Scaling Secure Group Communication Systems: Beyond Peer-to-Peer

Yair Amir [*] Cristina Nita-Rotaru [*] Jonathan Stanton [†] Gene Tsudik [‡]

Abstract

This paper proposes several integrated security architecture designs for client-server group communication systems. In an integrated architecture, security services are implemented in servers, in contrast to a layered architecture where the same services are implemented in clients. We discuss the performance and accompanying trust issues of each proposed architecture and present experimental results that demonstrate the superior scalability of an integrated architecture.

1 Introduction

Many routine activities in modern, everyday life involve the Internet: shopping for goods (such as books, cars, software and even groceries), administering bank or credit card accounts and making financial transfers, participating in voice or video-conferences, or simply playing games. Most such activities are in fact supported by collaborative applications running over an integrated software platform, namely, a group communication system.

Group communication systems (GCSs) are essentially application-level multicast techniques providing reliable and ordered message delivery, as well as a group membership service. GCSs have been built around a number of different architectural models, such as peer-to-peer libraries, 2- or 3-level middleware hierarchies, modular protocol stacks, and client-server. Prior research on such systems has tended to favor a client-server or a hierarchical model in order to obtain good scalability. However, security research for GCSs has focused mainly on peer-to-peer, or abstract group models, by that diminishing the scalability of the non-secure client-server systems.

Currently, the need for security in computing and communications is widely recognized. Although not an independent service, security is an enabling feature without which the actual end-services cannot be trusted or relied upon. To this end, the research community has invested a lot of effort in investigating and developing effective and efficient security services. Numerous algorithms, protocols, frameworks and policy languages have been developed to provide security services in general, and, in a group setting, in particular. However, there has not been much research into the integration of security techniques into GCSs, while maintaining a reasonable level of performance.

This work tries to fill this gap, by designing practical secure GCSs and investigating issues that arise in the course of integration.

The minimal set of security services that should be provided by any GCS include:

- *Client authentication*: authenticate a client when it requests access to the GCS, e.g., when it connects to a GCS server.
- *Access control*: check if a given client is authorized to access system resources. Typical group communication resources are: joining or leaving a group and sending messages to a group.
- *Group key management*: generate a shared group key that can be used to bootstrap other group services, i.e., data integrity and confidentiality.
- *Integrity and confidentiality*: protect the contents of the communication both from eavesdropping as well as undetected modification.

We distinguish among two basic approaches to integrate security services into a client-server GCS. The first approach (referred to as the *layered architecture*) places security services in a client library layered atop the GCS client library. The second approach entails housing some (or all) security services at the servers in order to obtain a more scalable design (referred to as the *integrated architecture*).

To put this work into context, we briefly outline our earlier efforts. Some of our recent results [2] demonstrate how authentication and access control for a client-server GCS can be efficiently addressed in the context of a particular GCS, called Spread [3]. Another recent work proposes and analyzes a layered architecture for Spread, focusing on robustness and correctness of group key agreement [4]. In the present work, we propose scalable and efficient secure

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

†Department of Computer Science, George Washington University, Washington, DC 20052, USA. Email: jstanton@gwu.edu

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

architectures for Spread, focusing on providing authentication, data confidentiality and data integrity. Our specific contributions are two-fold:

- A high-performance security architecture for Spread, under two well-known group semantics: Virtual Synchrony [5] and Extended Virtual Synchrony [6]. Both models support network partitions and merges. Our approach entails using contributory group key management in a light-weight/heavy-weight [5] group architecture such that the cost of key management is amortized over many groups, while each group has its own unique key.
- Three variants of an integrated architecture that trade off encryption cost for complexity and group communication model support. We discuss their performance and security guarantees and compare them to a layered approach, demonstrating the increased scalability.

The rest of the paper is organized as follows. In Section 2 we survey notable prior work in designing secure GCSs. We then describe Spread and the group communication semantics it supports. Next, we define our security goals and propose three variants of an integrated security architecture. We show the improved scalability of our integrated architecture and provide a discussion in Sections 5 and 6. Finally, we summarize this work and discuss some potential future research directions.

2 Related Work

Research in group communication systems has been quite active in the last 15-20 years. Initially, high availability and fault tolerance were the main goals. This resulted in systems like ISIS [7], Horus [8], Transis [9], Totem [10], and RMP [11].

With the increased use of GCSs over insecure open networks, some research interest shifted to securing GCSs. Recall that the core of any GCS is its membership protocol. Some of the work in securing group communication focused on protecting the membership protocol in the presence of byzantine faults. This includes systems such as Rampart [12] and SecureRing [13]. Rampart builds its group multicast over a secure group membership protocol achieved by the means of two-party secure channels. The SecureRing system protects the low-level ring protocol by using digital signatures to authenticate each transmission of the token and each data message received.

In addition to the membership service, GCSs provide reliable ordered message delivery within a group. To make this secure, group members (senders) must be authenticated and confidentiality and integrity of client data must be guaranteed. One notable result in this area is the Horus/Ensemble work at Cornell [14, 15, 16]. In it, data confidentiality is achieved by using a shared group key obtained by means of group key distribution protocols, while authentication is provided by using the popular PGP [17] method. Moreover, the system allows application-dependent trust models in the form of access control lists which are treated as replicated data within a group.

A collaborative application can have its own specific security requirements and its own security policy. The Antigone policy [18] framework allows flexible application-level group security policies in a more relaxed model than the one usually provided by GCSs. Policy flavors addressed by Antigone include: re-keying, membership awareness, process failure and access control.

Unlike the aforementioned systems, our approach focuses on the use of contributory key agreement as a building block for other security services in Spread [3]. We investigated the inter-relation between key agreement and group communication system and designed and implemented the first robust contributory group key agreement protocol. Moreover, we designed several scalable integrated architectures for client-server group communication systems that support strong group communication semantics: Virtual Synchrony and Extended Virtual Synchrony.

3 Spread

The work presented in this paper evolved from integrating security services into the Spread GCS. In this section we provide an overview of Spread's architecture and the group communication models it supports.

3.1 Spread Architecture

Spread [3] is a general-purpose GCS for wide- and local-area networks. It provides reliable and ordered delivery of messages (FIFO, causal, total ordering) as well as a membership service.

The system consists of a server and a client library linked with the application. The client and server memberships follow the model of light-weight and heavy-weight groups [19]. This architecture amortizes the cost of expensive distributed protocols, since these protocols are executed by a relatively small number of servers, as opposed to having all clients participating. A simple join or a leave of a client process translates into a single message, instead of a full-fledged membership change. Only network partitions [1] incur the heavy cost of a full-fledged membership change.

Spread offers a many-to-many communication paradigm where any group member can be both a sender and a receiver. Although designed to support small- to medium-size groups, it can accommodate a large number of collaboration sessions, each spanning the Internet. Spread scales

[1] By a network partition we mean connectivity changes due to networking hardware, routing, or a machine crash.

well with the number of groups used by the application without imposing any overhead on network routers.

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

3.2 Group Communication Services

Spread supports two well-known group communication semantics, Virtual Synchrony (VS) [5, 20] and Extended Virtual Synchrony (EVS) [6, 21] (see [22] for a comprehensive survey of group communication models). The VS service is provided by a client library implemented on top of the EVS semantics.

Both group semantics guarantee that group members see the same set of messages between two sequential group membership events and that the order of messages requested by the application (such as FIFO, Causal, or Total) is preserved. They also guarantee that all messages are delivered in the same view. However, there is a major difference in this last aspect: while VS guarantees that messages are delivered to all recipients in the same view as the sending application thought it was a member of at the time it sent the message (also known as Sending View Delivery), EVS guarantees that messages will be delivered in the same group view to connected members (also known as the Same View Delivery property). Note that, in the EVS case, the delivery view can be different from the sending view.

The VS service is easier to program and understand, while the EVS service is more general and has better performance. The VS service is slower, since it requires application-level acknowledgments for every group change. Moreover, it requires closed groups semantics, allowing only current members of the group to send messages to the group. EVS, in contrast, allows open groups where non-member clients can send to a group.

When securing a GCS providing VS, it is both natural and efficient to use a shared group key (securely refreshed upon each membership change) for data confidentiality. A message is guaranteed to be encrypted, delivered and decrypted in the same group view and, hence, with the same current key. This property does not hold in EVS since a message can be sent in one view and delivered in another, and also because of the open groups support. Therefore, a natural solution for EVS is to use two kinds of shared keys: one shared between the client and the server it connects to, and another – shared among the group of servers. The former is used to protect client-server communication, while the latter – to protect server-server communication.

4 Secure Group Communication Architecture

We now define our security goals, provide a brief overview of the Spread layered architecture and then describe the new integrated architecture and its variants.

4.1 Security Goals

One of our main goals is to protect client data from being eavesdropped by both passive and active adversaries that are not current members of the group. Insider attacks are not relevant for this work since the confidentiality of the data relies on the secrecy of the group key, and any malicious insider can always reveal the group key or its own private key, thus compromising the communication.

The way the group key is computed is essential for the security of the system. A group key agreement protocol should provide: *Key Independence*, *Perfect Forward Secrecy* and *Backward/Forward Secrecy*. Informally, key independence means that a passive adversary who knows any proper subset of group keys cannot discover any future or previous group key. Forward Secrecy guarantees that a passive adversary who knows a subset of old group keys cannot discover subsequent group keys, while Backward Secrecy guarantees that a passive adversary who knows a subset of group keys cannot discover preceding group keys. Perfect Forward Secrecy means that a compromise of a member's long-term key cannot lead to the compromise of any short-term group keys. For a more precise definition of the above terminology, the reader is referred to [23], [24].

The key agreement protocol we use in our design is called Tree-Based Group Diffie-Hellman [25] (TGDH). It provides key independence and perfect forward secrecy; it was also proven secure with respect to passive outside (eavesdropping) adversaries [26]. In addition, active outsider attacks – consisting of injecting, deleting, delaying and modifying protocol messages – that do not aim to cause denial of service are prevented by the combined use of timestamps, unique protocol message identifiers, and sequence numbers which identify the particular protocol execution. Impersonation of group members is prevented by the use of public key signatures: every protocol message is signed by its sender and verified by all receivers. Attacks aiming to cause denial-of-service are not considered.

4.2 Layered Architecture

In previous work we proposed a layered architecture for Spread, focusing on key agreement robustness and correctness. The result is a client library, Secure Spread [4, 27], that provides data confidentiality and integrity. Secure Spread is built on top of the VS Spread client library; it uses

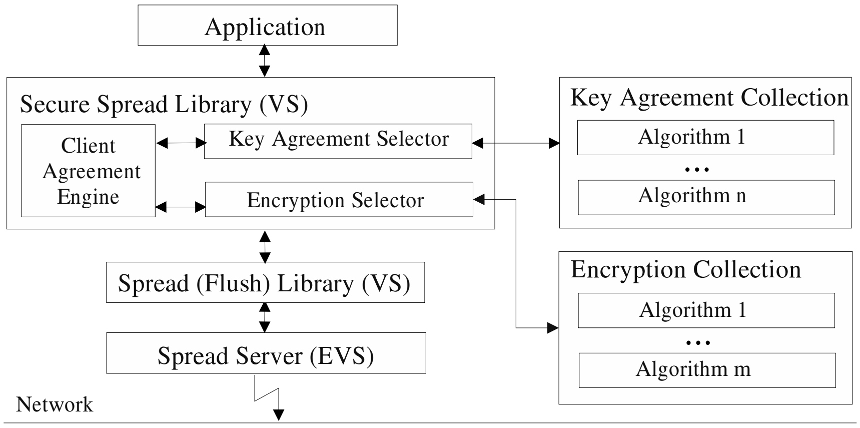


Figure 1. A layered architecture for Spread

Spread as its communication infrastructure and Cliques [28] group key management library primitives for key management implementation. To make the present paper self-contained and facilitate the discussion of different architectures in Section 6, we briefly summarize Secure Spread. For further details, we refer to [4, 27].

Figure 1 presents the layered architecture for Spread. Recall that, the Sending View Delivery property of VS enables the use of a shared view-specific key to encrypt client data, since the receiver is guaranteed to have the same view as the sender and, therefore, the same key. The core of Secure Spread is the Client Agreement Engine that is notified about group membership changes by the GCS. Whenever the group membership changes, the Client Agreement Engine initiates an instance of the group key agreement protocol, ensuring its correct execution. When this protocol terminates, a secure group membership change is delivered to the application and a new group key is ready for use. Applications are not allowed to send any messages while the key agreement protocol is executed.

The computation of a group key is group-specific. A client can be a member of multiple groups, each group managing keys with its own key agreement protocol. A Key Agreement Selector and an Encryption Selector modules are used to identify a group-specific key management and encryption algorithms. Secure Spread currently supports five key management protocols. One of them implements centralized key distribution and is referred to as the Centralized Group Key Distribution (CKD). The other four are key agreement protocols: Burmester-Desmedt (BD) [29], Steer et al. (STR) [25], Group Diffie-Hellman (GDH) [24] and Tree-Based Group Diffie-Hellman (TGDH) [27]. Each of the latter four protocols are based on various group extensions of the well-known (2-party) Diffie-Hellman key exchange [30]. For encryption, only one algorithm (Blowfish [31]) is currently supported.

4.3 Integrated Architecture

Early GCSs were implemented as libraries, which meant that all distributed protocols were performed between all clients. A substantial increase in performance and scalability was obtained by applying a client-server architecture to this model: a smaller number of servers run the expensive distributed protocols and, in turn, serve numerous clients.

Group key agreement protocols are, by nature, distributed and represent the most expensive security building block. Therefore, to improve the performance of the system in settings with multiple groups (or many clients) we amortize the cost of key management by placing the key agreement at the servers and having the servers generating client group keys . This follows the integrated architecture model where security services are implemented at the server.

Since the server population is smaller and more stable than that of clients, server-based key agreement is both faster and less frequent. Specifically, the servers' shared secret key is refreshed only when network connectivity changes, and not when some client group changes. This results in fewer costly key refreshes in contrast to client-based key agreement because network connectivity changes are far less frequent than normal client group changes. Note that the shared server key can be vulnerable if it changes very infrequently and a security policy should impose additional refreshing operations, triggered, for example, by maximum elapsed time between successive key changes (time-out) or maximum volume of data exchanged (data-out).

Generating client group keys is much less costly in the integrated architecture, since, if no change occurs in the servers configuration, the cost of generating a new key for a group amounts to one keyed MAC (HMAC [1]) operation. When network connectivity does change (and so does the membership of the servers' group), the group key shared by

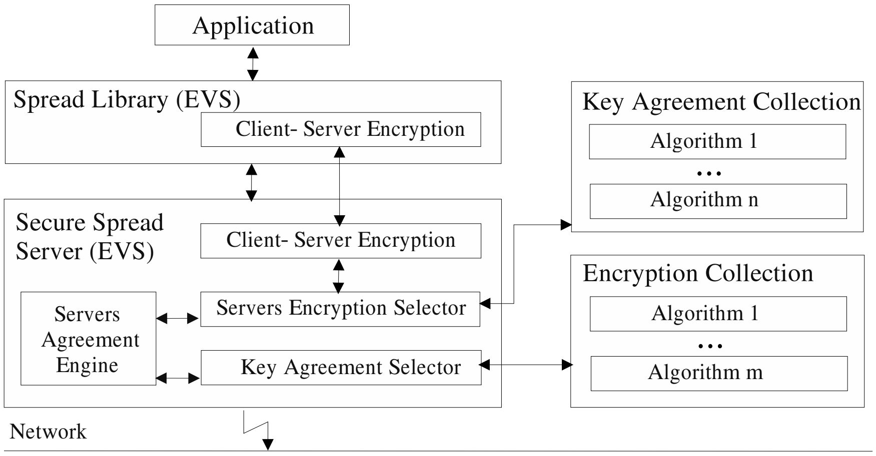


Figure 2. A Three-Step Client-Server architecture for Spread

the servers is refreshed using a full-blown group key agreement protocol. For this, we use the TGDH [25] protocol because of its superior performance.

The use of encryption for bulk data confidentiality results in decreased system throughput due to the extra consumption of CPU resources. Regardless of the location and particulars of the key management, bulk data encryption can be done by either clients or servers. In the following, we describe three integrated architecture variants that trade off encryption cost for complexity and group communication model support. We discuss their different performance and security guarantees and then compare them to the layered approach.

4.3.1 Three-Step Client-Server

We start with an architecture that provides EVS semantics at the expense of decreased (due to encryption) throughput. We refer to it as *Three-Step Client-Server*.

This architecture is based on the client-server model of the group communication system. We distinguish between two communication channels: client-server and intra-servers. A remote client connects to a server using a two-party secure communication protocol, such as SSL/TLS [32]. If a client connects to a server running on the same machine, the architecture uses IPC. In this case, no data protection is needed and client-server communication is not encrypted. The intra-server communication channel is protected by a shared group key multicast encryption protocol that we developed.

Figure 2 presents such an architecture. The Servers Agreement Engine detects changes in the server group connectivity and for each connectivity change performs a key management protocol. In addition, time-based or data-based key refresh can be enforced. As mentioned above, we use the TGDH [25] protocol for key management.

One of the challenges with integrating a key agreement protocol into a GCS is the interactions between the former and the membership protocol. Until the membership protocol completes, the key agreement protocol cannot run, since there is no fixed group of servers among which to perform key agreement. While the membership protocol is running, the set of known servers may change again (referred to as *cascaded membership*), and basic communication services between them may become unavailable.

To cope with this issue, the group key is provided only when the server group membership is stable and while the GCS membership protocol is not executing. This allows the key agreement protocol to run with its normal assumptions once the membership protocol completes, yet prior to notifying the client applications about the change. Thus, applications do not experience any change in semantics or the APIs (such as a new key message) but do experience an additional delay during each server membership change. (This is in order for the key agreement protocol to execute following the completion of the membership protocol.)

The membership protocol can be secured by using public key cryptography to encrypt and sign all membership messages, since the shared key is not available during its execution. The small number of messages sent during the membership algorithm and their small size, ensures that the overhead of public-private encryption can be tolerated.

The Three-Step Client-Server architecture allows individual policies for rekeying the server group key and the per-client SSL keys, as each is handled separately.

Once the master server group key is generated, the servers communication is protected by encryption using a key derived from it. The default protocol to encrypt communication between servers is Blowfish in CBC mode; however, the system supports any encryption algorithm in the OpenSSL [33] library, including AES [34].

The total end-to-end cost of sending an encrypted data

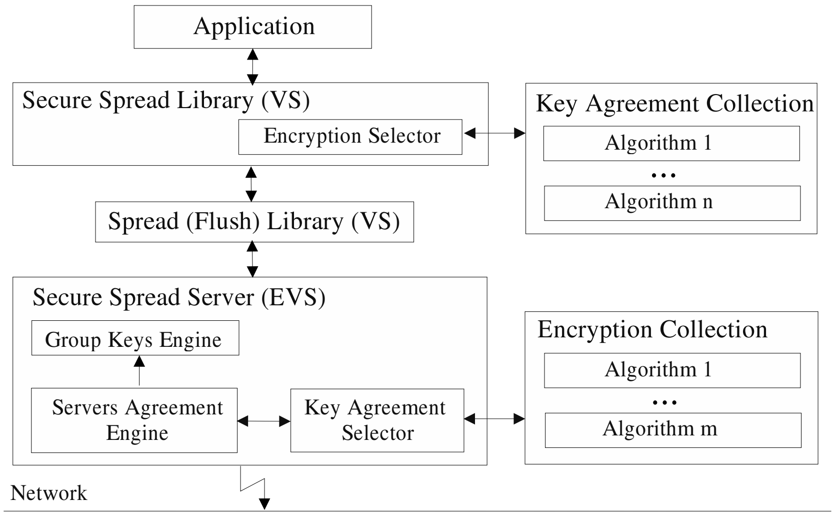


Figure 3. An Integrated VS architecture for Spread

message from one client to another (both are connected to the Spread server remotely) includes six encryption and decryption operations: client encrypts the message and sends it over SSL to the server; server decrypts it and then re-encrypts using the server group key; servers that receive this message decrypt it and then re-encrypt it again using SSL for the receiving client; finally, each receiving client decrypts the message.

Note that the receiving servers need to encrypt the message separately for each remote client who needs to receive it. This is potentially a large number since each server can support about $1,000$ client connections. Thus, if more than one receiver is connected remotely on the same server, the load on the server will increase linearly with each remote receiver, since each remote receiver receives the same message encrypted separately on its own SSL connection. Local receivers do not require client-server encryption.

If two clients (sender and receiver) are executing on the same machine as the server that they connect to, then the cost of encryption under the Three-Step Client Server model reduces to one encryption by the sending server and one decryption by the receiving server.

4.3.2 Integrated VS

The second variant, referred to as *Integrated VS*, supports the VS group communication model and is similar to the layered architecture in that encryption and decryption are performed by clients only. The client groups are closed, i.e., a client needs to be a member of a group in order to send messages to that group. This design requires client groups keys. However, unlike the layered architecture where the key agreement was performed individually by each group, in this case, group keys are generated by servers without involving costly key agreement protocols.

Figure 3 depicts the Integrated VS architecture. The Servers Agreement Engine (SAE) initiates a key agreement protocol between the servers whenever it detects a change in server group connectivity. The Group Keys Engine (GKE) generates, for each group, a shared key whenever group membership changes. In case of a network connectivity change, the SAE is invoked first, followed by the GKE. (The latter refreshes the key for each group that suffered changes in membership due to a change in server connectivity.) The new group key is attached to the membership notification and delivered to the group. Client group keys are generated by the servers based on three values:

1. server group shared key K_s,
2. group name, and
3. unique number that identifies the group view (list of members at a certain time)

The group key for group g in view i uniquely identified by $view\_id(g,i)$ is $K_{g,i} = HMAC(K_s, g\|view\_id(g,i))$.

The shared server group key is computed in a manner identical with the one used by the Three-Step Client-Server architecture and can be refreshed as needed. The client group key is changed whenever a group event (join, leave, etc.) occurs. The new key is delivered within the secure membership message informing the clients about the group change. All client group members receive the same key for the same membership as a result of the VS semantics. If a key change is required because of the security policy (not caused by an underlying group membership change), the key refresh notification is delivered as an "artificial" group membership change. This is needed to preserve the semantic guarantees of VS that messages encrypted by a client with one key will be received by everyone while they also have that same key.

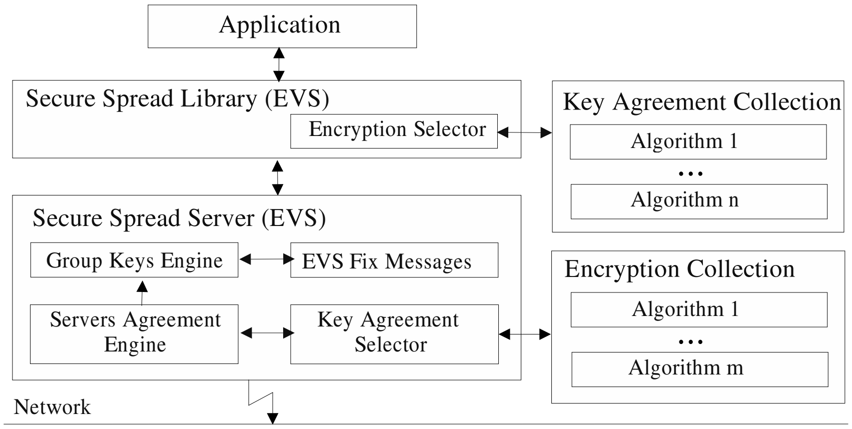


Figure 4. An Optimized EVS architecture for Spread

The encryption and decryption costs for Integrated VS consist of one encryption by the sender and multiple decryptions, one for each receiver. The worse case is when all receivers are situated on the same machine, whereas, the best case is when all receivers are running on distinct machines. In the latter case, decryption takes place in parallel. Once again, Blowfish is the preferred encryption algorithm.

4.3.3 Optimized EVS

Out of the architecture variants presented thus far, only Three-Step Client-Server supports the EVS model and open groups. As discussed in Section 3.2, EVS is faster, thus, it is desirable to have a secure GCS supporting this model. The Three-Step Client-Server serves this purpose, but incurs a heavy encryption overhead when clients connect remotely.

One method to alleviate this large number of encryption operations, is to have clients encrypt with a shared per-view group key. However, EVS does not guarantee that all messages are delivered to receivers in the same view in which they were sent. Therefore, there might be messages that group members will not be able to decrypt as they do not have the key used to encrypt that message in the first place. Our next architecture variant addresses this issue.

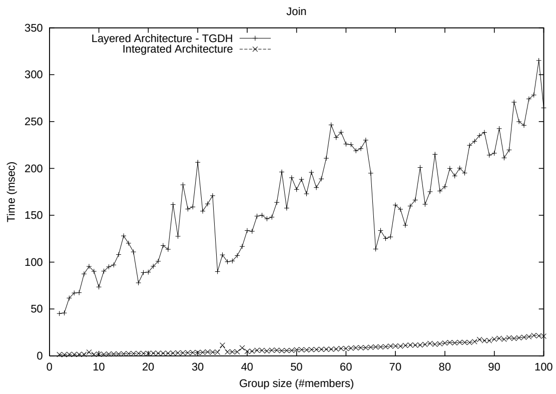
In order to support EVS semantics and client message encryption, we developed an architecture that relies on the servers not only to generate client group keys, but also to "adjust" messages that are not encrypted with the current group key. Figure 4 presents this architecture, referred to as *Optimized EVS*. The Servers Agreement Engine and Group Keys Engine perform key management of the servers' shared secret and client group keys, respectively. The method of generating client group keys is the same as the Integrated VS variant. The main change is that we add a new EVS-Fix-Messages module, that detects when a message for a certain group is encrypted with a key which is no longer valid. Each such message is decrypted and then re-encrypted with the current group key before delivery to the clients. The clients, in turn, decrypt all group messages normally. TGDH is used as the server group key agreement protocol and Blowfish is used for data encryption.

The first problem is addressed by having the sender include in each message a unique Key_id of the group key that was used to encrypt it. This Key_id is independently and randomly computed each time a new key is generated (it is also distributed along with each new client group key). However, since it does not provide integrity, but merely identifies the client group key, the Key_id can be relatively short, e. g., 32 bits. It is transported in the un-encrypted portion of the message header.

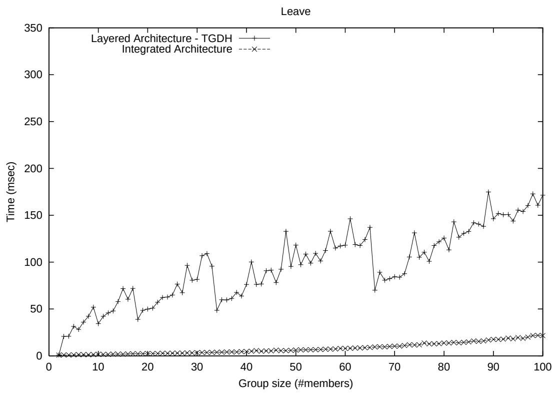
To detect messages encrypted with an "old" key, the server stores each client group along with its Key_id. The server also tags one key as the "current" key for each client group. The current key is the key that matches the last membership (or key refresh) delivered to the group members. Then, before delivering a message to a client, it checks if the Key_id on the message matches that of the current key. If so, the message is immediately delivered. Otherwise, the message is decrypted with the appropriate stored "old" key and re-encrypted under the current key. Since the message stream delivered to each client is a reliable FIFO channel, the client eventually receives the message in the same view that the server expects it to.

Accumulating old keys and Key_ids *ad infinitum* is not a viable solution. Thus, old keys have to be periodically flushed from by each server. Two different expiration metrics can be used either alone or in concert: time-outs and key-outs. A time-out occurs when no message encrypted under a given key has been received for a certain length of time. A key-out takes place when some pre-set maximum number of keys-per-group is exceeded. Many combinations and variations on the theme are clearly possible.

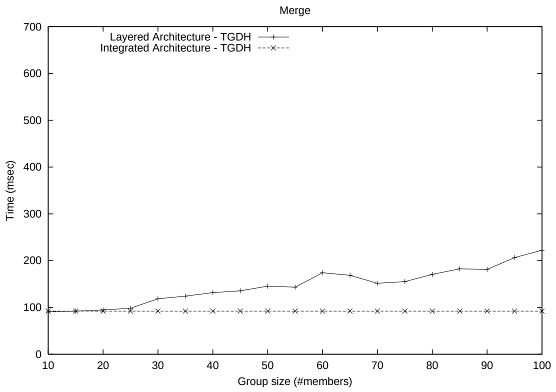
The choice of a key expiration methodology can affect the risk of a message being "undecipherable" even when



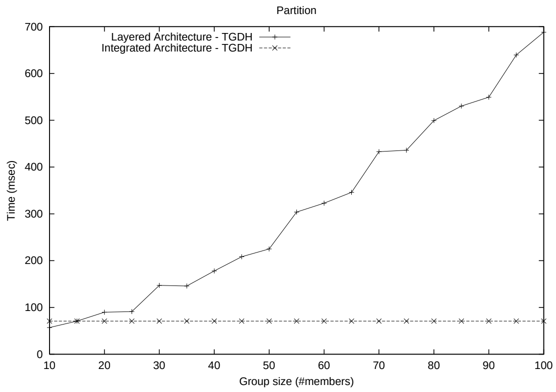
(a) Join



(b) Leave



(c) Merge



(d) Partition

Figure 5. The cost of key agreement - layered architecture vs. integrated architecture

the server, in theory, could have kept the required key.

5 Experimental Results

In this section we present experimental results for the group key management and data encryption building blocks. The experiments cover all architecture variants described in Section 4 measured in a local-area network environment and show the superior scalability of an integrated, over that of a layered, architecture.

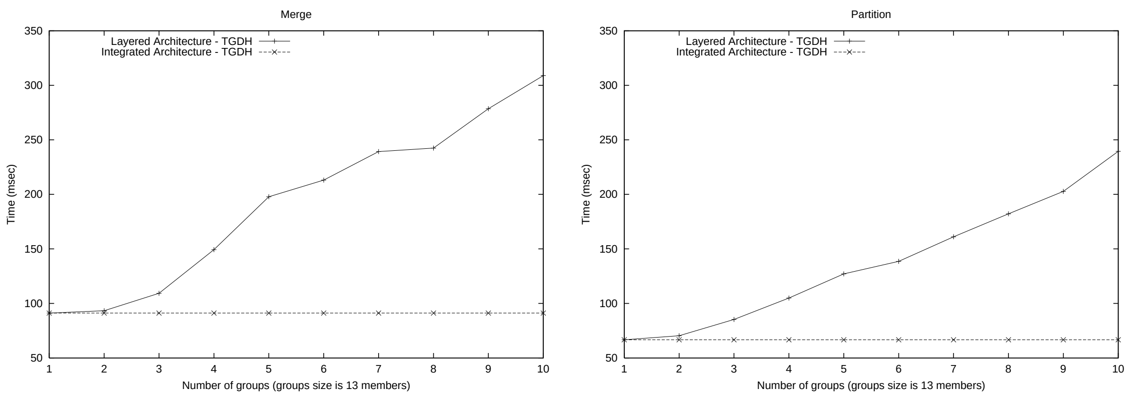
5.1 Group Key Management

We now compare the cost of establishing a shared group key in a layered architecture and in an integrated architecture. For the layered architecture we chose the most efficient key agreement protocol that we have experience with, TGDH [35]. For the integrated architecture we also chose TGDH as a key agreement protocol between the servers.

We used an experimental testbed consisting of a cluster of thirteen 667 MHz Pentium III dual-processor PCs running Linux. Each machine runs a Spread server. Clients are uniformly distributed on the thirteen machines. Therefore, more than one process can be running on a single machine (which is frequent in many collaborative applications).

For the most common group changes, join and leave, the cost of establishing a new group key is reduced to almost the cost of the group communication membership protocol, since the servers can compute a new group key without performing any other key agreement protocol, just one HMAC operation is needed per group change. The results presented in Figure 5(a) and Figure 5(b) for the integrated architecture are for a VS group membership protocol. This is because the cost of the VS group membership protocol is in some sense the worst case: VS uses closed groups and it requires acknowledgments from each group member before changing the group membership. In the EVS case, the numbers for the integrated architecture will be much smaller.

In Figure 5(c) and Figure 5(d) we present the cost of



(a) Merge (b) Partition

Figure 6. Scalability with number of groups

establishing a secure membership for merge and partition. Remember that such a group event is triggered by a network connectivity change which determines a modification in the servers configuration change, or by a server crash. In this case, a new key needs to be computed by the servers, and only then the group keys are computed. In Figure 5(c) and Figure 5(d) we present the cost of establishing a secure group membership for a test scenario where the servers are partitioned in half and then brought back together.

As it can be seen in Figures 5(c) and 5(d) the cost of the key management for the integrated architecture is slightly higher than in the case of join and leave because of the cost of the key agreement protocol performed between servers. However, since the number of servers is much smaller than the number of clients, the impact of the key agreement protocol is less significant. The cost of the secure membership decreases from about 220 milliseconds, to about 90 milliseconds where the size of the group after partition is 100 users, in case of a merge and from about 680 milliseconds to about 60 milliseconds for a partition, where the size of the group before partition is about 100 members.

The above results are for a scenario when only one group exists in the system. In practice, this is not the case. When more than one group exists in the system and a change in the servers' configuration that affects more than one group occurs, the layered architecture performs a key agreement protocol for each of the existing groups affected by the change. For the integrated architecture, there is only one small scale key agreement performed between servers, and then a number of HMAC operations equal with the number of groups affected by the change. Figure 6 shows the average cost of recomputing a shared key for all groups, when more than one group exists in the system. All the groups have the same number of clients 13. We chose this number, because this is also the number of the servers in our configuration. Even in this favorable setup for the layered architecture (small size groups), the integrated architecture scales much better than the layered architecture when the number of groups in the system increases. Based on the results we present in Figure 6 we estimate that even with a very small group size (13 in our case), it will take more than 4 seconds to refresh the key for 200 groups in a layered architecture, while it will take about 50 times less to perform the same operation for an integrated architecture.

5.2 Data Encryption

Another important building block in the architecture of secure group communication is the encryption module. In Figure 7 we present the data throughput for three different setups: the Layered Architecture, the Integrated VS and the Three-Step Client-Server, in a local area network. We consider a scenario where clients connect to servers running locally, so in the Three-Step Client-Server setup, encryption is performed only between servers. As expected, the results for the Integrated VS are similar with the results for the Layered Architecture, because in both models encryption and decryption are performed by the clients.

The results for the Three-Step Client-Server are about 0.66 of the throughput achieved in the other two models. The major reason for this decrease is due to the fact that both headers and data are encrypted and the message delivery protocol employed by our system can not detect if it needs to process a message further or not, without first decrypting it. In addition, since the maximum message size exchanged by the servers is about the size of an Ethernet frame (minus the UDP protocol header), a message of large size that gets encrypted in a client in only one encryption operation, translates into a number of encryption operations in the server. We note that since the encryption operation

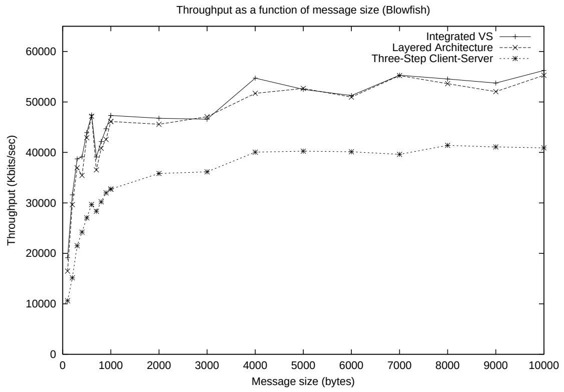


Figure 7. Data throughput

takes place at the data link layer, the servers encrypt not only client data, but also control information, so this model provides a stronger service than the other two models.

This experiment only had one sender and that server was the bottleneck. In cases where several servers are sending messages, this cost will be amortized and the throughput will considerably increase (by a bit less than a factor of 2).

We did not include results for the Three-Step Client Server architecture when clients connect remotely, but from the results in Figure 7 we can extrapolate that the achieved throughput in this case will be much smaller, and therefore unacceptable. The Optimized EVS architecture throughput will be similar to the Integrated VS throughput if no servers membership occurs, and will degrade when membership changes occur, since some messages will need to be decrypted and re-encrypted under new keys. The Three-Step Client-Server architecture performance should be worse in all cases.

Note that the drop in throughput for all of the methods (as seen in Figure 7) at a message size of around 700 bytes is actually a positive thing. Up to 700 bytes, Spread is able to pack multiple messages into one network packet, thus paying less per packet and increasing throughput considerably. Above 700 bytes, that optimization cannot be employed because of the Ethernet maximum packet size.

6 Discussion

The layered and each the new proposed integrated architectures have benefits and limitations. We compare them by investigating the following aspects: trust, encryption overhead, key management overhead, impact of the compromise of the shared secret, complexity and supported group communication model.

Table 1 summarizes the architectures we proposed. The layered architecture has the advantage that no trust is put into anything outside of the end user's control with respect to protecting the client's data. The client needs to trust the servers with respect to the membership service and ordered and reliable delivery, but these are outside the scope of our security goals for this work. The compromise of a group key does not affect the security of the rest of the groups in the system, since each group is running its own protocol and computes its shared key independently of the other groups. In addition, this architecture is less complex and easier to develop, allowing us to explore the inter-relationship between key agreement and group communication [2]. However, this model, due to the security- strong, but expensive key agreement protocols we used, has limited scalability, to no more than 100 members for the best protocol.

All of the integrated architectures we proposed overcome the key management scalability problem by using a secret key shared by the servers, and thus, putting more trust in the servers. This architecture is also appropriate for providing other security services such as client authentication upon connection and access control to perform group-specific operations. A security policy can be easily configured and enforced by an administrator controlling a server configuration file.

The Three-Step Client-Server approach does not use client group keys, but requires a client to share a key with the server it connects to. Although it uses a less complex key management mechanism, this approach is expensive in encryption and decryption operations when clients connect to servers remotely. If clients connect to servers locally, this is the best architecture since theoretically it only requires one encryption/decryption of each message and it can easily protect not only client data, but also the control information exchanged by the servers, something the layered architecture can not provide. Note, that depending on the implementation, even when clients connect locally, more than one encryption/decryption of each message can

[2] For example, we used this architecture to design robust contributory protocols, resilient to any sequence of group changes, possibly cascading.

	Group Keys	Servers Key	Encryption	Group Comm. Model
Secure Spread Library	Client	None	Client-Clients	VS
VS Integrated Architecture	Server	Yes	Client-Clients	VS
Three-Step Client-Server	None	Yes	Client-Server, Server-Server	VS and EVS
Optimized EVS	Server	Yes	Client-Clients mostly	EVS

Table 1. Secure group communication architectures

take place at the sender as discussed in Section 5.2.

Both the VS Integrated solution and the Optimized EVS solution use client group keys generated by servers. Our experimental results in Section 5 show that the scalability is group size is improved substantially with respect to the layered architecture. However, the security of the groups relies on the security of the servers' shared key, which is used in generating the group keys. If the servers' key is compromised, the security of all the groups in the system is compromised, as opposed to the layered model where the compromise of a group key, does not affect the security of the rest of the groups in the system. The encryption overhead is smaller that of the Three-Step Client-Server approach. The VS Integrated approach has the same encryption overhead as the layered architecture. The Optimized EVS solution has almost the same encryption cost as the layered architecture, for the messages not delivered in the membership they were sent in, four additional encryption/decryption operations per message are performed.

Although protecting the control messages exchanged by the servers is out of the scope of this paper, we note that the Three-Step Client-Server solution also provides confidentiality and integrity for the servers' control messages. The rest of the approaches are confidentiality client-driven so they do not. This can be corrected by addressing the control data flow at the servers' level: integrity, confidentiality and, if needed, non-repudiation by signing the messages.

Choosing the most appropriate architecture depends on the desired scalability and trust guarantees. An integrated approach scales better, but the security of all groups relies on one key; a layered architecture scales worse, but the security of a group is independent of the security of the rest of the groups and gives more control to the client.

7 Conclusions and Future Work

This paper presented several secure integrated architectures for client-server GCS, discussing their different performance and security guarantees. The experimental results we present demonstrate the increased scalability of integrated approaches over layered approaches, without a significant decrease in throughput performance.

The work presented in this paper focuses on strong semantics (e.g. membership) and security services that have a significant impact on computational overhead and scalability. Some applications might need in fact weaker service semantics. We intend to explore the security aspects of systems providing communication to groups, but with looser semantics.

8 Acknowledgments

This work was supported by grant F30602-00-2-0526 from the Defense Advanced Research Projects Agency.

We would like to thank to Giuseppe Ateniese and Fabian Monrose for helpful discussions about modes of providing confidentiality and integrity. We thank Yongdae Kim, one of the authors of the TGDH protocol, for discussions that helped us designing an effective integration of TGDH with Spread. We also thank John Schultz, the author of the Flush library, the client library that provides Virtual Synchrony semantics for Spread, for numerous discussions about group communication semantics and his help in designing Secure Spread.

References

[1] *The Keyed-Hash Message Authentication Code (HMAC).* No. FIPS 198, National Institute for Standards and Technology (NIST), 2002. http://csrc.nist.gov/publications/fips/index.html.

[2] Y. Amir, C. Nita-Rotaru, and J. Stanton, "Framework for authentication and access control of client-server group communication systems," in *3rd International Workshop on Networked Group Communication*, (London, UK), November 2001.

[3] Y. Amir and J. Stanton, "The Spread wide area group communication system," Tech. Rep. 98-4, Johns Hopkins University, Center of Networking and Distributed Systems, 1998.

[4] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Schultz, J. Stanton, and G. Tsudik, "Exploring robustness in group key agreement," in *Proceedings of the 21th IEEE International Conference on Distributed Computing Systems,*, pp. 399–408, IEEE Computer Society Press, April 2001.

[5] A. Fekete, N. Lynch, and A. Shvartsman, "Specifying and using a partitionable group communication service," in *Proceedings of the 16th annual ACM Symposium on Principles*

of Distributed Computing, (Santa Barbara, CA), pp. 53–62, August 1997.

[6] 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.

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

[8] 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.

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

[10] 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.

[11] 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.

[12] M. K. Reiter, "Secure agreement protocols: reliable and atomic group multicast in Rampart," in *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pp. 68–80, ACM, November 1994.

[13] 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.

[14] 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.

[15] O. Rodeh, K. Birman, and D. Dolev, "Using AVL trees for fault tolerant group key management," *International Journal on Information Security*, vol. 1, February 2002.

[16] O. Rodeh, K. Birman, and D. Dolev, "The architecture and performance of security protocols in the Ensemble Group Communication System," *ACM Transactions on Information and System Security*, vol. 4, pp. 289–319, August 2001.

[17] P. Zimmermann, *The Official PGP User's Guide*. MIT Press, 1995.

[18] P. McDaniel, A. Prakash, and P. Honeyman, "Antigone: A flexible framework for secure group communication," in *Proceedings of the 8th USENIX Security Symposium*, pp. 99–114, August 1999.

[19] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 784–803, December 1997.

[20] J. Schultz, "Partitionable virtual synchrony using extended virtual synchrony," Master's thesis, Department of Computer Science, Johns Hopkins University, January 2001.

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

[22] G. V. Chockler, I. Keidar, and R. Vitenberg, "Group communication specifications: A comprehensive study," *ACM Computing Surveys*, pp. 427–469, December 2001.

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

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

[25] Y. Kim, A. Perrig, and G. Tsudik, "Communication-efficient group key agreement," in *Proceedings of IFIP SEC 2001*, June 2001.

[26] Y. Kim, A. Perrig, and G. Tsudik, "Tree-based group key agreement," 2002. In Submission.

[27] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik, "On the performance of group key agreement protocols," in *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems,*, (Viena, Austria), June 2002.

[28] Cliques Project team, "Cliques." http://sconce.ics.uci.edu/cliques/.

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

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

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

[32] *The TLS Protocol Version 1.0*. No. RFC2246, T. Dierks and C. Allen, 1999. http://www.faqs.org/rfcs/rfc2246.html.

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

[34] *Advanced Encryption Standard (AES)*. No. FIPS 197, National Institute for Standards and Technology (NIST), 2001. http://csrc.nist.gov/encryption/aes/.

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