Hydra: A decentralised group key management

Sandro Rafaeli and David Hutchison
{rafaeli,dh}@comp.lancs.ac.uk
Computing Department, Lancaster University, LA1 4YR, UK

Abstract

In this paper, we describe Hydra, a scaleable decentralised architecture to create and distribute symmetric cryptographic keys to large multicast-based groups. In order to support large multicast groups, the group is divided into a number of TTL-scoped regions. The purpose of having several regions is to achieve flexible and efficient key management, particularly in face of group membership changes. Hydra servers are designated as subgroup managers. They are responsible for managing group membership and distributing secret keys at the subgroup level. Hydra does not employ a manager for subgroup managers, and hence, it is not vulnerable to failures of single entities.

1 Introduction

The use of IP multicast in supporting distributed applications has brought several benefits in terms of data distribution. However, it is a challenge to effectively control the access of such data since anyone can join a multicast group and receive the data being exchanged. Secure multicast sessions can be realised through encryption. Messages are protected by encryption using a cryptographic key, which in the context of group communication is called *group key*. Only those who know the group key are able to recover the original message.

One of the issues that must be addressed in secure multicast sessions is *key distribution*, i.e., how to distribute the group key to all members of a group. In recent years, many different proposals have been presented to solve the problem of key distribution. Some of the proposals employ a central entity that manages the whole group, and thus may not scale for large groups. Other proposals distribute the group key generation among all members of the group. This also does not scale to large groups because every single member of the group participates in the key generation. Moreover, other proposals divide large groups into smaller ones, employing a controller for each subgroup.

In this paper, we present and describe the Hydra system. Hydra employs a decentralised architecture to generate and distribute the group key. The large group is split into smaller subgroups and each subgroup is managed by a Hydra server. Hydra does not employ a manager for subgroup managers, and hence, it is not vulnerable to failures of single entities.

The work presented here was done within the context of ShopAware - a research project funded by the European Union in the Framework V IST Programme.

2 Decentralised Group-Key Management Systems

A decentralised architecture for group-key management is required to have the following features [2][8]:

Distributed control: there should not be a central manager controlling the subgroup managers. The central manager rises the same issues as centralised systems, namely if the centralising manager is unavailable, the whole group is compromised;

Local rekey: membership changes in a subgroup should be treated locally, which means that rekey of a subgroup should not affect the whole group. This is also known as the *1–affects–n* problem;

Data independency: the data path should be independent of the key management path, which means that rekeying the subgroup should not impose any interference or delays to the data communication;

Communication type: groups with a single data source are said to use *1–to–n* communication, and groups with several or all members being able to transmit are characterised by using *n–to–n* communication.

These features will be used to compare Hydra with other decentralised systems in Section 5.

3 The Hydra Architecture

3.1 Overview

The Hydra architecture is composed of two hierarchical levels (see Figure 1). It employs a hierarchy in order to organise a large group into smaller subgroups. The top level is composed exclusively of Hydra servers (HSs). HSs are the entities responsible for managing the subgroups. Group members are placed in the bottom level separated into subgroups.

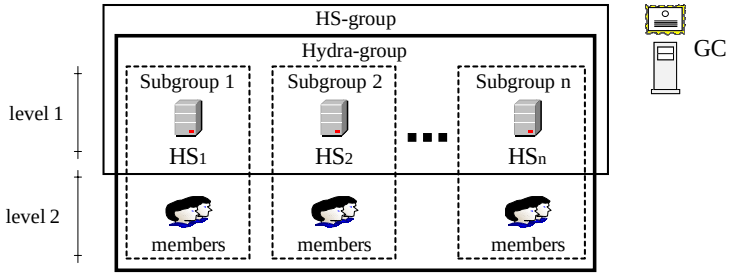

Figure 1: Hydra System.

A group key, namely GroupKey, protects the group communication. All members in the group share the GroupKey. Hydra uses two secure multicast groups for the key management. The first one, called HS-group, is used by the HSs in the top level to agree on a common group key. The second one, called Hydra-group, is used by the HSs to distribute the agreed key to their respective subgroup members in the bottom level. The Hydra-group is best described as a single "physical" multicast group "virtually" divided among the HSs managing it by means of *time-to-live* (TTL) scoping [7].

A key called HydraKey protects the communication on the HS-group. HydraKey is shared by all HSs managing a session. It is valid for a short period and after the period expires, HydraKey is refreshed. The HS performing the rekey operation at the time HydraKey expires is responsible for generating the new one and sending it together with the new Groupkey.

A subgroup key protects the GroupKey within a subgroup in the Hydra-group. Each subgroup has its own subgroup key and it is shared between the HS controlling that subgroup and all the subgroup members. When a new GroupKey is distributed among the HSs controlling a group, the HSs encrypt the new group key with their respective subgroup key and then send it out.

Hydra does not use a central entity to generate the new group key; if a membership change takes place at HS_i, and a new key must be generated, it can generate the new group key and send this key to the other HSs involved in that session. Then, the HSs relay the new group key to their respective subgroup members. One or more HSs being unavailable does not interfere with the remaining HSs. Members associated with a failing HSs can rejoin the group session by simply connecting to another HS.

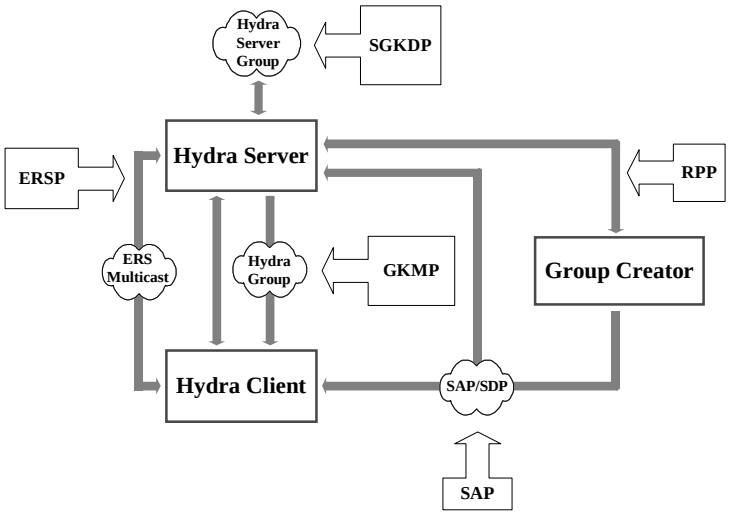

Figure 2: Hydra's building blocks.

In Hydra, the group creator takes up important roles during the group creation. Firstly, it distributes certificates to all entities involved in the group (servers and members). Secondly, it is responsible for setting the policies of the group and choosing the cryptographic parameters. Thirdly, it has to allocate the multicast addresses needed by the architecture and then advertise the session. Finally, it waits for Hydra servers to connect and request permission to manage the group. The GC participates in the group setup phase, but it does not take any roles during group management or key distribution.

Figure 2 shows the protocols that make the Hydra system. Firstly, the GC announces its group session using the Session Announcement Protocol (SAP) [4] and Session Description Protocol (SDP) [3]. HSs listen to the well-known SAP address for Hydra group announcements. When a new Hydra group is identified, the HS contacts the group creator requesting permission to manage it. The request is made

using the Permission Request Protocol (PRP).

Hydra Clients also listen on the SAP address and when a session is interesting to them, they contact the closest HS to acquire the group key. Hydra clients use the Expand Searching Ring Protocol (ERSP) to find the closest HS.

The Synchronised Group Key Distribution Protocol (SGKDP) protocol maintains the group key synchronised among the HSs participating in the group key management. If there is a membership change in the HS, then it generates a new group key and communicates it to the other HSs via the SGKDP (see Section 4).

Finally, when HSs agree on a new group key, they relay this key to their respective subgroup members using the Group Key Management Protocol (GKMP). The GKMP protocol is independent of the SGKDP protocol.

3.2 Authentication

Hydra employs a Public Key Infrastructure (PKI) model to authenticate all parties in the system. The PKI root certification authority is the group creator. Its functions include personal authentication, token distribution and revocation reporting.

The GC's certificate is the root of the hierarchy. The GC also maintains three other certificates. Each certificate is used to issue specific sets of certificates. The first set, which contains Hydra server certificates (hCerts), is used to identify HSs to the group creator. The second set, which contains Session Authorisation Tokens (SATs), is used to authenticate an HS to other HSs and group members. The third set, which contains member certificates (mCerts), is used to authenticate members to HSs and authorise them to participate in the secure group.

4 Synchronised Group Key Distribution Protocol

Hydra does not employ a manager of managers as other decentralised schemes do. The central manager is a central point of failure and if it becomes unavailable, it compromises the security of the group, especially if the central manager is in charge of updating the group key. Hydra gives opportunity to all HSs managing a group session to change the group key. When a membership change takes place at an HS, and a new key must be generated, it can generate the new group key and send this key to the other HSs involved in that session. Thus, one or more HSs become unavailable do not interfere with the remaining HSs.

SGKDP messages are sent over a group communication service. The group communication is in charge of guaranteeing that all HSs will receive SGKDP messages in the same order. The order in which the messages are transmitted is not important, but it is fundamental that the messages are delivered to all HSs in exactly the same order. Therefore, the group communication in the HS-group must provide *totally ordered* [6] delivery of messages. A totally ordered service has the following property: If both processes i and j deliver the same messages a and b, they deliver these messages in the same order.

Hydra uses the SPREAD group communication system [1]. SPREAD supports total order and provide complete extended virtual synchrony [9] with delivery guarantees.

The SGKDP protocol is divided into two parts. In the first part, a new HS joining the group requests the current GroupKey and HydraKey keys from other HSs already

Table 1: Notations.

$\{X\}_K$ and $\{X\}_K^{-1}$	encryption and decryption of X with key K
$\{X\}_{sign}^{-K_i}$	X is signed with i's private key, $-K$
$\mathrm{E}_i \rightarrow \mathrm{E}_j$: X	Entity E_i sends message X to E_j
$\mathrm{E}_i \rightarrow *$: X	Entity E_i multicasts message X

in the group. The GC does not participate in any key management protocol, hence it cannot provide any keys to newly joined HSs. Therefore, newly joined HSs must request from other HSs (participating in the group management) for a copy of the current keys.

In the second part, the HSs can update these keys with new values. This part is executed with either of two messages: `NEWKEY` message or `ALLKEYS` message. Whenever an HS needs to rekey the group (due to membership change, for example), it multicasts a `NEWKEY` message containing the new GroupKey. Other HSs receiving this message learn the new value of the GroupKey key and then update their respective subgroups with that key. When it is time to change the HydraKey, the HS sends an `ALLKEYS` message instead.

Part I of protocol SGKDP between HS_i and HS_j goes as follows (see notations in Table 1):

Part I:

1. $\mathrm{HS}_i \rightarrow * : 1, \mathrm{G}, \mathrm{HS}_i, 0, \texttt{JOIN}, nonce_i, \mathrm{SAT}_i$
2. $\mathrm{HS}_j \rightarrow \mathrm{HS}_i : 1, \mathrm{G}, \mathrm{HS}_j, \mathrm{V}_{hk}, \texttt{OK}, \left\{ \{nonce_i, \mathrm{V}_{gk}, \mathrm{K}_{gk}, \mathrm{V}_{hk}\}_{sign}^{-\mathrm{K}_{\mathrm{HS}_j}}, \mathrm{SAT}_j \right\}_{\mathrm{K}_{hk}}, \{\mathrm{K}_{hk}\}_{+\mathrm{K}_{\mathrm{HS}_i}}$

HS_i multicasts a `JOIN` message to the HS-group. The message contains a *nonce* and the HS_i's SAT.

HSs receive the request and verify whether HS_i is allowed in the group or not. If it is, then one of the HSs sends back an `OK` message with current HydraKey and GroupKey. The *nonce* received in the request is copied to the `OK` message. The response is properly signed and the respective SAT is sent with it. The `OK` message is put in an envelope and HS_i's public key (from SAT_i) is used to seal the secret key. Only the HS holder of the counter-part private key can open the envelope.

HS_i decrypts the envelop and verifies HS_j's signature. It also checks if HS_j is a valid HS for group session G. HS_i then verifies if the *nonce* received matches the *nonce* it had sent. If they do, the HS_i knows that this is a fresh `OK` message and accepts the keys.

HS_i may not receive any responses. This means that it is the first HS in the group and then it generates the first HydraKey and GroupKey keys.

Part II of protocol SGKDP goes as follows:

Part II:

1. $\mathrm{HS}_i \rightarrow * : 1, \mathrm{G}, \mathrm{HS}_{id}, \mathrm{V}_{hk}, \texttt{NEWKEY}, \left\{ \{\mathrm{V}_{gk}^m, \mathrm{K}_{gk}^m\}_{sign}^{-\mathrm{K}_i}, \mathrm{SAT}_i \right\}_{\mathrm{K}_{hk}}$

or

1. $\mathrm{HS}_i \rightarrow * : 1, \mathrm{G}, \mathrm{HS}_{id}, \mathrm{V}_{hk}, \texttt{ALLKEYS}, \left\{ \{\mathrm{V}_{gk}^m, \mathrm{K}_{gk}^m, \mathrm{V}_{hk}^m, \mathrm{K}_{hk}^m\}_{sign}^{-\mathrm{K}_i}, \mathrm{SAT}_i \right\}_{\mathrm{K}_{hk}}$

An HS can use either a `NEWKEY` or `ALLKEYS` message for updating GroupKey. The `NEWKEY` message updates only GroupKey and `ALLKEYS` updates both GroupKey and HydraKey. A `NEWKEY` message carries the new GroupKey version (incrementing the previous version value by one) and the new GroupKey value. The HS signs the new values and then encrypts it with the current HydraKey. The `ALLKEYS` message carries also a new version value for HydraKey and a new HydraKey.

No acknowledgement is needed due to the total order property. When an HS is delivered a message, it knows that all other HSs were delivered the same message.

When an HS is delivered a `NEWKEY /ALLKEYS` message, it executes Algorithm 3. Every HS_i has some local state variables, namely GV_i and GK_i, respectively, the current group key version and the current GroupKey key, and HV_i and HK_i, respectively, the current Hydra key version and the current HydraKey key. Initially, GV_i and HV_i are set to zero and GK_i and HK_i to *void*. When an HS requires a new group key, it increments the version number by one and chooses a new GroupKey key value. These new values are sent in the `NEWKEY/ALLKEYS` message:

Due to the total ordering property, all HS members of the group communication executing Algorithm 3 receive the same message m in step 1. If all HS started executing the algorithm with the same value GV, after several executions of the algorithm, they are all holding the same value GV, and hence, GK.

Figure 3: Part II SGKDP algorithm.

1. receive and decrypt message m
2. verify m's authenticity
3. discard m and stop if $(V_{gk}^m \leq GV)$ or $(V_{gk}^m \geq GV + 1)$
4. accept m and make $GK = K_{gk}^m$ and $GV = V_{gk}^m$
   if message type is `ALLKEYS` then $HK = K_{hk}^m$ and $HV = V_{hk}^m$
5. distribute GK to subgroup members

The correctness of the algorithm can be verified with an example. Let us say that there are two Hydra servers, HS_i and HS_j, managing a certain group. Both hold values GV and GK, where $GV_i = GV_j$ and $GK_i = GK_j$. They communicate with each other using a totally ordered delivery of messages system. HS_i and HS_j send, respectively, m_i and m_j rekey messages at virtually the same time. Before sending m_i, HS_i sets $V_{gk}^{m_i}$ to $GV_i + 1$ and HS_j sets $V_{gk}^{m_j}$ to $GV_j + 1$. By definition, HS_i and HS_j have m_i and m_j delivered in the exact same order. Let us assume that m_i is delivered first and then m_j is delivered. Hence, HS_i and HS_j receive m_i in step 1. In step 2, both verify authenticity of m_i. Then, in step 4, HS_i and HS_j accept m_i, since $GV_i < V_{gk}^{m_i} \leq GV_i + 1$ and $GV_j < GV_{gk}^{m_i} \leq GV_j + 1$, and set, respectively, $GK_i = K_{gk}^{m_i}$ and $GV_i = V_{gk}^{m_i}$ and $GK_j = K_{gk}^{m_i}$ and $GV_j = V_{gk}^{m_i}$. In step 5, both HSs distribute the new GroupKey to their respective subgroup. When message m_j is delivered, both HSs discard it in step 3 because $V_{gk}^{m_j} = GV_i$ and $V_{gk}^{m_j} = GV_j$.

HSs update the HydraKey key in the same way. However, an `ALLKEYS` message is sent instead of a `NEWKEY` message. The `ALLKEYS` is treated exactly as the `NEWKEY` message, and the example above works in the same way.

5 Related Work

In this section, we go through other main contributions in the group key management area that also use decentralised subgroups. Although all proposals have in common the subgrouping within the large group, they use different techniques to achieve the distribution. The similarity is that all of them divide the large group into smaller subgroups and employ a subgroup manager to control each subgroup.

In Iolus [8], each subgroup has its own key. The absence of a general group key

makes membership changes in a subgroup to be treated locally. It means that changes that affect a subgroup are not reflected to other subgroups.

Hardjono proposed the IGKMP protocol [5]. In IGKMP, the subgroup are divided in administratively scoped areas [7]. There is one Domain Key Distributor (DKD) and several Area Key Distributors (AKDs). A group key is generated by the DKD and is propagated to the members through the AKDs.

Setia, Koussih and Jajodia proposed Kronos [10]. Kronos is an approach driven by periodic rekey, which means a new key is generated after a certain period of time, disregarding any membership changes in the group.

Table 2 shows a summary of the characteristics of each decentralised scheme. The value between brackets represents the best option, accordingly to sections 2.

Table 2: Features comparison

Feature/Scheme	**Hydra**	Iolus	IGKMP	Kronos
Distr. Control (Y)	**Y**	Y	N	Y
Fault-tolerant (Y)	**Y**	Y	N	Y
Secure Keys (Y)	**Y**	Y	Y	N
Local rekey (Y)	**N**	Y	N	na
Data indep.(Y)	**Y**	N	Y	Y
Membership rekey(Y)	**Y**	Y	Y	N

Y=feature present, N=feature not present, na=not applicable

Hydra has a fundamental advantage over IGKMP, because Hydra does not have a central controller. IGKMP relies on the central controller for authentication and key generation, and if the central controller is not available, the whole group is affected. In Hydra, the group creator appears only during setup time and it is contacted neither during authentication nor for group key updates.

Another advantage that Hydra has over IGKMP is that Hydra is more flexible regarding the subgroup formation. Whenever an HS stops working, its members can always find the new nearest HS and rejoin the group session. This is not possible in IGKMP due to its administrative scoped multicast subgrouping. There may not be another reachable AKD.

Although Kronos does not use a central controller and the subgroup controllers can generate the new keys independently, it compromises the group security because it generates the new key based on the previous one. If one key is disclosed then it compromises all following keys. Additionally, Kronos servers must have their clocks perfectly synchronised to be able to change keys at exactly the same time. Hydra uses keys that are independently generated, so that they cannot be used to recover either past keys or future keys.

Hydra presents the undesirable effect of one change affecting the whole group that Iolus manages to avoid. However, Hydra separates the control path (key management) from the data path, which means that group communication is not interrupted or spoiled by key management operations. Although Hydra needs a longer time to rekey the whole group after a membership change, the delay will be noticed only in the group key updates (control path). The group communication (data path) will not be affected by that. In Iolus, the group data must be translated when it moves from one subgroup to another.

6 Conclusion

In this paper, we have described Hydra, a new approach for distributing group key management. A large multicast session is split in smaller subgroups and a group controller manages each subgroup. Hydra drops the central controller for subgroup controllers and distributes the key generation and group management among several HSs. It makes the system fault-tolerant, since it allows any HS to generate the group key and it will prevent the interruption of the group communication.

Hydra draws the line between control (key management) path and data path. Although Hydra may need a long time to rekey the whole group after a membership change, the delay will be noticed only in the group key updates (control path). The group communication (data path) will not be affected by that. Yet, the separation between the control and data path allows Hydra to serve both *1-to-n* and *n-to-n* group communications.

References

[1] Y. Amir and J. Stanton. The Spread wide area group communication system. Technical Report 98-4, Department of Computer Science, Johns Hopkins University, 1998.

[2] L. Dondeti, S. Mukherjee, and A. Samal. Survey and Comparison of Secure Group Communication Protocols. Department of Computer Science, University of Maryland, 1999.

[3] M. Handley and V. Jacobson. SDP: Session Description Protocol. RFC 2327, November 1997.

[4] M. Handley, C. Perkins, and E. Whelan. Session Announcement Protocol. RFC2974, October 2000.

[5] T. Hardjono, B. Cain, and I. Monga. Intra-domain Group Key Management Protocol. IETF Internet draft (work in progress), September 2000.

[6] K.P.Birman. *Building Secure and Reliable Network Applications.* Manning Publications Co., 1996.

[7] D. Meyer. Administratively Scoped IP Multicast. RFC 2365, July 1998.

[8] S. Mittra. Iolus: A Framework for Scalable Secure Multicasting. In *ACM SIGCOMM*, volume 27,4 of *Computer Communication Review*, pages 277–288, New York, September 1997. ACM Press.

[9] L. Moser, Y. Amir, P. Melliar-Smith, and D. Agarwal. Extended virtual synchrony. In *In Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*, pages 56–65, 1994.

[10] S. Setia, S. Koussih, and S. Jajodia. Kronos: A Scalable Group Re-keying Approach for Secure Multicast. In *2000 IEEE Symposium on Security and Privacy*, Oakland CA, May 2000.