International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 3, ISSN No. 2455-2143, Pages 94-99
Published Online December – January. 2016 in IJEAST (http://www.ijeast.com)

SECURE SPREAD: AN INTEGRATED ARCHITECTURE FOR SECURE GROUP COMMUNICATION

Arjeeta Tripathi
G.L.B.I.TM

Anshika Mishra
G.L.B.I.T.M

Shudhanshu Purwar
G.L.B.I.T.M

Kirti Sharma
G.L.B.I.T.M

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 – sharing information about the groups in the system – 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

Index Terms— Protocol architecture (C.2.2.b) and Distributed applications (C.2.4.b)

I. INTRODUCTION

UBIQUITOUS information access and communication have become essential to everyday life, global business, and national security. Activities, including personal, commercial and international financial transactions, studying and teaching, shopping for goods 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 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. Processes communicate via multicast in an asynchronous environment where failures can occur. More specifically, a GCS provides two services: group membership as well as reliable and ordered message delivery. The membership service provides all members of a group with information about the list of currently connected and alive group members [1] 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. All the above events can trigger corresponding changes in group membership.

The core of GCS is 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]. Practical 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 can be excluded from the group membership. If the network is stable, GCS membership reflects the current list of connected and alive group members.

Membership and message delivery services were formalized in two models: Virtual Synchrony [3] (VS) and Extended Virtual Synchrony [4] (EVS). The main difference between the two models has to do with the relation between the views in which messages are sent and delivered.

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: 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 [7], a GCS resilient to process crashes and network partitions. To put this work into context, we briefly outline our earlier efforts. Some of our previous results [8] demonstrate how authentication and access control for a client-server GCS can be efficiently addressed. The framework specified that clients are authenticated when connecting to a server, while access control to group resources is enforced by the local server. Another recent work focused on designing a robust contributory group key agreement [9], [10]. In the present work, complimentary to previous work, 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 makes them appealing for secure group communication. However, when used in a layered architecture, they scale poorly. We show how this limitation can be overcome by using an integrated approach in a light-weight/heavy-weight [11] group architecture, such that the cost of key management is amortized over many groups, while each group has its own unique key.

- 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 semantics 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, such systems have better performance; thus, 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.

II. RELATED WORK

RESEARCH in group communication systems 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 [12], Transis [13], Horus [14], Totem [15].

These systems explored several different models of group communication such as Virtual Synchrony [3] and Extended Virtual Synchrony [4].

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. Although efficient, this method does not provide certain security properties such as key independence and perfect forward secrecy. Ensemble is used for authorization. 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 and the Spin glass 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.

Some other approaches focus on building highly configurable dynamic distributed protocols. Cactus 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 specific security services. Redundancy of data confidentiality is obtained by encrypting data multiple times, each time using a different encryption algorithm. This approach is not appropriate for data-stream applications where throughput is a concern.

Enclaves are used for secure group communication. 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. Policy flavors addressed by Antigone include: rekeying, 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 violate perfect forward secrecy, while the session rekeying mechanism – although able to detect the leader's failure – cannot recover from it.

Unlike aforementioned systems, we focus on using contributory group key agreement as a building block for other security services in Spread [7]. Contributory key agreement protocols provide strong security properties. In particular, they can guarantee that: (1) compromise of any subset of old group keys does not lead to compromise future group keys; (2) compromise of any subset of group keys does not lead to compromise of previous group keys; and, (3) more generally, compromise of all-but-one group keys does not lead to compromise of the one "missing" group key. Moreover, even compromise of the members' long-term secret keys does not lead to compromise of any group keys. Our work investigates trade-offs between security and group communication semantics support. Our secure GCS supports two strong group communication semantics: Virtual Synchrony and Extended Virtual Synchrony.

III. SPREAD

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

Spread [7] 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. 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 [1] incur the heavy cost of a full-fledged membership change.

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) 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 due to the support for open groups. 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.

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.

IV. SECURITY ASSUMPTIONS

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

We do not consider insider attacks in this work. We acknowledge that such threats are significant, especially, for the underlying group membership protocols; some of our ongoing work focuses on this direction. However, in this paper we assume that each entity (client or server) 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*.

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.

Tree-Based Group Diffie-Hellman (TGDH) protocol provides key independence and perfect forward secrecy; it was also proven secure with respect to passive outside eavesdropping. 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.)

1) Three-Step Client-Server*:* The most intuitive architecture is one derived from 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*.

We note that the communication taking place in the system can be classified in two logical communication 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 is using a standard two-party secure communication protocol, such as SSL/TLS. If a client connects to a server running on the same machine, Spread architecture uses IPC. In this case, no data protection is needed and client-server communication is not encrypted.

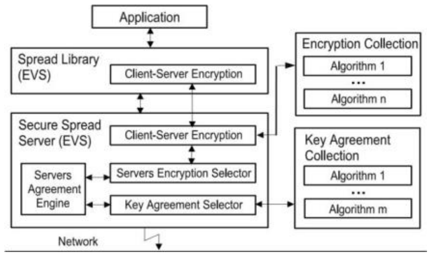


Fig.1. A Three-Step Client-Server architecture for Spread

The intra-server communication channel is provided by a multicast protocol developed on top of UDP. In order to provide confidentiality of this communication, a block cipher encryption protocol based on a key shared by the servers is a good solution.

Figure 1 presents such architecture. The Servers Agreement Engine detects changes in the server group connectivity and for each connectivity change performs a key management protocol between servers. In addition, time-based or data-based key refresh can be enforced. As mentioned above protocol for key management. Servers can distinguish between communication coming from peer servers and communication from the clients, and therefore, use the appropriate key in order to encrypt/decrypt the information.

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; however, the system supports any encryption algorithm in the OpenSSL library, including AES [6], while integrity and authentication are performed using HMAC-SHA1 [5]. Two different shared keys are derived, one used for encryption and one for the HMAC computation. In addition, the system can 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 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. 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

2) Integrated VS: Although the Three-Step Client-Server architecture presented above is relatively simple, it suffers from decreased throughput due to encryption performed by servers.

Therefore, it is not recommended when clients connect remotely. Recall that we aim to design 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 in order to send messages to the group. As mentioned above, this requires client group's 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 views changes.

Figure 2 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.

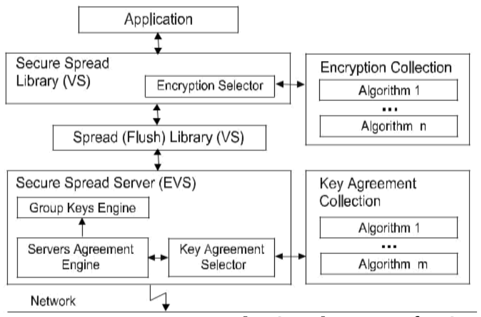
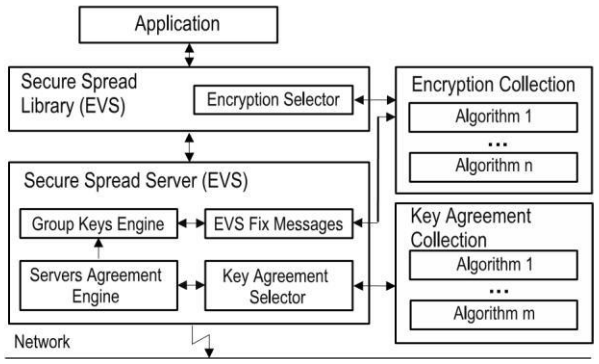


Fig. 2. Integrated VS architecture for Spread

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 Ks, 2) group name (unique within the system), and 3) unique number that identifies the group

Fig. 3. An Optimized EVS Architecture for Spread



The group key for group g in view v, where v is uniquely identified by view_idgv is

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 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 3 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 reencrypted 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 solves 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). However, 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.

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.

If the servers' key 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.

V. REFERENCES

[1] The choice of a key expiration methodology can affect the 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.

[2] *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.

[3] 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.

[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] 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 (TPDS), vol. 15, pp. 468–480, May 2004.

[6] 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.

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

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

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

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

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

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

[14] M. A. Hiltunen, R. D. Schlichting, and C. Ugarte, "Enhan cing surviv-ability of security services using redundancy," in *Proceedings of The International Conference on Dependable Systems and Networks,* June 2001.

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