Exploring Adaptability of Secure Group Communication using Formal Prototyping Techniques

Sebastian Gutierrez-Nolasco, Nalini Venkatasubramanian
School of Inf. and Comp. Science
University of California Irvine
Irvine, CA 92697-3430, USA
{seguti,nalini}@ics.uci.edu

Mark-Oliver Stehr
Dept. of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
stehr@uiuc.edu

Carolyn Talcott
Computer Science Laboratory
SRI International
Menlo-Park, CA 94025, USA
clt@cs.stanford.edu

ABSTRACT

Traditionally, adaptability in communication frameworks has been restricted to predefined choices without taking into consideration tradeoffs between them and the application requirements. Furthermore, different applications with an entire spectrum of requirements will have to adapt to these predefined choices instead of tailoring the communication framework to fit their needs. In this paper we extend an executable specification of a state-of-the-art secure group communication subsystem to explore two dimensions of adaptability, namely security and synchrony. In particular, we relax the traditional requirement of virtual synchrony (a well-known bottleneck) and propose various generic optimizations, while preserving essential security guarantees.

1. INTRODUCTION

Dynamic peer groups are common in collaborative applications of all kind such as server replication, clustering, distributed logging, grid computing, factory control, video conferencing, distributed interactive simulations, on-line games, air traffic control, and financial markets. These applications ideally run on top of a group communication system (GCS), which provides reliable and ordered message delivery and protects sensitive information against unauthorized entities. Due to the dynamic membership of peer groups, the expensive cryptographic protocols, and the potential real-time requirements of applications, securing group communication in dynamic environments is a challenging task.

In recent years some secure GCS have been developed [18, 12, 13, 7, 10] and several useful techniques have been proposed to deal with scalability, performance and security in peer groups with dynamic membership and decentralized control[1, 24, 11]. However, GCS were designed to be highly efficient in local (wired) networks, assume a relatively small group size (up to few hundred), and do not consider mobility, temporary disconnections and real time constraints. In particular, scalability and high performance are both currently achieved via the light-weight/heavy-weight model[2, 15], where powerful servers (daemons) residing in each host execute relatively expensive distributed protocols and several clients can connect to a server to share the GCS services on each host.

The next generation of adaptable GCS is driven by constantly changing application requirements, real-time data delivery, intermittent membership changes due to temporary disconnections and mobility patterns, performance requirements and non-uniform security and fault tolerance levels. For example, in current troop deployment simulations (TDS), a computer at the headquarters gathers all the information from the battlefield and displays the current locations of troops, vehicles and obstacles (such as mine fields). The headquarters computer is networked via a secure link to a set of commander PDAs which are connected directly to one another forming a dynamic peer group system. In a dynamic peer group system, members may join or leave the group dynamically either intentionally or due to communication interference or failures.

Due to the high computational overhead of public key cryptography, symmetric keys are commonly used to encrypt the data. To fully exploit the multicasting nature, a shared group key is typically considered to be the most efficient solution. Consequently, the main problem now becomes the efficient establishment and management of keys. The current Secure Spread system [22] uses key establishment protocols that stall all communication (at the application level), while the key is generated and rely on strong synchronization guarantees to assure that no member can receive and decrypt messages after he left the group (*forward secrecy*) and no new member can receive and decrypt messages sent before he joined the group (*backward secrecy*).

However, in many applications, such as the TDS example, disconnections are common and expected and data in transit must not only be protected against unauthorized users, but also must be delivered in a timely manner, so that decisions can be made from accurate and fresh data. Triggering a blocking rekey after every join or leave (to preserve forward and backward secrecy) may preclude timely delivery of sensitive information and even may lead to potential denial of service attacks if a trusted member is compromised and joins and leaves the group intermittently. In this case, it would be desirable to employ a less constrained GCS that does not require the generation of a new key after every join or leave, but still maintains a certain degree of security.

In fact, we believe that an application should be able to tailor the secure GCS according to its needs not only in terms of security but also synchrony, timeliness and reliability, because there is no one-size-fits-all solution.

In this paper we study two dimensions of adaptability, namely se-

3rd Workshop on Adaptive and Reflective Middleware Toronto, Canada

curity and synchrony. Our starting point is a formal prototype of the Secure Spread GCS. The formal prototype has been developed earlier in the DARPA Fault-Tolerant Network (FTN) Program, and we have now generalized this specification along various lines to support secure communication with fewer synchronization constraints and adaptability along several new dimensions. In particular, our approach opens a spectrum of new security guarantees, which are weaker than in the synchronized case, but still sufficient for many applications. Thanks to the use of abstract APIs, our generalizations are to a large degree independent of the group communication system and the key establishment algorithm, and hence can be combined with improvements along other dimensions, such as the choice of specific group communication protocols and key establishment protocols. The use of formal prototyping techniques based on the executable specification language Maude enabled us to explore and validate design decisions without the need to carry out an actual implementation.

2. STATE OF THE ART IN GCS

After a brief explanation of the relevant group communication system semantics, this section gives an overview of a state-of-the-art group communication system (Spread) and a framework for key establishment protocols (Cliques), and discusses how these components are assembled to provide a secure group communication architecture (Secure Spread). In this paper we use Spread and Secure Spread without further qualification to refer to the publicly released versions that can be found at `http://www.spread.org/`.

2.1 Semantics of Group Communication

The most well-known group communication model is the *virtual synchrony semantics* (VS semantics) [2] which was originally developed for Isis/Horus [14], a primary component GCS, but later extended to partitionable GCS. One of these extensions is the *extended virtual synchrony semantics* (EVS semantics) [8], a model that extends the virtual synchrony model of Isis to support continued operation in all components of a partitioned network. The central concept of group communication is that of a *view*, *i.e.* a snapshot of membership in a group. In each execution of a partitionable GCS, views and transitions between them form a partial order. Both, the VS and the EVS semantics, share the key property of *virtual synchrony*, namely that every two processes that participate in the same two consecutive view changes, deliver the same set of messages between the two changes.

Virtual synchrony, however, is only one property of the VS semantics. The VS semantics furthermore ensures that messages are delivered in the same view they were sent in (sending view delivery). To accomplish this, an extra round of acknowledgment messages is needed every time before a view change, preventing applications to send other messages until the next view is installed. Furthermore, the VS semantics is a closed group semantics, allowing only current members of the group to send messages to the group.

The EVS semantics, on the other hand, allows message delivery in a different view than it was sent in, as long as the message is delivered in the same view to all members (same view delivery). Consequently, the synchronization phase which allows the application to be aware of the sending view is not needed in the EVS semantics. The EVS semantics also allows open groups, where non-members of the group can send messages to a group.

2.2 Spread

The Spread group communication system [17] emerged from the work on Transis[19] and Totem[21] and has been designed to cope with node failure and network partitions. Spread supports the EVS semantics and provides different levels of service with different reliability and ordering guarantees: Messages can be reliable, fifo, causally ordered, totally ordered (also called agreed), or safe, where the later means that messages are only delivered if it is known that everybody in the group has actually received it.

The Spread architecture consists of two layers, which are correspondingly reflected in our formal specification: the heavy-weight group layer and the light-weight group layer. The heavy-weight group layer provides extended virtual synchrony semantics at the level of the *physical group*, i.e. the group of hosts (servers). Due to changing network connectivity, we are really concerned with snapshots of group membership, which are called *configurations*. This layer provides services to multicast data messages which should be sent ideally to every host and to retrieve messages that have been delivered to the application, which can be either application data messages or messages that represent configuration change events.

The primary mode of operation is to deliver messages to all hosts which are part of the most recently established regular configuration. According to the EVS semantics all messages should be delivered at each of these hosts in the same regular configuration or the following transitional configuration (see below). This delivery is furthermore subject to ordering constraints that depend on the service level that was requested when the message was sent. In the case of safe messages, it is also subject the constraint that every host in the configuration has received this message, and hence can deliver it unless it crashes.

If a change in the connectivity is detected, two different configuration change events are generated: First, there is an event to introduce a transitional configuration, which is a reduced configuration in which certain messages can be delivered that could not be delivered in the previous regular configuration. After this transitional phase, a new regular configuration is introduced which reflects the new connectivity of the network.

The light-weight group layer provides EVS semantics at the level of logical groups, i.e. groups of agents (clients), simply called groups in the following. Groups are identified by names and the different snapshots of group membership are called views. The API is similar to that at the heavy-weight group layer, except that messages and changes refer to groups instead of configurations, but in addition the API offers two new services at this level: A client can request to join or leave a group, and in response Spread generates corresponding group change events when the actual transition to the new view has occurred.

It is worth to emphasize that in the EVS semantics the application cannot determine or even know the view in which the message is sent by the GCS. The application passes messages to the GCS where they can be buffered. Hence, the most recently established view at the time when the application sends the message is not necessarily the view in which the message is sent out by the GCS, let alone the view when the message is delivered to the receiving application.

2.3 Secure Spread

Secure Spread [22] provides secure group communication for closed groups and can operate with different protocols that establish a single key shared by all members of the current view. Secure Spread is built on top of Flush Spread [20] and the Cliques toolkit [11]. Flush Spread has a similar functionality as Spread but provides the stronger virtual synchrony semantics, which requires acknowledgments by all members for each view change. In [20] it is explained how VS semantics can be implemented using the weaker EVS semantics. The Flush Spread implementation is essentially a

refinement of these ideas.

The Cliques toolkit [3] provides a generic API and implementations of various group key agreement protocols, among them the Group-Diffie-Hellman protocol (GDH) [11] and a tree-based variant (TGDH). Authentication is not provided by the key agreement protocol, but instead all messages are authenticated using digital signatures. An interesting feature of GDH and its variants is that they are contributory, which means that every member contributes a key share, but the entire key is never transmitted over the channel (not even in encrypted form). However, this leads to the essential requirement that all members actively participate in the key agreement.

Secure Spread simply uses the underlying Flush Spread to exchange the messages required and produced by the Cliques toolkit, whenever a group change occurs. If the key agreement is itself interrupted by a new group change the Cliques protocol is restarted. Furthermore, Secure Spread implements some optimizations allowing several subsequent joins and leaves to be batched into a single call of the delete/merge subprotocol.

3. FORMAL METHODOLOGY

The general methodology we employ for system design and analysis is based on an executable specification language called Maude [9]. Its theoretical foundation is rewriting logic [5], a logic with an operational as well as a model-theoretic semantics. Formal prototyping is a key ingredient of our methodology, which allows us to experiment with a abstract mathematical but executable specification of the system early in the design phase. Our experience indicates that the combination of mathematical rigor with execution and analysis tools such as Maude leads to better understanding of the system and often pinpoints potential problems.

To employ this methodology in the exploration of adaptive secure group communication, we build upon abstract executable specifications of all relevant components of Secure Spread. This includes the physical and logical group layers, providing the functionality of Spread [17] with its EVS semantics. The more constrained VS semantics is provided by a specification of Flush Spread [6] on top of this. Independently, a specification of the Cliques toolkit [3] instantiated to the GDH protocol [11] has been developed. On top of all these components an executable specification of Secure Spread has been built, more precisely the basic algorithm described in [22]. This effort, which has been mainly conducted in the context of the DARPA FTN Program, was based on [16, 6, 22], the source code and discussions with the developers, in particular Y. Amir, J. Schultz, and G. Tsudik. We will not discuss the formal details of the specifications in this paper, but the interested reader can find all the components on the web [4].

4. HIGH-LEVEL ADAPTABILITY

Although security on top of the VS semantics enables perfect forward and backward secrecy in a straightforward manner by forcing to rekey after every group membership change, it possesses a high overhead when view changes are very frequent or real-time constraints have to be met. In fact, if a group membership change occurs while the key establishment is in progress, the key establishment protocol is restarted, further exacerbating the time required to generate a new key.

As we briefly explained in Section 1, the application should be able to tailor the secure GCS according to its needs in terms of synchrony and security. In order to provide this level of adaptability, we need to identify what assumptions need to be relaxed, what are the tradeoffs between these different levels and what parameters can be adjusted to tune the performance.

4.1 Adaptable Synchrony

Secure Spread implements security on top of Flush Spread, a layer providing the VS semantics, which guarantees that messages are sent and delivered in the same view. This synchronization makes it easier to implement the key establishment protocol because every message is encrypted with the same key as the receiver believes is current when the message is delivered.

In order to provide security on top of EVS semantics, the secure GCS can not longer assume that the received message was encrypted with the current key. The paper [18] proposes a solution to this problem based on two levels of keys used by the heavy-weight and the light-weight layer, respectively. In the present paper we use the idea of [18] to maintain a history of keys indexed by key identifiers (keyids), but we stick to the use of light-weight group keys without assuming underlying heavy-weight keys. This enables us to study the interaction between security and EVS semantics in its pure form and makes the solution independent of the implementation of Spread. Furthermore, given that we already have a specification of Secure Spread, it makes it easy to obtain an integrated solution which can be adapted to both, the original VS-based security, exactly as implemented in Secure Spread, and to the new EVS-based security.

Hence, we have modified the formal prototype of Secure Spread as follows: First, for EVS groups (we added VS and EVS group synchrony modes as adaptation parameters) we removed the synchronization constraints imposed by the Flush Spread layer. Second, every key generated is associated with a keyid, *i.e.* a key identifier, every message is tagged with the corresponding keyid of the key used to encrypt the message and every member of the group keeps a list of (possibly old) keys and their associated keyids. Thus, every time a message is received its keyid is checked and the corresponding key is fetched from the list so it can be properly decrypted. Thus members can move from one view to another one and rekey asynchronously. Every rekey phase adds the current key to the list of older keys and the newly generated key is used as the current key.

Obviously, the dynamics of this approach is far less constrained than in the VS case. Specifically, we observed the following difficulties: Although keyids allow to decrypt messages sent in previous views, they do not guarantee that every message received can be decrypted and delivered to the application. In particular, it may be possible that a new member receives an old message sent in a previous view. If he joined the group very recently, he does not have the key required to decrypt. One possibility would be to drop the message, but this would violate the EVS semantics (only a network change can justify dropping a message). We have addressed this issue by introducing the concept of a *nondecryptable* message, *i.e.* a message with content that is not accessible, to inform the application of this situation. However, there is also the possibility that the new member can find a key in his list associated with the keyid of the message, but it is not the keyid associated with the new view. In this case, we say that the message was encrypted under an *old keyid*, and we tag the message as *delayed* to inform the application of this situation.

Security on top of EVS allows us to increase concurrency and hence performance by providing non-blocking (application level) communication that uses the most recently established key to send messages, while the key establishment for the new view is in progress. However, this new added flexibility relaxes the degree of consistency in the system and eliminates some security guarantees, *i.e.* messages may not be encrypted with a key for the current view and

two new message tags need to be added to preserve the property that all received messages are delivered to the application (*nondecryptable* message) and to warn the application that a possibly (very) old message has been received and its contents may be suspicious (*delayed* message).

4.2 Adaptable Security

The choice of the key establishment protocol is a natural dimension of adaptability in secure group communication. However, even with the most efficient key establishment protocols, network connectivity changes and membership changes can cascade while the key establishment is in progress, causing a restart the key establishment protocol from scratch. Thus, delaying the execution of the key establishment protocol and carefully avoiding its execution in certain situations can improve system performance while preserving forward and backward secrecy.

We have explored two approaches to reduce the number of key establishment phases. The first approach is based on key caching and the second one is based on lazy key establishment, that is delaying key establishment until the key is really needed. Both approaches are *generic*, that is independent of the underlying protocol, and can be composed to further improve system performance without sacrifying security guarantees. As an important by-product, key caching allows us to deal efficiently with temporary disconnections (as opposed to voluntary join/leave events), which are quite common in groups with mobile participants and their consequences are similar to a network connectivity changes.

Interestingly, the decision to (partially) relax virtual synchrony has opened a variety of new possibilities, which includes not only the possibility to perform lazy key establishment but also new secure delivery modes.

4.2.1 Key Establishment Protocols

One of the most important security guarantees is data confidentiality, which protects data from being eavesdropped. The way the secret shared group key is computed, how often, and when it is computed are critical for the security of the GCS.

There are two basic approaches to generate a secret shared key in GCS. In the centralized approach, one member (typically a group leader) chooses the group key and distributes it to all group members (*group key distribution*); while in the contributory approach every member contributes to the creation of the secret shared key (*group key agreement*). Although the centralized approach works reasonably well for static (possibly large) groups, it turns out that the contributory approach is more robust for non-hierarchical (midsize) groups with dynamically changing memberships [24].

The relevant properties for key establishment algorithms are of purely computational nature [23]: *Cryptographic forward secrecy* guarantees that a passive adversary who knows a contiguous subset of old group keys cannot discover subsequent group keys. *Cryptographic backward secrecy* guarantees that a passive adversary who knows a contiguous subset of group keys cannot discover preceding group keys.

In a GCS like Secure Spread that supports the VS semantics, tightly synchronizing view changes with key agreement phases, backward and forward secrecy are immediate consequences of cryptographic forward and cryptographic backward secrecy, respectively [22]: *Forward secrecy* guarantees that nobody should be able to read messages sent to a group after he left this group (assuming he will not become a future member of the group). *Backward secrecy* guarantees that nobody should be able to read messages sent to a group before he joined this group (assuming he was not a past member of the group).

However, to be precise, we need to define what are the join/leave events referenced in these definitions. It obviously would not make sense to take them to be the events of requesting a join/leave at the GCS. These events would be of no use for the client applications. They are not (immediately) observable for the applications, because the processing of such requests can be delayed. This suggests to define leave/join events to be the events where the GCS delivers leave/join (with the new view) to the application which sends the message. Similarly, we have to be precise about what the send event in these definitions refers to. Since a message carries sensitive data, we should adopt the most conservative definition, namely the event when the application requests the GCS to send a message.

Forward secrecy under the EVS semantics is fairly straightforward: Assume a member A leaves the group G, the GCS delivers a new view to B, and B sends a message M to G. The new view can have only been delivered after successful completion of a key agreement phase between the members of the new view. Since M is encrypted with the resulting key that A does not know, forward secrecy is guaranteed.

Backward secrecy under the EVS sematics, however, does not hold, as the following counterexample shows: Assume A requests the GCS to send a message M to a group G, but the processing of this request is delayed. In the meantime B joins G, and the GCS delivers the new view $\{A, B\}$ to A. Now the GCS processes the send request in the new view, which means that the message is encrypted using using the key associated with this view. Hence, B can decrypt the message, which is a violation of backward secrecy.

To solve this problem we have adopted the following solution: We add the view in which we would like to send the message (*requested sending view*) as an argument to the multicast service. This view determines the key to be used for encryption. Even if the message is sent out in the new view, the key of the requested sending view should be used. Note that there are two possibilities for a member of the new view. If it was a member of the earlier sending view it can decrypt the message. If it was not a member of the earlier sending view it just joined the group and will not be able to decrypt in accordance with backward secrecy. In this case, the message is delivered but as *nondecryptable.* The possibility to specify a requested sending view is optional, so that if backward secrecy is not a concern the original implementation can be used.

The high-level rationale for this solution is the following: The EVS semantics leads to a loss of sending view awareness at the application, but the benefits of sending view awareness can be recovered by always sending messages with a *requested sending view*, which prevents members joining unexpectedly to decrypt messages not intended for them. The drawback is that we have to internally keep track of former keys, and some messages received will be *nondecryptable.* Both of these mechanisms, however, were already added when we moved from the VS to the EVS semantics (see Section 4.1) so that this extension does not cause any additional overhead.

4.2.2 Key Caching

Frequent network connectivity changes may trigger patterns of membership changes, where new views tend to have the same members as earlier views. Current implementations of secure GCS generate a new key for each view. Thus, if a subset of members of a group becomes temporary isolated due to a network partition, the key establishment protocol will be invoked for each new partition, and again when the partitions merge together. No member has left/joined the group, but several new keys have been generated. Obviously, this is unnecessary, because the group member-

ship has not changed in the end. Ideally, the key establishment protocol should be executed only if the current set of members has not shared a secret key before; otherwise, a previously agreed upon key can be used instead. Since the reuse of keys increases the vulnerability to crypto-analysis attacks, key caching like all forms of key reuse need to be carefully constrained. To this end, keys can be equipped with an expiration or some other attribute limiting key reuse, and they are removed from the list when this limit is reached.

In detail we have made the following modifications to our formal prototype to accommodate for key caching:

1. Every member keeps a list of keys and the associated set of members that share that key. The list is updated whenever a new key is generated.
2. If a membership change or network connectivity change happens, every member receives a message with the updated membership.
3. Every member checks its list of keys and if the updated membership shared a key before, the key is retrieved and used as the current key; otherwise the key establishment is triggered and a new key is generated.

Forward and backward secrecy are still satisfied, but *key freshness*, i.e. the property that each view uses a fresh key to encrypt messages, is given up. Therefore, a new group security mode (*fresh secure*) is added to enforce freshness if the application requests this level of security. If the group security mode is fresh secure, a normal key establishment is triggered even if the members shared a secret key before. It is important to point out that a keyid associated with a nonfresh key should not be confused with an old keyid, *i.e.* a keyid associated with a previous view, and hence it does not imply that the message is delivered as delayed (see Section 4.2.4).

4.2.3 Lazy Key Establishment

Current GCS have been designed under the assumption that network connectivity changes occur rarely and that members exchange a considerable amount of messages between membership changes. However, membership changes (due to unpredictable network connectivity changes or join/leave operations) may occur quite frequently in certain environments (wireless, mobile), and with many view changes taking place it is highly unlikely that messages are sent in every intermediate view. Under these circumstances, delaying the execution of the key establishment protocol until a message needs to be send will avoid unnecessary key establishment phases. We say that a key establishment phase is unnecessary if a key is generated but not used because no message is sent before a new key is generated.

As a possible solution we explored *delayed key establishment*. Instead of a synchronized initiation of the key establishment algorithm by a view change event, the member who wants to send a message triggers the key establishment asymmetrically. Our formal prototype is modified as follows:

1. Any membership change or network connectivity change is treated normally and the membership is updated, but the key establishment protocol is not executed.
2. When a member needs to send a message, it checks if a current key exists and if it is up to date, i.e. belongs to the most recently established view.
3. If the key is up to date, then the message is encrypted and sent normally.
4. If the key does not exist or is not up to date:
   (a) The member starts the key establishment protocol, notifies the other group members and stalls the message till the new key is generated.
   (b) Members are notified and each one of them starts the key establishment protocol, which proceeds normally.
   (c) If another member wants to sent a message, the key establishment has been triggered by some other member and no view change has been triggered, the message is stalled until the new key is generated and the member continues with the normal key establishment execution (*i.e.* the key algorithm is not restarted).
   (d) If a view change event is triggered at any time, the membership is updated and the key establishment protocol is restarted.
   (e) Once the key has been generated, the current key is updated, the up-to-date flag is set and members proceed to encrypt and send the message normally.

4.2.4 Secure Delivery Modes

Traditionally, secure delivery in GCS has been restricted to the delivery of an encrypted message, assuming that all members of the group are able to decrypt the message using the unique shared group key. When we relax the virtual synchrony semantics, messages encrypted with different group keys may be received at any time and we can not longer assume that the receiver is able to decrypt every message using the most recent key or even to decrypt the message. As a result, EVS semantics leads to a new variety of secure delivery modes based on key freshness and an extended concept of *safe messages* as follows:

- Non-secure: Message is sent and received in clear-text
- Secure: Message is encrypted and can be decrypted with any (possibly old) known key; otherwise delivered as *non-decryptable.*
- Strongly secure: Message is encrypted and must be decrypted with the most recent known key; otherwise delivered as *non-decryptable.*
- safe-secure: Message is encrypted and can be decrypted with any (possibly old) known key, but can only be delivered if everybody else received and decrypted the message using any (possibly old) known key.
- Strongly safe-secure: Message is encrypted and must be decrypted with the most recent known key, but can only be delivered if everybody else received and decrypted the message using the most recent known key.

5. CONCLUDING REMARKS

In this paper we have focussed on two dimensions of high-level adaptability in group communication, namely synchrony and security, as opposed to low-level adaptability of the underlying communication protocols, which we leave as future work. We have explored several solutions and built a formal prototype to validate our ideas and explore the properties of the new design. We have emphasized adaptability, because there is no one-size-fits-all solution given the diversity of application requirements that we are concerned with.

We developed adaptation parameters that allow us to tailor (dynamically) the communication framework to specific application requirements. In the synchrony dimension, groups with different degrees of synchrony can coexist given that every group specifies its synchrony (VS or EVS), members can participate in several groups with different synchrony modes simultaneously. In the security dimension, each group specifies the degree of laziness of the key establishment protocol, which is not entirely independent of the degree of synchrony selected: (i) eager keying will trigger a rekey after every membership change; (ii) key caching will reuse previous cached keys accordingly; and (iii) lazy keying will delay rekeying until a message needs to be send. It is noteworthy that our approach is entirely generic in the sense that it is independent of the key establishment protocol and the implementation of the group communication system.

Possible directions for future work include further generic optimizations for key management and secure multicasting, dynamic access control for a high-level enforcement of security requirements, adaptability to support group communication in mobile environments, and adaptability to QoS requirements such as timeliness constraints.

6. ACKNOWLEDGMENTS

This research was supported by the Office of Naval Research under MURI Research Contract N00014-02-1-0715 and is based on earlier work supported by the Project "Composable Formal Models for High-Assurance Fault Tolerant Networks" sponsored by DARPA in the Fault-Tolerant Network Program under Contract AF SRI 27-000803. We would also like to thank Grit Denker for her collaboration in the specification of Secure Spread, and Yair Amir, John Schultz, and Gene Tsudik for various discussions in this context.

7. REFERENCES

[1] The Keyed-Hash Message Authentication Code (HMAC). In *No. FIPS 198, National Institute for Standards and Technology*, 2002.

[2] A. Fekete, N. Lynch and A. Shvartsman. Specifying and using a Partitionable Group Communication Service. In *16th Annual ACM Symposium on Principles of Distributed Computing*, 1997.

[3] C. Talcott and M.-O. Stehr. Specification of the Group Diffie-Hellman Protocol as a Component of the Cliques Toolkit. Website: `http://formal.cs.uiuc.edu/stehr/cliques_eng.html`, 2003.

[4] C. Talcott, M.-O. Stehr and G. Denker. Towards a Formal Specification of the Spread Group Communication System. Website: `http://formal.cs.uiuc.edu/stehr/spread_eng.html`, 2004.

[5] J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. In *Theoretical Computer Science 96(1):73-155*, 1992.

[6] J. Schultz. *Partitionable Virtual Synchrony Using Extended Virtual Synchrony*. Master Thesis, Department of Computer Science, Johns Hopkins University, 2001.

[7] K.P. Kihlstrom, L.E. Moser and P.M. Melliar-Smith. The SecureRing Protocols for Securing Group Communication. In *IEEE 31st Hawaii International Conference on System Sciences*, 1998.

[8] L. E. Moser, Y. Amir, P. M. Melliar-Smith and D. A. Agarwal. Extended Virtual Synchrony. In *14th International Conference on Distributed Computing Systems*, 1994.

[9] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer and C. Talcott. The Maude 2.0 System. In Robert Nieuwenhuis, editor, *Rewriting Techniques and Applications (RTA 2003)*, number 2706 in Lecture Notes in Computer Science, pages 76–87. Springer-Verlag, June 2003.

[10] M. K. Reiter. Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart. In *2nd ACM Conference on Computer and Communications Security*, 1994.

[11] M. Steiner, G. Tsudik and M. Waidner. Key Agreement in Dynamic Peer Groups. In *IEEE Transactions on Parallel and Distributed Systems*, 2000.

[12] O. Rodeh, K. Birman, M. Hayden, Z. Xiao and D. Dolev. Ensemble Security. Technical Report TR98-1703, Cornell University, 2000. Department of Computer Science.

[13] P. McDaniel, A. Prakash and P. Honeyman. Antigone: A Flexible Communication for Secure Group Communication. In *Proceedings of the 8th USENIX Security Symposium*, 1999.

[14] R. van Renesse, K. Birman and S. Maffeis. Horus: A Flexible Group Communication System. *Communication of the ACM*, 39(4):76-83, 1996.

[15] S. Floyd,V. Jacobson, C. Liu, S. McCanne and L. Zhang. A Reliable Multicast Framework for Light-weight Session and Application Level Framing. In *IEEE/ACM Transactions on Networking, (5):784-803*, 1997.

[16] Y. Amir. *Replication Using Group Communication Over a Partitioned Network*. Ph.D. Thesis, Hebrew University of Jerusalem, 1995.

[17] Y. Amir and J. Stanton. The Spread Wide Area Group Communication System. Technical Report Technical Report CNDS-98-4, Johns Hopkins University, 1998.

[18] Y. Amir, C. Nita-Rotaru, J. Stanton and G. Tsudik. Scaling Secure Group Communication Systems: Beyond Peer-to-Peer. In *DARPA Information Survivability Conference and Exposition*, 2003.

[19] Y. Amir, D. Dolev, S. Kramer and D Malki. Transis: A Communication Subsystem for High Availability. In *22nd International Symposium on Fault-Tolerant Computing Systems*, 1992.

[20] 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 *20th International Conference on Distributed Computing Systems*, 2000.

[21] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. Agarwal and P. Ciarfella. The Totem Single-Ring Ordering and Membership Protocol. In *ACM Transactions on Computer Systems*, 1995.

[22] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Schultz, J. Stanton and G. Tsudik. Secure Group Communication Using Robust Contributory Key Agreement. In *IEEE Transactions on Parallel and Distributed Systems*, 2004.

[23] Y. Kim, A. Perrig and G. Tsudik. Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups. In *ACM CCS*, 2000.

[24] Y. Kim, A. Perrig and G. Tsudik. Communication-efficient Group Key Agreement. In *IFIP SEC 2001*, 2001.