Secure Spread: An Integrated Architecture for Secure Group Communication

Yair Amir, *Member, IEEE,* Cristina Nita-Rotaru, *Member, IEEE,* Jonathan Stanton, *Member, IEEE,* and Gene Tsudik, *Member, IEEE*

***Abstract*— Group communication systems are high-availability distributed systems providing reliable and ordered message delivery as well as a membership service, to group-oriented applications. Many such systems are built using a distributed client-server architecture where a relatively small set of servers provide service to numerous clients.**

In this work, we show how group communication systems can be enhanced with security services without sacrificing robustness and performance. More specifically, we propose several integrated security architectures for distributed 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 performance and accompanying trust issues of each proposed architecture and present experimental results that demonstrate the superior scalability of an integrated architecture.

***Index Terms*— Group Key Management, Secure Communication, Peer Groups, Group Communication**

I. Introduction

UBIQUITOUS information access and communication have become essential to everyday life, global business, and national security. Many activities, including personal, commercial and international financial transactions, studying and teaching, shopping, or managing modern battlefields have fundamentally changed over the last decade as a result of the expanding capabilities of computers and networks. Most such activities are supported by distributed applications which, in turn, increasingly rely on messaging systems to provide secure and uninterrupted service within acceptable throughput and latency parameters. This is difficult to guarantee in a complex network environment that is susceptible to a multitude of human and/or electronic threats, especially, as network attacks have become more sophisticated and harder to contain.

A distributed messaging system is essentially an abstraction layer built on top of an underlying network. It provides distributed applications with: (1) services not available from the native network, e.g., security, ordered message delivery, or (2) services that are enhanced, e.g., higher availability, improved reliable delivery. Group communication systems, overlay networks, and middleware are all examples of messaging systems serving as infrastructure for applications, such as: web clusters, replicated databases, scalable chat services and streaming video.

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

A preliminary version of this article was presented, in part, at DISCEX III [1].

Since many applications are expected to run over the Internet, security becomes a real necessity. To this end, the research community has invested significant effort in investigating and developing efficient security services. Numerous algorithms, protocols, frameworks and policy languages have been developed to provide security services in point-to-point or group-based communication models. However, there has not been enough research into the integration of security techniques into distributed systems, while maintaining a reasonable level of performance. This work tries to fill this gap, by showing how high-availability systems (such as group communication systems) can be enhanced with security services without sacrificing robustness and performance.

A. Group Communication Systems

Group communication systems (GCS) are distributed messaging systems that enable efficient communication between a set of processes logically organized in groups and communicating via multicast in an asynchronous environment where failures can occur. Examples of group-oriented applications that can take advantage of the services provided by a GCS include: peer groups of long-running servers, conferencing, distributed logging and mobile state transfer.

A GCS provides a group membership service as well as reliable and ordered message delivery. The membership service informs all members of a group about the list of currently connected and alive group members (often referred to as a *view*), and notifies group members about every group change. A group can change for several reasons. In an idealized fault-free setting, a change can be caused by members voluntarily joining or leaving the group. In a more realistic environment, faults can occur, e.g., processes can become disconnected or simply crash and network partitions can prevent members from communicating. When faults are healed, group members can communicate again. These events can also trigger corresponding changes in group membership.

The core of GCS lies in achieving agreement between multiple participants about group membership views and about the order of messages to be delivered. Many agreement protocols were proved to have no solution in asynchronous systems with failures [2]. GCS's overcome the problem by using time-out based failure detection to sense network (dis-)connectivity and process faults. One risk of this approach is that alive and connected members communicating over high-delay links, may be excluded from the group membership. However, if the network is stable, GCS membership reflects the current list of connected and alive group members.

The membership and message delivery services were formalized in two models: Virtual Synchrony [3] (VS) and Extended Virtual Synchrony [4], [5] (EVS). The main difference between the two models lies in the relation between the views in which messages are sent and delivered. Essentially, both models guarantee that all group members see the same set of messages between two sequential group views and that the order of messages requested by the application is preserved. The major difference is that 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). Thus, in EVS, the delivery view can be different from the sending view, while in VS the sending view and the delivery view are the same.

GCS's have been built around a number of different architectural models, such as, peer-to-peer libraries [6], [7], 2- or 3-level middleware hierarchies [8], [9], modular protocol stacks [10], [11], and client-server [12]. To improve performance, modern GCS's use a client- server architecture where expensive distributed protocols are run between a set of servers, where each server provides services to multiple clients. In this architecture, the client membership service is implemented as a "light-weight" layer that communicates with a "heavy-weight" layer asynchronously using a FIFO buffer.

B. Security Services for Group Communication Systems

Security is crucial for distributed and collaborative applications that operate in a dynamic network environment and communicate over insecure networks, such as the Internet. Basic security services needed in such a dynamic peer group setting are largely the same as in point-to-point communication. 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 resources that can be controlled by access control methods are: joining a group and sending or receiving messages to a group.
- *Integrity and data source authentication*: protect the contents of the communication from being modified by an outsider. Data source authentication guarantees that the message was generated by a trusted source and protects against injections. Efficient integrity and data authentication mechanisms (such as. HMAC [13]) require a shared key between participants. For many protocols data integrity and authentication is an essential service.
- *Confidentiality*: protect the contents of communication from passive eavesdroppers. Symmetric encryption algorithms (such as AES [14]) require participants to share a secret key.

Data integrity, source authentication, and confidentiality, can be efficiently provided if group members share a secret group key. We refer to protocols that generate and maintain a shared group key as *group key management* protocols. Desired properties for key management protocols are *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 [15], [16].

There are two basic architectural approaches to providing security services in a client-server GCS. The first approach (referred to as as the *layered architecture*) places security services in a client library layered on top of the GCS client library. The second approach (referred to as the *integrated architecture*) entails housing some (or all) security services at the servers. Potentially an integrated architecture can provide a more scalable design because it can amortize the cost of security services over several groups.

C. New Contributions

The main goal of this work is to investigate scalable solutions for securing GCS's that do not result in the severe degradation of performance and preserve the fault-tolerance properties. In particular, we focus on securing Spread [12], a GCS resilient to process crashes and network partitions. We propose scalable and efficient secure architectures for Spread, focusing on providing authentication, data confidentiality and data integrity. More specifically, our contributions are:

- **Improved scalability of group key generation**: Contributory key agreement protocols provide strong security properties, which make them appealing for secure group communication. However, when used in a layered architecture, they scale poorly. We show that by using an integrated approach in a light-weight/heavy-weight [17] group architecture, we can improve the performance of key regeneration, substantially reducing the cost for process group join and process group leave, the most common group change operations.
- **Group confidentiality support for EVS semantics:** We discuss the relationship between group communication semantics and group confidentiality. Providing confidentiality in systems supporting the VS model is an easier task (than in EVS) since the model provides a form of synchronization between the group membership and data message delivery. The task is more challenging in systems supporting the EVS model. However, since such systems have better performance it is desirable to provide solutions for them as well.
- **Experimental evaluation and comparison of secure group architectures:** We proposed three variants of scalable integrated architectures for Spread, supporting both

VS and EVS semantics. We discuss the accompanying trust issues and present experimental results that offer insights into their scalability and practicality.

Roadmap: The rest of the article is organized as follows. We survey notable prior work in Section II. We then describe Spread and the group communication semantics it supports. Next, we specify our security assumptions. We overview a layered architecture design in Section III and propose three variants of the integrated security architecture for Spread in Section V. We demonstrate and discuss the improved scalability of our integrated architecture in Sections VI and VII, respectively. Finally, we summarize our work and discuss potential future research directions.

II. Related Work

RESEARCH in GCS's operating in a local area network (LAN) environment 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 [6], Transis [18], Horus [10], Totem [19], and RMP [20]. These systems explored several different models of group communication such as VS [3] and EVS [4]. More recent work in this area focuses on scaling group membership to wide area networks (WAN) [21], [22].

With the increased use of GCS's over insecure open networks, some research interests shifted to securing these systems. Research on securing group communication is fairly new. The only implemented GCS's that focus on security (in addition to ours) are: the SecureRing [23] system at UCSB, the Horus/Ensemble systems at Cornell [24], [25], and the Rampart system at AT&T [26].

At the core of any GCS is a 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 [26] and SecureRing [23]. Rampart builds its group multicast over a secure group membership protocol achieved via two-party secure channels. SecureRing protects its low-level ring protocol by using digital signatures to authenticate each token transmission and each data message received. Both systems exhibit limited performance since they use relatively costly protocols and make extensive use of public key cryptography.

In addition to the membership service, GCS's provide reliable ordered message delivery within a group. To secure this service, group members (senders) must be authenticated and both confidentiality and integrity of client data must be guaranteed. One notable work in this area is the Horus/Ensemble work at Cornell [24], [25]. Ensemble achieves data confidentiality by using a shared group key obtained via group key distribution protocols. The key management protocols in Ensemble provide backward and forward secrecy. For authentication, Ensemble uses the popular PGP [27] method. In addition, the system allows application-dependent trust models in the form of access control lists which are treated as replicated data within a group. Recent research on Bimodal-Multicast, Gossip-based protocols [28] and the Spinglass system has largely focused on increasing scalability and stability of reliable group communication services in more hostile environments – such as wide-area and lossy networks – by providing probabilistic guarantees about delivery, reliability, and membership.

Other approaches focus on building highly configurable dynamic distributed protocols. Cactus [29] is a framework that allows the implementation of configurable protocols as composition of micro-protocols. Survivability of the security services is enhanced by using redundancy. For example, in [30], redundancy of data confidentiality is obtained by encrypting data multiple times, each time using a different encryption algorithm. This approach is not appropriate for applications where throughput is a concern.

Another toolkit that can be used to build secure group oriented applications is Enclaves [31]. It provides group control and communication (both point-to-point and multicast) and data confidentiality using a shared key. The group utilizes a centralized key distribution scheme where a member of the group (group leader) selects a new key every time the group changes and securely distributes it to all members of the group. The main drawback of this system is that it does not address failure recovery when the leader of the group fails.

A collaborative application can have its own specific security requirements and its own security policy. The Antigone policy [32] framework allows flexible application-level group security policies in a more relaxed model than the one usually provided by GCS's. Policy flavors addressed by Antigone include: re-keying, membership awareness, process failure and access control. The system implements group rekeying mechanisms in two flavors: session rekeying - all group members receive a new key, and session key distribution - the session leader transmits an existing session key. Both schemes present some problems: distributing the same key when the group changes violates perfect forward secrecy, while the session rekeying mechanism – although able to detect the leader's failure – does not attempt to recover from it.

Unlike aforementioned systems, we focus on using contributory group key agreement as a building block for other security services. Contributory key agreement protocols provide strong security properties such as backward/forward secrecy and perfect forward secrecy [1]. Our work investigates trade-offs between security and group communication semantics support.

III. Spread

THE work presented in this article evolved from integrating security services into Spread. In this section we present an overview of the group communication semantics supported by Spread and describe its architecture.

Spread [12] 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. A client can obtain access to the group services by connecting to a server. Any process, client or server, can fail. If a server fails, all clients connected to that

[1]We note that contributory key agreement protocols do not provide key independence when they use static keys.

server are also considered failed. When a network partition [2] takes place, Spread servers detect it and continue to provide operation within each connected component. The client and server memberships follow the model of light-weight and heavy-weight groups [33]. This architecture amortizes the cost of expensive distributed protocols, since such protocols are executed only by a relatively small number of servers (as opposed to all clients). This way, 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 incur the heavy cost of a full-fledged membership change.

In Spread any group member can be both a sender and a receiver. A client can be a member of many groups. Spread supports a large number of small- to medium-size groups.

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.

Spread supports two well-known group communication semantics, VS [17], [5] and EVS [4], [34]. (See [35] for a comprehensive survey of group communication models). The VS service is provided by a client library implemented on top of the EVS semantics. The two models define the relationship between group views and message delivery. Both group semantics guarantee that all group members see the same set of messages between two sequential group changes and that the order of messages requested by the application (such as FIFO, Causal, or Total) is preserved. The major difference is that 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 EVS, the delivery view can be different from the sending view.

The VS semantic is easier to program and understand, while the EVS semantic is more general and has better performance. VS is slower, since it requires application-level acknowledgments for every group change in order to guarantee Sending View Delivery. Moreover, it requires closed groups semantics, allowing only current members of a group to send messages to that 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 per view (securely refreshed upon each membership change to preserve key independence) for data confidentiality. In VS, 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, because a message can be sent in one view and delivered in another, and because of the support for open groups. One possible 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. We discuss in more details security architectures for Spread supporting VS and EVS in Section V.

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

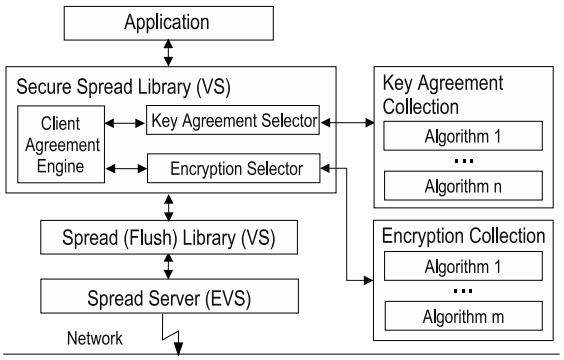


Fig. 1. A Layered Architecture for Spread

IV. Security Assumptions

Our goals are protecting client data from eavesdropping by passive adversaries and preventing impersonation and data modification/fabrication attacks by active adversaries. An adversary is thus anyone who is not a current group member.

We do not consider insider attacks in this work. We acknowledge that such threats are significant; some of our ongoing work focuses on this direction. However, in this article we assume that each entity can be directly authenticated and each has an X.509v3 public key certificate that allows it to sign messages.

The method of computing the group key is essential for the security of the system. An ideal group key management protocol should provide: *key independence*, *perfect forward secrecy* and *backward/forward secrecy*. The key agreement protocol used in our design is the so-called Tree-Based Group Diffie-Hellman [36] (TGDH) protocol. It provides key independence and perfect forward secrecy; it was also proven secure with respect to passive outside (eavesdropping) adversaries [36]. 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.)

V. Secure Group Communication Architecture

IN this section we provide a brief overview of the Spread layered architecture and then describe the new integrated architecture and its variants.

A. Layered Architecture

Our previous work proposed a layered architecture for Spread, focusing on robustness and correctness of group key agreement. The result is a client library [37] that provides data confidentiality and integrity. The library is built on top of the VS Spread client library and it uses Spread [38] as its communication infrastructure and Cliques [39] group key management library primitives for group key management. To make the present article self-contained and facilitate the discussion of different architectures in Section VII, we briefly summarize the layered architecture. For further details, including complete formal specifications of the VS semantic model and correctness proofs that the layered architecture maintains the VS semantics, we refer to [37].

Figure 1 presents the layered architecture for Spread. The library has as main functionalities providing confidentiality of the data by encrypting/decrypting client data using a group shared key and managing the shared key for each group in the system. A client that desires to communicate securely is required to connect to a server and then join a group before proceeding with the communication. The library provides an API interface very similar with the Spread interface allowing a client to connect/disconnect to a server, to join and leave a group, and to send and receive messages.

At the core of the Secure Spread library is the Client Agreement Engine (CAE) which operates as follows: upon every group membership change, the CAE receives notifications from the membership service about the change. Then, the CAE initiates the refresh group key by starting an instance of the group key agreement protocol and ensuring its correct execution (making sure that the messages are sent to the correct destinations in the right order, and that all the members make consistent decisions with respect to installing the new secure membership). When this protocol terminates, a secure group membership change is delivered to the application and the new group key is ready for use. Applications are not allowed to send any messages while the key agreement protocol is executed. In addition, the library ensures that the VS semantics are preserved (see [37] for formal proofs).

A client can be a member of multiple groups, each group managing its shared key 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. The CAE is the one that manages the key agreement protocol for each group.

The layered architecture currently supports five key management protocols. One of them implements centralized key distribution and is referred to as the Centralized Group Key Distribution (CKD) protocol. It is adapted to provide the same security properties as the other four key agreement protocols. The other four are key agreement protocols: Burmester-Desmedt (BD) [40], Steer et al. (STR) [41], Group Diffie-Hellman (GDH) [16] and Tree-Based Group Diffie-Hellman (TGDH) [36]. Each of the latter four protocols are based on various group extensions of the well-known (2-party) Diffie-Hellman key exchange [42] and provide similar security properties: key independence and perfect forward secrecy.

B. Integrated Architecture

Early GCS's were implemented as libraries, which means that all distributed protocols were performed between all clients, per group. 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 propose to amortize the cost of key management by placing the key agreement protocols at the servers and having the servers generate client group keys in a "light-weight" manner. 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. 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 elapsed time between successive key changes (time-out) or 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, our scheme reduces the cost of generating a new key for a group amounts to one keyed MAC (HMAC [13]) operation. When network connectivity does change (and so does the membership of the servers' group), the group key shared by the servers is refreshed using a full-blown group key agreement protocol. For this, we use the TGDH [36] protocol because of its good performance and strong security properties.

The use of encryption for bulk data confidentiality results in decreased throughput due to the extra consumption of CPU resources. Regardless of the location and particulars of the key management, data encryption can be done by either clients or servers. Below we describe three integrated architecture variants that trade off encryption cost for complexity, overhead and group communication model support.

1) Three-Step Client-Server: The most intuitive architecture is one derived from the the client-server model of the group communication system. The architecture can support both VS and EVS semantics at the expense of decreased (due to encryption) throughput. We refer to it as *Three-Step Client-Server*.

The communication taking place in the system can be classified in two logical channels: client-server and intra-servers. The goal is to protect these two channels. Spread's architecture uses a TCP connection when a client connects remotely to a server. In this case, the best approach to protect the client-server communication is using a standard two-party secure communication protocol, such as SSL/TLS [43]. If a client connects to a server running on the same machine,

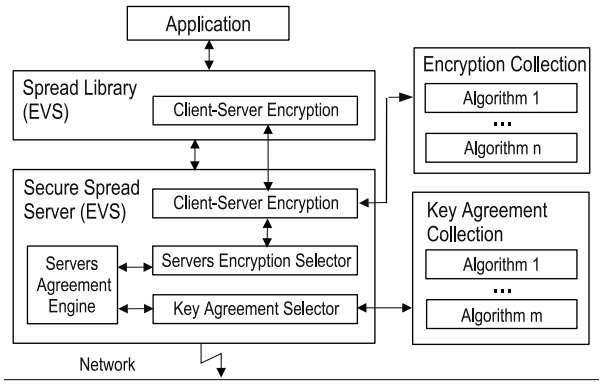


Fig. 2. A Three-Step Client-Server architecture for Spread

Spread architecture uses IPC. In this case, no data protection is needed and client-server communication is not encrypted.

The intra-server communication channel is provided by a multicast protocol developed on top of UDP. Using a protocol like SSL/TLS for confidentiality, integrity and authentication is extremely expensive in this case since SSL/TLS is a two-party communication protocol and will require a number of encryption operations scaling linearly with the number of servers. In this case, the desired security services are provided using a block cipher encryption scheme for confidentiality and the HMAC [13] algorithm for integrity and authentications, both based on a key shared by the servers.

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 between the servers. In addition, time-based or data-based key refresh can be enforced. As mentioned above, we use the TGDH [36] protocol for key management.

One of the challenges with integrating a key agreement protocol into a group communication system 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 servers' group membership is stable and while the group communication 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 servers' membership protocol is 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 [44]; however, the system supports any encryption algorithm in the OpenSSL [45] library, including AES [14], while integrity and authentication are performed using HMAC-SHA1 [13]. Two different shared keys are derived, one used for encryption and one for the HMAC computation. The system can also be configured to use only HMAC and no encryption.

The total end-to-end cost of sending an encrypted data message from one client to another (both are connected to a 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.

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 that 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. We note that several solutions can be defined to decrease the number of encryption operations, particularly for the server that needs to decrypt and re-encrypt all the messages under the SSL client pair-wise keys. We discuss them in more details in Section VII.

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.

The described architecture maintains all of the traditional GCS membership semantics (such as VS and EVS). The Spread membership protocol is unchanged from the non-secure version except for encrypting and signing each membership message using public-private keys which does not change the content or pattern of message exchange. The new intra-server key agreement executes after the regular membership has completed, but before any new client messages are introduced so it adds some delay to the membership process the client experiences but does not allow any reordering of messages. If a new membership view change occurs before the intra-server key agreement is complete then the key agreement is abandoned and restarted once a new stable membership view has been established. Thus clients will see the same set of messages in the same order as they would have in the non-secure Spread protocol.

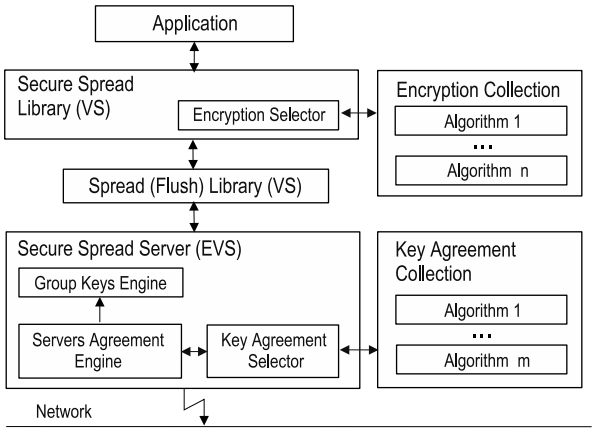


Fig. 3. An Integrated VS architecture for Spread

2) Integrated VS: Although the Three-Step Client-Server architecture presented above is relatively simple, it suffers from decreased throughput due to the cost of the encryption operations performed by the servers. Therefore, although less complex, it is not recommended when clients connect remotely since we aim to design an architecture with reasonable performance, not only in key management, but also in throughput. This can be achieved if encryption is pushed to the clients, which, in turn, requires client group keys.

We now describe a second variant of our architecture, referred to as *Integrated VS*. It supports the VS group communication model and combines the advantage of a less expensive key management building block (by integrating it in the servers) with the advantage of encryption done in the client library. In this aspect, Integrated VS is similar to the layered architecture. 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. As mentioned above, this requires client groups keys. However, unlike the layered architecture where key agreement was performed by each group, in this case, client group keys are generated by servers, without involving costly key agreement protocols. Since the library operates in the VS model, in a manner similar to the layered architecture (see Section V-A), a per-view shared key associated with the group can be used to provide confidentiality. The key is refreshed by the servers when the group view changes.

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 the 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 (unique within the system), and 3) unique number that identifies the group view at a certain time [3].

The group key for group g in view v, where v is uniquely identified by $view\_id_{gv}$ is

$$K_{gv} = HMAC(K_s, g||view\_id_{gv})$$

The shared server group key is computed in a manner identical to that in 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 any group membership change), the key refresh notification is delivered as an "artificial" group membership change. This is in order to preserve the semantic guarantees of VS stipulating that messages encrypted by a sender with a given key must be received by everyone while they also perceive (have) the same key as their current key. One of the servers acting as a leader on behalf of all the servers, can send a signal to 'refresh' the key, sent with the appropriate (SAFE) ordering service to ensure that these "extra" membership view are delivered at the same point in the message stream to all clients.

In this architecture the sequence of messages and membership notifications is identical to those seen by identical clients running the non-secure Spread with the exception of the additional membership notifications described in the previous paragraph. The VS model allows these "extra" membership view notifications as long as they are delivered at the same point in the message stream to all clients. In our case, this is ensured by the SAFE ordering service. In addition, note that one of the requirements for preserving VS, is that clients accept not the send new messages while the membership is changing; therefore, no de-synchronization between encrypted messages and group keys can occur. The server membership protocol and key agreement is the same as in the Three-Step Client-Server architecture and thus preserves the formal GCS semantics in the same way.

Encryption costs for Integrated VS consist of one encryption by the sender and multiple decryptions, one for each receiver. The worst 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 operations take place in parallel.

3) Optimized EVS: Out of the variants presented thus far, only Three-Step Client-Server supports the EVS model and open groups. As discussed in Section I-A, EVS is faster, thus, it is desirable to have a secure group communication system supporting this model. The Three-Step Client-Server serves this purpose, but incurs heavy encryption overhead when clients connect remotely to servers.

One way to alleviate the large number of encryption operations is to have clients perform encryption by using a shared

[3]This number is generated based on a timestamp, the identifier of the servers' representative, and a counter that is incremented every time the group changes

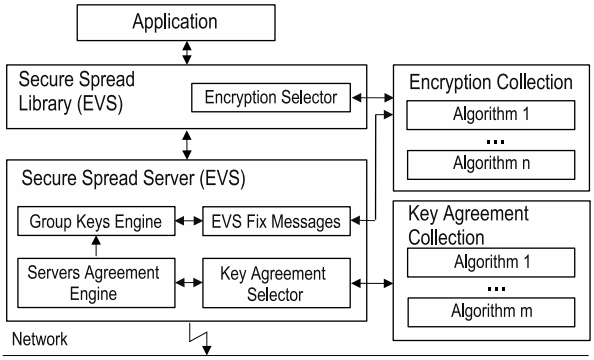


Fig. 4. An Optimized EVS Architecture for Spread

per-view group key, in a manner similar to the Integrated VS architecture. However, unlike VS, 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 be unable to decrypt as they do not have the key used to encrypt that message in the first place. Our next variant addresses this issue.

In order to support EVS semantics and client message encryption, we developed an architecture that relies on servers not only to generate client group keys, but also to "adjust" messages that are not encrypted with the current group key. Clients operate without any disruption since servers guarantee that all messages delivered to the clients are encrypted with the current group key.

Figure 4 presents this variant, 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 in Integrated VS. The main change is the addition of the EVS-Fix-Messages module, that detects when a message for a certain group is encrypted with a key that is no longer valid. Each such message is decrypted and re-encrypted with the current group key before being delivered to the clients. Clients, in turn, decrypt all group messages normally. TGDH is used as the server group key agreement protocol.

The EVS-Fix-Messages module addresses two problems: it detects whenever a message is encrypted with the wrong key and determines the correct key to use for encrypting the message. 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). Since it does not provide integrity, but merely identifies the client group key, Key_id can be relatively short, e.g., 64 bits. It is transported in the un-encrypted portion of the message header.

To detect messages encrypted with an "old" key, a server stores each client group along with its Key_id. Each 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.

Thus, the sequence of messages and views delivered by the server to each client is identical to the sequence in the non-secure Spread case, so the Optimized EVS architecture still maintains all of the typical EVS semantics. The only new risk is that a message could be delivered in the correct order, but be encrypted with the wrong key and thus not be readable by the client. This can occur only if there is a collision in the Key_id field, so that two different keys generate identical Key_id's, or, as discussed in the next paragraph, if the server does not store the old keys for a long enough time. Both of these risks can be managed to be arbitrarily small at the cost of some resources.

Accumulating old keys and Key_ids *ad infinitum* is clearly not viable. Thus, old keys have to be periodically flushed by each server. Different expiration metrics can be used either by each server individually 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 "indecipherable" even when the server, in theory, could have kept the required key.

VI. 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 V measured in a local-area and wide-area network environments.

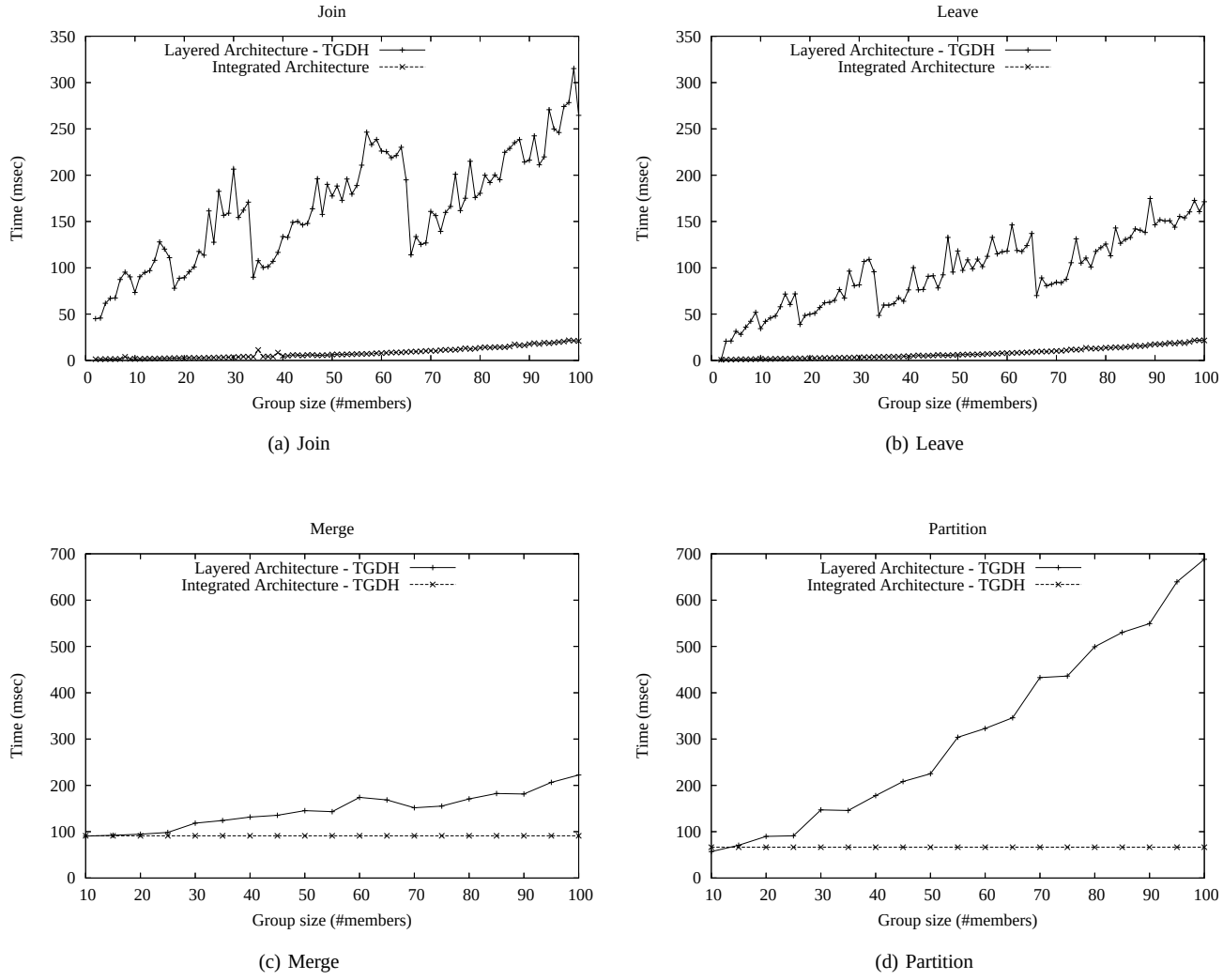
A. Group Key Management

We first compare the cost of establishing a shared group key in a layered architecture and in an integrated architecture. To ensure a fair comparison we use the same key agreement protocol in both architectures, TGDH [36]. The communication and computation costs of the TGDH protocol are summarized in Table I, where h denotes the height of the tree built by TGDH. More details about why TGDH is our protocol of choice can be found in [46].

We used an experimental testbed consisting of a cluster of 13 667 MHz Pentium III dual-processor PCs running Linux. Each machine runs a Spread server. Clients are uniformly distributed on the machines. Therefore, more than one process can be running on a single machine (which is frequent in many collaborative applications). We present results both in local and wide area network. For the WAN experiments, machines were distributed at three sites: Johns Hopkins University (JHU), Maryland, University of California at Irvine (UCI) and Information and Communications University (ICU), Korea.

TABLE I
TGDH COMMUNICATION AND COMPUTATION COST

Event	Rounds	Messages	Unicast	Multicast	Exponentiations	Signatures	Verifications
Join, merge	2	3	0	3	$3h/2$	2	3
Leave	1	1	0	1	$3h/2$	1	1
Partition	h	$2h$	0	$2h$	$3h$	h	h



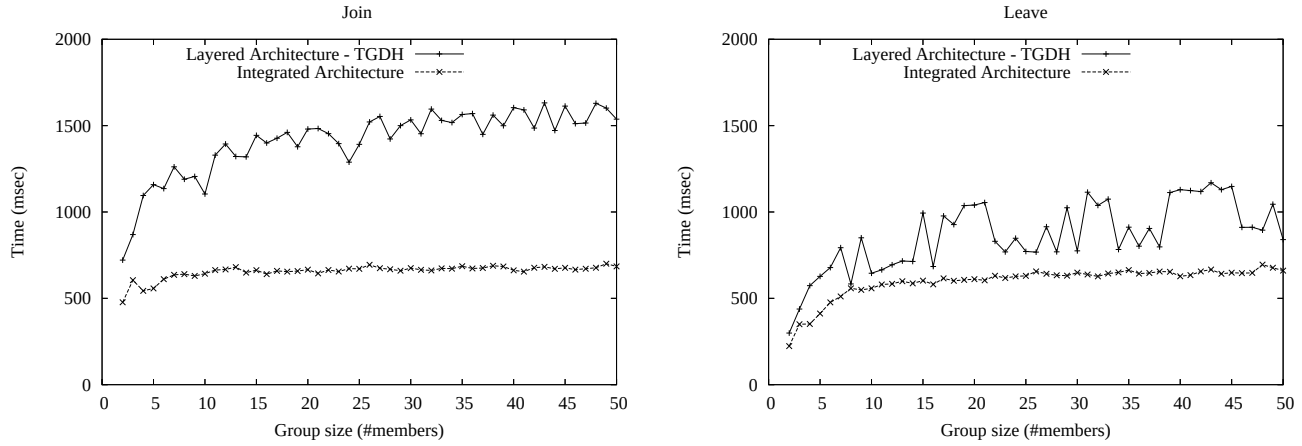
(a) Join (b) Leave

(c) Merge (d) Partition

Fig. 5. The cost of key agreement in LAN - layered architecture vs. integrated architecture

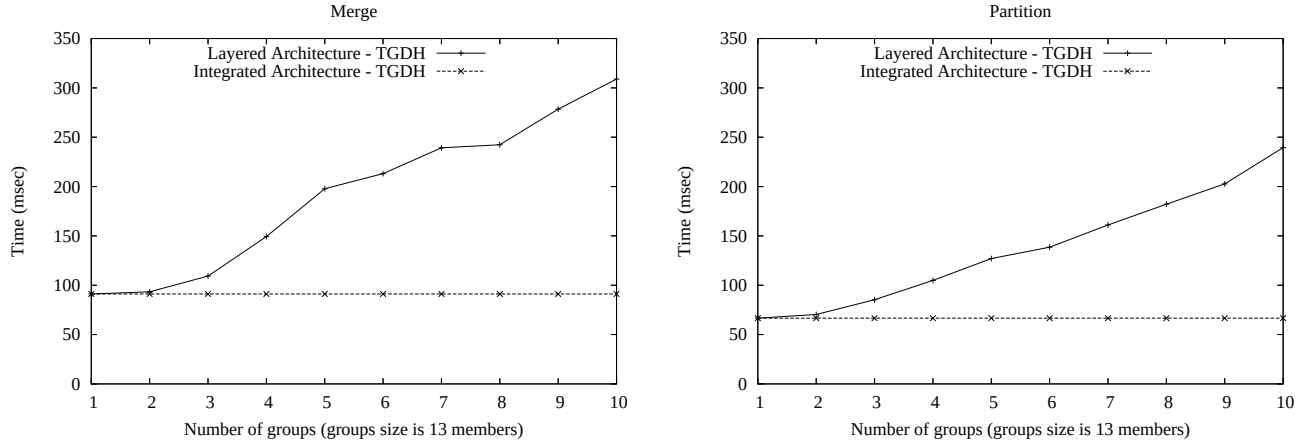
For the most common group changes, join and leave, the cost of establishing a new group key is reduced to almost the cost of the group communication membership protocol, since the servers can compute a new group key without performing any other key agreement protocol, just one HMAC operation is needed per group change. The results for the experiments performed in a LAN setting, for join and leave are presented in Figure 5(a) and Figure 5(b). The results for the integrated architecture are for a VS group membership protocol. This is because the cost of the VS group membership protocol represents 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. The saw tooth aspect of the TGDH protocol is due to the heuristics used by TGDH to balance the tree. New members are always added to the right-most leave as long as they do not increase the height of the tree. In this case, the new member will be added to the root and the cost of refreshing the key will be minimal (this corresponds to the drop in the saw tooth). While the height increases, the cost of refreshing the key also increases, corresponding on an ascending slope on the graph.

Results for join and leave in a WAN environment are presented in Figure 6. In this case the predominant cost is the



(a) Join (b) Leave

Fig. 6. The cost of key agreement in WAN - layered architecture vs. integrated architecture



(a) Merge (b) Partition

Fig. 7. The cost of key agreement in LAN - multiple groups

communication cost, and over high-delay networks like the one we use for our experiments, extra communication rounds can degrade the scalability significantly.

In Figure 5(c) and Figure 5(d) we present the cost of establishing a secure membership for merge and partition, also in a LAN environment. Such a group event is triggered by a network connectivity change which requires a modification in the set of reachable servers, 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. For example, in the case of a LAN, the cost of the secure membership merge decreases from about 220 milliseconds, to about 90 milliseconds where the size of the group after partition is 100 users, 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 (smaller scale) key agreement performed between servers, and then a number of HMAC operations equal with the number of groups affected by the change. Figure 7 shows the average cost of recomputing a shared key for all groups, when more than one group exists

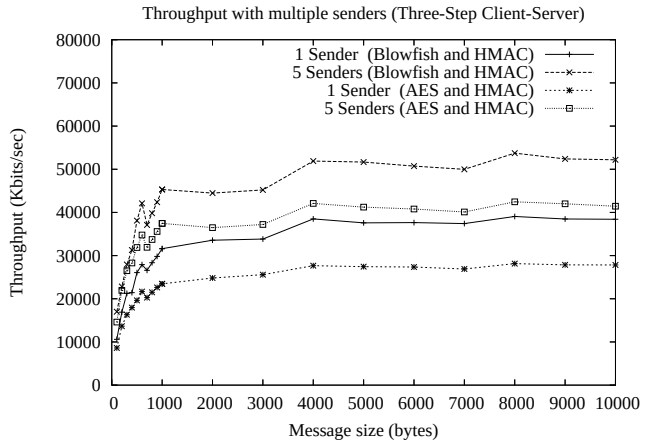


Fig. 9. Data throughput as a function of varied number of senders and message sizes

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 7 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.

B. Data Encryption

Another important building block in the architecture of secure group communication is the encryption module. Figure 8 presents our results for data throughput. Figure 8 (a) shows the throughput achieved by an integrated architecture (i.e. Three-Step Client-Server) under different configurations: using a 64-bit encryption algorithm, Blowfish with HMAC-SHA1, using a 128-bit encryption algorithm, AES also with HMAC-SHA1, and finally, no encryption is used, just HMAC-SHA1 for integrity and source authentication. As expected, adding security services decreases the throughput of the system, with the most expensive configuration being the one using AES. It is interesting to note the performance dip for messages around 700 bytes. Spread uses message packing for very small packets, to improve throughput. The dip at 700 bytes occurs because messages can no longer be packed into one network packet.

In Figure 8 (b) we compare the throughput of an Integrated Architecture (Three-Step Client-Server) with a Layered Architecture, in two encryption configurations, AES and Blowfish. 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.

The throughput for the Three-Step Client-Server is less than that of the throughput achieved in the Layered Architecture. The major reason for this decrease is that both headers and data are encrypted and the message delivery protocol employed by Spread can not detect if it needs to process a message further or not, without first decrypting it. Since the encryption operation 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. Both Integrated VS and Layered architecture have the same throughput since encryption is performed by clients.

This experiment only used one sender and the server that the sender was connected to was the bottleneck. In a case where several senders exist in the group and therefore several servers will send messages, this cost will be amortized and the throughput will increase considerably. The results presented in Figure 9 demonstrate this behavior. Both in the Blowfish and AES configuration a higher throughput is achieved when there are 5 senders in the system instead of 1.

We did not include results for the Three-Step Client Server architecture when clients connect remotely, but from the results in Figure 8 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 one of Integrated VS if no server membership occurs, and will degrade when membership changes occur, as some messages will need to be decrypted and re-encrypted under new keys. The Three-Step Client-Server architecture performance should be the worst in all cases, when clients connect remotely to servers.

VII. Discussion

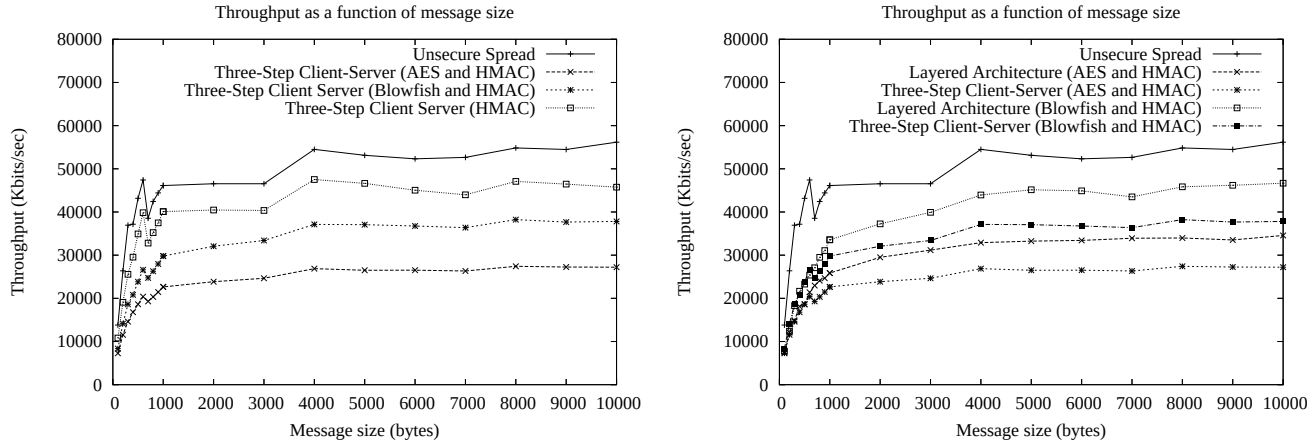
The layered architecture and each of the new proposed integrated variants have benefits and limitations. In the following we first compare the layered and integrated approaches and then discuss the three variants of integrated architectures.

A. Layered Architecture vs. Integrated Architecture

In this section we compare a layered architecture approach to an integrated architecture approach, when providing security services to a GCS. We compare them by investigating the following aspects: trust, key management scalability, impact of the compromise of the shared secret, complexity, and ability to efficiently support other group services.

The layered architecture has the advantage that no trust is put outside of the end user's control with respect to protecting the data generated by a client. The client needs to trust the servers with respect to the membership service and ordered and reliable delivery. 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. However, this model, due to the high security, but expensive key agreement protocols we used, has limited scalability, to no more than 100 members for the best performance key protocol.

The integrated architectures we proposed overcome the key management scalability problem by using the key agreement to compute a secret key shared by the servers, and thus putting more trust in the servers. This is because the security of the groups relies on the security of the servers shared key which is used in generating the client group keys. If the servers' key



(a) Integrated Architecture

(b) Integrated vs. Layered Architecture

Fig. 8. Data throughput as a function of varied encryption algorithms and security architectures

TABLE II
SECURE GROUP COMMUNICATION ARCHITECTURES

	Group Keys	Servers Key	Encryption	Group Comm. Model
Layered Architecture	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

is compromised, the confidentiality of the communication of all the groups in the system is compromised, as opposed to the layered model where in order to compromise the confidentiality of all the groups in the system, an attacker needs to compromise the shared key for each group. We note that in the case of the layered architecture, an attacker can perturb service availability by attacking the servers' communication.

An integrated architecture is more 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.

Another advantage of an integrated architecture vs. a layered architecture involves the protection of the control information messages exchanged by the servers. If designed appropriately, an integrated architecture can provide this service based on the secret key shared between servers, while the layered architecture can not. Combinations of the two approaches are also possible. For example, the clients who do not trust the servers will encrypt their data end-to-end, while the servers will also provide either secure channels, or only integrity checks between themselves.

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 poorly, but the security of a group is independent of the security of the rest of the groups and gives more control to the client.

B. Integrated Architectures Variants Comparison

As we discussed in Section V-B there is no one-size-fits-all architecture solution that will perform the best in all possible environments, under both VS and EVS group communication semantics. Therefore, we proposed three integrated architecture variants that trade off encryption cost for complexity, overhead and group communication model support. In this section we compare them by focusing on the group communication model supported, design and implementation of the key management building block (do they use client group keys or not) and the place where the encryption and decryption operations are performed (only between clients, only between servers, or between a client and a server).

Table II summarizes their features. 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. The approach is very appealing because it uses a less complex key management mechanism. However, it 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. Note, that depending on the implementation, even when clients connect locally, more than one encryption/decryption of each message can take place as discussed in Section VI-B. This architecture supports both the VS and the EVS semantics.

Both the Integrated VS and the Optimized EVS archi-

tectures use client group keys generated by servers. Our experimental results show that the scalability of the system is improved substantially with respect to the layered architecture. For all the integrated architectures the confidentiality of the data ultimately relies on the secret shared by the servers.

The smallest encryption overhead is exhibited by the Integrated VS approach. The Optimized EVS solution has the same encryption cost as the Integrated VS if the group membership is stable. When membership changes occur and there are messages not delivered in the membership they were sent in, four additional encryption/decryption operations per message are performed, to decrypt the messages encrypted with an old key and re-encrypt them under the current key. The encryption overhead incurred by the Three-Step Client-Server approach, even when clients connect locally, is larger than that of Integrated VS. However, it provides a stronger service since it also protects the information exchanged by the servers.

As mentioned in Section V-B.1 the cost of Three-Step Client-Server is quite high, when clients connect remotely. Possible solutions to decrease the number of encryption/decryption operations, use an asymmetric architecture as follows: the sending client encrypts the message using a pairwise key and sends it (via SSL) to its server; the server decrypts and re-encrypts the message, each receiving server decrypts and re-encrypts but re-encryption is done under a group key (a key common for all clients, on that server, that belong to the appropriate client-group, clients receive and decrypt. The overhead of encryption is still 6 operations but, on delivery, a server only performs one encryption instead of one for each client who is a group member.

VIII. Conclusions

The main focus of this work was designing a high-performance security architecture for a client-server group communication system. In particular, we focused on designing a security architecture for Spread, under two well-known group communication semantics: VS and EVS. Both models support network partitions and merges and present their particular challenges. Contributory key agreement protocols when used in a layered architecture have limited scalability. We overcame this by using an integrated approach that relies on contributory group key management in a light-weight/heavy-weight group architecture such that the cost of key management is amortized over many groups, while each group has its own unique key. The experimental results we present demonstrate the increased scalability of integrated approaches over layered approaches, without a significant decrease in throughput performance.

One limitation of this work is that it does not tolerate insider attacks and hence relies on servers not being compromised. Current architectures for distributed systems tolerating insider threats have strong connectivity requirements and multiple all-peer communication exchanges that prevent them from scaling well to wide-area networks. One way of overcoming this obstacle is to use a hierarchical approach, which combines intrusion-tolerant protocols with fault-tolerant protocols. In such an approach, security services of the type provided by Secure Spread, can be very useful.

References

[1] Y. Amir, C. Nita-Rotaru, J. Stanton, and G. Tsudik, "Scaling secure group communication systems: Beyond peer-to-peer," in *Proceedings of DISCEX3*, Washington, DC, USA, April 2003.

[2] T. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost, "On the impossibility of group membership." in *Proceedings of the 15^{th} ACM Symposium on Principles of Distributed Computing (PODC)*, May 1996, pp. 322–330.

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

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

[5] J. Schultz, "Partitionable virtual synchrony using extended virtual synchrony," Master's thesis, Department of Computer Science, Johns Hopkins University, January 2001, available at www.cnds.jhu.edu/publications/.

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

[7] K. Birman, "The process group approach to reliable distributed computing," *Communications of the ACM*, vol. 36, no. 12, pp. 36–53, December 1993.

[8] A. Montresor, R. Davoli, and Ö. Babaoğlu, "Enhancing Jini with group communication," in *Proceedings of the ICDCS Workshop on Applied Reliable Group Communication*, April 2001.

[9] O. M. Group, "Fault-tolerant CORBA: Joint revised submission," OBG Document orbos/99-12-08, December 1999.

[10] 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.

[11] M. Hayden, "The ensemble system," Ph.D. dissertation, Department of Computer Science, Cornell University, 1998.

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

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

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

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

[16] M. Steiner, G. Tsudik, and M. Waidner, "Key agreement in dynamic peer groups," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 8, August 2000.

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

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

[19] 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, no. 4, pp. 311–342, November 1995.

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

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

[22] I. Keidar, J. Sussman, K. Marzullo, and D. Dolev, "A client-server oriented algorithm for virtually synchronous group membership in wans," in *Proceedings of the 20^{th} International Conference on Distributed Computing Systems (ICDCS 2000)*. IEEE Computer Society, 2000, p. 356.

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

[24] O. Rodeh, K. Birman, and D. Dolev, "Using AVL trees for fault tolerant group key management," *International Journal on Information Security*, vol. 1, no. 2, February 2002.

[25] ——, "The architecture and performance of security protocols in the Ensemble Group Communication System," *ACM Transactions on Information and System Security*, vol. 4, no. 3, pp. 289–319, August 2001.

[26] 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*. ACM, November 1994, pp. 68–80.

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

[28] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," *ACM Transactions on Computer Systems*, vol. 17, no. 2, May 1999.

[29] M. A. Hiltunen and R. D. Schlichting, "Adaptive distributed and fault-tolerant systems," *International Journal of Computer Systems Science and Engineering*, vol. 11, no. 5, pp. 125–133, September 1996.

[30] M. A. Hiltunen, R. D. Schlichting, and C. Ugarte, "Enhancing survivability of security services using redundancy," in *Proceedings of The International Conference on Dependable Systems and Networks*, June 2001.

[31] L. Gong, "Enclaves: Enabling secure collaboration over the Internet," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 3, pp. 567–575, April 1997.

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

[33] 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, no. 6, pp. 784–803, December 1997.

[34] Y. Amir, "Replication using group communication over a partitioned network," Ph.D. dissertation, Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel, 1995.

[35] G. V. Chockler, I. Keidar, and R. Vitenberg, "Group communication specifications: A comprehensive study," *ACM Computing Surveys*, no. 4, pp. 427–469, December 2001.

[36] Y. Kim, A. Perrig, and G. Tsudik, "Tree-based group key agreement," *ACM Transactions on Information and System Security*, vol. 7, no. 1, 2004.

[37] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Stanton, and G. Tsudik, "Secure group communication using robust contributory key agreement," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 5, pp. 468–480, May 2004.

[38] Spread Project team, "Spread," http://www.spread.org.

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

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

[41] Y. Kim, A. Perrig, and G. Tsudik, "Group key agreement efficient in communication," *IEEE Transactions on Computers*, vol. 33, no. 7, 2004.

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

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

[44] *Recommendation for Block Cipher Modes of Operation - Methods and Techniques*. National Institute for Standards and Technology (NIST), 2001, no. SP 800-38A.

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

[46] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik, "On the performance of group key agreement protocols," *ACM Transactions on Information Systems Security*, vol. 7, no. 3, August 2004.

Yair Amir received the BS (1985) and MS (1990) degrees from the Technion and the PhD degree (1995) from the Hebrew University of Jerusalem. Prior to his PhD, he gained extensive experience building C3I systems. He is currently with the department of Computer Science, The Johns Hopkins University where he served as Assistant Professor since 1995, Associate Professor since 2000 and Professor since 2004. He has been a member of the program committees of the IEEE International Conference on Distributed Computing Systems (ICDCS) in 1999 and 2002, the ACM Conference on Principles of Distributed Computing (PODC) in 2001, and the IEEE International Conference on Dependable Systems and Networks (DSN) in 2001 and 2003. He is a member of the IEEE Computer Society.

Cristina Nita-Rotaru is an Assistant Professor in the Computer Science department of the Purdue University and a member of Center for Education and Research in Information Assurance and Security at Purdue University. She received the BS and MSc degrees in Computer Science from Politechnica University of Bucharest, Romania, in 1995 and 1996, and the MSE and PhD degrees in Computer Science from The Johns Hopkins University in 2000 and 2003. Her Ph.D thesis focused on secure group communication. Her research interests include secure distributed systems, network security protocols and security aspects in wireless networks. She is a member of the ACM and IEEE.

Jonathan Stanton is an Assistant Professor in the Computer Science department of the George Washington University. He also holds an appointment as an adjunct assistant professor in the Computer Science department of The Johns Hopkins University. He received the BA degree in Mathematics in 1995 from Cornell University, and the MSE and PhD degrees in Computer Science from The Johns Hopkins University in 1998 and 2002. His research interests include distributed systems, secure distributed messaging, network protocols, and middleware support for clustered systems. He is a member of the ACM and the IEEE Computer Society.

Gene Tsudik is a Professor of Computer Science at the University of California, Irvine. He has been conducting research active in internetworking, network security and applied cryptography since 1987. He obtained a Ph.D. in Computer Science from USC in 1991; his dissertation focused on access control in internetworks. Before coming to UC Irvine in 2000, he was a Project Leader at IBM Research, Zurich Laboratory (1991-1996) and USC Information Science Institute (1996-2000). Over the years, his research interests included: routing, firewalls, authentication, mobile/wireless network security, secure e-commerce, anonymity, secure group communication, digital signatures, key management, ad hoc network routing, and, more recently, database privacy and secure storage. Some of Professor Tsudik's notable research contributions include: Inter-Domain Policy Routing (IDPR), IBM Network Security Program (KryptoKnight), IBM Internet Keyed Payment (iKP) protocols, Peer Group Key Management (CLIQUES) and Mediated Cryptographic Services (SUCSES). Professor Tsudik has over 100 refereed publications and 7 patents. Since 2002 he has been serving as Associate Dean of Research and Graduate Studies in the Donald Bren School of Information and Computer Sciences at UCI. He is a member of the IEEE.