm.com*
[‡]Computer Networks Division, USC Information Sciences Institute; {*hasse,yongdaek,gts*}*@isi.edu*
[§]**Contact author.**

These distributed applications and other inherently collaborative applications (such as conferencing, whiteboards, shared instrument control, as well as command-and-control systems) are very difficult to implement. One common and successful approach to developing these types of applications is to use a reliable group communication toolkit as a base messaging and fault-tolerant service.

Reliable group communication systems offer a set of low-level services that provide efficient messaging, membership, ordering and fault-detection services to peer groups. Typically, distributed and collaborative applications require fairly low latency message delivery of both small and large messages. They may involve many-to-many communication patterns. The number of membership events that these applications generate can vary from one every few minutes or hours to tens or hundreds of membership changes per second. Commonly, however, the number of joins or leaves is at most a few per second and network failures or recoveries causing merges of groups or partitions are at most a few an hour. Thus, we aim for the secure group communication system to support that level of performance in a practical setting..

The rest of this paper is organized as follows. The next two subsections outline, in general terms, security problems in peer groups and justify our focus on group key management. Section 2 discusses, in detail, our security goals and the various issues arising in group key management. Then, Sections 3 and 4 provide an overview of the Spread group communication toolkit and the Cliques group key management service, respectively. Next, the secure Spread architecture is described in Section 5. Section 6 illustrates and discusses some experimental results obtained with secure Spread and Section 7 summarizes the related work. Section 8 concludes the paper with the discussion of on-going and future work.

1.1 Security in Peer Groups

Just as many-to-many peer group communication tends to be much more complex than two-party communication, security in a multi-party setting is much harder to define, specify and achieve, than in a two-party case.

Communication channel specifics have relatively little impact on the ability of **two** parties to communicate securely. An unreliable communication channel may drop or corrupt data but, at worst, it can only prevent communication. Well-known cryptographic methods can be utilized to assure data privacy, data integrity, source authentication and other properties. Moreover, techniques exist for hiding the communication patterns, thus preventing hostile traffic analysis. As a result, the adversary's choice is reduced to a binary one: either allow communication or prevent it altogether.

In contrast, secure group communication is very much dependent on the composition of the group. A group, unlike a pair of end-points, mutates over time. If we collapse all pair-wise channels within a group into a single group channel, its state cannot be expressed as a binary value. Fluctuations in the group channel state cause and are caused by members joining and leaving the group. To achieve the highest level of security, every state fluctuation must be accompanied by a corresponding adjustment to group security parameters. Of these, the most apparent is the group shared keying material or group secret.

This leads to a classical case of a *chicken-and-egg* problem: security events must immediately

follow group state change events yet the latter are not themselves secure. However, we argue that certain group communication events are fundamentally impossible to secure. Notably, these include all kinds of fault-like events leading to group partitions and involuntary member disconnects. As a concrete example, consider the situation where network faults and individual node disconnects constantly perturb group membership. In this setting, trying to differentiate between two possible causes: 1) a clever adversary and 2) a truly faulty network, is impossible. Other events, such as members joining a group (whether as singletons or *en masse*) can, indeed, be made secure.

Finally, there is the all-important matter of trust. Trust, a very vague notion in a two-party case, becomes even vaguer in a group context. As group membership changes, trust among group members may change over time.

1.2 Our Security Focus

The purpose of the above discussion is to motivate the need for specialized group security mechanisms. Since routine security services such as bulk data privacy and data integrity are usually contingent upon sharing a common secret (group key), establishing and managing group keying material is the most fundamental group security mechanism.

We thus concentrate on group key management, its integration with a reliable group communication platform and its impact on the latter. We also consider the impact of data privacy and data integrity, however, these services are not much different in peer groups from those in a traditional two-party communication setting.

There are tradeoffs to be made when choosing what type of key management protocols a security system should use. These include number of messages sent per event, number of participants per event, amount of serial computation and overall computation done by the group per event, fault tolerance, amount of trust in members of the group and fairness of load distribution. We discuss these tradeoffs as we evaluate two group key management protocols: a distributed key management protocol suite based on the Cliques work, and a centralized protocol providing a roughly equivalent level of security.

In tackling the security issues in reliable group communication we do not (yet) consider certain components that are needed for comprehensive security.

- Group access control and, more generally, group policy
- Group and member certification

While we do not aim to underestimate their importance, the research in this area is just beginning [1]. We believe that well-designed group key management and data security protocols can be coupled with an *ad hoc* policy framework and be deployed rapidly. We further believe that the same group key management system can be coupled with a better policy framework when such exists.

2 Security Goals

Our main security goal is both natural and fairly standard: to achieve authentic and private communication within a group. Although this requirement can be expressed in any secure group setting–in a dynamic peer group context–it leads to a number of interesting corollaries. The second security goal is to provide authentic and private communication between a secure group (i.e., its members) and other entities (non-members). Finally, the third goal is to obtain strong authentication and non-repudiation of individual group members both within and outside a group.[1]

So-called *conventional* cryptography requires two (or more) parties to share a common secret in order to communicate securely. In contrast, public key cryptography [2] allows two parties who do not share a common secret to communicate securely. Owing to its computational cost (orders of magnitude greater than that of conventional encryption) public key encryption can be used to secure communication between two parties only when a very small amount of data is involved. In practice, public key encryption is used only as a means to distribute (or agree on) a common secret key which is subsequently used with conventional cryptography to provide authenticity and privacy of bulk data.

To obtain security in a peer group setting, we can construct an obvious extension to any two-party security mechanism (such as SSL or SSH) by establishing N^2 (where N is the group size) pair-wise secure channels among all group members. Each pair-wise channel would then be associated with a unique secret key known only to the two endpoints. In order to send private data to the group, the sender would have to encrypt it $(N - 1)$ times (once for every recipient) producing as many copies which would then be sent individually to each recipient or conglomerated into a single message and broadcasted to the entire group. It is easy to see that this approach involves horrendous overhead both in terms of computation and bandwidth consumed.[2]

2.1 Group Key Management

The above discussion essentially implies that–practically speaking–neither private nor authentic communication within a group can be achieved without some sort of a common group secret. Consequently, the starting point for security is a set of mechanisms for obtaining and maintaining such a secret. We refer to this collectively as the group key management problem.

Besides the fact that it forms the basis of all other security services, there are two other important reasons for our focus on group key management:

- The security of group key management itself is of paramount importance. Data privacy and authenticity (integrity) mechanisms are formed by defining a message format and selecting a basic underlying algorithm, i.e., a block cipher such as DES[4][3] for privacy and a keyed

[1] That is, a secure group can be viewed as a microcosm of sorts where authentication and non-repudiation of individuals is at the granularity of members, not permanent identities.

[2] We note that the only exception is its use in secure multi-destination electronic mail, such as PEM [3], where recipient "groups" are constructed on a per-message basis.

[3] Data Encryption Standard.

MAC[4] such as HMAC [5] for data integrity. Whereas, key management mechanisms involve, in addition, intricate *protocols*. An adversary aiming to attack a system is unlikely to waste efforts on attacking privacy and integrity mechanisms since key management presents a much more attractive target.

- The cost of group key management represents a "pure" form of security overhead. Compared with the cost of encryption (which can be done with almost no overhead if certain types of stream ciphers are used) or integrity (which can be obtained with certain Gbit-speed MACs), key management typically requires relatively heavy-weight arithmetic operations and additional communication among group members.

2.2 Centralized vs Distributed Key Management

The main issues in group key management center around **who** generates a group key as well as **how** and **when** it is generated. The **when** part is relatively simple, since, in the extreme, a new key must be generated following every group membership change. The **who** and **how** issues are more involved as they collectively determine the actual key management protocol(s).

If the group key is generated by a single party who then distributes it to all other group members we call the key management *centralized*; whereas, if all members participate in key generation we call the key management *distributed*.[5]

Centralized key management takes on two flavors: TTP-based or controller-based. The first is based on the notion of a Trusted Third Party (TTP) which is a fixed, highly secure entity (e.g., a Kerberos Authentication Server [6]) charged with user authentication as well as generation and distribution of keys. Since a TTP is fixed, this approach cannot tolerate TTP partitions or TTP failures and is thus of limited utility in a peer group. An alternative is to fix a certain group member (e.g., oldest or newest) as a group controller whose duty is to generate and distribute keys to the group. Failures or partitions of a group controller can be effectively dealt with by selecting an appropriate member within the remaining group. This approach is workable and we are in fact using it as a point of comparison in the experimental results (see Section 6 and Appendix.) However, it does not allow for the authentication of individual group members which is one of our security goals. Furthermore, a fixed controller presents an attractive attack target since it is the only member entrusted with the security of the entire group. Another drawback is the inability to authenticate certain membership changes.

Distributed key management involves all group members collectively generating (or agreeing upon) a group key. The Cliques protocol suite, described in detail in Section 4, falls into this category. A group secret is essentially a function of all group members' individual contributions. At the same time, a member's contribution is known only to that member; this property aids in the authentication of individual members (since a member can show that it knows a unique and secret share of a common group secret.) The often-cited drawbacks of distributed key management are the relative complexity and the computational overhead of the cryptographic protocols that implement it. On the other hand, as illustrated by experimental results in Section 6, the overhead is

[4]Message Authentication Code.

[5]For the sake of clarity we do not consider hybrid methods, i.e., a subset of members generating a group key.

actually comparable to that of centralized controller-based key management.

3 The Spread Group Communication Toolkit

The group security services discussed in this paper are built on top of the Spread wide-area group communication system [7].

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

Spread creates an overlay network that can impose any arbitrary network configuration including for example, point-to-multi-point, trees, rings, trees-with-subgroups and any combinations of them to adapt the system to different networking environments. The Spread architecture allows multiple protocols to be used on links between sites and within a site.

Spread is very useful for applications that need the traditional group communication services such as causal and total ordering, and membership and delivery guarantees, but also need to run over wide area networks.

In addition, other applications may find Spread useful because of some other technical properties:

- Scalability with the number of collaboration sessions. Spread can support large number of different collaboration sessions, each of which spans the Internet but has only a small number of participants. The reason is that Spread utilizes *unicast* messages on the wide area network, routing them between Spread nodes on the overlay network.

- Scalability with the number of groups. Spread can scale well with the number of groups used by the application without imposing any overhead on network routers. Group naming and addressing is no longer a shared resource (the IP address for multicast) but rather a large space of strings which is unique per collaboration session.

- Spread supports the Extended Virtual Synchrony model [8] and the View Synchrony model [22] which provide strong semantic guarantees about membership events and message delivery.

- Spread uses a daemon-client architecture. This architecture has many benefits, the most important for wide-area settings is the resultant ability to pay the minimum necessary price for different causes of group membership changes. Simple join and leave of processes translates into a single message. A daemon disconnection or connection does not pay the heavy cost involved in changing wide area routes. Only network partitions between different local area components of the network requires the heavy cost of full-fledged membership change. Luckily, there is a strong inverse relationship between the frequency of these events and their cost in a practical system. The process and daemon membership correspond to the more common model of ”Lightweight Groups” and ”Heavyweight Groups”[RG98].

The Spread toolkit is available publicly. An early version of the system is 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.[6]

3.1 Supported Group Communication Semantics

Spread supports the Extended Virtual Synchrony (EVS) model [8] [9] and the View Synchrony (VS) model [22]. Both EVS and VS guarantee that group members see the same set of messages between two sequential group membership events. They also both guarantee that the order of messages requested by the application (such as FIFO, Causal, or Total) is preserved. EVS provides a more general service, for VS semantics can be implemented on top of EVS semantics, but EVS can not be implemented on top of VS.

EVS provides two general benefits over VS: it has better performance and it allows open groups, where non-members of a group can send messages to the group. EVS guarantees that messages are delivered to all recipients in the same membership as the message was originally sent on the network.

The last EVS property above is the critical difference between VS and EVS. VS, in contrast, guarantees the stricter property that messages are delivered to all recipients in the same membership as the sending application thought it was a member of at the time it sent the message. Providing this property requires a round of application acknowledgement messages before installing a new membership. This need for application level acknowledgements requires that the groups be closed, only allowing members of the group to send messages to it.

This knowledge that a message is received in the membership the application believed it was sent in which is provided by VS semantics makes implementing a secure group system much easier because every message is encrypted with the same key as the receiver believes is current when the message is delivered to them.

The Spread system provides Extended Virtual Synchrony. A flush layer built atop spread (provided with the Spread system) provides the View Synchrony model.

4 Overview of Cliques and CLQ_API

Cliques [10, 11, 12] is a cryptographic protocol suite which provides authenticated contributory group key management and other security services. Cliques protocols guarantee key independence, key confirmation, perfect forward secrecy and resistance to known key attacks. (We refer to [13] for detailed definitions of these attacks.) In short, Cliques guarantees that group keys cannot be obtained by either active or passive attackers. Moreover, past group members (alone or in collusion) cannot obtain group keys subsequent to laving the group and, similarly, current group members cannot obtain group keys used before they joined the group. Cliques is based on group extensions of the well-known Diffie-Hellman [2] key exchange technique.

[6] More details on the Spread system can be found at *http://www.spread.org/* along with a white paper and programming documentation.

Cliques defines a special role for a group controller, the last member to join a group. This role floats as the group membership changes. A controller is charged with initiating key adjustments following membership changes. Other than that, it has no special security tasks or privileges.

CLQ_API [14] is a group key agreement API built on top of the Cliques protocol suite. Its main purpose is to implement the cryptographic primitives of Cliques. The underlying communication system is assumed to deal with the group communication and network events such as partitions, failures and other abnormalities. CLQ_API is small and concise containing only eight function calls.

Cliques guarantees two system invariants:

1. All group members always agree on the identity of the current group controller.
   At any point in time, the controller is the newest (most recently joined) group member.

2. A group secret key is always contributed to equally by each and every group member.
   At any point in time, a group secret is a function of all members' private shares. Only current group members have access to the group secret.

Cliques supports the following group key agreement operations:

- Join: a new member is added to the group.
- Merge: one or more members are added to the group.
- Leave: one or more members are removed from (or leave) the group.
- Key Refresh: generates a new group secret.

All of these result in the same outcome, i.e., the group secret is changed such that: all members obtain the same authenticated key.

The rest of this section briefly explains how the CLQ_API performs the above operations. (For details, the reader is referred to [14].) The group secret for n members is of the form $g^{N_1 N_2 \ldots N_n} \pmod p$, where N_i is the member M_i's private share, p is large prime number and g is the generator in Z_p^*. Both p and g are agreed-upon systemwide parameters just as in the plain two-party Diffie-Hellman setting [2].

4.1 Join

1. The group controller generates a new private share and computes partial group secrets, one for each existing group member. Then, it hands over the partial group secrets to the joining member, who is slated to become the new controller.

2. After receiving the information, the new member adds its own share to the partial group secrets of all users and broadcasts the result to the entire group.

3. Upon reception of the broadcasted message, every user computes the group secret.

4.2 Merge

The MERGE operation in Cliques requires the list of merging members to be available to the current group controller. At the end of MERGE, the last member in this list becomes the new group controller.

1. The current controller generates a new private share and computes a new partial group secret. This value is then sent openly to the first new (merging) member.
2. Each new member, in turn, adds its own private share to the group secret and sends it on to the next new member.
3. The last new member does not add its share but only broadcasts the partial group secret to the group.
4. Upon receipt of the broadcast, every member, except the last one, removes (factors out) its private share from the partial group secret and sends the result back to the last new member who now becomes the new group controller.
5. Having received all messages, the new controller computes a new partial group secret for all members by adding its own private share to every value it received. It then broadcasts the set of partial group secrets to the entire group.
6. Upon reception of the broadcast, every member computes the new group secret.

4.3 Leave

1. The current controller updates its private share, recomputes the partial group secrets for all remaining group members and broadcasts the result to the group.
2. Upon reception of the broadcast, each user computes the new group secret.

4.4 Key Refresh

Key refresh is identical to LEAVE with the exception that it is triggered unilaterally by the controller. (The API also support an option whereby any group member, not just the controller, can cause key refresh by updating its private share.)

5 Secure Spread Architecture

As mentioned previously, Spread supports both VS and EVS group semantics. Currently secure Spread uses the VS group semantics. Also, Spread uses a daemon-client communication architecture. The current implementation of secure Spread is as a layer that extends the Spread client library and adds the new security features discussed in this paper. Client applications then link

with this secure library and use the provided API which is similiar to Spread's API. A different way secure Spread could be implemented would be to graft the security features directly onto the Spread daemons themselves. These two approaches will be referred to as the client model and the daemon model, respectively. The benefits of each model are the drawbacks of the other.

The main advantage of the client model is that no trust is put into anything outside of the end user's control. The secure Spread client implements all of the security operations in the system. Because of this and the fact that the client is open source, an end user can easily verify for himself that the secure Spread client is trustworthy. Another advantage is that this particular design demonstrates how to secure any group communication system that supports the VS model.

The daemon model denies an end user access to the security implementation, especially if the daemon network is out of the end user's control, however, it does offer several other advantages. The main benefit gained from this approach is a large increase in performance. Keys would no longer need to be established for every group. Instead, the daemons could encode all of their intercommunication using one private key. Daemons are long lived entities and might only update their private key if their interconnectivity changed or a daemon crashed or recovered (daemons could also choose to rekey on a time out as well). These kinds of events occur rarely in practice and the number of daemons in a configuration is usually small. Therefore, in the daemon model the number of rekeyings occurring in the system as a whole would be drastically reduced when compared to the client model. Note, however, that communication between clients and daemons would still have to be secured in some manner. For example, a client could assume that IPC is secure, or instead use encrypted communications such as SSL sockets.

Another advatage of the daemon model is that security policy can be easily configured and enforced. In this model an administrator could change policy by editing a daemon configuration file. End users connecting to daemons could be authenticated upon connection. Access control to the daemons or specific groups could be easily enforced. Daemons could also easily authenticate that users are members of specific groups.

Despite all of the apparent advantages in using the daemon model, the client model has several security advantages over the daemon model. As stated above, no trust is placed in another entity, and furthermore, the system as a whole is less vulnerable to attack. For example, in the daemon model, if a daemon key is obtained, all messages in every group are compromised until the daemons rekey. In the client model, however, if a group key is obtained then only the messages in that group are compromised until the group rekeys. Due to the different levels of security offered by the two models there will be places for both approaches to be applied.

5.1 High Level Design

A major consideration when designing a secure communications toolkit is the security and trust that go along with the algorithms used in the toolkit. As time goes on, trust in these algorithms may change: ciphers are broken, algorithms are proven insecure, better algorithms are designed, etc. Therefore, any system hoping to secure communications and be viable in the long run, needs to be flexible and easily modified. Another desirable feature is if security policy is easily changed by administrators and/or users. One way to achieve these goals is to design an extremely modular

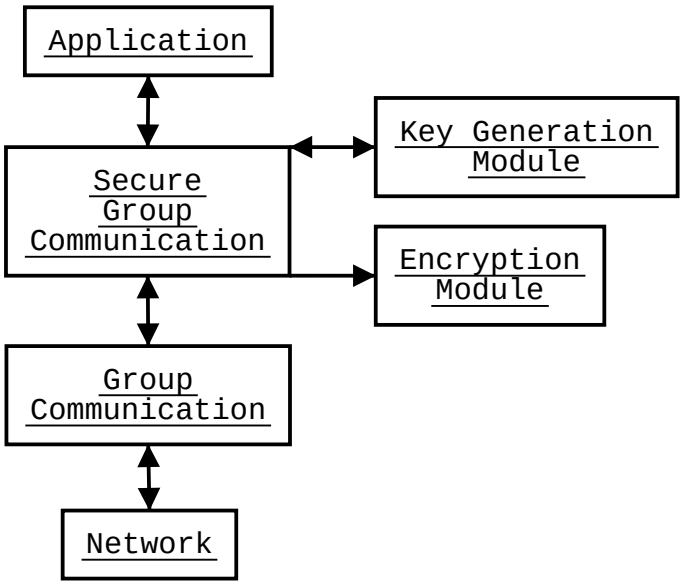


Figure 1: A Generic Secure Architecture

system.

Figure 1 shows a basic architecture for a modular secure system. When necessary, the secure group layer calls different modules that implement encryption, key management, and key generation routines. If the secure layer only needs to know when to call these modules, but not their functions, then this type of design allows for easy drop-in replacement of different modules. Therefore, if a new module needed to be added to this type of layer, then only two modifications would have to be made. First, code must be written implementing the new module, and second this module must be chosen to run at the proper times.

5.2 Secure Spread Modules

The basic design of secure Spread follows this modular design closely. The core functionality of secure Spread is an event handling loop. Network events are generated by the VS group communication layer. Secure Spread takes each of these events and depending on their context, passes it on to the proper module that handles the event.

An interesting addition to the design in secure Spread is that the modules that implement the security policies and algorithms of the system can be chosen at run time as new groups are created. This mechanism allows different groups to choose different event handlers while simultaneously running in the same group communication system. For example, one group could decide to use centralized key management, while another group is using distributed key management at the same time.

Currently, secure Spread is designed to allow for easy drop in replacement of encryption and of key agreement protocols. In the future we hope to extend the architecture to easily allow for modules that would make access control and other policy decisions as well. At this early stage of

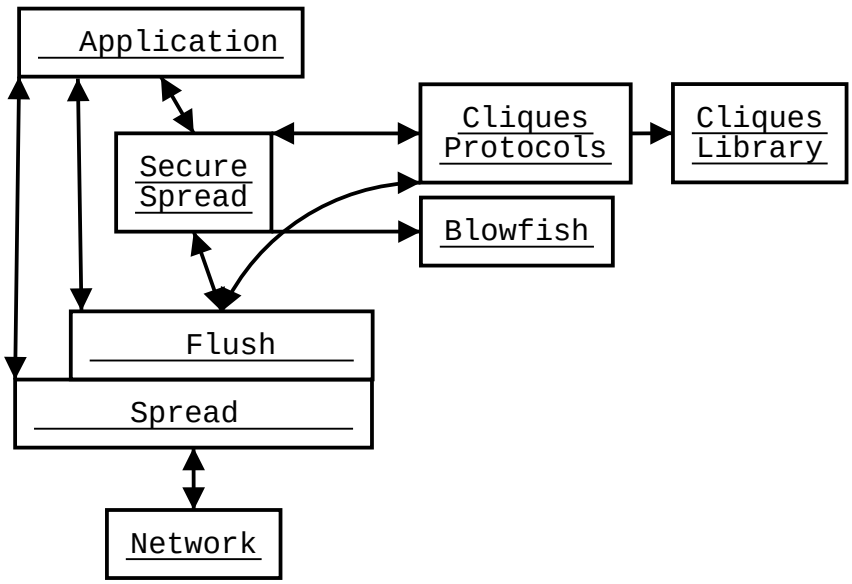


Figure 2: Secure Spread Implementation

development, secure Spread has only a few different modules that have been implemented.

Figure 2 shows the current actual implementation of secure Spread. The Flush layer provides View Synchrony semantics to the secure Spread layer. Note that the application can have several connections open at the same time and each of them can be in multiple groups. Some of these groups can be secure, while others can instead use the View Synchrony semantics of the Flush layer or the EVS semantics provided by Spread. Choosing which services are used is entirely up to the application.

Currently, for bulk data encryption secure Spread uses an open source implementation of Bruce Schneier's Blowfish algorithm [15]. In the near future we hope to add the ability to use the OpenSSL[16] cryptographic library which provides an abundant selection of encryption algorithms. With this selection of encryption algorithms available, the user could be allowed to decide which encryption algorithm they are most comfortable using.

Secure Spread currently has two different modules for key agreement, both based on the Cliques protocol suite.

- Cliques key management (using group Diffie-Hellman)
- Simple centralized key management (described in the Appendix)

The complexity of secure Spread is contained almost solely within these modules.

5.3 Implementation of Cliques Group Key Management

The Cliques key management suite and the corresponding API presented in Section 4 are independent from a particular implementation of a group communication system.

How these protocols are implemented specifically in secure Spread is the subject of this section.

In order to easily accommodate the Cliques protocols, the underlying group communication system needs to provide certain fundamental features. These include: group multicast, group member to group member unicast, FIFO ordering on messages, and a mechanism for knowing and identifying all of the members of a group. All of these services are provided by Spread.

Furthermore, the Cliques API is tuned to allow for the following types of membership changes: singleton join, singleton leave, multi-join, and multi-leave. Again, all of these types of membership events are provided by Spread. In addition, the VS model of Spread generates an event that is a combination multi-join and multi-leave, and an event requesting the application to OK a group membership change.

To actually implement the Cliques protocols to be used with a group communication system a mapping must be made between group communication events and Cliques events. Presented in Table 1 is the simple mapping of Spread's VS group events to Cliques events.

Table 1: Mapping of Spread Events to Group Key Events

Spread VS Group Membership Events	Group Key Management Operations
Join	Join
Leave	Leave
Disconnect	Leave
Partition	Leave
Merge	Merge
Partition + Merge	Leave then Merge
Group Change Request	N/A
N/A	Key Refresh

When a VS group event is generated by Spread it is passed up to the security layer. Secure Spread then examines the event and determines to which group the message corresponds. It retrieves the event handler for that particular group and gives the handler the event. This handler is responsible for mapping the network VS event into the proper actions to be taken with respect to the current state of the group and the protocols it implements as described in Section 4.

In our implementation, the Cliques protocols are implemented using state machines. The current state of a group member is associated with each process connection and a specific group. This state machine and the current state of the entity in the group will be collectively referred to as the "member state."

The current member state and the current event affecting the group determine what actions are taken by each member. The Cliques library keeps most of state information associated with a group, such as who is the current group controller. When calls are made to the CLQ_API, in general, all group members make the same function call and the return values of these calls determine how the member state of each member changes and what actions they take.

FIFO ordered messages were used for all messages used to communicate partial keys or key shares

to the group or between particular entities. This was done because FIFO ordered messages have extremely low overhead associated with them, and stronger message orderings are not required.

5.4 Cascading Failure Handling

When merging security protocols with high reliability group communication protocols we must begin with an obvious premise that security does not imply robustness, i.e., a system can be secure but not robust. A security protocol does not become more secure when built over a reliable communication platform; it becomes more robust. For instance, many security protocols for group communication assume that messages are received in certain order or that all protocol parties are (at the same time) connected and ready to receive messages. In real systems these assumptions are often not fulfilled. Adding reliability through a group communication system can make these assumptions realistic and protocols themselves – usable.

Focusing on the problem at hand recall that – as discussed above – fluctuations in group channel state must be coupled with group key adjustments. However, adjustments require the availability and reachability of each group member. If the group channel state changes while an adjustment (stemming from an earlier change) is in progress, appropriate actions must be taken not to maintain security but to preserve the overall group integrity. This is in stark contrast to two-party communication where such "cascading" fluctuations simply do not occur.

Thus the greatest challenge in integrating reliable group communication and group security protocols lies not in straightforward mapping of well-spaced communication events into their security counterparts but in the proper and robust handling of various **cascading** events that perturb group channel state.

Thus far we have only implemented key agreement for non-cascading membership events. This allows us to benchmark the system and learn about its performance in essentially all of the common cases. However we do have solutions handling these cases that will be implemented in the near future. We discuss the ideas behind these solutions here.

The basic difficulty arises after the secure layer has received a new membership event from the VS layer and started to do whatever security work needs to be completed before the new membership configuration can be considered secure and the application can continue working. The security layer now has to send more messages and do computation to, for example, generate a new key if someone joined the group. However, during this time new membership events can occur, which might invalidate the current membership (say a partition occurred) or might necessitate more work (another member joined and also needs to be added to the secure group). Actually, an essentially unlimited number of such events can happen before the security layer has completed handling the first membership event. This implies that there is an upper limit on how many membership events can occur per unit time if the system is to perform any useful work.

All membership events can be divided into two types: those that affect the current secure membership handling, and those that can be safely deferred until the current event is handled.

For example, if someone joins a group and during the handling of the join someone else leaves, the leave must be included in the handling of the join or else the leaving member will either learn the key with the new member in it (which they should not know), or they have left because they died

and if they are needed to complete the secure join then the system can deadlock (i.e. if they were the previous controller).

An example of a membership event type which is safe to defer, is a join to a group while a join of another member is being completed. Since the second join has no effect on a group that they have not officially/securely joined, the first join will be able to securely and successfully complete.

Note that one can not just use the VS properties which allow each application to delay the new membership until it is finished, because at the time the security layer is asked to OK a new membership change it does not yet know what the membership event is (Join/Leave/Partition) or who is involved. Thus, it can not know whether the event is safe to defer or not.

The security layer must OK the new membership. After OK'ing it and receiving the new membership the security layer evaluates the event and determines if it is deferable. If it is, it queues it and continues with the security protocol. If it is not, then it follows a state dependent protocol that includes the new information into the current proposed secure membership and restarts the key agreement protocol based on the new secure group.

6 Experimental Results

In this section we present the experimental results obtained with Secure Spread. The experiments were performed in two different architecture sets under the same group scenario. In the first set, all machines were SUN Ultra-s 2 Model 1200 (200 MHz UltraSPARC, 128MB) running Solaris 5.5.1, while in the other set, all machines were Pentium II (450 MHz, 128MB) running RedHat Linux 2.2.7. The timings were obtained performing multiple batches of each operation 50 times and then averaging across batches. The setup consists of three identical machines, each running its own Spread daemon. Two machines had a single member each and the third machine contained all other members (processes) utilizing a single daemon. Machines were connected with an Ethernet 10 BaseT network for the SUN Ultra-2s and an Ethernet 100 BaseT network for the Pentium II machines. CLQ_API was linked with OpenSSL 0.9.3a [16], where one Diffie-Hellman (DH) exponentiation with 512-bit modulus costs 12 and 2.5 msecs for the SUN and Pentium platforms, respectively.

The total number[7] of serial exponentiations required for Join and Leave operations is illustrated in Tables 2 and 3, respectively. Table 4 summarize the total number of serial exponentiations performed in the course of each operation.

Figure 3 shows the total time of one join or leave operation, which costs the same, versus the group size.[8] The time in this graph includes the network overhead. The Flush layer timings in the graph are not linear due to the fact that n messages have to be broadcasted from each member to every other. This interaction caused a significant amount of overhead[9] in the machine that has most of the group members.

[7] During the join operation n includes the new member and during the leave operation n includes the leaving member.

[8] The Flush layer values on the Sparc architecture are not included.

[9] This overhead could have been caused by the operating system scheduler as well.

Table 2: Detailed number of exponentiation for Join

Cliques	Controller	Update key share with every member	$n-1$
		Long term key computation with new member	1
		New session key computation	1
		Total:	$n+1$
	New Member	Long term key computations	$n-1$
		Encryption of session key	$n-1$
		New session key computation	1
		Total:	$2n-1$
CKD	Controller	Long term key computation with new member	1
		Pairwise key computation with new member	1
		New session key computation	1
		Encryption of session key	$n-1$
		Total:	$n+2$
	New Member	Long term key computation with controller	1
		Pairwise key computation with controller	1
		Encryption of pairwise secret for controller	1
		Decryption of session key	1
		Total:	4

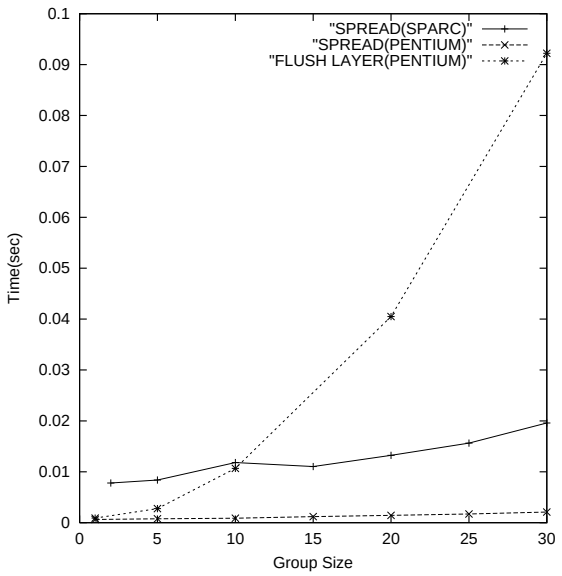
Table 3: Detailed number of exponentiation for Leave

Cliques	Remove long term key with previous controller	1
	New session key computation	1
	Encryption of session key	$n-2$
	Total:	n
CKD	New session key computation	1
	Encryption of session key	$n-2$
	Total:	$n-1$
CKD, when controller leaves	Long term key computations	$n-2$
	Pairwise key computation with new user	$n-2$
	New session key computation	1
	Encryption of session key	$n-2$
	Total:	$3n-5$

Table 4: Total number of serial exponentiation

Operation	Join	Leave	Controller leaves
Number of members after operation	n	$n-1$	$n-1$
Cliques	$3n$	n	n
CKD	$n+6$	$n-1$	$3n-5$

Figure 3: Spread and Flush Layer Timings



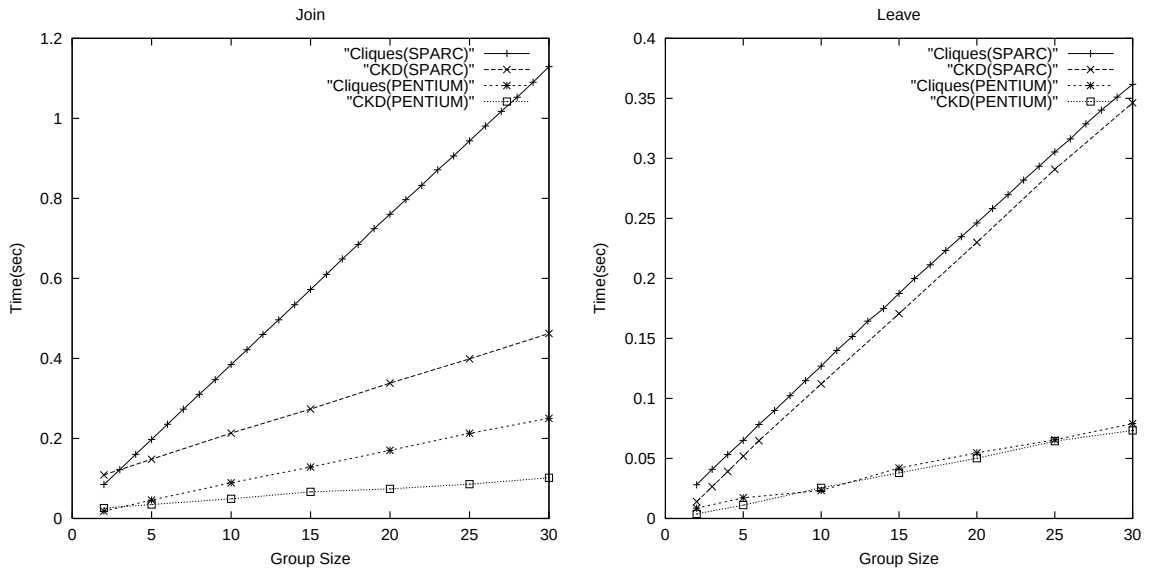


Figure 4: Timings

Figure 4 shows the total time of Join and Leave operations versus the group size in two separate graphs. The graphs seem to follow closely the total number of expected exponentiations in Tables 2 and 3. In other words, the results clearly show that exponentiation is the most dominant operation, as compared with network overhead, other arithmetic operations, and disk I/O. For example, a join operation in a group of fifteen members takes 0.1125 secs for the modular exponentiation and the total CPU time experimentally takes 0.1285 secs for the Pentium. Hence, 88% of the CPU was used for modular exponentiation.

7 Related Work

Related work falls into three categories: 1) cryptographic protocols for group key management, 2) architectures and frameworks for secure multicast and 3) implementation of secure group communication systems. Given the "systems" orientation of this paper, we concentrate on (3). Readers interested in (1) are referred to [10] and [12]. We choose not to dwell on the related work in secure multicast since both its security and its communication models differ greatly from those of dynamic peer groups. Suffice it to say that it typically assumes one-to-many communication paradigm and emphasizes scalability over strong security; more specifically, the focus is on minimizing bandidth overhead from re-keying a very large group.[10] Generation and distribution of group keys is assumed to be performed by trusted servers which, as discussed above, is an approach unsuitable for a peer group setting.

Group communication systems in LAN environments have a well-developed history beginning with ISIS [17], and more recent systems such as Transis [18], Horus [19], Totem [20], and RMP

[10]Most recent work in secure multicast has taken place under the aegis of the Internet Research Task Force's (IRTF) Secure Multicast Group (SMUG); see *http://www.ipmulticast.com* for further information.

[21]. These systems explored several different models of Group Communication such as Virtual Synchrony [23] and Extended Virtual Synchrony [8]. More recent work in this area focuses on scaling group membership to Wide Area Networks (WANs) [24].

Research in securing group communication systems is fairly new and is only beginning to be explored. The only actual implementations of group communication systems which focused on security issues are the secure distributed CORBA (Immune) system built on top of SecureRing[25] group communication work at UCSB [26] and Horus/Ensemble work at Cornell [27].

The Immune system from UCSB provides protection against Byzantine failures using cryptographic techniques to secure a low-level ring protocol (that forms the base of the Totem system) and replicating the protected CORBA objects sufficiently to detect and recover from upto a fixed number of compromised objects or machines.

The Ensemble security work exemplifies the state-of-the-art in reliable group communication security and addresses several of the same problems we consider in this paper. It also allows application-dependent trust models and optimizes certain aspects of group key generation and distribution protocols. The problems we both address are in generating shared group keys and rekeying, however, our respective approaches are quite different. The Ensemble security protocols are tied into the Horus/Ensemble layered protocol stack architecture and interact with the group communication system at several levels, not just as an application. Ensemble also uses a different group key structure which is not contributory and provides a different set of security guarantees.

More specifically, Ensemble relies on *ad hoc* extensions of conventional (non group-oriented) security tools, namely PGP [28], to distribute and refresh group keys. The main drawbacks of such approaches are (recall that these are also summarized in Section 2.2):

- Generation of group keys is centralized (done by group leader) which goes contrary to the peer group model. In particular, the entire group relies on one member's ability to generate strong, secure keys. It is unclear how a group leader is selected/elected in the event of partition or failure.
- Key distribution is performed with the help of encryption (via PGP) which is expensive since the key must be encrypted individually for each member. Moreover, this method is insecure since compromise of just one member's long-term secret (PGP private key) exposes all previous session keys, and, hence, all prior group communication. (A more secure approach would avoid this problem by providing *Perfect Forward Secrecy*, i.e., it would preclude distribution of keys encrypted under long-term secrets.)

8 Future Work

This paper represents only the tip of the proverbial iceberg. Much work remains to be done in constructing a comprehensive architecture and implementation for secure and reliable group communication.

As the immediate next step, secure handling of cascaded group events must be implemented as sketched out in Section 5. Also, more experimentation is needed to better assess the impact of

security on group communication. In the short term, we need to consider other important and more specialized group security services that can be built on top of the basic services, i.e., distributed key management and data privacy/integrity. These include:

- intra-group member authentication
- secure communication with non-members (authentication, privacy, integrity)
- group integrity (how to authenticate the entire membership at once)
- group/member anonymity
- membership non-repudiation

There are also several fundamental, long-term research topics that this work has not addressed. Dynamic peer groups present interesting policy considerations that traditional two-party communication does not. New policy constraints might include: altering group membership; policy contingent on current membership; time based group admission constraints. Group certification is also an open research problem, including such issues as: how to issue, manage and revoke certificates for constantly changing groups. Finally, there is the need to extend peer group security to two-tiered groups composed of small number of senders and a comparatively large number of receivers.

References

[1] H. Harney and C. Muckenhirn, "Group key management protocol (GKMP) specification," Request for Comments (Experimental) 2093, Internet Engineering Task Force, July 1997.

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

[3] S. T. Kent, "Internet privacy enhanced mail," *Communications of the ACM*, vol. 36, pp. 48–60, Aug. 1993.

[4] National Bureau of Standards, "Data Encryption Standard DES, Data Encryption Standard Model of Operation." Government Printing Office, Washington, D.C., 1980. Federal Information Processing Standards FIPS-81.

[5] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: keyed-hashing for message authentication," Request for Comments (Informational) 2104, Internet Engineering Task Force, Feb. 1997.

[6] J. G. Steiner, C. Neuman, and J. I. Schiller, "Kerberos: An authentication service for open network systems," in *Usenix Winter Conference*, pp. 191–202, Jan. 1988.

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

[8] 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.

[9] Y. Amir, "Replication using group communication over a partitioned network", Ph.D Thesis from the Institute of Computer Science, The Hebrew University of Jerusalem, 1995.

[10] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-hellman key distribution extended to groups," in *3nd ACM Conference on Computer and Communications Security*, pp. 31–37, ACM Press, Mar. 1996.

[11] M. Steiner, G. Tsudik, and M. Waidner, "CLIQUES: A new approach to group key agreement," in *IEEE International Conference on Distributed Computing Systems*, May 1998.

[12] G. Ateniese, M. Steiner, and G. Tsudik, "New multi-party authentication services and key agreement protocols," in *IEEE Journal of Selected Areas in Communication*, 1999. Accepted for publication, to appear in early 2000.

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

[14] G. Ateniese, O. Chevassut, D. Hasse, Y. Kim, and G. Tsudik, "The design of a group key agreement api," in *Submitted to First International Workshop on Networked Group Communication*, July 1999.

[15] B. Schneier, "The blowfish encryption algorithm," *Dr. Dobb's Journal*, pp. 38–40, Apr. 1994.

[16] OpenSSL Project team, "Openssl," May 1999. http://www.openssl.org/.

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

[18] Y. Amir, D. Dolev, S. Kramer, and D. Malki, "Transis: A communication subsystem for high-availability," in *Digest of Papers, The 22nd International Symposium on Fault-Tolerant Computing Systems*, pp. 76–84, IEEE Computer Society Press, Los Alamitos, CA, 1992.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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* (M. Mavronicolas, M. Merritt, and N. Shavit, eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1998.

[25] 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.

[26] P. Narasimhan, K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, "Providing support for survivable corba applications with the immune system," in *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, (Austin, TX), pp. 507–516, May 1999.

[27] 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.

[28] P. Zimmerman, *The Official PGP User's Guide*. prz@acm.org, 1994. The MIT Press In press. More in http://www.pegasus.esprit.ec.org/people/arne/pgp.html.

A Centralized Key Distribution protocol

The Centralized Key Distribution (CKD) protocol is a simple group key distribution scheme. It provides the same level of security as Cliques, as far as key independence, key confirmation, perfect forward secrecy and resistance to known key attacks. However, it is not contributory as the group secret is always generated by one member, namely, the current group controller.[11] Following each membership change, the current controller generates the secret and distributes it to the group in a secure manner. Unlike Cliques, the controller is always the oldest member.

Regardless of the group operation, the CKD key distribution protocol consists of two phases:

1. Each group member and the controller agree on a key using authenticated two-party Diffie-Hellman.
   This key does not need to change as long as both users remain in the group. If the controller leaves the group, the new controller has to perform this operation with every member. On the other hand, if a regular member leaves, the controller simply discards this key.

[11] We use the term *current* to mean that, even in the CKD protocol suite, a controller can fail or be partitioned out thus causing the contoller role to be reassigned to the oldest suviving member.

Protocol CKD: Let (x_1, α^{x_1}) and (x_{n+1}, $\alpha^{x_{n+1}}$) be the secret and public keys of M_1, the group controller, and M_{n+1} respectively. Let $K_{1n+1} = \alpha^{x_1 x_{n+1}}$ mod p. Assume that the group has n members, and that M_{n+1} wants to join the group.
The protocol runs as follows:
Round 1:
M_1 selects random $r_1 mod q$ (this selection is performed only once),
$M_1 \longrightarrow M_{n+1} : \alpha^{r_1} mod p$
Round 2:
M_{n+1} selects random $r_{n+1} mod q$,
$M_1 \longleftarrow M_{n+1} : \alpha^{r_{n+1} K_{1n+1}} mod p$
Round 3:
M_1 selects a random group secret Ks and computes
$M_1 \longrightarrow M_i : Ks^{\alpha^{r_1 r_i}} mod p \forall i \in [2..n+1]$

Table 5: CKD protocol

2. The group controller unilaterally generates and distributes the group secret.