Universitat Politècnica de Catalunya

Tesis Doctoral

Contribución a la Seguridad en Servicios Multimedia. Aplicación a Entornos Multicast.

Autor: Josep R. Pegueroles Vallés.
Director: Dr. Francisco Rico-Novella.
Tutor: Dr. Luis J. de la Cruz Llopis.

2003

Programa de Doctorado de Ingeniería Telemática.
Departamento de Ingeniería Telemática.

"Bien está lo que bien acaba"

Refranero popular español

a Marta

Resumen

Los servicios multimedia pueden dividirse en dos fases: acceso al servicio e intercambio de datos. La primera fase se realiza normalmente sobre protocolos fiables de transporte y comunicaciones punto a punto. La segunda usa protocolos de transporte no fiable y comunicaciones multicast. Para añadir seguridad a la primera fase basta con aplicar las soluciones unicast existentes. Añadir seguridad sobre IP multicast, en cambio, introduce una nueva problemática no resoluble mediante las técnicas unicast clásicas. Esta tesis doctoral estudia los problemas que aparecen al ofrecer seguridad en multicast y propone distintas soluciones prácticas.

De todos los posibles ataques en la fase de distribución, la escucha de datos es probablemente el de mayor repercusión. El servicio de protección frente a escuchas es el cifrado. El cifrado multicast añade el problema de gestión de claves de grupos. Para conseguir confidencialidad estricta, la clave de sesión debe actualizarse cada vez que un miembro se da de alta o de baja en el grupo. Cuando el número de miembros es elevado y muy dinámico se hace inviable el reparto punto a punto de claves. En esta tesis se propone un algoritmo de actualización de claves multicast basado en los árboles lógicos de claves presentados en la literatura, pero que incluye el uso de funciones pseudo-aleatorias y reducción modular. Nuestra propuesta minimiza la cantidad de memoria necesaria en el Servidor de Claves y reduce considerablemente el ancho de banda requerido por los algoritmos existentes.

En muchos casos la actualización de claves cada vez que un miembro se da de alta o de baja en el grupo no es necesaria. Servicios como la televisión en red, permiten un decremento en seguridad a cambio del ahorro en ancho de banda que supone la espera en la actualización. Ahí operan los algoritmos de renegociación por lotes o en *batch* que procesan todas las solicitudes de los miembros conjuntamente y de forma periódica. Este tipo de algoritmos deben mantener balanceado el árbol lógico de claves para ser eficientes. Este trabajo propone, implementa y testea un algoritmo en *batch* que da lugar a árboles lógicos completamente balanceados.

Las técnicas en *batch* se tratan independientemente al algoritmo de renegociación, pero el estudio conjunto de las técnicas de gestión de claves con estos algoritmos da lugar a mejoras en la eficiencia global. En esta tesis doctoral se propone la combinación de la propuesta de algoritmo de gestión de claves con las técnicas en *batch*. Con ello se consigue una mejora en los parámetros de eficiencia.

El algoritmo en *batch* propuesto, no es soportado por el estándar del IETF de gestión de claves multicast (GDOI). Ello es debido a que nuestro algoritmo permite que los miembros del grupo cambien de posición en el árbol y descarga la responsabilidad de actualización de la posición al mismo cliente. Finalmente, se propone la modificación del protocolo GDOI de forma que se pueda usar con las propuestas de algoritmos de esta tesis.

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos y Glosario

CGI *Common Gateway Interface*
Es un interfaz para que programas externos puedan operar bajo un servidor de información. Actualmente, los servidores de información soportados son servidores HTTP (Hypertext Transfer Protocol).

GCKS *Group Controller Key Server*
Acrónimo del elemento que hace las funciones de controlador de grupo y servidor de claves en el Marco de Referencia de seguridad en multicast.

GDOI *Group Domain of Interpretation*
Acrónimo del protocolo de gestión de la seguridad en multicast propuesto por el IETF.

GSA *Group Security Agent*
Denominación genérica de elementos controladores en la arquitectura de seguridad en multicast *Iolus*.

GSC *Group Security Controller*
Elemento adicional controlador de subgrupos en la arquitectura de seguridad en multicast *Iolus*.

GSEC *Group SECurity*
Nombre del grupo de investigación del IRTF encargado de la seguridad en comunicaciones de grupo.

GSI *Group Security Interface*
Elemento adicional traductor entre subgrupos en la arquitectura de seguridad en multicast *Iolus*.

HTML *HyperText Markup Language*
Lenguaje usado para escribir documentos para servidores World Wide Web. Es una aplicación de la ISO Standard 8879:1986 (SGML, Standard Generalized Markup Language).

HTTP *HyperText Transfer Protocol*
Protocolo para distribuir y manejar sistemas de información hipermedia. Una característica es la independencia en la visualización y representación de los datos, permitiendo a los sistemas ser contruidos independientemente del desarrollo de nuevos avances en la representación de los datos.

IETF *Internet Engineering Task Force*
Organismo estandarizador de protocolos en internet.

IKE *Internet Key Exchange*
Intercambio de claves para Internet. Protocolo que establece unas directivas de seguridad compartidas y autenticación de claves para aquellos servicios (por ejemplo, IPSec) que requieran claves. Definido en el RFC2409.

IP *Internet Protocol*
Conjunto de reglas que regulan la transmisión de paquetes de datos a través de Internet. La versión actual es IPv4 mientras que en el proyecto Internet2 se intenta implementar la versión 6 (IPv6), que permitiría mejores prestaciones dentro del concepto QoS (Quality of Service).

IPSec *Internet Protocol Security*
Conjunto de protocolos desarrollados por IETF para soportar el intercambio seguro de paquetes en el nivel IP. IPsec se utiliza ampliamente para implementar Redes Virtuales Privadas (VPNs).

IRTF *Internet Research Task Force*
Organismo de la Internet Society compuesto por diversos grupos que trabajan sobre temas relacionados con los protocolos, la arquitectura y las aplicaciones de Internet. Lo forman personas individuales y no representantes de empresas u organismos.

KEK *Key Encryption Key*
Clave usada en protocolos de gestión de claves para cifrar la clave de sesión u otras claves que ayuden a obtenerla.

KS *Key Server*
Nomenclatura del servidor de claves en el Marco de Referencia de la seguridad en multicast.

LKH *Logical Key Hierarchy*
Protocolo de gestión de claves basado en árboles lógicos. Nombre que recibe una de las propuestas del IETF.

MIME *Multipurpose Internet Mail Extensions*
Conjunto de especificaciones Internet de libre distribución que permiten tanto el intercambio de texto escrito en lenguajes con diferentes juegos de caracteres como el correo multimedia entre ordenadores y aplicaciones que sigan los estándares de correo Internet. Las especificaciones MIME están recogidas en numerosos RFCs, entre los que se encuentran los RFC1521 y 1848.

MMDBMS *MultiMedia DataBase Management System*

MSEC *Multicast Security*
Grupo de trabajo del IETF encargado de la estandarización de protocolos relacionados con la seguridad en multicast

MTU *Maximum Transfer Unit*
La MTU de una red es la máxima cantidad de datos o tamaño de los paquetes que pueden ser transferidos de forma física en dicha red. Se producirá fragmentación si un paquete es enviado por una red con una MTU menor que la longitud estructural del paquete. Esto producirá una reducción del rendimiento y la necesidad de recomponer fragmentos.

OFC *One-way Function Chain*
Protocolo de gestión de claves basado en árboles lógicos. Nombre que recibe una de las propuestas del IETF.

OFT *One-way Function Tree*
Protocolo de gestión de claves basado en árboles lógicos. Nombre que recibe una de las propuestas del IETF.

OKS *Optimal Key Storage*
Protocolo de gestión de claves basado en árboles lógicos. La propuesta minimiza el número de claves a guardar en el gestor de claves.

RSVP *Resource ReSerVation Protocol*
Protocolo desarrollado para soportar distintas clases de QoS (calidad de servicio) en aplicaciones IP.

RTP *Real-time Transport Protocol*
Protocolo de comunicaciones utilizado en la transmisión de información en tiempo real, como por ejemplo el audio y vídeo de una videoconferencia.

SEK *Session Encryption Key*
Clave de sesión. Equivalente a TEK.

TCP *Transmission Control Protocol*
Protocolo de internet de nivel de transporte orientado a conexión y fiable.

TEK *Transmission Encryption Key*
Clave de sesión. Equivalente a SEK.

TLS *Transport Layer Security*
Es un protocolo que ayuda a proporcionar a las comunicaciones, privacidad y seguridad entre dos aplicaciones que se comunican a través de una red. También permite que los clientes autentifiquen servidores y que los servidores autentifiquen a los clientes.

UDP *User Datagram Protocol*
Protocolo de internet de nivel de transporte no orientado a conexión y no fiable.

Capítulo 1

Introducción

1.1. Motivación

En los últimos años hemos asistido al auge de las aplicaciones multimedia en red. Servicios como la multivideoconferencia, el vídeo quasi-bajo demanda o los juegos en red que tenían como principal inconveniente el uso ineficiente de ancho de banda han ido superando sus dificultades tecnológicas y han irrumpido en el plano comercial.

Un servicio multimedia puede dividirse en dos fases de funcionamiento: acceso al servicio, que normalmente se realiza de forma individual y sobre protocolos de transporte fiable, y distribución de información, que utiliza protocolos de transporte no fiable y usualmente es compartida por diversos usuarios.

En la fase de distribución, la tecnología multicast ha sido uno de los aspectos que más ha contribuido al uso eficiente de los recursos de la red. El transporte multicast permite a los emisores enviar un mensaje a una única dirección IP de forma que los receptores sólo deben "suscribirse" a esa dirección y son los *routers* los encargados de ponerse de acuerdo para entregar la información a los receptores interesados. Así, para hacer llegar la misma información a N usuarios, basta con un paquete multicast por cada tramo físico de red en vez de los N necesarios en unicast.

Con el creciente uso comercial de estos servicios, introducir aspectos de seguridad se vuelve imprescindible. Cada una de las citadas fases requiere distintos servicios como distintos son los tipos de ataques más frecuentes en cada una de ellas. De igual manera, a la hora de implementar estos servicios, deben tenerse en cuenta tanto el protocolo de transporte utilizado como si la comunicación es individual o de grupo. Esta tesis aborda la introducción de servicios de seguridad en la fase de distribución de servidores multimedia.

Esquema de funcionamiento de los servicios multimedia

En la Fig 1.1 se esquematiza el funcionamiento genérico de los servicios multimedia en red [Szuprowicz, 1995]. En la fase de solicitud de servicio, el cliente accede a la oferta del servidor, normalmente a través de un navegador web. A continuación, realiza la petición de servicio o contenido (paso (1) en Fig 1.1), generalmente esta petición se realiza mediante un formulario insertado en una página HTML que será interpretado por un CGI en el servidor HTTP (paso (2) en Fig 1.1). La naturaleza del "producto" demandado por el cliente puede ser distinta:

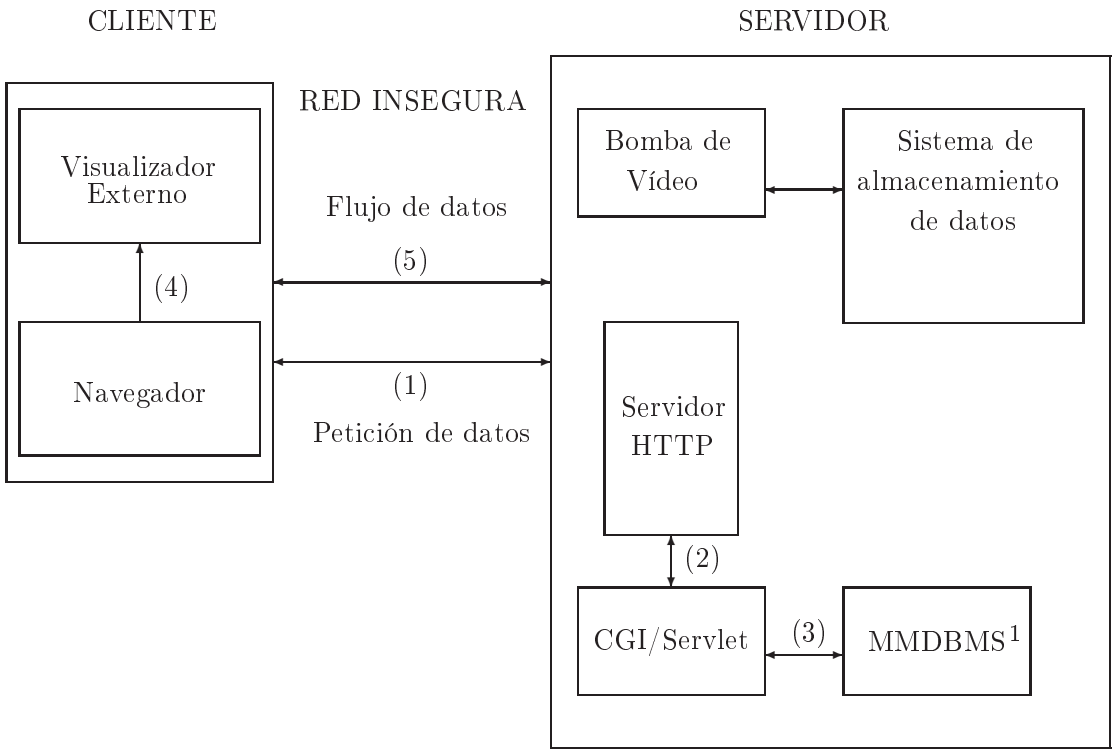


Figura 1.1: Modelo de servidor multimedia

visualización de un evento deportivo o película de vídeo, participación en un juego en red, participación en un multivideochat, etc. En cualquiera de los casos, el CGI deberá interpretar la petición del cliente y consultar a su base de datos (MMDBMS[1]) dónde y cómo debe servir ese "producto" (paso (3) en Fig 1.1), por ejemplo, en el caso en que se trate de un servicio de vídeo quasi bajo demanda, deberá consultar en qué bomba de vídeo y puerto se encuentra el recurso. Con esta información, finalmente el CGI devuelve al cliente la información necesaria para que pueda disfrutar del servicio, esto puede realizarse mediante la generación de un documento MIME que se devuelve al cliente por transferencia HTTP.

Hasta el momento el cliente aún no está haciendo uso del servicio, pero ya dispone de toda la información necesaria para hacerlo. En esta fase de solicitud del servicio todas las transacciones en red se realizan sobre el protocolo de transporte fiable TCP ya que no implican gran cantidad de información pero la pérdida de esa información puede ser crítica.

En la segunda fase de distribución de contenidos el cliente interpreta la información remitida por el servidor, por ejemplo la contenida en el documento MIME, y ejecuta una aplicación (paso (4) en Fig 1.1) que inicia el intercambio de información con la fuente real del contenido ((5) en Fig 1.1), por ejemplo, en el caso del vídeo, con la bomba multimedia correspondiente. Esta segunda fase es crítica en cuanto a limitaciones temporales y no importa tanto que se pierda algún paquete como que éste no llegue en un determinado instante. Todo este proceso normalmente se lleva a cabo mediante Protocolos de ReSerVa de recursos (RSVP) y Protocolos de Transporte en tiempo Real (RTP) sobre la capa no fiable UDP.

El esquema sirve tanto para unicast como multicast. En el primer caso, la conexión con la fuente de datos se realiza mediante conexión simple y puede iniciarse en cualquier instante

[1]MultiMedia Data Base Management System

de tiempo. En el caso multicast, todos los clientes se conectan de forma conjunta y deben esperarse a que el servidor empiece a ofrecer el servicio.

Servicios de seguridad requeridos

A grandes rasgos, en un esquema como el presentado, debe evitarse que usuarios no autorizados accedan a la información (fraude), o se hagan pasar por un usuario autorizado sin realmente serlo (suplantación de identidad), que alguien ofrezca el servicio sin poder luego prestarlo (timo), o se haga pasar por un servidor de vídeo conocido sin realmente serlo (suplantación de marca). Además, deben protegerse los derechos de autor (violación de la propiedad intelectual) y evitar que se puedan analizar los contenidos requeridos por un determinado usuario (recopilación no autorizada de datos personales) [Schneier, 2000].

La forma habitual de protegerse frente a los ataques referidos a la identidad de los participantes es mediante servicios de control de acceso, identificación de usuario, no repudio e integridad de la información. Estos servicios, ampliamente estudiados, se basan en certificados digitales de identidad e infraestructura de clave pública [Adams and Lloyd, 2002, Nash et al., 2001]. Por su parte, el fraude, el timo y la recopilación no autorizada de preferencias de usuario son ataques que pueden ser resueltos mediante el servicio de protección frente a escuchas. Éste se consigue cifrando los datos [Schneier, 1996].

Campos abiertos

Como hemos comentado, en la fase de acceso sólo es necesario adaptar las soluciones unicast existentes. Sin embargo, la inclusión de servicios de seguridad en multicast presenta problemáticas que no pueden solventarse mediante las técnicas unicast conocidas y precisan de nuevas propuestas que les den solución. En concreto, multicast vuelve a poner de actualidad la gestión de la seguridad en grupos de usuarios.

Seguridad en comunicaciones de grupo significa básicamente privacidad, autenticación tanto de miembro como de grupo y acuerdo de políticas [Hardjono et al., 2000b]. Cifrar los datos mediante una clave conocida por todos los miembros del grupo solventa dos de dichos problemas: privacidad y autenticación de grupo. Sólo los poseedores de esa clave tendrán acceso a los datos de la comunicación y sólo ellos podrán generar mensajes válidos.

Hay que tener en cuenta que se cifra información que posteriormente usará protocolos de transporte no fiable (UDP) y normalmente con restricciones temporales muy fuertes. Así, el cifrado debe realizarse sobre unidades mínimas de transmisión (MTU) y en un tiempo muy limitado para evitar la propagación de errores y el descarte de paquetes debido a retardos excesivos.

Además, el cifrado en comunicaciones en grupo introduce la necesidad de gestión de las claves de grupo. Si se quiere preservar la confidencialidad durante toda la transmisión, la clave de sesión debe actualizarse cada vez que un usuario se da de alta o de baja en el grupo.

En grupos grandes, que pueden llegar a miles o centenares de miles de usuarios, y muy dinámicos, donde se producen muchas altas y bajas en periodos relativamente cortos de tiempo, el método trivial de enviar la nueva clave de sesión mediante una conexión unicast a cada uno de los miembros del grupo se vuelve completamente inviable. En concreto, la minimización del ancho de banda requerido para la renegociación de claves es aún una área de

estudio abierta en comunicaciones seguras de grupo. Los objetivos de esta tesis están básicamente relacionados con este campo de investigación.

1.2. Objetivos

El objetivo de este trabajo es el estudio de los requisitos de seguridad en los servidores multimedia multicast. En concreto, se aportan soluciones alternativas a los problemas que presenta la integración de seguridad en la fase de distribución de contenido multimedia especialmente en lo referente a la gestión de claves de grupos.

Hasta hace poco tiempo no se empezó a prestar atención a la seguridad de la fase de distribución de la misma forma que se hizo con la fase de acceso y los campos abiertos aún son numerosos. Las contribuciones principales de esta tesis, cuyos objetivos concretos se enumeran a continuación, se enmarcan en esta área.

Estudio de algoritmos de gestión de claves de grupo

Una vez determinado el algoritmo simétrico más adecuado para cifrar comunicaciones multimedia, debe tenerse en cuenta que en multicast la clave de cifrado debe cambiarse cada vez que un miembro se da de alta o de baja. Éste es el denominado problema de *rekeying* o redistribución de claves.

En la literatura existen distintas propuestas de protocolos de redistribución de claves aún en fase de discusión y estandarización. Desde el punto de vista teórico, dichos protocolos presentan parámetros de eficiencia susceptibles de ser mejorados. Además, la validación empírica de las propuestas y posibles mejoras no podía ser contrastada ya que no existía ninguna implementación real de dichos protocolos. Así los objetivos relacionados con la gestión de claves se centran en la mejora, implementación y evaluación de algoritmos de gestión de claves de grupo:

- Estado del arte de algoritmos de gestión de claves de grupo.
- Propuestas de mejoras de protocolos de renegociación de claves de grupo basado en funciones pseudo-aleatorias y propiedades de la teoría de números.
- Evaluación del comportamiento de los distintos protocolos, obtención de parámetros de eficiencia y validación de resultados teóricos.
- Programación de las clases necesarias para incluir gestión de claves en aplicaciones multicast.

Estudio de algoritmos de gestión de claves por lotes

Aunque para obtener confidencialidad estricta, deben cambiarse las claves de grupo cada vez que un miembro se da de alta o de baja en el grupo, algunas aplicaciones multimedia no requieren de un nivel de seguridad tan elevado. En estos casos se opta por la renegociación periódica de claves. Estos algoritmos consisten en el proceso por lotes (o en *batch*) de todas las peticiones de alta y de baja que se producen en un determinado intervalo de tiempo. Estos

algoritmos ahorran ancho de banda pero decrementan el nivel de seguridad y además introducen problemáticas de gestión adicionales, especialmente en el mantenimiento balanceado de los árboles de claves. Los objetivos y contribuciones conseguidos en cuanto a la gestión de claves por lotes son los siguientes:

- Estado del arte de algoritmos de gestión de claves de grupo en *batch*.
- Propuesta de algoritmo de renegociación de claves de grupo en *batch* balanceado.
- Programación de las clases necesarias para incluir gestión de claves en *batch* en aplicaciones multicast.
- Evaluación del comportamiento de los distintos protocolos, obtención de parámetros de eficiencia y validación de resultados teóricos.

Integración de algoritmos de gestión de claves de grupo y proceso por lotes

Normalmente, las técnicas en *batch* se tratan independientemente al algoritmo de renegociación. Esto permite la compatibilidad de estos algoritmos con distintas técnicas en batch, y viceversa, pero también incide en un decremento en cuanto a eficiencia. En esta tesis doctoral también se combina la propuesta de algoritmo de gestión con las técnicas en *batch*. Con ello se consigue una mejora en los parámetros de eficiencia. Los objetivos son los que se marcan a continuación.

- Combinación de la propuesta de algoritmo de renegociación de claves con técnicas en *batch*.
- Obtención y comparación de los parámetros de eficiencia de las propuestas.

Adaptación de las propuestas a los estándares actuales

El IRTF e IETF a través de sus grupos de investigación y trabajo GSEC y MSEC están dedicando grandes esfuerzos a la estandarización de un protocolo universal para comunicaciones seguras en grupo (*Group Domain of Interpretation* o GDOI). Este protocolo define una arquitecura común y el formato de los mensajes que deben intercambiarse los participantes de una comunicación de grupo para gestionar la seguridad. Este protocolo permite el uso de distintos algoritmos de renegociación de claves pero no incluye las propuestas de esta tesis doctoral. El último objetivo pretende que estas propuestas sí sean soportadas por el GDOI.

- Propuesta de modificación del GDOI para adaptar los algoritmos de gestión de claves y proceso por lotes propuestos.

1.3. Desarrollo de la tesis

Todas las contribuciones de esta tesis doctoral y la forma como se han conseguido sus objetivos se explican en los siguientes capítulos estructurados de la siguiente manera.

En el Capítulo 2 se ponen las bases para el estudio de las comunicaciones seguras de grupo. En primer lugar se clasifican y caracterizan estas comunicaciones, sus requisitos de seguridad

y los parámetros de eficiencia que se usarán para su evaluación. Seguidamente, todo esto se concreta para el caso multicast; se explican los objetivos clave de la seguridad en multicast: manipulado de los datos, gestión de material de claves y gestión de políticas y se describe el escenario de referencia que servirá para abordar su estudio.

El Capítulo 3 trata el problema de la distribución de claves en multicast. En primer lugar, se exponen los problemas existentes en la seguridad en multicast: escalabilidad y uso eficiente del ancho de banda. A continuación, se presentan algunas soluciones a estos problemas presentes en la literatura y se discuten sus ventajas e inconvenientes. Finalmente se realizan dos propuestas de mejora de algoritmo de renegociación de claves multicast basadas en árboles lógicos de claves, funciones pseudo-aleatorias y teoría de números. En el mismo capítulo se evalúan sus parámetros de eficiencia y se comparan con otras soluciones existentes.

En el Capítulo 4 se estudian los algoritmos de renegociación de claves por lotes o en *batch*. Se trata de una alternativa a los algoritmos clásicos, con un nivel de seguridad menor pero con un uso de ancho de banda mucho menor. Consiste en procesar periódicamente y de forma conjunta todas las peticiones que se hayan realizado en un intervalo determinado de tiempo. Estos algoritmos, para conseguir máxima eficiencia, requieren que los árboles de claves se mantengan balanceados durante toda la vida del grupo. Las propuestas actuales de la literatura dan lugar a árboles desbalanceados. En este capítulo se propone un algoritmo en *batch* que mantiene los árboles balanceados y así mejora la eficiencia. La evaluación y comparación con otros algoritmos existentes también se presentan en este capítulo. El proceso en *batch* no es incompatible con los algoritmos de gestión de claves presentados en el Capítulo 3. En este capítulo también se combinan las propuestas presentadas anteriormente con el proceso por lotes, con ello se consigue mejorar su eficiencia.

En el Capítulo 5 se describe la implementación realizada tanto del GDOI como de las aplicaciones de gestión de seguridad programadas para dotar de seguridad a un servidor de vídeo multicast. Además, y debido a que las contribuciones de esta tesis no se soportan por el estándar de comunicaciones de grupo del IETF, se presenta una propuesta de adaptación del estándar del IETF para comunicaciones seguras de grupo.

Finalmente, las conclusiones y líneas futuras se discuten en el Capítulo 6.

Para permitir una mejor comprensión de los conceptos explicados en este documento, se adjuntan los anexos presentados a continuación.

En el Anexo A se expone el trabajo realizado referente a generación de tráfico, caracterización y modelado del comportamiento de usuarios multicast. Este trabajo fue imprescindible para simular los algoritmos propuestos teniendo en cuenta unas condiciones de contorno lo más reales posibles, y servirá al lector para comprender mejor la problemática de la gestión de claves en multicast.

En el Anexo B se presenta el GDOI o *Group Domain of Interpretation* por ser el estándar del IETF para protocolos de comunicaciones seguras de grupo. El estudio y comprensión de este protocolo fue necesario para realizar la propuesta de adaptación de dicho protocolo a las contribuciones de esta tesis. Igualmente servirá al lector para entender cómo se articulan de forma real los mensajes de gestión de claves a los que se hace referencia en el cuerpo de esta tesis.

1.4. Artículos publicados

En este apartado se relacionan las publicaciones a que ha dado lugar este trabajo de tesis.

Nacionales

- J. Pegueroles, F. Rico-Novella. *Evaluación de Herramientas Java para ofrecer Confidencialidad en Servidores de Vídeo.* VII Reunión Española sobre Criptología y Seguridad de la Información.VII RECSI. Septiembre 2002, Oviedo. ISBN 84-699-8931-6

- J. Pegueroles, F. Rico-Novella. *Rekeying de grupo en multicast seguro usando el Teorema de Fermat.* XVII Simposium Nacional de la Unión Científica Internacional de Radio. XVII URSI. Septiembre 2002. Alcalá de Henares, Madrid. ISBN 84-8138-517-4

- J. Hernández-Serrano, J. Pegueroles. *Comercio electrónico a través de web de contenido multimedia sobre transporte multicast.* II Simposio Español de Comercio Electrónico. SCE'03, Junio 2003. Barcelona. ISBN 84-932902-0-3

- J. Pegueroles, F. Rico-Novella. *LKH mejorado para gestión de claves en grupos multicast.* IV Jornadas de Ingeniería Telemática. Jitel'03, Septiembre 2003. ISBN 84-96131-38-6

- J. Hernández-Serrano, J. Pegueroles, M. Soriano. *Método heurístico de generación de tráfico sintético de juegos en red multicast.* IV Jornadas de Ingeniería Telemática. Jitel'03, Septiembre 2003. ISBN 84-96131-38-6

Internacionales

- J. Pegueroles, F. Rico-Novella. *Performance evaluation of cryptographic algorithms for multicast secrecy protection.* 7th Nordic Workshop on Secure IT Systems. Nordsec2002. November 2002, Karlstad University, Sweden. ISBN 91-89422-96-1 ISSN 1403-8099

- J. Pegueroles, F. Rico-Novella. *Balanced LKH for Secure Multicast.* Proceedings of the 2nd NATO PfP/PWP International Scientific Conference on Security and Protection of Information. SPI2003. April 2003, Brno, Czech Republic. ISBN 80-85960-50-8

- J. Pegueroles, F. Rico-Novella. *Balanced Batch LKH: New Proposal, Implementation and Performance Evaluation.* Proceedings of the IEEE Symposium on Computers and Communications . ISCC'2003. June 2003, Antalya, Turkey. ISBN 0-7695-1961-X ISSN 1530-1346

- J. Pegueroles, F. Rico-Novella. *Enabling Secure Multicast Using a New Java LKH Rekeying Tool.* Proceedings of the 3rd International Conference on Web Engineering. ICWE2003. July 2003, Oviedo, Spain. Published by Springer Verlag, LNCS 2722, ISBN 3-540-40522-4 ISSN 0302-9743

- J. Pegueroles, J. Hernández-Serrano, F. Rico-Novella, M. Soriano. *Adapting GDOI for balanced batch-LKH*, draft-irtf-gsec-gdoi-batch-lkh-00.txt. IETF Draft. Work in Progress. Proceedings of the 57th IETF Meeting. July 2003, Vienna, Austria.

- J. Pegueroles, F. Rico-Novella, J. Hernández-Serrano, M. Soriano. *Improved LKH for Batch Rekeying in Multicast Groups.* IEEE International Conference on Information Technology Research and Education. ITRE2003. August 2003, New Jersey, USA. ISBN 0-7803-7878-4

- J. Hernández-Serrano, M. Soriano, J. Pegueroles. *Heuristic method for synthetic traffic generation for multicast networked games.* IASTED International Conference on Communication Systems and Networks. CSN 2003. September 2003. Benalmádena, Spain. ISBN 0-88986-388-1

- J. Pegueroles, W.Bin, M.Soriano, F. Rico-Novella. *Group rekeying algorithm using pseudo-random functions and modular reduction.* Accepted to be published in the proceedings of the 2nd International Workshop of Grid and Cooperative Computing (GCC2003) by Springer Verlag Lecture Notes in Computer Science series. December 2003.

Capítulo 2

Seguridad en comunicaciones de grupo

2.1. Introducción

Tal como se ha puesto de manifiesto en el Capítulo 1, las aplicaciones comerciales en redes multicast han empezado a tomar relevancia como fuente de negocio en internet. Servicios como la distribución de noticias o *news*, difusión de cotizaciones de bolsa, transmisión de vídeo, teleconferencias, actualizaciones de software, etc. son claro ejemplo de ello [Quinn, 1999].

Añadir seguridad a estos servicios plantea problemas más complejos que los existentes en las comunicaciones unicast. A los aspectos clásicos de autenticidad de fuente e integridad y secreto de los datos se les debe añadir otros temas como el control de acceso al grupo, autenticidad de grupo, tamaño y dinámica del mismo, confianza en el controlador del grupo, etc.

Si bien es cierto que cualquier comunicación multicast puede descomponerse en múltiples comunicaciones unicast, también es verdad que aprovechar los servicios de seguridad punto a punto para comunicaciones en grupo es completamente inviable. Se deben buscar soluciones adecuadas a estos entornos y que tengan un coste similar al que supone la seguridad en comunicaciones punto a punto.

Como primer paso en este capítulo se enumeran de forma genérica las características relevantes a la seguridad de las comunicaciones de grupo y se introducen los servicios de seguridad requeridos en cualquier interacción de grupo.

La soluciones adoptadas deben tener en cuenta el protocolo de transporte. En los servicios multimedia que requieren comunicaciones de grupo, el mecanismo de transporte natural es el multicast. Por ello, como continuación se particulariza todo lo discutido de forma genérica a las comunicaciones multicast.

Finalmente se expone el escenario que se usará como referencia en todo el documento de tesis, se explican los entornos multicast más comunes y se discuten los parámetros de evaluación que se usarán para comparar los protocolos de seguridad sobre multicast.

2.2. Introducción a la seguridad en comunicaciones de grupo

Quizás el rasgo más característico de las comunicaciones multicast es su naturaleza grupal. La criptografía se ha encargado del estudio de la seguridad en comunicaciones de grupo desde hace más de dos décadas. En particular, la compartición de secretos [Shamir, 1979,

Simmons, 1992] y las firmas digitales múltiples [Okamoto, 1988], han sido dos campos muy fructíferos, pero cuyos resultados han sido mayormente teóricos y que difícilmente pueden ser puestos en práctica. Al abordar el problema de la seguridad en multicast este estudio debe realizarse mucho más desde el enfoque de las comunicaciones. De cualquier forma, los principios de la seguridad en grupo son los mismos en los que se basan los estudios mencionados. A continuación se presenta la caracterización genérica de los grupos y sus requisitos de seguridad, válidos tanto desde el enfoque criptográfico como de comunicaciones.

2.2.1. Características de las comunicaciones en grupo

Existen distintos tipos de comunicaciones en grupo, como existen distintos tipos de grupos. Las soluciones de seguridad adoptadas en cada uno de ellos dependerán en gran medida de sus características. En esta sección se relacionan los parámetros más relevantes a tener en cuenta en las comunicaciones de grupo [Canetti and Pinkas, 2000].

Tamaño del grupo

Ésta es quizás la característica más importante por los problemas de escalabilidad que comporta. El número de participantes en un grupo puede variar desde unas pocas decenas, por ejemplo en las reuniones de empresa por videoconferencia; pasando por unos miles, en congresos o clases virtuales; a varios millones, en difusiones cerradas de grandes eventos deportivos, por ejemplo.

Características del miembro.

Bajo esta denominación básicamente se incluyen dos características de los miembros: potencia computacional y atención. Como potencia computacional se entiende la capacidad de cálculo, velocidad, memoria disponible, etc. que tienen los miembros (o los dispositivos pertenecientes a los miembros que les dan acceso al grupo). Por atención, entendemos la condición de los miembros de estar on-line. Por ejemplo, diremos que un miembro tiene atención máxima si durante todo el transcurso de la comunicación está "escuchando" al grupo. Un miembro con atención intermitente será el que durante el transcurso de la comunicación pueda estar escuchando o no (a voluntad propia o porque el transporte no fiable se lo impide).

No todos los integrantes de un grupo tienen porque tener la misma potencia computacional o atención. En este sentido diremos que las características de los miembros de un grupo son homogéneas o no.

Dinámica del grupo.

Este parámetro refleja la variación en el número e identidad de los componentes de un grupo. El caso más simple es que éste sea estático y conocido a priori. Es decir, ningún miembro abandona ni se añade al grupo después de su inicialización y, además, su identidad es conocida desde el principio. El segundo caso es que el grupo mantenga el número de miembros pero su identidad cambie. Finalmente, el caso más habitual es que el grupo pueda crecer, mantenerse o disminuir indistintamente, en número de miembros, y también pueda variar la identidad de los mismos.

La dinámica también refleja cuán a menudo cambia la composición del grupo y cómo de rápido debe ser éste actualizado. Una condición a resaltar cuando se describe la dinámica del grupo es si los cambios pueden producirse a ráfagas o no.

Tiempo previsto de vida

Es el tiempo en que el Gestor de Grupo debe mantener la seguridad del mismo. Pueden ser minutos, días, o incluso tiempo ilimitado.

Número y tipo de emisores.

Puede darse el caso que sólo uno de los miembros del grupo envíe datos (1 a M), o que los emisores sean un subconjunto de éstos (N a M), o que todos los miembros puedan actuar como emisores (M a M). En este último caso, además, se puede dar la situación que, aunque todos estén autorizados para ser emisores, la mayoría de tráfico lo genere un subconjunto reducido de miembros.

Volumen y tipo de tráfico

La cantidad de información útil que desean intercambiarse los miembros también puede variar mucho. El caso más crítico es cuando el volumen de tráfico es elevado. El tipo de tráfico hace referencia a si la comunicación debe realizarse en tiempo real o no, en cuyo caso debe determinarse la latencia permitida. El tipo de tráfico suele dividirse en: datos (requisitos de tiempo real poco estrictos y poco volumen), audio (debe ser en tiempo real y volumen reducido o moderado) y vídeo (tiempo real y volumen elevado de datos).

2.2.2. Requisitos de seguridad en comunicaciones de grupo

La descripción de las características anteriores da una idea del tipo de grupo al que se añadirá seguridad. Además, distintos tipos de grupos requerirán distintos servicios de seguridad.

En [Canetti and Pinkas, 2000] se relacionan los posibles servicios de seguridad susceptibles de ser introducidos en las comunicaciones de grupo. Por supuesto, no todos los grupos necesitarán todos los servicios, y la particularización de estos será distinta dependiendo de si el grupo es de una naturaleza u otra.

Gestión de grupo y control de acceso

El primer requisito, y tal vez el más importante en un grupo seguro, es garantizar que sólo los miembros registrados y legítimos tengan acceso a él. También debemos ser capaces de diferenciar los privilegios de cada miembro, a menudo se debe permitir sólo a un miembro (o subconjunto) del grupo enviar datos.

Muchas veces el control de acceso se realiza de forma individual y usando técnicas PKI unicast, otras veces se establecen jerarquías de miembros, dependiendo de sus privilegios, y así se distribuye también su gestión. De todos modos, la forma más simple de ofrecer acceso restringido a la comunicación de grupo es mediante el uso de una clave de grupo conocida sólo por los miembros autorizados.

Como parte de la gestión del grupo, también deben tenerse en cuenta otras implicaciones de seguridad como:

- Autenticación de miembros potenciales del grupo.
- Distribución segura de claves de grupo.
- Revocación de miembros dados de baja.
- Impedir la confabulación de miembros.
- Periodicidad con la que deben actualizarse las claves.
- Cómo guardar la información y permitir auditorías externas.

Confidencialidad a corto plazo (*Ephemeral Secrecy*)

Un grupo puede requerir que se establezca un secreto entre ellos para intercambiarse información de forma que sólo se dificulte el acceso a la misma aunque la aplicación de técnicas avanzadas de criptoanálisis puedan acceder a ella. En esos casos, puede ser suficiente un mecanismo que sólo retarde el acceso o restrinja el acceso a la parte crucial de los datos. Por ejemplo, para mantener confidencialidad a corto plazo al transmitir vídeo sería suficiente con cifrar los coeficientes de Fourier de menor orden en la codificación MPEG.

Confidencialidad a largo plazo (*Long term secrecy*)

En cambio, cuando por ejemplo la información intercambiada es crítica, se require confidencialidad a largo plazo. Ésta asegura que los datos cifrados se mantienen secretos a todo aquél ajeno al grupo durante un tiempo considerable después de haberse realizado la transmisión. A menudo, éste requisito no puede ofrecerse para el tráfico de grupo y tiempo infinito. En particular, cuanto más grande es el grupo, más débil es la confidencialidad a largo plazo (incluso si las técnicas criptográficas son perfectas), ya que la información de cifrado revelada es mayor y las posibilidades de confabulación entre miembros aumentan.

Confidencialidad hacia adelante (*o Perfect Forward Secrecy*)

Asegura que los datos se mantienen secretos si la clave en uso se ve comprometida (por ejemplo, por criptoanálisis o por robo). Este requisito sólo es necesario para aplicaciones que necesiten secreto a largo plazo. En muchos usos de las comunicaciones de grupo no es necesario.

Confidencialidad hacia atrás (*o Perfect Backward Secrecy*)

Evita que un miembro del grupo que llega en fecha posterior a su establecimiento pueda acceder a los datos que se ha intercambiado el grupo previamente a su alta.

Autenticidad de los datos y del remitente

Asegura que los datos recibidos provienen del remitente y que no han sido modificados en el camino. La autenticidad tiene dos vertientes:

- Autenticidad de grupo: un miembro del grupo puede reconocer si el mensaje ha sido enviado por otro miembro del grupo o se puede demostrar a un miembro externo del grupo que un mensaje ha sido enviado por un integrante legítimo del grupo.
- Autenticidad individual: es posible identificar al remitente particular de un mensaje dirigido al grupo.

Anonimato

En algunas ocasiones, los integrantes del grupo quieren preservar su identidad. En esos casos el servicio de anonimato se hace imprescindible. Existen dos niveles de anonimato. El primero se corresponde con los servicios clásicos de anonimato y consiste en mantener secreta la identidad de los miembros del grupo o frente a otros miembros del grupo o frente a entidades externas o los dos casos a la vez. El segundo nivel consiste en desvelar la identidad del grupo, pero que un agente externo no pueda saber cuál de todos los integrantes del grupo es el remitente real de un mensaje. Muy relacionado con este caso está la protección frente a análisis de tráfico.

No repudiabilidad

Como contrapartida, y servicio contrario al anonimato, un miembro del grupo puede querer probar que él ha enviado un mensaje y el destinatario lo ha recibido. El servicio de no repudiabilidad da la habilidad a la fuente de probar a terceras partes que los datos han sido transmitidos y recibidos correctamente. Pueden existir distintos tipos de no repudiabilidad, dependiendo si se considera al miembro individual o al grupo como fuente de datos.

Servicio de disponibilidad.

Finalmente, un servicio clásico pero que resulta mucho más dificultoso en entornos de grupo es la disponibilidad. El mantenimiento de la disponibilidad frente a ataques de negación de servicio es mucho más difícil en grupos, ya sea porque son más fáciles de realizar o porque éstos son mucho más dañinos.

2.3. Particularización a multicast

La particularización al entorno multicast de los conceptos presentados se realiza a través de dos ejes. Un primer punto es la división de todos los servicios de seguridad presentados en tres grupos u objetivos directrices. El segundo punto es la definición de un modelo de referencia que incluya todos los agentes implicados en una comunicación multicast y contemple tanto las posibles interacciones entre ellos como los objetivos directrices que deben cumplir.

2.3.1. Objetivos directrices de la seguridad en multicast

A fin de abordar la seguridad en multicast de forma sistemática, el IETF dividió los requisitos de seguridad de grupos en tres áreas problemáticas distintas [Hardjono et al., 2000b]: manipulado de los datos, gestión del material de claves y gestión de políticas.

2.3.1.1. Área Problemática 1: Manipulación de los datos (*Multicast Data Handling*)

Dentro del área problemática de manipulación de los datos se engloban todos los servicios que para ser implementados requieren de una transformación criptográfica de los datos. En multicast seguro los datos necesitan estar encriptados usando una clave de grupo, así se solucionan principalmente el control de acceso y la confidencialidad. Además, los datos también deben estar autenticados para verificar la fuente y su integridad, para ello normalmente se realizan *hashes* y firmas sobre los mismos.

La autenticación tiene dos aspectos:

- Autenticación de la fuente e integridad de los datos: garantiza que los datos se han originado en la fuente requerida y no se han modificado por el camino (por un miembro del grupo o por un atacante externo).
- Autenticación de grupo: garantiza que los datos se han generado (o modificado) por un miembro del grupo. No garantiza integridad de los datos si no hay confiabilidad con todos los participantes.

El cifrado multicast y la autenticación de grupo son bastante estándares y similares al cifrado y autenticación unicast. Existen dos posibles soluciones para el cifrado de datos en mutlicast. La primera consiste en adaptar soluciones existentes ajenas a la aplicación. En este sentido las soluciones disponibles para cifrado en unicast (IPSec, TLS, etc.) pueden ser suficientes. La segunda integra el cifrado en la misma aplicación. En estos casos, deben realizarse consideraciones referentes a la naturaleza no fiable del transporte multicast y a la latencia permitida de los datos [Pegueroles and Rico-Novella, 2002b]. La autenticación de la fuente en multicast, en cambio, es considerablemente más compleja, necesita algoritmos especiales, y queda fuera del propósito de esta tesis.

2.3.1.2. Área Problemática 2: Gestión de material de claves (*Management of Keying Material*)

Para conseguir los objetivos del área problemática 1 es imprescindible el uso de claves criptográficas. Con el término material de claves (del inglés *keying material*) hacemos referencia tanto a las claves criptográficas del grupo como al estado asociado a cada clave (en uso, obsoleta, comprometida, en fase de actualización...) y demás parámetros de seguridad relacionados con ellas. Por tanto, la gestión de las claves del grupo también requiere necesariamente la gestión de su estado y de sus parámetros asociados.

El área problemática 2 engloba todo el estudio y objetivos referentes a los siguientes problemas:

- Gestión de la identificación y autenticación de los miembros.
- Gestión de la identificación y autenticación del grupo.
- Gestión del establecimiento de un canal de confidencialidad a corto plazo.
- Gestión del establecimiento de un canal de confidencialidad a largo plazo.

- Métodos para efectuar el cambio de material de claves.
- Métodos para detectar y señalar los fallos y percibir qué cantidad y qué parte del material de claves se ha visto comprometido.

Los capítulos 3 y 4 de esta tesis doctoral aportan soluciones prácticas en esta área problemática. Las necesidades relacionadas con la gestión del material de claves debe considerarse siempre dentro del contexto de las políticas concretas adoptadas.

2.3.1.3. Área Problemática 3: Gestión de políticas (*Policy Management*)

Las políticas de seguridad multicast son los conjuntos de reglas que deben seguir todos los participantes en la comunicación de grupo para realizar todas las operaciones implicadas en las áreas problemáticas 1 y 2. Este tercer grupo de objetivos directrices engloba todo lo referente a la gestión, distribución y acuerdo de estas reglas entre los comunicantes.

Casi todo el trabajo existente en cuanto a políticas se basa en la existencia de una entidad Controladora de Políticas (al igual que el área problemática 2 se basa en la existencia de un Gestor de Claves centralizado). El problema de dinamismo no es grave en este entorno ya que normalmente la política se decide al inicio del grupo y se mantiene durante toda la vida del mismo. Además, la mayoría de soluciones exitentes en gestión de políticas ya estandarizadas por el IETF son aplicables a entornos multicast [Dinsmore et al., 2000]. La gestión de la política de seguridad multicast sólo amplia los conceptos desarrollados por las comunicaciones unicast en las áreas de:

- Creación de la política.
- Traducción a alto nivel de la política.
- Representación de la política.

La creación de las políticas de seguridad en multicast añade la posibilidad de que los miembros que acuerdan una política puedan extenderse por diversos dominios administratrivos (cada uno con sus reglas locales correspondientes) o incluso que un mismo usuario pueda abarcar varios dominios (en entornos móviles puede ser altamente útil).

Existen diversos métodos para la creación de una única política de grupo segura y coherente. Éstos incluyen una especificación de la política de grupo del grupo iniciador y una negociación de la política entre los miembros del grupo. Esta negociación puede ser tan simple como una estricta intersección de las políticas de los distintos miembros o una más compleja que usase sistemas de votación por pesos.

Las traducciones a lenguaje de alto nivel de las políticas son complejas en entornos multicast, especialmente cuando el grupo de miembros pertenecen a distintos grupos administrativos. Así, cuando las políticas se especifiquen a alto nivel, deben darse también las reglas precisas de traducción a bajo nivel correspondientes a cada dominio administrativo.

En la representación de las políticas de seguridad multicast se debe incluir más información que en las políticas unicast. Además de representar los mecanismos de seguridad para la comunicación del grupo, la política también debe representar las reglas para el gobierno del

grupo seguro. La política ha de establecer tanto las operaciones básicas de alta y baja de miembros, como las más avanzadas de partición del grupo, re-unión y resincronización.

Aunque la mayoría de problemas referentes al área problemática 3 están resueltos para los escenarios actuales, últimamente, la fuerte irrupción de entornos móviles y redes *ad-hoc* ha puesto de manifiesto la existencia de áreas de trabajo totalmente inexploradas aún a fecha de escritura de este documento. De todos modos, esta área problemática queda fuera del propósito de esta tesis doctoral.

2.3.2. Modelo de referencia

El segundo eje para particularizar las comunicaciones de grupo a multicast es la definición de un modelo de referencia. Éste incorpora las entidades que intervienen en una comunicación multicast y asigna a cada una de ellas las áreas problemáticas de las que se encargará. A su vez, representa de forma esquemática las relaciones existentes entre estas entidades, cada una de estas relaciones supone la definición de un protocolo.

El modelo de referencia ayuda a enfocar la seguridad desde la perspectiva de la arquitectura (centralizada y distribuida), del tipo de multicast (1 a M o M a M) y de los protocolos (mensajes de intercambio), todo según la clasificación de las áreas problemáticas. Su objetivo es proporcionar un contexto general donde los problemas puedan ser identificados y clasificados, y se puedan reconocer las relaciones entre ellos.

El esquema aceptado por el IETF como modelo de referencia es el presentado en la Fig 2.1 [Hardjono et al., 2000b]. Debe remarcarse que las cajas no se corresponden necesariamente con entidades individuales reales que implementan una determinada función. Una caja en el modelo de referencia se debe interpretar libremente como perteneciente a una función de alguna área problemática. Si esta funcionalidad está implementada por una o más entidades dependerá de la solución particular. Por ejemplo, la caja Servidor de Claves o *Key Server* agrupa las funcionalidades de la gestión de claves y, de manera similar, el modelo de referencia acepta que algunas implementaciones puedan agrupar cajas en una sola entidad física.

El modelo de referencia se puede interpretar de forma horizontal y vertical. Horizontalmente, muestra las entidades y funciones como cajas particulares que dan solución a cada una de las áreas problemáticas. Verticalmente, muestra los diseños de arquitecturas básicas, denominadas centralizada y distribuida.

Los protocolos que se estandarizan se representan en la Fig. 2.1 como las flechas que conectan las distintas cajas.

2.3.2.1. Elementos del Modelo de referencia

El diagrama del modelo de referencia contiene cajas que simbolizan las entidades funcionales y flechas que representan los interfaces entre ellas. Los protocolos estándares se corresponden con los interfaces, que dan soporte a los servicios multicast que se ofrecen entre entidades funcionales.

Existen tres entidades funcionales tanto en los diseños centralizados como distribuidos.

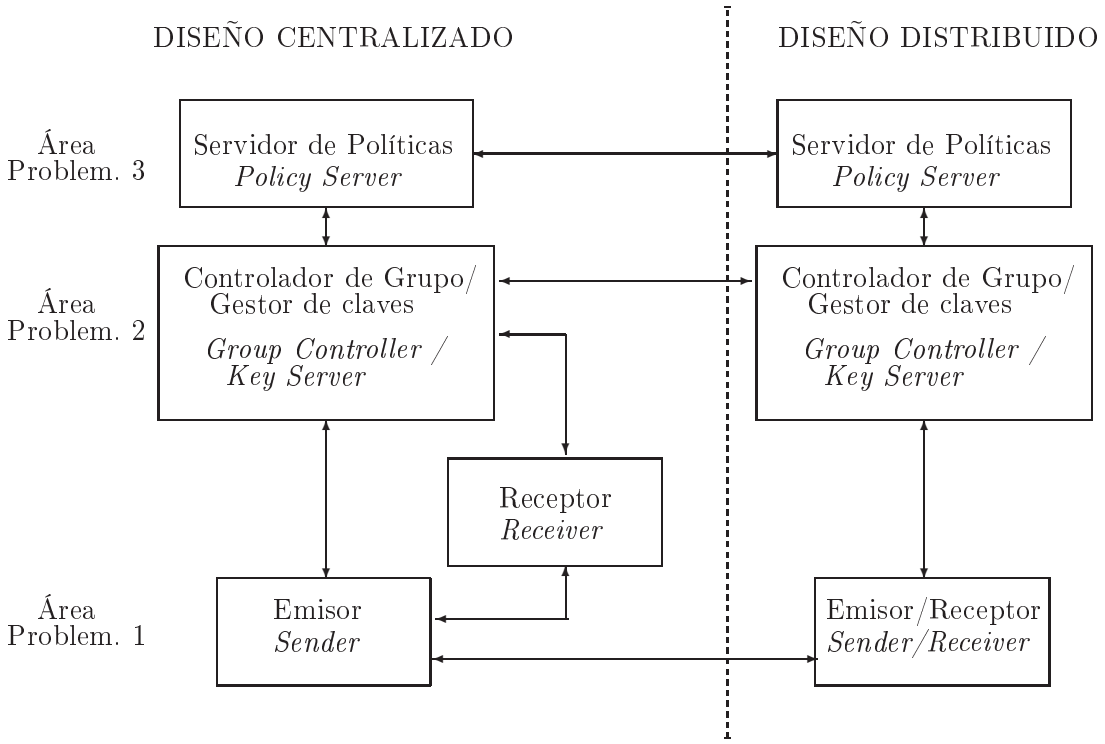


Figura 2.1: Modelo de referencia de entornos de seguridad en multicast

Servidor de claves *Key Server* **y Controlador de grupo** *Group Controller* **(GCKS**[1]**)**

El servidor de claves y controlador de grupo (*Key Server and Group Controller*) es la entidad con las funciones referentes a la emisión y gestión de claves criptográficas en un grupo multicast. Éste además deberá revisar la autenticidad y autorización de usuarios.

En una arquitectura distribuida, el GCKS interactúa con otras entidades GCKS para conseguir escalabilidad en los servicios relacionados con la gestión de claves. En este caso cada miembro del grupo multicast puede operar con una o más entidades GCKS. De la misma forma en una arquitectura distribuida un GCKS puede relacionarse con uno o más servidores de política.

Debe remarcarse que el servidor de claves (KS) y el controlador de grupo (GC) tienen funcionalidades distintas y en principio pueden verse como entidades diferentes. Actualmente, el modelo unifica las dos entidades en una sola caja para simplificar el diseño y por no forzar a la estandarización de un protocolo de comunicación entre el KS y el GC.

Emisor *Sender* **/Receptor** *Receiver* [2]

En un grupo multicast 1 a N sólo se permite a un único remitente transmitir datos al grupo. En un grupo multicast M a M muchos miembros del grupo (o todos) pueden transmitir datos al grupo.

El remitente y el receptor han de ponerse conjuntamente de acuerdo con el GCKS para gestionar las claves. Esto incluye la autenticación de usuario, la obtención del material de las claves en concordancia con la política de gestión de claves del grupo, la obtención de nuevas

[1] A efectos de nomenclatura, en todo el cuerpo de la tesis se usarán indistintamente los términos Servidor de Claves (*Key Server o KS*) y Controlador de Grupo (*Group Controller o GC*) o su variante combinada GCKS.

[2] Cuando se haga referencia a ellos de forma genérica se les denominará "miembros"

claves durante la actualización de las mismas y la obtención de otros mensajes relacionados con la gestión del material de claves y los parámetros de seguridad.

El acuerdo de política por parte del remitente y el receptor se da indirectamente a través del GCKS. La unión a un grupo multicast implica la adquisición del material de claves desde el GCKS, y por tanto, a través de él se asumen las reglas del grupo. Esto no impide que remitente/receptor puedan interacccionar directamente con el servidor de política en caso en que, por ejemplo, quieran verificar la política obtenida a través del GCKS.

El modelo de referencia muestra dos cajas de receptor, una se corresponde con la situación en la que emisor y receptor utilizan la misma entidad GCKS (arquitectura centralizada) y la otra con la que remitente y receptor usan entidades GCKS distintas (arquitectura distribuida).

Servidor de políticas *Policy Server*

El servidor de política es la entidad con las funciones de crear y gestionar las políticas de seguridad específicas para cada grupo multicast. El servidor de política interactúa con la entidad GCKS para instalar las políticas seguras relacionadas con los miembros y el material de claves.

Las interacciones entre el servidor de política y otras entidades en el modelo de referencia (por ejemplo con emisores y receptores) dependen de los niveles de seguridad de la política concreta.

2.3.2.2. Diseño centralizado y distribuido

La necesidad de soluciones escalables para grupos grandes, que a su vez puedan comprender distintas regiones geográficas en internet, requiere que los elementos del modelo de referencia funcionen también como un sistema distribuido. Esto implica que una entidad GCKS debe poder interactuar de manera segura con otra entidad GCKS de otra localización. De manera similar, los Servidores de Política deben interactuar con otros de forma segura para permitir la comunicación y aplicación de las políticas a través de Internet.

2.3.2.3. Bloques funcionales

Para reducir la complejidad de la seguridad en multicast y facilitar su implementación, el IETF decidió dividir las áreas problemáticas ligadas a cada entidad del modelo de referencia en bloques funcionales. Los distintos bloques funcionales agrupan algunos de los servicios de seguridad discutidos en 2.2.2 y ofrecen una descripción a alto nivel de cómo se deben solucionar. Esta descripción es más abstracta que los protocolos o los algoritmos concretos (como Diffie-Hellman o RSA), con ello se consigue la reutilización de protocolos existentes para nuestro cometido (Diffie-Hellman y RSA, por ejemplo, también se pueden usar para alguno de los bloques funcionales de seguridad en multicast). Los bloques funcionales, además, deberán ser independientes, es decir, la ejecución de uno no debe afectar a la ejecución de los demás.

Confidencialidad de los datos multicast

Este bloque se encarga del cifrado de los datos multicast en el emisor y el descifrado en el receptor. Presumiblemente deberá usar el material de claves que le proporcionará el bloque

funcional de Gestión de Claves y deberá hacerlo de acuerdo a la política del bloque funcional de gestión de políticas, pero debe ser independiente de estos dos.

Un importante trabajo a tratar en este bloque funcional es la especificación e identificación de cifradores apropiados a datos multicast. Como es obvio, no todos los cifradores serán adecuados para cifrar multicast (a nivel de IP o a nivel de aplicación). Los cifradores en flujo no sincronizantes, por ejemplo, son especialmente nocivos para transmisiones no fiables. Sin embargo, la combinación de algoritmos en flujo con algoritmos en bloque o los algoritmos en flujo autosincronizantes pueden llegar a ser más exitosos que la aplicación estricta de algoritmos en bloque. Las tareas de investigación en este bloque funcional deben ir orientadas en este sentido.

Este bloque, en el modelo de referencia de la Fig 2.1, se sitúa en el área problemática 1, en el interfaz entre los emisores y los receptores.

Autenticación de la fuente multicast e integridad de los datos

Este bloque trata de la autenticación de la fuente y la integridad de los datos multicast. Debe encargarse de las transformaciones criptográficas a realizar por el emisor y receptor a tal efecto. Al igual que en el bloque funcional anterior, se asume que el bloque de Gestión de Claves proporciona las claves necesarias para realizar las firmas digitales y sus correspondientes verificaciones de acuerdo con las normas del bloque de Gestión de Políticas. El trabajo en esta área se presume crítico debido a los requerimientos de conexión y tiempo real de la mayoría de aplicaciones sobre IP multicast.

Este bloque, en el modelo de referencia de la Fig 2.1, se sitúa en el área problemática 1, en el interfaz entre los emisores y los receptores.

Autenticación de grupo multicast

Este bloque debe encargarse de proporcionar una cierta autenticidad a los datos transmitidos: sólo garantiza que los datos se han originado (o han sido modificados en última instancia) por un miembro del grupo. No garantiza la autenticidad si existe algún miembro del grupo que no es fiable.

La ventaja que presenta la autenticidad de grupo es su simplicidad y que se puede basar en protocolos criptográficos muy eficientes ya existentes. Por estos motivos, cuando la autenticación de fuente no es estrictamente necesaria, la autenticación de grupo nos da las funcionalidades necesarias. Además, la autenticación de grupo es útil en aquellos casos en los que la autenticación de fuente se realiza a posteriori, ya que provee de una prueba de integridad débil, muy útil contra ataques de negación de servicio.

El bloque de autenticación de grupo multicast se enmarca en el área problemática 1 en el interfaz entre emisores y receptores.

Gestión de Miembros del grupo multicast

Este bloque describe las funcionalidades que permite a los miembros de un grupo darse de alta y de baja del mismo.

El alta (o registro) de un miembro incluye la autenticación del miembro, la notificación y negociación de los parámetros de seguridad, y el mantenimiento de *logs*; todo según la política acordada entre el Controlador de Grupo y el futuro miembro. En la literatura es usual referirse a esta acción con el anglicismo *joining*.

Normalmente, y de forma previa a que un usuario desee registrarse, se ha realizado un anuncio de los servicios del grupo.

La baja de un miembro la puede iniciar el mismo miembro o el controlador. Consiste en el mantenimiento del *log* de la baja y en las acciones oportunas por parte del controlador de grupo para expulsar al miembro. Al igual que en al alta, en numerosos trabajos se hace referencia a esta acción con el término inglés *leaving*.

En el caso de diseño distribuido, este bloque también describe la comunicación entre los distintos servidores GCKS que esté relacionada con la gestión de los miembros.

En la Fig. 2.1, el bloque de Gestión de los Miembros está situado en el área problemática 2 en los interfaces entre el GCKS y los emisores y receptores.

Gestión de Claves multicast

Este bloque describe la distribución y actualización del material de claves criptográficas a lo largo de la vida del grupo. Los componentes de este bloque incluyen:

- Notificación del GCKS al cliente sobre el material de claves actual (por ejemplo, cifrado de grupo y claves de autenticación, claves auxiliares usadas para la gestión de grupo, claves de autenticidad de la fuente, etc.)
- Actualizaciones del material de claves , dependiendo de las circunstancias y las políticas.
- Finalización de los grupos de forma segura, incluyendo el mismo grupo multicast y el material de claves asociado.

En este bloque se solucionan los problemas de gestión segura de las claves entre el servidor de claves (KS) y los clientes, la administración de la distribución multicast del material de claves y la escalabilidad u otros requisitos de funcionamiento para la gestión de claves multicast. Además, como la capa IP multicast y la de aplicación se pueden solapar, este bloque debe considerar tanto la gestión de claves en IP multicast, como la gestión de claves para la capa de aplicación y decidir qué requisitos del bloque de gestión de claves multicast satisface a cada uno.

En un diseño distribuido, este bloque también debe ocuparse de la comunicación entre distintos GCKS siempre que esté relacionada con la gestión de claves.

El bloque de Gestión de Claves multicast aparece en los diseños centralizados y distribuidos mostrados en la Fig 2.1 y está situado en el área problemática 2.

Gestión de políticas multicast

Este bloque trata todos los temas relacionados con la política de grupo multicast incluyendo políticas de miembros y de gestión de claves. También incluye el diseño del servidor de políticas, las definiciones concretas a usar por IP multicast y la capa de aplicación, y los protocolos de comunicación entre el KS y el servidor de Política. La información que se transmiten

estos últimos incluye las políticas de los grupos, los miembros, la definición del material de claves y su uso y demás información.

En un diseño distribuido, este bloque también debe describir la comunicación entre los distintos servidores de Políticas.

El bloque de Gestión de Políticas multicast está situado en el área problemática 3 en el interfaz entre los servidores de claves y los servidores de políticas. No se necesita que los miembros del grupo participen directamente de este bloque, aunque no se desecha esta posibilidad.

2.4. Escenarios de referencia

Para finalizar la particularización de las comunicaciones de grupo a multicast, en esta sección se reagrupan las características de los distintos tipos de grupos a estudiar presentadas en 2.2.1 a tres escenarios de referencia. Éstos se usarán para evaluar las propuestas de algoritmos de gestión de claves expuestos en esta Tesis.

En [Canetti et al., 1999a] sólo se consideran dos posibles escenarios para la simulación y evaluación de sistemas multicast seguros: difusión con un único emisor (o *Single Source Broadcast*) y videoconferencias (*Virtual conferences*). Sin embargo, a fin de tener un abanico de escenarios que contemplen todas las características, se ha introducido un tercero: Juegos en red (*Multiplayer networked games*).

A continuación se describen las características que reúnen cada uno de ellos y se presentan ejemplos de perfiles de comportamiento. En el Anexo A se presenta el trabajo realizado en cuanto a caracterización para generar perfiles sintéticos, éste permitió la simulación de los algoritmos y la obtención de sus parámetros de eficiencia.

2.4.1. Difusión con emisor único (*Single Source Broadcast*)

Se corresponde con el escenario real más antiguo, consiste en una única fuente emisora de datos que envía continuamente flujo de información a un número muy elevado de receptores. El ejemplo más claro de este escenario es la televisión en red (*web-tv*).

El número de integrantes del grupo puede llegar a ser de cientos de miles o incluso más. Normalmente, el tiempo de vida de grupo suele ser largo. Los receptores en el grupo son dinámicos (no así la fuente): pueden unirse al grupo y abandonarlo a voluntad y con una tasa relativamente alta. La capacidad computacional de los clientes puede variar mucho, puede tratarse desde *set top box* estáticos con gran capacidad de cálculo hasta terminales móviles con recursos muy limitados. Casi siempre las características de los miembros son heterogéneas. En general la atención de los participantes es máxima, pero en algunas aplicaciones se puede requerir que los receptores presten atenciones intermitentes. Pueden llegar a darse ráfagas de peticiones de alta o baja en horas punta (que normalmente se corresponden con el inicio o finalización de un evento previamente anunciado). El tipo de datos también puede variar mucho de unas aplicaciones a otras, en el ejemplo de *web-tv*, por ejemplo, se tratará de flujos multimedia (video, audio...); en la difusión de noticias, por ejemplo, o transmisiones militares, puede requerirse sólo la transmisión de texto.

Como ejemplo, la Tabla 2.1 muestra valores del número de peticiones de alta, duración

media de una conexión y tiempo medio entre llegadas para distintas sesiones en un servidor multicast. Los datos extraidos de [Almeroth and Ammar, 1996] son el resultado de medidas empíricas sobre la red MBone. Estos valores dan una idea más concreta de las características que apuntamos en el párrafo anterior.

Tabla 2.1: Estadísticas del comportamiento de usuarios en servidor multicast de la MBone

Nombre de Sesión	Número de Altas	% Altas del total	Tiempo medio entre llegadas	Duración media
NASA:STS-75	4230	27.55	7.51 min	4.3 hrs
IMS: World Radio Network	1998	13.01	22.22 min	1.7 hrs
IETF-1 Audio	1343	8.75	22.71 min	2.6 hrs
IETF-2 Audio	1234	8.04	38.87 min	2.4 hrs
Radio Free Vat	1179	7.68	38.94 min	49 min
MBone Audio	839	5.46	53.13 min	4.4. hrs
FreeBSD Lounge	712	5.33	67.57 min	35 min
Otras Sesiones IMS	1314	8.56	56.00 min	2.0 hrs
Sesiones de corta duración	2507	16.33	18.11 min	42 min
Totales	15356	100 %	1.24 min	2.5 hrs

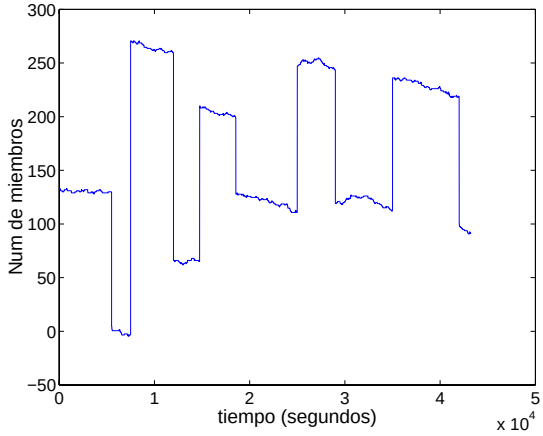
A partir del trabajo anterior, [Almeroth and Ammar, 1997] iniciaron la caracterización del comportamiento de usuarios para sesiones multicast con una única fuente. Las conclusiones a las que llegaron se han usado para generar perfiles de comportamiento sintético que permitan la simulación de los algoritmos (Anexo A).

La Fig 2.2 muestra un ejemplo de patrón sintético acorde al modelo *Single Source Broadcast*. En las subgráficas se representa tanto la evolución en número de miembros en el servidor multicast (a) como el tiempo medio entre llegadas al servidor (b) y el histograma del tiempo entre llegadas (c).

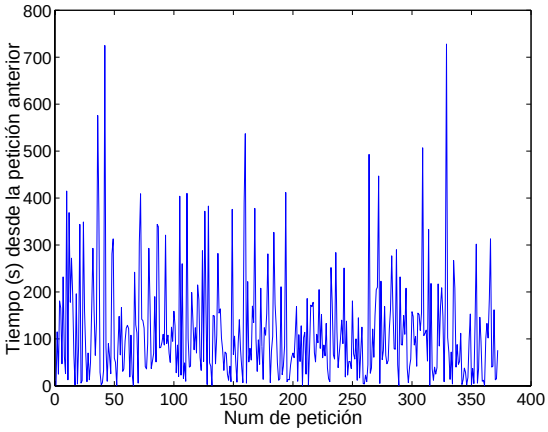
2.4.2. Videoconferencias (*Virtual conferences*)

Las videoconferencias son también casos clásicos de entornos multicast, ejemplos de aplicación son las reuniones corporativas on-line y clases interactivas a distancia, etc. Una videoconferencia normalmente implica a decenas o centenares de interlocutores, la mayoría de los cuales, en casi todos los casos, actuarán como emisores y receptores (o al menos querrán tener la posibilidad). El grupo se forma usualmente por evento y su tiempo de vida es relativamente corto (pocas horas o incluso minutos). La dinámica del grupo es baja: los miembros se conectan al inicio de la conferencia y permanecen dados de alta durante toda la vida del grupo. En cuanto a características de los miembros, suelen tener una capacidad computacional relativamente elevada y su atención es máxima. Los datos enviados suelen limitarse a tres tipos: audio, vídeo e instrucciones de pizarra virtual.

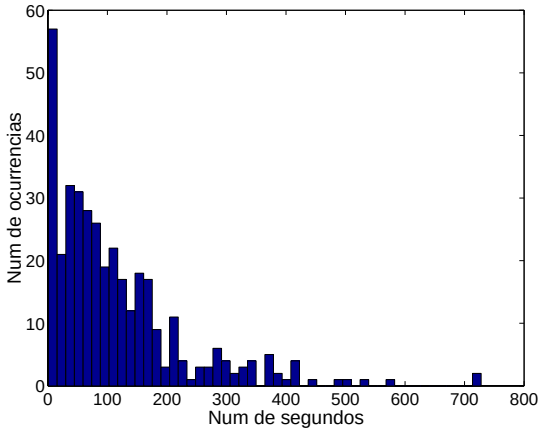
Debido a sus caracterísitcas, este tipo de grupo no presenta mucha problemática en cuanto a su gestión (dinámica baja y suficientes recursos por parte del cliente). Puede que ello sea el motivo de que no hayamos encontrado estudios de medidas empíricas del comportamiento de usuarios en estos entornos. De cualquier forma, y para poder obtener medidas de la eficiencia



(a) Evolución del número de miembros en el grupo



(b) Tiempo entre llegadas



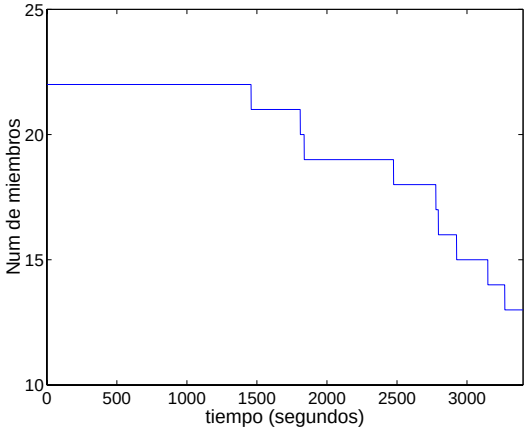
(c) Histograma del tiempo entre llegadas

Figura 2.2: Patrón sintético de comportamiento de usuario según *Single Source Broadcast*

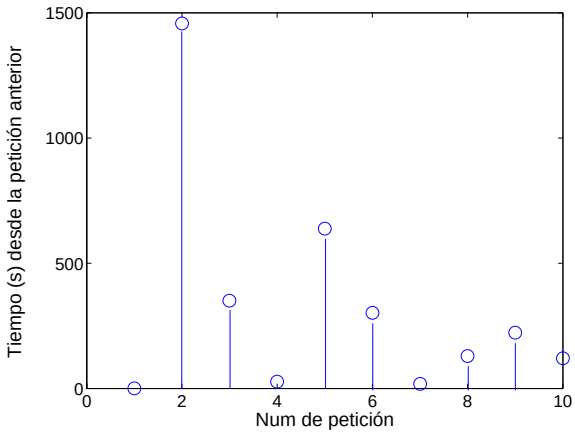
de nuestras propuestas para todos los escenarios posibles, desarrollamos un modelo muy simple explicado en el Anexo A.

El modelo distingue dos fases en toda videoconferencia: la fase de exposición, en que todos los participantes que quieren hablar van haciéndolo de forma ordenada y la fase de debate, en que se realizan réplicas y se inician discusiones sobre las exposiciones presentadas. El modelo sigue el comportamiento habitual en cualquier reunión tanto sea presencial como virtual. La primera fase se corresponde con el caso menos dinámico y de atención máxima y, por tanto, más fácil de gestionar. La fase de debate incluye decrementos en la atención de los miembros, que pueden llegar a pedir la baja o a subdividirse en grupos más reducidos de discusiones más particulares, presentando una dinámica superior a la de la fase de exposición.

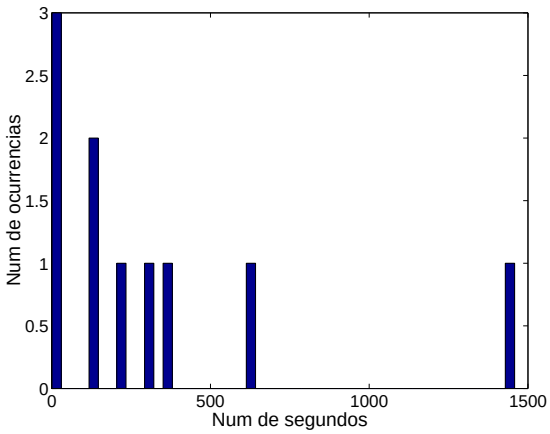
En la Fig 2.3, al igual que en el escenario *Single Source Broadcast*, se muestra un ejemplo de patrón sintético acorde al modelo *Virtual conferences*.



(a) Evolución del número de miembros en el grupo



(b) Tiempo entre llegadas



(c) Histograma del tiempo entre llegadas

Figura 2.3: Patrón sintético de comportamiento de usuario según *Virtual conferences*

2.4.3. Juegos en red (*Multiplayer networked games*)

El tercer escenario, no contemplado por el IETF, pero que por sus características, es especialmente interesante para el simulado de algoritmos de seguridad en multicast, es el de juegos en red con múltiples jugadores (*Multiplayer networked games*).

En este tipo de grupos suelen participar unas decenas de jugadores y no suelen llegar nunca a centenares, aunque en servidores con distintos juegos sí que se puede llegar a estas cifras si se considera los usuarios de forma conjunta. El tiempo de vida del grupo es muy variable y puede ir desde pocos minutos (si la partida no ha tenido éxito) hasta varias horas, días e incluso semanas, si se van dando de alta y baja nuevos jugadores al ver que una partida tiene éxito. Los clientes prestan atención intermitente (un jugador puede desear hacer un descanso durante la partida) y se debe diseñar el sistema para que soporte dispositivos con capacidad de cómputo muy reducida (por ejemplo terminales de teléfono móviles).

Existen dos modos de funcionamiento en cuanto a número de emisores multicast. El

primero en el que toda la comunicación pasa a través de un servidor central se corresponde con *Single Source Broacast* (ya que la respuesta de los miembros al servidor no tiene porque ser multicast). El segundo es que todos los usuarios sean capaces de interpretar las instrucciones de otros usuarios y todos emitan a todos, en cuyo caso el número y tipo de servidores es el mismo que en *Virtual conferences*. De cualquier forma, las demás características de este tipo de grupos no se corresponden con los modelos anteriores.

El tipo de datos emitido suele ser una combinación de texto (o instrucciones al cliente de juego para que muestre cierto comportamiento) y contenido multimedia (sonidos y melodías nuevas, figuras y objetos animados...). En cualquier caso, lo más característico en este escenario es su dinámica.

El comportamiento de usuario en los juegos en red tiene unas peculiaridades que no presentan los otros escenarios. En primer lugar, el número de usuarios y el tiempo entre llegadas al sistema presentan una relación de proporcionalidad inversa, es decir, el alta por parte de usuarios propicia que otros también se den de alta (un juego en red con muchos participantes es más atractivo que uno con pocos jugadores). A su vez, la presencia de otros usuarios en el servidor también afecta de forma inversa al tiempo de conexión (que haya muchos jugadores en un servidor incrementa la posibilidad de que fuercen mi baja).

En [Henderson and Bhatti, 2001] se encontró que el tiempo entre llegadas en los juegos en red sigue una distribución *heavy-tailed*. Estos resultados sirvieron para la generación de patrones de usuarios usados en las simulaciones. La explicación de como se consiguieron se detalla en el Anexo A.

En la Fig 2.4, como se ha hecho con los otros escenarios, se muestra un ejemplo de patrón sintético acorde al modelo *Multiplayer networked games*.

2.5. Parámetros para la evaluación

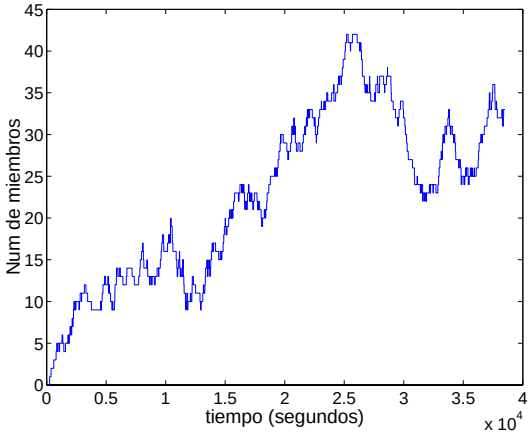
Para conseguir los objetivos directrices de multicast (2.3.1) deben implementarse los distintos bloques funcionales (descritos en 2.3.2.3). Ello dará lugar a distintas propuestas de algoritmos y protocolos que será necesario comparar para decidir qué opción es mejor en cada caso. A continuación se presentan los parámetros de eficiencia que se usarán para evaluar los algoritmos expuestos en esta tesis. La clasificación se ha realizado en función de los elementos emisor/receptor y GCKS del modelo de referencia (2.3.2). Los parámetros deberán evaluarse para los distintos escenarios expuestos en 2.4.

2.5.1. Parámetros genéricos

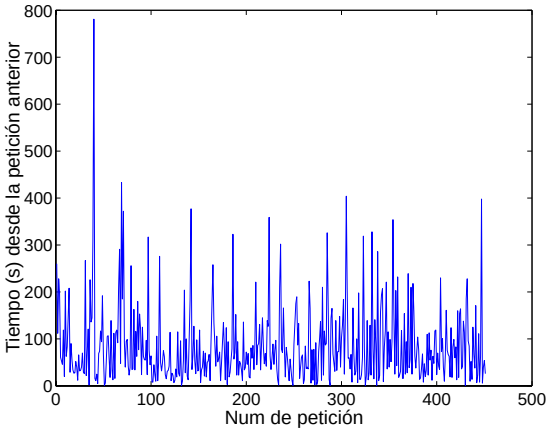
En primer lugar se presentan parámetros generales de algoritmo/protocolo, poco cuantificables, pero que su consideración puede ser importante para según que casos.

Impacto de orden N de acciones individuales (en la literatura *1 afects n scalability*)

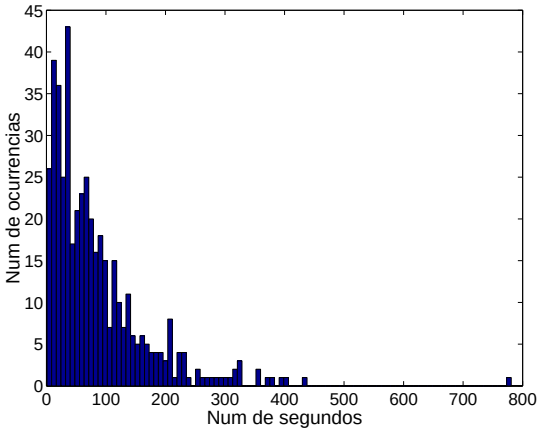
Este parámetro refleja como afecta al resto del grupo las acciones de un elemento de forma individual. Si una alta o baja en el grupo requiere acciones del algoritmo para con los N miembros del grupo diremos que sí presenta un impacto de orden N de acciones individuales.



(a) Evolución del número de miembros en el grupo



(b) Tiempo entre llegadas



(c) Histograma del tiempo entre llegadas

Figura 2.4: Patrón sintético de comportamiento de usuario según *Multiplayer video games*

Confianza en nodos intermedios

Un algoritmo requerirá de confianza en los nodos intermedios si los elementos del modelo de referencia deben fiarse de los nodos de transporte de la red para garantizar servicios de seguridad.

Vulnerabilidad a ataques de confabulación

Un algoritmo o protocolo se considerará vulnerable a ataques de confabulación si un grupo de miembros (autorizados o no) del grupo pueden manipular conjuntamente su información privilegiada para acceder a información para la que no estaban autorizados.

Control de congestión

Diremos que un protocolo/algoritmo tiene control de congestión si es capaz de soportar ráfagas de solicitudes comprendidas entre el tiempo de *sign-on* y *sign-off* máximo o recuperarse de los efectos nocivos que ellas le hayan podido causar. Un posible escenario donde el control de congestión es crítico es una difusión en tiempo real donde muchos miembros se unen al grupo justo antes de que empiece la difusión y se dan de baja a los pocos instantes de que ésta haya concluido.

2.5.2. Desde el punto de vista del canal

Desde el punto de vista del canal o de las transmisiones entre los elementos del modelo de referencia, la eficiencia se mide según propiedades de los mensajes de control. Estos valores de eficiencia ya son cuantificables, las unidades más habituales también se presentan a continuación.

Ancho de banda de paquetes de control

Es el ancho de banda requerido para realizar una determinada acción en un protocolo. Éste es uno de los parámetros críticos y que más se tienen en cuenta al diseñar un algoritmo (de renegociación de claves, por ejemplo). Muchas veces se usa el término "ancho de banda del algoritmo" cuando en realidad se hace referencia a este parámetro.

Se mide, normalmente, en número de bits de los mensajes de control.

Latencia de mensaje de control

Es el tiempo que transcurre desde que se envía un mensaje de control hasta que la acción que contiene puede hacerse efectiva. Por ejemplo, en un algoritmo de renegociación de claves es el tiempo que transcurre desde que el GCKS notifica que se debe cambiar la clave de sesión hasta que el cambio (en los paquetes de datos) puede realizarse.

Aunque por definición su unidad sean segundos (o fracciones de segundo), en la mayoría de estudios consultados se mide en número de paquetes. Esto es debido a que en esos estudios se considera el canal ideal y con velocidad infinita, por tanto la distancia y el tiempo de propagación se reducen a cero, y lo único que debe considerarse es cuántos paquetes debe enviar un centro de control hasta que pueda hacer efectiva la orden.

Workoverhead de paquetes de control

Este parámetro da una idea de lo costoso que puede ser dividir una orden de control en dos o más paquetes. Debido al *overhead* de cada paquete, el número de bits efectivos que deben transmitirse (*bits on the wire*) será superior si la misma orden se envía dividida en 2 paquetes que si se envía sólo en uno.

Su unidad de medida es el número de bits.

2.5.3. Desde el punto de vista del Gestor de Claves

Respecto al Gestor de Claves, los parámetros de eficiencia miden el coste en cuanto a recursos que necesita esta entidad para realizar las acciones que le corresponden.

Coste computacional de Inicialización de grupo

Normalmente se considera el número de operaciones criptográficas necesarias para establecer una comunicación segura de grupo. Obviamente, el coste computacional real dependerá del algoritmo criptográfico usado y del número de miembros del grupo. Por eso sólo deben compararse evaluaciones que usen los mismos criterios.

Cantidad de memoria necesaria para mantener el grupo

Es el número de bits que necesita almacenar el Gestor de Claves en su memoria para poder realizar las operaciones de mantenimiento del grupo. Normalmente, este parámetro también presenta una dependencia en el número de miembros, por eso para comparar evaluaciones se proporcionan expresiones abiertas en función de N.

Coste computacional de Alta de miembro

Se define como el número de operaciones criptográficas a realizar por el Gestor de Claves cada vez que un miembro se da de alta en el grupo.

Coste computacional de Baja de miembro

Se define como el número de operaciones criptográficas a realizar por el Gestor de Claves cada vez que un miembro se da de baja en el grupo.

2.5.4. Desde el punto de vista del Miembro

Finalmente, de forma muy similar a lo realizado con el Gestor de Claves, se presentan los parámetros de eficiencia referentes a los miembros.

Inicialización de remitente

Hace referencia al número de acciones (generalmente operaciones criptográficas) que debe realizar el remitente cuando empieza a transmitir al grupo.

Cantidad de memoria necesaria para mantenerse en el grupo

Es el número de bits que necesita almacenar en su memoria un miembro del grupo para poder pertenecer a él.

Coste computacional de Alta

Se define como el número de operaciones criptográficas a realizar por el miembro cuando se da de alta en el grupo.

Coste computacional de Baja

Se define como el número de operaciones criptográficas a realizar por el miembro cuando se da de baja en el grupo.

Trabajo de reactivación

En los casos en los que la atención del miembro es intermitente, el trabajo de reactivación refleja el esfuerzo que supone para un miembro (en cuanto a operaciones criptográficas) volver a estar activo después de estar un tiempo inactivo (off-line).

Capítulo 3

Gestión de claves en grupos multicast

3.1. Introducción

Tal como se introdujo en el Capítulo 2, el segundo de los objetivos directores de la seguridad en multicast es la gestión del material de claves. Este término (en inglés *Keying Material*) hace referencia tanto a las claves con las que se cifran los datos como a su estado (por ejemplo si una clave está obsoleta o se ha visto comprometida), y demás información necesaria para poder gestionar la clave de grupo.

Con el uso de claves, y sus mecanismos de gestión relacionados, debemos ser capaces de:

1. Identificar y autenticar a los miembros del grupo.
2. Identificar y autenticar al grupo de miembros.
3. Establecer canales seguros entre el GCKS y los miembros, que soporten tanto confidencialidad a corto plazo (*Ephemeral Secrecy*) como confidencialidad a largo plazo (*Long term secrecy*).
4. Efectuar el cambio del material de claves que se ha visto comprometido, en inglés *rekeying*
5. Detectar, señalar fallos y percibir qué parte del material de claves se ha visto comprometido.

Los puntos 1 y 3 hacen referencia a acciones en las que sólo intervienen 2 entidades: el *Key Server* y los emisores/receptores, y pueden considerarse interacciones punto a punto. En estos casos, mecanismos clásicos de identificación y acuerdo de claves como Diffie-Hellman [Diffie and Hellman, 1976] satisfacen a la mayoría de sus requisitos.

Por su parte, tanto para el punto 2 como para el área problemática de manipulado de datos se usa una única clave de cifrado común a todos los miembros del grupo.

Una clave puede verse comprometida por causas directas o indirectas. Las causas indirectas se deben a un comportamiento malicioso de uno o un conjunto de agentes (pertenecientes o no al grupo). Las causas directas, en cambio, resultan del desarrollo normal de la actividad del grupo multicast: altas y bajas de miembros.

Para que un usuario no tenga acceso a la información que el grupo se intercambió antes de su alta (Confidencialidad Estricta hacia Atrás o *Perfect Backward Secrecy*) ni pueda descifrar

los datos que se mandan después de que él abandone el grupo (Confidencialidad Estricta hacia Adelante o *Perfect Forward Secrecy*) debe cambiarse la clave de grupo cada vez que un miembro se da de alta o de baja en el grupo. Es decir, la clave en uso se ve comprometida cada vez que se produce un cambio en la composición del grupo multicast seguro.

Por tanto, en grupos grandes y dinámicos, como los que nos ocupan, debe encontrarse una forma eficiente de realizar el cambio de material de claves. En este capítulo se enumeran los retos que supone la gestión del material de claves, se exponen las propuestas más representativas del estado del arte y se presentan alternativas y mejoras a las mismas.

3.2. El problema de la escalabilidad

El método más simple de actualizar una clave de sesión es mediante técnicas unicast. Es decir, el Gestor de Claves calcula una nueva clave de sesión y la envía a cada uno de los miembros que permanece en el grupo. La comunicación entre GCKS y miembro se realiza a través del canal seguro que deben tener establecido (normalmente de largo plazo).

El primer protocolo del IETF para la gestión de claves de grupo está basado en este comportamiento. Se denomina GKMP (*Group Key Management Protocol*) y fue propuesto por [Harney and Muckenhirn, 1997]. Este método, aunque es válido para grupos reducidos y no requiere de infrastructura adicional, es inviable cuando el número de integrantes del grupo crece o presentan un grado elevado de dinamismo. Como hemos visto en el Capítulo 2, los grupos multicast suelen tener estas dos características.

Como ejemplo, supóngase el caso del servidor multicast expuesto en la Tabla 2.1. El número total de usuarios que accedieron al servicio fue 15356. El tiempo medio entre llegadas era 1,24 min, pero el tiempo entre eventos de *rekeying* se reduce a la mitad $\simeq 40$ ya que se deben considerar las bajas. Consideramos que en media había simultánemente en el servidor poco menos de la mitad de los usuarios totales ($\simeq 7500$) y la longitud de la clave de sesión se fija en 16 bytes[1]. Con estos datos, se hubieran necesitado $\simeq 900$ Kbits cada 40 s. para distribuir las actualizaciones de las claves de sesión, sin tener en cuenta el overhead de las cabeceras de los mensajes.

Además, el sistema se bloquearía si todo el proceso de actualización tardase más de 40 s en finalizar. Si el GCKS no fuera computacionalmente capaz de realizar todas las operaciones criptográficas necesarias o los mensajes de actualización tardasen en llegar a su destino se produciría una nueva petición de alta o de baja antes de finalizarse la notificación de la actualización anterior.

Para hacer frente a este problema de escalabilidad, últimamente se han realizado propuestas de algoritmos de distribución de claves que mejoran la eficiencia de GKMP: ancho de banda de *rekeying* reducido, pocas claves a guardar por el Gestor de Claves y por los miembros, pocos mensajes a enviar por el Gestor, latencia baja, etc.

En la literatura se pueden encontrar distintas clasificaciones de estos algoritmos, una división atendiendo al número de gestores de claves implicados se encuentra en [Rafaeli, 2000] y en [Dondeti et al., 1999] se agrupan dependiendo de si la comunicación es 1 a M o M a M. En los siguientes apartados, sin embargo, se presentan los algoritmos de renegociación de

[1] clave típica de 128 bits

claves más representativos según los mecanismos empleados para reducir el ancho de banda. También se presenta su evaluación y se discuten sus ventajas e inconvenientes.

3.3. Algoritmos basados en agrupaciones físicas

Una forma lógica de simplificar la gestión de claves multicast es mediante la reducción del tamaño del grupo. Esta reducción puede conseguirse mediante reagrupaciones de miembros. Si se subdivide todo el grupo multicast en grupos más pequeños podemos dirigirnos a estos subgrupos de forma conjunta (como si fueran un único miembro) sin tener que conectarnos con ellos individualmente y la gestión de cada uno de estos subgrupos será más sencilla que la de la globalidad del conjunto.

3.3.1. *Iolus*

Una primera propuesta en ese sentido fue el Proyecto *Iolus* [Mittra, 1997]. Éste se basaba en la simple división del grupo de usuarios en subgrupos más pequeños atendiendo a su posición geográfica (*routers* a los que están conectados).

Inicialización

En este esquema la escalabilidad se consigue manteniendo a los subgrupos relativamente independientes, es decir, cada subgrupo del esquema se considera un grupo multicast diferente con su propia dirección IP. Además, cada subgrupo tiene su propio material de claves y no existe ninguna clave global a todos los miembros del grupo.

Alta/Baja de miembro

Cuando un miembro se da de alta o de baja en el grupo multicast, de hecho, sólo lo está haciendo en el subgrupo que le corresponde. Así, sólo se debe cambiar la clave de subgrupo local y se solventa el problema de escalabilidad global.

De cualquier forma, la información que envía un miembro de un subgrupo aún debe llegar a todos los miembros del grupo multicast, por ello se hace necesaria la introducción de unas entidades intermedias entre grupos: las GSI o *Group Security Intermediaries*. Éstas deben encargarse de "traducir" los mensajes de un subgrupo a otro, ya que estarán cifrados con claves distintas.

Un ejemplo de arquitectura *Iolus* puede verse en la Fig 3.1. El ejemplo consta de un subgrupo núcleo (G^1) con tres interfaces (GSIs) a subgrupos de segundo nivel (G^{2C},G^{2B}y G^{2A}) que a su vez tiene interfaces a subgrupos de tercer nivel (G^{3A} y G^{3B}). Para controlar a los distintos interfaces existe un controlador de grupo global (GSC o *Group Security Controller*). Tanto los GSIs como los GSCs se denominan de forma genérica GSAs (*Group Security Agents*).

Ventajas e inconvenientes

Iolus reduce el ancho de banda de *rekeying* ya que la dinámica de un usuario sólo afecta a los miembros de su subgrupo. La notificación de nueva clave a los miembros del subgrupo

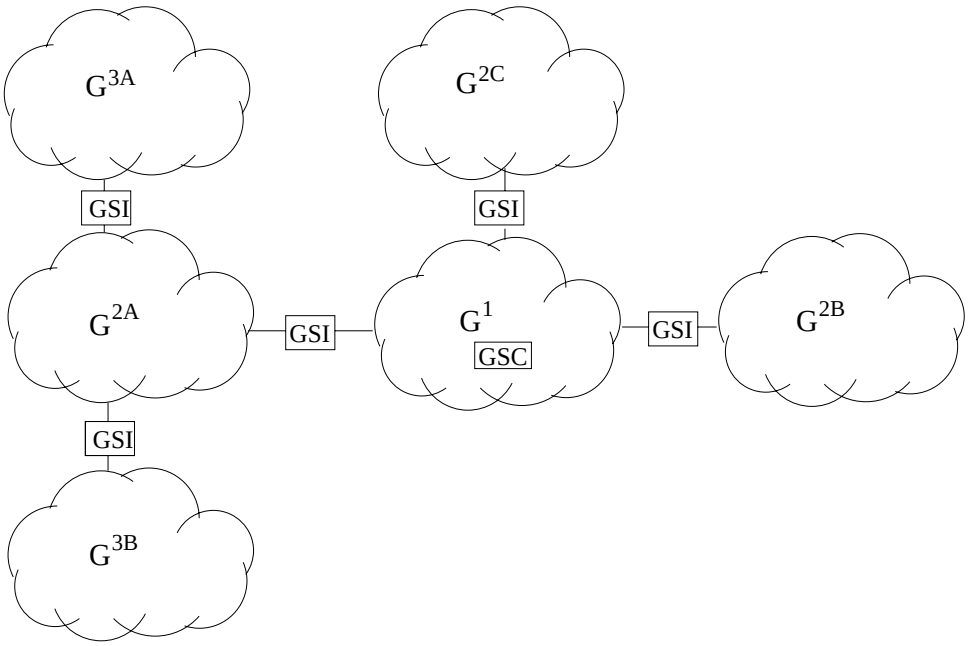


Figura 3.1: Ejemplo de arquitectura *Iolus*

se realiza según GKMP. Esto puede verse como ventaja ya que permite compatibilidad con dicho protocolo.

De todas formas, este mecanismo presenta numerosos inconvenientes. En primer lugar se debe depositar confianza en los nodos intermedios (GSI); éstos deben descifrar y volver a cifrar la información, con lo cual se permite que ésta permanezca en claro en un punto intermedio de la transmisión. En segundo lugar, requiere de una infrastructura paralela a los propios miembros (las GSAs) que a su vez también debe gestionarse.

3.4. Algoritmos basados en agrupaciones lógicas.

Como contraposición a los esquemas basados en agrupaciones físicas existen los basados en agrupaciones lógicas. En estos algoritmos no importa donde esté situado el miembro emisor/receptor dentro de la topología de la red sino dentro de una representación abstracta que construye el GCKS.

Un tipo de estructura que ayuda a agrupar elementos de manera lógica es el árbol. En este tipo de estructuras los miembros se sitúan en las hojas de forma que cada uno de los nodos del árbol es la raíz de un subconjunto (o agrupación lógica) del grupo global. Usando este principio y asociando a cada nodo una "clave de subgrupo lógico" se consiguen los algoritmos de renegociación basados en árboles lógicos de claves.

El uso de árboles binarios de claves (o *Hierarchical Binary Trees*) fue introducido para la gestión de seguridad en grupos dinámicos por [Caronni et al., 1998] . Para su ejecución se utilizan dos tipos de claves de cifrado:

- Claves de Sesión o Transmisión, en inglés *Transmission or Session Encryption Keys (TEKs o SEKs)*
- Claves que Cifran Claves o *Key Encryption Keys (KEK)*

Las SEK[2]s se utilizan para cifrar los datos que se intercambian dentro de un grupo multicast, por ejemplo flujos de vídeo en una multivideoconferencia. Las KEKs, en cambio, se utilizan para cifrar las claves que los distintos miembros del grupo van a necesitar para obtener la SEK.

Aunque existen extensiones para su uso descentralizado, estos algoritmos son normalmente centralizados. En el caso más simple, un único Gestor de Claves organiza las KEKs en árboles lógicos. En la raíz de dichos árboles sitúa la clave que conocerán todos los miembros del grupo multicast. Cada uno de esos miembros se corresponde con una hoja del árbol, que a su vez tiene asignada una clave que sólo conoce el miembro que reside en ella. Esta clave es un secreto individual entre el KS y cada miembro y la denominaremos Clave Individual o *Individual Key (IK)*, para diferenciarla de la clave secreta (correspondiente al par de criptografía de clave pública) de cada miembro. Las KEKs de nodos intermedios se revelan a los miembros que están en las hojas del árbol descendientes de esos nodos.

Para seguir un convenio en notación, todos los árboles que se utilizarán a partir de ahora se denominarán siguiendo el criterio *(nivel del nodo, posición en el nivel)*, empezando el contador tanto de nivel como de posición en el valor 1. De esta forma, el nodo raíz será el *(1,1)*; los hijos de éste se corresponderán con *(2,1)* y *(2,2)* y así sucesivamente. En la Fig 3.2 se muestra un ejemplo de árbol lógico de claves binario. La clave correspondiente al nodo *(X, Y)* la denominaremos $K_{(x,y)}$.

Como ejemplo, considere el grupo de 8 usuarios de la Fig 3.2. El árbol tiene 15 nodos, cada nodo se corresponde con una KEK. Los miembros se localizan en los nodos hoja. Las claves de las hojas son sus claves individuales (IK), mientras que $K_{(1,1)}$ se revela a los 8 usuarios. Además, cada una de las claves en los nodos intermedios se reparte a los usuarios hijos del nodo. Por ejemplo, $K_{(3,1)}$ se revela sólo a los usuarios en las hojas *(4,1)* y *(4,2)*, y $K_{(2,2)}$ se revela a los usuarios en los nodos *(4,5)*, *(4,6)*, *(4,7)* y *(4,8)*.

Ninguna de las claves pertenecientes al árbol lógico (KEKs) se usará para cifrar datos útiles multicast. Para ello sólo se usará la clave de sesión (SEK), que no está contenida en el árbol. Las KEKs servirán para cifrar los mensajes de control del GCKS para distribuir la nueva SEK cuando ésta deba actualizarse.

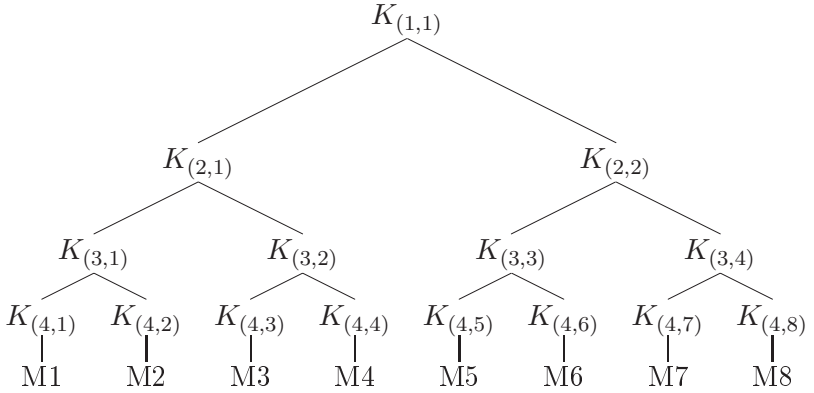


Figura 3.2: Ejemplo de árbol binario de claves con 8 miembros

La forma en la que se aprovecha la estructura en árbol para reducir el número de mensajes de actualización cambia de un algoritmo a otro. Aunque la mayoría de propuestas utiliza ár-

[2]Las nomenclaturas SEK y TEK se utilizarán indistintamente.

boles binarios, nada impide utilizar árboles de grado superior o incluso híbridos. Algunas propuestas de algoritmos con árboles de grado superior pueden encontrarse en [Wong et al., 1998].

Por convenio, a partir de ahora se considerarán todos los árboles de grado $d = 2$. Si el árbol se mantiene balanceado, su profundidad seguirá la expresión (3.1).

$$Profundidad = 1 + \lceil log_2 N \rceil \tag{3.1}$$

A continuación se presentan 3 de las propuestas más significativas de algoritmos de gestión basados en árboles lógicos de claves.

3.4.1. Algoritmo *Logical Key Hierarchy* ***(LKH)***

El primer protocolo que utilizó algoritmos en árbol fue propuesto conjuntamente por [Harney and Harder, 1999, Wallner et al., 1999]. Es el método más simple en cuanto a gestión de claves en árboles lógicos y hasta el momento el de mayor aceptación.

Las KEKs del árbol se generan de forma totalmente aleatoria y sin tener ninguna dependencia unas con otras. Esto obliga al GCKS a almacenar muchas más claves que en el caso trivial (GKMP) pero reduce a orden logarítmico el número de mensajes necesario para la actualización de claves o *rekeying*.

Para entender mejor el modo de operación de este algoritmo a continuación se presenta un ejemplo sencillo de las tres fases del mismo: inicialización, alta de miembro y baja de miembro.

Inicialización

En la fase de inicialización del grupo, el GCKS debe construir el árbol binario de claves y situar a los miembros en las hojas. Las KEKs se generan de forma independiente y son localizadas en cada uno de los nodos. A su vez, el gestor de claves debe encargarse de notificar a cada miembro el conjunto de KEKs que le corresponde.

Como ejemplo, considere la Fig 3.3 con $N = 7$ miembros ($M1...M7$). Cada integrante del grupo conocerá el subconjunto de claves del árbol correspondiente al camino que existe desde su hoja hasta la raíz. Éstas le permitirán recuperar la nueva SEK cada vez que ésta se cambie. En nuestro ejemplo, $M1$, situado en *(4,1)*, conocerá $K_{(4,1)}$, $K_{(3,1)}$, $K_{(2,1)}$ y $K_{(1,1)}$.

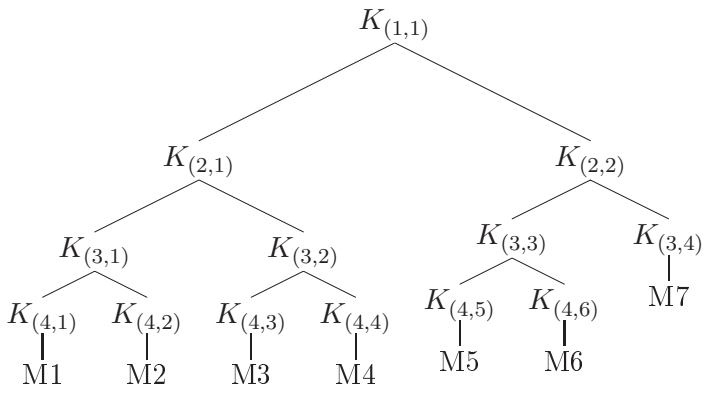


Figura 3.3: Árbol lógico de claves para LKH con 7 miembros

Alta de miembro

Un nuevo miembro que quiera añadirse al grupo ($M8$) debe primero contactar con el KS vía un canal unicast seguro. En esa fase negociarán una clave compartida sólo entre ellos dos que les servirá para comunicarse de forma secreta ($K_{(4,8)}$). Después de esto, el KS deberá actualizar todas las claves que existan en el camino desde la hoja donde ha situado a $M8$ hasta la raíz. De no ser así, posibles comunicaciones anteriores que hubieran sido cifradas con dichas claves, y que M8 podría tener grabadas, serían vulnerables en este momento.

En la Fig 3.4 puede observarse dicho comportamiento, donde las claves actualizadas se han marcado como $K'_{(x,y)}$. $K_{(1,1)}$ y $K_{(2,2)}$ son eliminadas y en su lugar se sitúan $K'_{(1,1)}$ y $K'_{(2,2)}$. $K_{(3,4)}$, por ser la IK de $M7$, es aprovechada y renombrada, situándose ahora en el nodo *(4,7)* con nombre $K_{(4,7)}$. Finalmente se genera una nueva $K'_{(3,4)}$de forma independiente.

El KS debe repartir las nuevas claves a los usuarios pertinentes. Para ello utiliza la jerarquía de claves junto con protocolos de transporte multicast, que por simplificación supondremos fiable. En primer lugar envía a cada miembro en los nodos *(4,7)* y *(4,8)* todas las claves de su subconjunto correspondiente. Esta transmisión se realiza usando sólo sus claves individuales y mediante una conexión unicast (también podría realizarse mediante multicast, pero su uso no supondría ningún ahorro de ancho de banda).

Seguidamente, envía un mensaje multicast con las claves $K'_{(2,2)}$ y $K'_{(1,1)}$cifradas con $K_{(3,3)}$, de forma que sólo los miembros en los nodos *(4,5)* y *(4,6)* puedan descifrarlas. Finalmente, se envía un mensaje multicast con la nueva clave raíz $K'_{(1,1)}$ cifrada con $K_{(2,1)}$, de forma que los miembros en los nodos *(4,1)* a *(4,4)* puedan recuperarla.

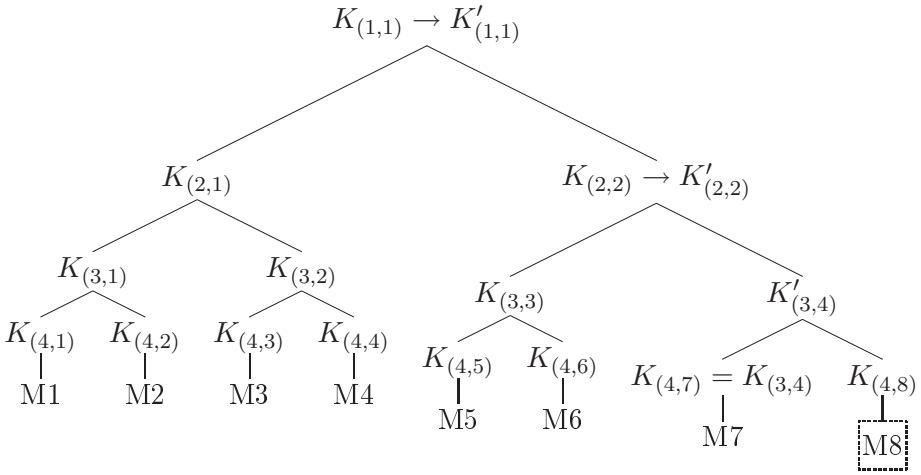


Figura 3.4: Alta de miembro en árbol lógico de claves LKH

Llegados a este punto, los 8 miembros en el grupo multicast conocen el subconjunto de claves actualizado que hay desde su posición en el árbol hasta la raíz. Todos los miembros conocen la clave raíz, así, ésta se puede usar para cifrar un mensaje multicast que contendrá la nueva clave de sesión o SEK. Los únicos mensajes de gestión de claves que se han enviado son los siguientes:

$\{K'_{(1,1)}\}_{K_{(2,1)}}$
$\{K'_{(2,2)} || \, K'_{(1,1)}\}_{K_{(3,3)}}$

$\{K'_{(3,4)}||\, K'_{(2,2)}||\, K'_{(1,1)}\}_{K_{(3,4)}=K_{(4,7)}}$ (enviado por canal unicast)[3]

Baja de miembro

Imagine ahora que el miembro $M4$ desea abandonar el grupo. Todas las claves que posee ($K'_{(1,1)}$, $K_{(2,1)}$, $K_{(3,2)}$ y $K_{(4,4)}$) se consideran comprometidas. $K_{(4,4)}$ simplemente se elimina. Vea Fig.3.5. Las demás KEKs son actualizadas por el KS y, al igual que en el caso de alta de miembro, son enviadas al resto de miembros cifradas con claves situadas en nodos inferiores a los nodos hermanos de las claves actualizadas.

En nuestro ejemplo, en primer lugar el KS envía al miembro $M3$, usando su clave única y por un canal unicast, el subconjunto de claves actualizadas. Seguidamente, envía un mensaje multicast que contendrá las claves $K'_{(2,1)}$ y $K''_{(1,1)}$ cifradas con $K_{(3,1)}$, de esta forma sólo $M1$ y $M2$ podrán recuperarlas. Finalmente, envía un mensaje multicast que contendrá la clave $K''_{(1,1)}$ cifrada con $K'_{(2,2)}$, de forma que los miembros en los nodos *(4,5)* a *(4,8)* podrán recuperarla.

Llegados a este punto, todas las claves que conocía $M4$ cuando era miembro autorizado del grupo han sido actualizadas, por lo que se puede asegurar que $M4$ no tendrá acceso a ninguna comunicación futura de dicho grupo.

Los mensajes que se han enviado en este caso son los siguientes:

$\{K''_{(1,1)}\}_{K'_{(2,2)}}$
$\{K'_{(2,1)}||\, K''_{(1,1)}\}_{K_{(3,1)}}$
$\{K'_{(2,1)}||\, K''_{(1,1)}\}_{K'_{(3,2)}=K_{(4,3)}}$ (enviado por canal unicast)

Ventajas e inconvenientes

Como principal ventaja de LKH debe destacarse la reducción del ancho de banda de *rekeying*[4] a $O(log_2 N)$. Esta característica junto con su simplicidad, le han hecho uno de los algoritmos más populares y el paradigma de otras propuestas basadas en árboles lógicos. De cualquier forma, no debe olvidarse que para ello se ha incrementado la cantidad de memoria necesaria en el gestor de claves y en los miembros, y quedan aún algunos parámetros susceptibles de ser mejorados. Este es el propósito de las restantes propuestas explicadas a continuación.

[3] El mensaje a $M8$ no se contabiliza ya que puede incluirse en la fase de alta al servicio, cuando se negocia $K_{(4,8)}$.
[4] El análisis formal de los parámetros de eficiencia se encuantra en la sección 3.6.

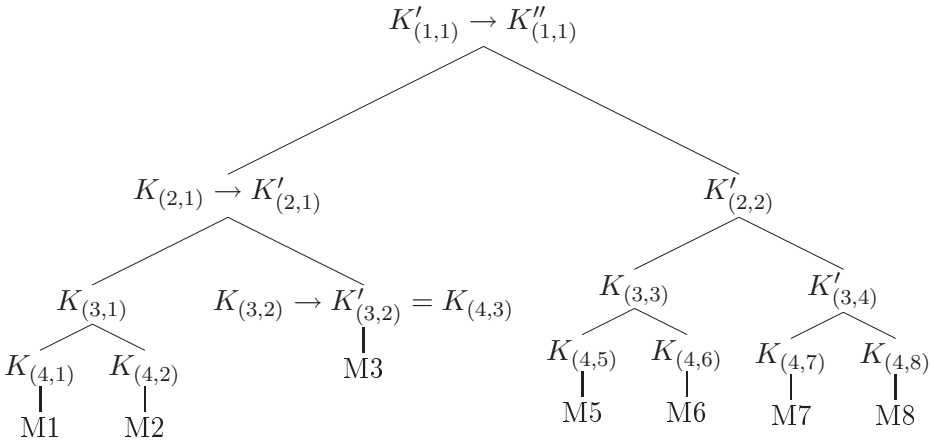


Figura 3.5: Baja de miembro en árbol lógico de claves LKH

3.4.2. Algoritmo *One-way Function Tree* ***(OFT)***

Una propuesta de mejora de LKH fue presentada por [Ballenson et al., 2000]. Este algoritmo reduce aproximadamente a la mitad tanto el ancho de banda de *rekeying* del LKH como la cantidad de memoria a almacenar por el GCKS. El sistema también se basa en árboles binarios de claves pero sus nodos no se generan de forma aleatoria e independientes unos de otros sino que siguen unas determinadas reglas.

Inicialización

Cuando se quiere formar un grupo OFT, el KS genera un árbol de claves con el mismo aspecto que el de la Fig 3.2 sólo que ahora cada uno de los nodos se construye siguiendo la expresión (3.2)

$$K_{(x,y)} = f\left(g\left(K_{(x+1,2y-1)}\right), g\left(K_{(x+1,2y)}\right)\right) \tag{3.2}$$

g denota una función unidireccional (por ejemplo una función de *hash*) y $f(A, B)$ se trata sólo de una combinación de A y B (podría ser por ejemplo una XOR, suma, etc). $K_{(x+1,2y-1)}$ denota la clave en el nodo "hijo izquierdo" de la clave $K_{(x,y)}$ y $K_{(x+1,2y)}$ se corresponde con el nodo "hijo derecho". Aplicar una función unidireccional ($g(K)$) a una de esas claves (K) implica "cegar K", ya que por definición de función unidireccional es computacionalmente infactible obtener el valor de K a partir sólo del conocimiento de $g(K)$.

En este esquema, cada miembro debe almacenar su clave individual (IK) junto con todas las claves cegadas de los nodos hermanos a los correspondientes a su camino hasta la raíz. Con esa información, y sabiendo qué funciones utiliza el KS para calcular la expresión (3.2), cada miembro tendrá suficiente para reconstruir su grupo de claves cada vez que éstas se actualicen.

Para entender mejor el funcionamiento de OFT, se presenta un ejemplo similar al seguido en LKH. Considere el grupo multicast con $N = 8$ miembros de la Fig 3.6. El miembro $M4$ por ejemplo, almacena en su memoria las claves $K_{(4,4)}$, $g(K_{(4,3)})$, $g(K_{(3,1)})$, $g(K_{(2,2)})$.

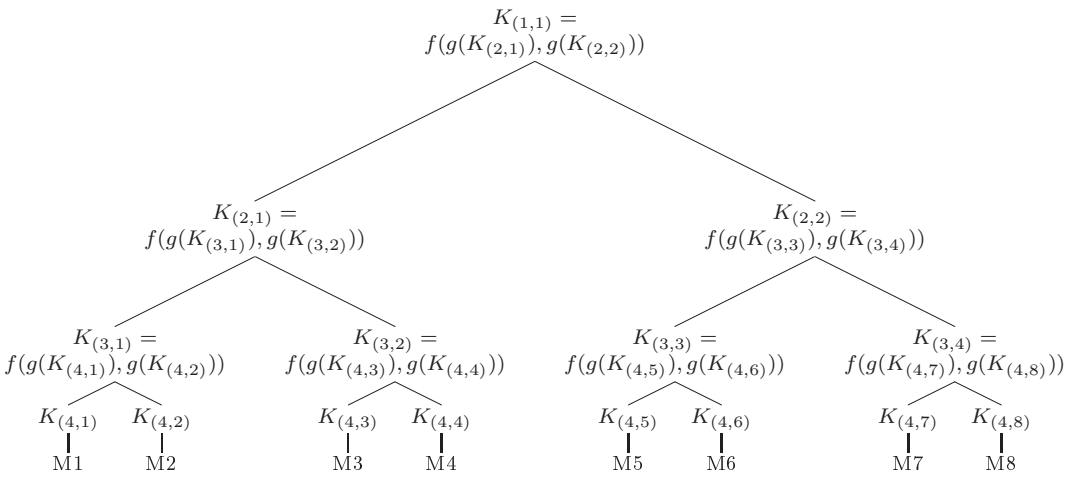


Figura 3.6: Árbol lógico de claves para OFT con 8 miembros

Baja de miembro

Supóngase que ese mismo miembro ($M4$) desea abandonar el grupo. Fig 3.7. Todas las claves desde el nodo donde estaba $M4$ hasta la raíz deben ser actualizadas.

En primer lugar, $K_{(4,4)}$ se elimina del árbol. El miembro $M3$ pasa a ocupar el nodo *(3,2)* que ahora se trata de un nodo hoja. La clave $K'_{(3,2)}$ pasa a ser ahora la clave individual de $M3$. Faltan por calcular las actualizaciones $K'_{(2,1)}$ y $K'_{(1,1)}$ que deberán ser notificadas a todos los miembros que estén por debajo de sus nodos. $M1$ y $M2$, situados en los nodos hijos de *(3,1)* sólo necesitan $g(K'_{(3,2)})$ para calcular todas las actualizaciones de su subconjunto. De igual forma, los miembros $M5$ a $M8$ sólo necesitan saber $g(K'_{(2,1)})$ para poder calcular $K'_{(1,1)}$. Así, los mensajes que el gestor de claves enviará por el canal multicast serán los siguientes:

$$\{g(K'_{(3,2)})\}_{K_{(3,1)}}$$
$$\{g(K'_{(2,1)})\}_{K_{(2,2)}}$$

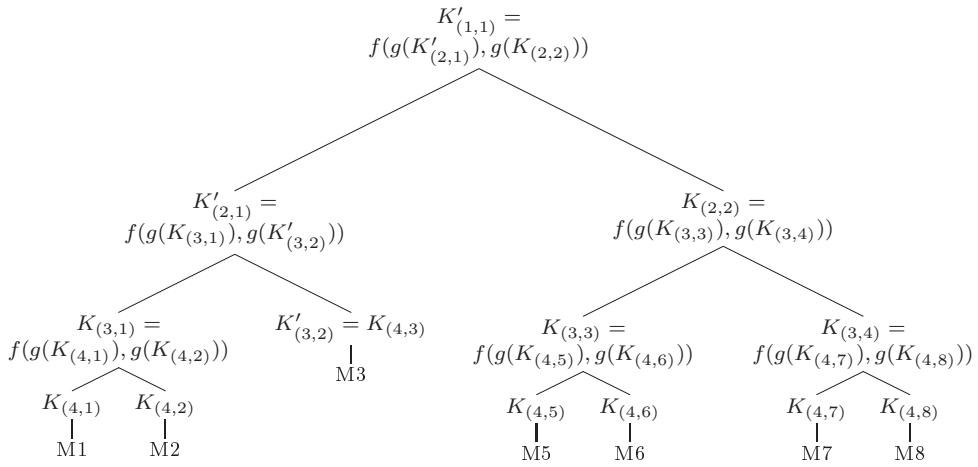


Figura 3.7: Baja de miembro en árbol lógico de claves OFT.

Alta de miembro

De forma similar, si ahora un miembro M_A desea darse de alta en el grupo (supongamos que para mantener el árbol balanceado se sitúa debajo del nodo *(3,2)*) deberán recalcularse $K''_{(3,2)}$, $K''_{(2,1)}$ y $K''_{(1,1)}$. Fig 3.8.

En primer lugar a $M3$ se le debe notificar $g(K'_{(4,4)})$, con este valor ya es capaz de reconstruir todo su camino de claves hasta la raíz. De igual forma a $M1$ y $M2$ les falta $g(K''_{(3,2)})$ para actualizar su subconjunto de claves, y a $M5$-$M8$ se les debe enviar $g(K''_{(2,1)})$. Con todo esto, los mensajes necesarios para la actualización en el caso que nos ocupa son los siguientes:

$$\{g(K'_{(4,4)})\}_{K_{(4,3)}}$$
$$\{g(K''_{(3,2)})\}_{K_{(3,1)}}$$
$$\{g(K''_{(2,1)})\}_{K_{(2,2)}}$$

Ventajas e inconvenientes

La eficiencia de OFT también será discutida con detalle posteriormente en este capítulo, pero podemos decir que respecto al LKH aporta una mayor reducción del ancho de banda de renegociación de clave y menor cantidad de memoria requerida en el GCKS. Como contrapartida introduce complejidad y coste computacional al algoritmo y lo vuelve menos flexible para negociaciones de clave por lotes[5].

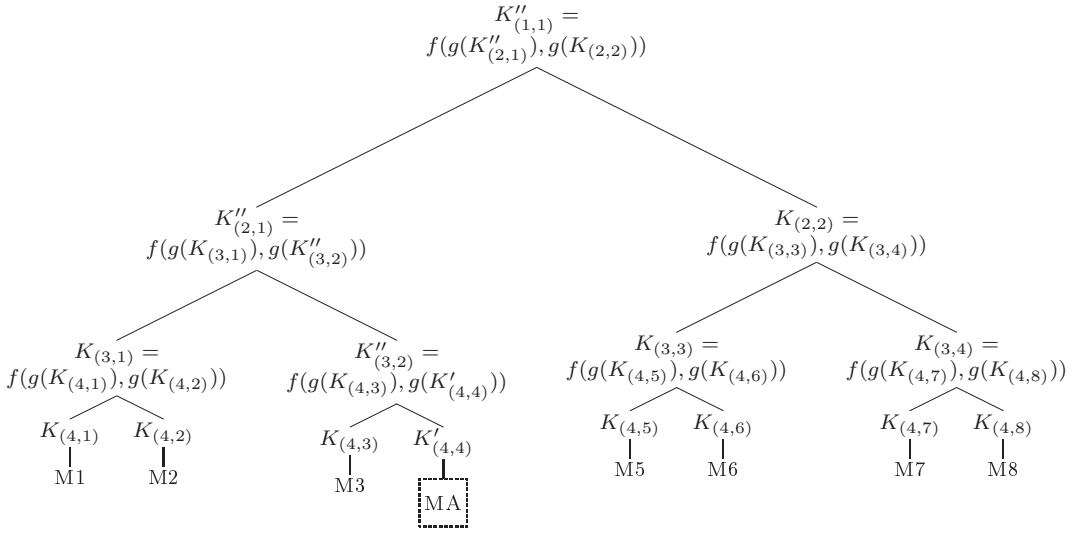


Figura 3.8: Alta de miembro en árbol lógico de claves OFT

3.4.3. Algoritmo *One-way Function Chain* ***(OFC)***

[Canetti et al., 1999a] propusieron una variación del algoritmo anterior con los mismos parámetros de eficiencia que OFT pero que, según afirmaban, presenta un mayor nivel de seguridad. Su esquema usa funciones pseudo-aleatorias para generar las nuevas KEKs en vez de las funciones unidireccionales de OFT y sólo se aplica en la baja de miembros.

[5]Este punto se discutirá en el Capítulo 4

Inicialización

En la fase de inicialización, OFC genera un árbol como el de la Fig. 3.2 y según las reglas de LKH (claves aleatorias e independientes) o OFT.

Alta de miembro

Las altas también son tratadas o como LKH o como OFT, por tanto, no aporta ninguna novedad ni mejora a las propuestas anteriores.

Baja de miembro

La novedad de este algoritmo respecto las propuestas anteriores reside en la fase de "baja de miembro". Esta fase se encarga de actualizar las claves que conocía el miembro revocado. Para ello establece una relación entre las claves de los nodos basada en funciones, pero que no son las mismas que las f y g usadas en OFT. Precisamente, la elección de la función f es el argumento en el que se basan [Canetti et al., 1999a] para justificar que presenta un mayor nivel de seguridad que el algoritmo anterior [Blum and Micali, 1982].

En OFC, g no se usa (no hay ningún parámetro cegado) y f consiste en una función pseudo-aleatoria con longitud de salida doble que la entrada. Si x representa la entrada de la función, entonces, como resultado se obtiene $f(x)$ según la expresión (3.3), donde $\|$ representa la concatenación y $f_{izquierda}(x)$ y $f_{derecha}(x)$ son computacionalmente independientes.

$$f(x) = f_{izquierda}(x) \ \| \ f_{derecha}(x) \tag{3.3}$$

Imagine que $M1$ abandona el grupo (Fig 3.9). Después de borrar su IK, el KS calcula un número aleatorio $rand_{(4,2)}$ y lo envía a $M2$ (hermano del miembro revocado). Con este valor $M2$ podrá calcular $K'_{(3,1)}$. En concreto, $M2$ calcula $f\,(rand_{(4,2)})$ y le asigna a su nodo padre la parte izquierda de su resultado ($f_{izq}(rand_{(4,2)})$), después de esto vuelve a aplicar f , pero ahora sobre la parte derecha de $f\,(rand_{(4,2)})$ y asigna la parte izquierda del resultado al nodo *(2,1)*. Finalmente vuelve a aplicar f sobre la parte derecha de ese resultado y le asigna el resultado izquierdo al nodo raíz (*(1,1)*)

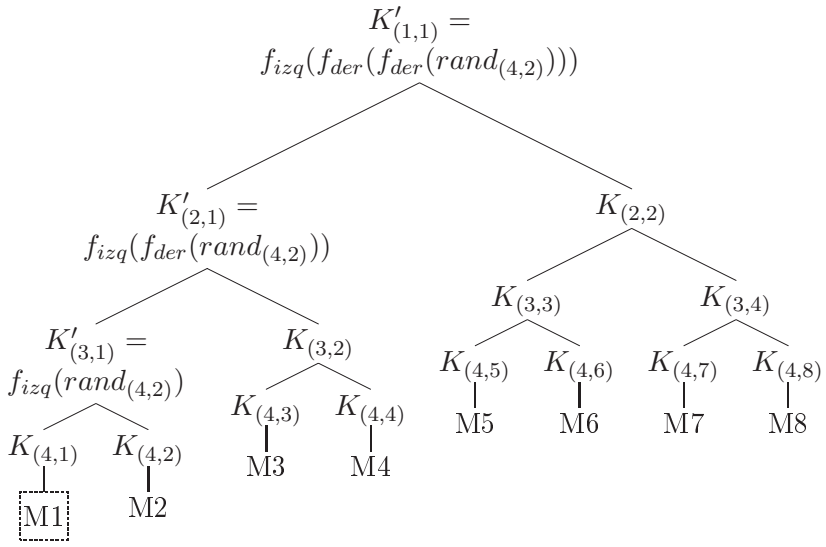


Figura 3.9: Baja de miembro en árbol lógico de claves OFC

Por su parte, el KS deberá realizar las mismas operaciones y notificar a los nodos correspondientes los parámetros necesarios para que puedan actualizar sus subconjuntos de claves. En concreto, el Gestor de Claves enviará los siguientes mensajes por el canal multicast.

$$\{rand_{(4,2)}\}_{K_{(4,2)}}$$
$$\{f_{derecha}(rand_{(4,2)})\}_{K_{(3,2)}}$$
$$\{f_{derecha}(f_{derecha}(rand_{(4,2)}))\}_{K_{(2,2)}}$$

Ventajas e inconvenientes

Tal como se apreciará en 3.6, OFC no aporta ninguna mejora en cuanto a parámetros de eficiencia respecto OFT. Sin embargo, hace mucho más sistemático el estudio de su nivel de seguridad, ya que éste se puede reducir formalmente al estudio de la seguridad de f.

3.5. Otros algoritmos

A parte de los algoritmos básicos, en la literatura también se encuentran propuestas menores consistentes en pequeñas variaciones de éstas que mejoran su eficiencia. Entre ellas cabe destacar el LKH mejorado (*LKH+*, *LKH híbrido* o *Enhanced LKH*), OFT mejorado (*OFT+*, *OFT híbrido* o *enhanced OFT)*) y *Subset Difference*.

Extensión a árboles híbridos

LKH+, propuesto por [Canetti et al., 1999a], y *OFT+* [Sampigethaya et al., 2002], generalizan los algoritmos básicos a árboles híbridos $2-N$. Controlan el número de hijos que puede tener cada nodo y así consiguen reducir la cantidad de memoria necesaria en el GCKS. Paradójicamente, estas propuestas no aprovechan la construcción de árboles híbridos para mantener el árbol balanceado.

Extensión a agrupaciones lógicas genéricas

La otra propuesta, *Subset Difference* [Lotspiech et al., 2001], se trata de una aproximación más matemática y está basada en los denominados subgrupos diferenciales.

Un subgrupo diferencial se define como "el grupo de elementos G_1 menos otro grupo G_2", donde $G_2 \subset G_1$. En estructuras en árbol estos "subgrupos" se notan con $S(a, b)$ donde a y b son nodos raíz de los subgrupos G_1 y G_2. Así por ejemplo, en la Fig 3.10, $S(N_{(1,1)}, N_{(3,2)})$ abarca a los miembros del subgrupo $\{M1, M2, M5, M6, M7, M8\}$.

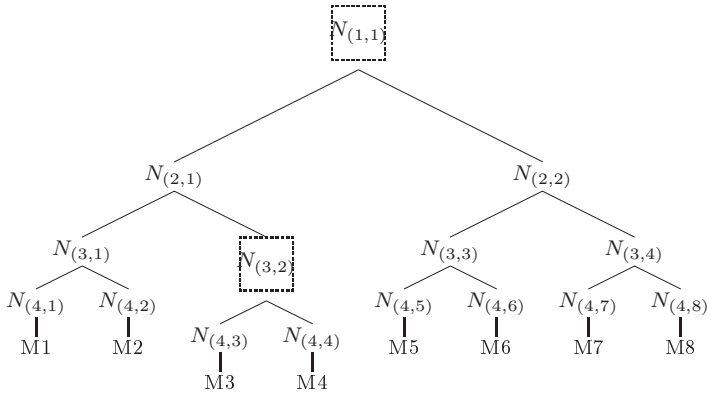


Figura 3.10: Ejemplo de subgrupo diferencial

Se puede demostrar que cualquier subconjunto $A \subset N$ se puede particionar usando como máximo $2R - 1$ subgrupos diferenciales, donde R es el cardinal de $N - A$.

Este método propone que cualquier miembro mantenga una clave distinta para cada subgrupo diferencial al que pueda pertenecer. Cuando R miembros abandonan el grupo, sólo se necesitarán $2R - 1$ mensajes para actualizar las claves.

Aunque la propuesta es original y teóricamente, en el caso de una alta/baja individual, reduce el número de mensajes de *rekeying* a 1, la cantidad de claves que debe guardar el GCKS (y cada uno de los miembros) así como la complejidad de la actualización hacen inviable de forma práctica este algoritmo. Como aspecto positivo debe resaltarse que incluye la posibilidad de actualización por lotes[6], aunque el incremento de miembros salientes/entrantes vaya en detrimento de su eficiencia (a mayor número de peticiones en un *batch*, mayor número de mensajes necesarios).

3.6. Contribución a la evaluación de algoritmos existentes

Todos los algoritmos expuestos hasta el momento pretenden la mejora respecto GKMP de los parámetros de eficiencia discutidos en 2.5. Para poder comparar hasta qué punto es eficiente cada propuesta, a continuación, se realiza la evaluación de todos ellos. Esto, además, permitirá detectar las carencias existentes con el fin de introducir propuestas de mejoras en las siguientes secciones.

[6]Este tipo de proceso se estudiará en el Capítulo 4.

3.6.1. Parámetros generales

Los parámetros generales no pueden evaluarse de forma matemática. De todos modos, en este apartado se discute brevemente, de forma cualitativa, cómo se comportan estos parámetros en los distintos algoritmos presentados hasta el momento.

Dependencia en N de la escalabilidad

El efecto más nocivo a superar por los algoritmos de gestión de claves de grupo es la dependencia en N de la escalabilidad. GKMP es un ejemplo claro de cómo la acción individual de un miembro comporta la ejecución de N acciones por parte del GCKS, que además afectan a todos los miembros restantes del grupo.

En los otros algoritmos presentados, aunque el alta/baja de un miembro requiere un cambio de clave, se rompe la dependencia en N de la escalabilidad (*1 afects N scalability*) ya que el GCKS no tiene que interactuar con los N miembros sino con subconjuntos de ellos.

De cualquier forma, esto se considera únicamente válido para algoritmos que utilizan agrupaciones lógicas. Se afirma que sistemas como *Iolus* suponen una "falsa" reducción de la dependencia en N ya que en cada subconjunto en los que se divide el grupo inicial vuelve a presentar una dependencia en N.

Confianza en nodos intermedios

Un protocolo requiere confianza en nodos intermedios si el mensaje permanece en claro en algún punto intermedio de la transmisión. Al igual que GKMP, todos los algoritmos que utilizan subagrupaciones lógicas son de naturaleza centralizada y los dispositivos de red no acceden a la comunicación del grupo. Los participantes de la comunicación sólo deben confiar en el GCKS. *Iolus*, en cambio, al distribuir geográficamente la gestión del grupo y ligarla al sistema de transporte de datos sí que requiere confianza en nodos intermedios.

Ataques de confabulación

El análisis en cuanto a ataques de confabulación es ligeramente más complicado que en los parámetros anteriores. En general se considerará que un algoritmo es vulnerable a ataques de confabulación si la combinación de información que varios miembros poseen de forma individual puede revelar información que posee un miembro no perteneciente al grupo confabulador.

La forma más sencilla de evitar este ataque es mantener las claves independientes entre sí. GKMP, *Iolus*, LKH y *Subset Difference* no son vulnerables a ataques de confabulación[7] por tener esta propiedad. El conocimiento de un subconjunto autorizado de KEKs no revela nada acerca de otras claves ni aunque se combinen distintos subconjuntos autorizados.

OFT y OFC reducen el número de claves a guardar en el GCKS y en los miembros mediante la introducción de relaciones entre claves. En estos casos, la no vulnerabilidad frente a confabulaciones debe ser demostrada de forma explícita. Tanto en [Ballenson et al., 2000]

[7] El único caso posible sería que un usuario autorizado revele toda su información a uno no autorizado, de forma que este último también pueda acceder al grupo. Este ataque es obvio en cualquier sistema e inevitable (solo detectable o rastreable) por lo cual la literatura no lo considera ataque de confabulación.

como en [Canetti et al., 1999a] se afirma que ninguno de los dos algoritmos es vulnerable a este tipo de ataques, aunque no se proporciona su demostración formal.

Comparativa de parámetros generales

En la Tabla 3.1 puede verse un resumen de los parámetros generales de eficiencia para los algoritmos expuestos hasta el momento.

Tabla 3.1: Comparativa de parámetros generales en algoritmos de gestión de claves de grupo

	GKMP	*Iolus*	LKH	OFT	OFC	*SubsetDiff.*
1 afects N scalability	Sí	Sí	No	No	No	No
Confianza en nodos intermedios	No	Sí	No	No	No	No
Ataques de confabulación	No	No	No	No	No	No

3.6.2. Ancho de banda

Una vez rota la dependencia en N, el parámetro al que se le ha prestado más interés en la literatura ha sido el ancho de banda necesario para inicializar y actualizar el material de claves.

Inicialización

Recordemos que en GKMP, cada vez que se establece un grupo seguro, con N miembros iniciales, el KS se comunica con cada uno de ellos para entregarles su clave. Si definimos L como la longitud de la clave, es trivial deducir que el número de mensajes necesarios y el ancho de banda de inicialización en GKMP siguen las expresiones (3.4) y (3.5).

$$Num\ Mensajes_{GKMP-inic} = N \tag{3.4}$$

$$BW_{GKMP-inic} = N \cdot L \tag{3.5}$$

En LKH el número de comunicaciones que el GCKS debe establecer al iniciar el grupo coincide con el caso GKMP (expr (3.6)). En cambio, en cada uno de estos mensajes debe enviar tantas claves como KEKs existan en el camino desde el nodo donde está el miembro hasta la raíz. Si se considera un árbol balanceado, este valor tiene como cota máxima la profundidad del árbol y por tanto, el ancho de banda total seguirá la expresión (3.7).

$$Num\ Mensajes_{LKH-inic} = N \tag{3.6}$$

$$BW_{LKH-inic} = N \cdot (L \cdot (1 + \lceil log_2 N \rceil)) \tag{3.7}$$

OFT y OFC tienen un comportamiento similar a LKH para la inicialización. Las claves que se envían son distintas, ya que se envía la IK de cada miembro junto con las KEKs

cegadas de los nodos hermanos al camino desde su posición hasta la raíz. Este valor también coincide con la profundidad del árbol y por tanto el número de mensajes y el ancho de banda de inicialización para OFT y OFC también siguen las expresiones (3.6) y (3.7).

Un caso a parte es el algoritmo *Subset Difference*. Recordemos que cada miembro debía mantener tantas claves como posibles grupos diferenciales a los que puede pertenecer. En la Fig 3.10 , es fácil observar que $M1$ pertenecerá a "cualquier subgrupo diferencial que excluya a 1 miembro siempre que ese miembro no sea él", a "cualquier subgrupo diferencial que excluya 2 miembros siempre que no sean los definidos por el subárbol de raíz su nodo padre", y así sucesivamente. Esta regla debe aplicarse para cualquier orden de grupo diferencial que lo contenga. Siguiendo ese criterio, el número de subgrupos diferenciales a los que puede pertencer un miembro sigue la expresion (3.8), cada uno de esos subgrupos tendrá asignada una clave de longitud L y el GCKS tendrá que enviar esa información por cada uno de los N miembros. Entonces, el número de mensajes y el ancho de banda de inicialización vienen dados por las expresiones (3.9) y (3.10).

$$Num\ Subgrupos\ Diferenciales\ de\ 1\ miembro = \sum_{a=1}^{\lceil log_2 N \rceil} \sum_{i=1}^{\left\lceil log_2 \frac{N}{2^{(a-1)}} \right\rceil} 2^i - 1 \tag{3.8}$$

$$Num\ Mensajes_{SubsetDiff-inic} = N \tag{3.9}$$

$$BW_{SubsetDiff-inic} = N \cdot L \cdot (\sum_{a=1}^{\lceil log_2 N \rceil} \sum_{i=1}^{\left\lceil log_2 \frac{N}{2^{(a-1)}} \right\rceil} 2^i - 1) \tag{3.10}$$

Los mismos autores en [Lotspiech et al., 2001] reconocen que este método es inviable para casos prácticos. En la misma referencia proponen una variación del algoritmo con funciones pseudo-aleatorias y dependencias entre nodos para reducir esta cantidad de claves. Este método no se analiza en detalle ya que introduciría vulnerabilidad respecto ataques de confabulación.

Comparativa de parámetros de inicialización

En la Tabla 3.3 puede verse el orden de magnitud de los valores del Número de mensajes y Ancho de banda de inicialización para los algoritmos expuestos hasta el momento. Asimismo, en la Fig 3.11 se observa la evolución del ancho de banda en bits[8] para distintos valores de N.

Tabla 3.3: Órdenes de magnitud de Número de mensajes y Ancho de banda necesarios para la inicialización.

	GKMP	*Iolus*	LKH	OFT	OFC	*SubsetDiff.*
Núm. mensajes	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$
Ancho de banda	$O(N)$	$O(N)$	$O(N \cdot log_2 N)$	$O(N \cdot log_2 N)$	$O(N \cdot log_2 N)$	$O(N!)$

[8]Se ha considerado $L = 128$bits

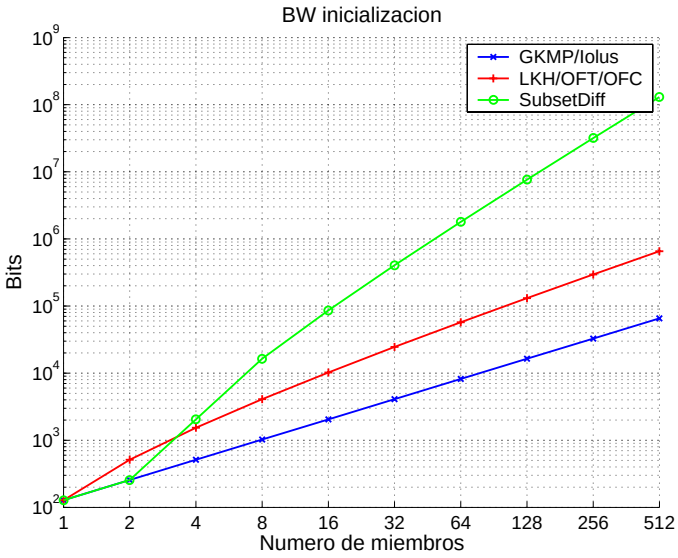


Figura 3.11: Ancho de banda de inicialización en función del número de miembros del grupo.

Debe notarse que, lejos de reducir este parámetro, las propuestas de mejora al GKMP aumentan considerablemente el ancho de banda de inicialización. Este efecto es debido al incremento de claves que debe guardar cada miembro. En las propuestas de mejora, el número de mensajes a enviar por el GCKS en la fase de inicialización se mantiene constante, pero en esos mensajes deben comunicarse todos los parámetros a guardar por cada miembro.

Tal como se verá en 3.6.4, el número de claves que debe guardar cada miembro es mayor y, por tanto, la longitud del mensaje de inicialización también.

Este comportamiento no deseado se ve ámpliamente compensado por la reducción en cuanto a ancho de banda que supone en las fases de alta y baja de miembro.

Actualización (Alta / Baja)

Recuérdese que en GKMP cuando un miembro se da de alta o de baja el KS debe enviar la nueva clave a los N miembros restantes de igual forma que hacía en la fase de inicialización y, por tanto, el número de mensajes y el ancho de banda de actualización son los mismos que en las expresiones (3.4) y (3.5).

$$Num\,Mensajes_{GKMP-alta/baja} = Num\,Mensajes_{GKMP-inici} \tag{3.11}$$

$$BW_{GKMP-alta/baja} = BW_{GKMP-inic} \tag{3.12}$$

En cambio, *Iolus* sólo deberá enviar un mensaje de actualización a los compañeros de subgrupo del miembro entrante/saliente. Si definimos n como el número de subconjuntos en los que se ha particionado el grupo inicial, el número medio de elementos en uno de esos subconjuntos será $(\frac{N}{n})$, que se corresponderá con el número de mensajes a enviar (expr (3.13)) y por tanto el ancho de banda se comportará según la expresión (3.14).

$$Num\,Mensajes_{Iolus-alta/baja} = \frac{N}{n} \tag{3.13}$$

$$BW_{Iolus-alta/baja} = L \cdot \frac{N}{n} \tag{3.14}$$

En LKH, sin embargo, para cifrar los mensajes se usan sólo las claves situadas en los nodos hermanos (*siblings*) a los del camino de claves a actualizar. Por tanto, se envían tantos mensajes como nodos hay en ese camino. Si el árbol está balanceado, este parámetro tiene como cota superior la profundidad del árbol menos uno (expr (3.15)).

$$Num\ Mensajes_{LKH-alta/baja} = \lceil log_2 N \rceil \tag{3.15}$$

Cada mensaje, además, contiene tantas claves de longitud L como saltos hay desde ese nodo hasta la raíz. Por tanto, el número total de claves de longitud L que se envían sigue la expresion (3.16) y el ancho de banda total es el de la expresión (3.17).

$$Num\ Claves_{LKH-alta/baja} = \sum_{i=1}^{\lceil log_2 N \rceil} i \tag{3.16}$$

$$BW_{LKH-alta/baja} = L \cdot \sum_{i=1}^{\lceil log_2 N \rceil} i = \frac{L}{2} \cdot (\lceil log_2 N \rceil^2 + \lceil log_2 N \rceil) \tag{3.17}$$

Por su parte OFT y OFC reducen el ancho de banda de actualización añadiendo dependencias entre las claves de sus árboles. El número de mensajes a enviar es el mismo que en LKH (expr (3.15)) pero ahora cada mensaje sólo contiene un único parámetro de longitud L ya que, con ese valor y conocidas las relaciones entre los nodos, los miembros pueden actualizar todas las claves de su *path*. Con todas estas consideraciones, el ancho de banda de actualización en OFT y OFC es el de la expresión (3.20).

$$Num\ Mensajes_{OFT/OFC-alta/baja} = Num\ Mensajes_{LKH-alta/baja} \tag{3.18}$$

$$Num\ Claves_{OFT/OFC-alta/baja} = \lceil log_2 N \rceil \tag{3.19}$$

$$BW_{OFT/OFC-alta/baja} = L \cdot \lceil log_2 N \rceil \tag{3.20}$$

Por último, en *Subset Difference*, se debe diferenciar entre la fase de alta y de baja. En [Lotspiech et al., 2001] se advierte que la aplicación directa del algoritmo no sirve para prestar confidencialidad hacia atrás, así, la fase de alta de miembro no tiene sentido ya que debería reiniciarse todo el sistema, en cuyo caso los parámetros serían los mismos que en la fase de inicialización.

El algoritmo en cambio sí que mejora mucho el ancho de banda de actualización cuando sólo hay bajas, ya que únicamente se tendrían que enviar $2R-1$ mensajes, siendo R el número de *leavings*. En el caso que nos ocupa, donde las bajas sólo pueden darse de una en una, un único mensaje bastaría para actualizar la clave.

Comparativa de parámetros de actualización

En la Tabla 3.5 puede verse el orden de magnitud del Número de mensajes y Ancho de banda de actualización para los algoritmos expuestos hasta el momento. Asimismo, en la Fig 3.12 se observa la evolución del ancho de banda de actualización en bits[9] para distintos valores de N.

Tabla 3.5: Órdenes de magnitud de Número de mensajes y Ancho de banda necesarios para la actualización.

	GKMP	*Iolus*	LKH	OFT	OFC	*SubsetDiff.*
Núm. mensajes	$O(N)$	$O(\frac{N}{n})$	$O(log_2 N)$	$O(log_2 N)$	$O(log_2 N)$	$O(1)$
Ancho de banda	$O(N)$	$O(\frac{N}{n})$	$O(log_2^2 N)$	$O(log_2 N)$	$O(log_2 N)$	$O(1)$

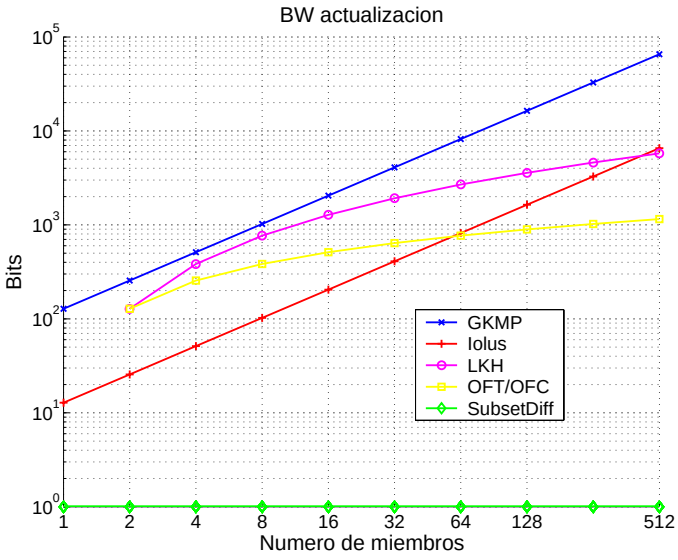


Figura 3.12: Ancho de banda de actualización en función del número de miembros del grupo.

Puede observarse que este parámetro sí que se reduce en las propuestas de mejora respecto al comportamiento presentado en GKMP.

LKH rompe la dependencia directa en N, y la diferencia absoluta en número de bits respecto GKMP es mucho mayor cuanto más crece el número de miembros en el grupo. De cualquier forma, este comportamiento logarítmico en N mejora aún más en la gráfica OFT/OFC donde se observa una pendiente menor.

Iolus sólo mejora el comportamiento de GKMP en cuanto a escalado. En la gráfica se observa un factor 10 por ser el valor asignado a n (número de subgrupos en los que se particiona N). El número de bits necesario para la actualización puede reducirse tanto como se quiera aumentando ese factor. Dependiendo de ese factor, la intersección con las gráficas LKH y OFT/OFC se dará para un determinado número de miembros u otro. Esto será a costa de

[9]Se ha considerado $L = 128$bits

aumentar los otros parámetros de eficiencia. La elección de cuál es el mejor algoritmo a escoger dependerá en cada caso.

Debe resaltarse el comportamiento de *Subset Difference*, que sólo requiere un mensaje de actualización, aunque, como ya hemos comentado, sólo sirve para baja de miembros y su incremento en memoria necesaria es mucho mayor que en otros algoritmos.

3.6.3. Latencia

En 2.5 se definió la latencia en el *rekeying* como el tiempo transcurrido desde que el KS empieza el proceso de notificación de cambio de SEK hasta que el cambio puede hacerse efectivo.

En el caso del GKMP se corresponde con el tiempo que tarda el Gestor de Claves en enviar el mensaje de cambio de SEK a N miembros. En *Iolus* el tiempo se reduce al intervalo entre el pimer envío de paquete y el último, pero en este caso sólo se deben enviar $(\frac{N}{n})$.

En los algoritmos de agrupaciones lógicas la mejora se basa en el mismo principio, se deben enviar menos mensajes, y por tanto, el tiempo transcurrido entre el inicio y el final del proceso de *rekeying* es menor.

Obviamente, este tiempo depende de la velocidad media (en bits por segundo) de la red, de la longitud del mensaje, la fiabilidad del canal, de lo distantes que estén los integrantes del grupo, etc. Todos estos factores deben ser normalizados para poder realizar una comparativa entre algoritmos.

Si consideramos el canal ideal (no se producen errores) y despreciamos los tiempos de propagación y transmisión, este parámetro puede medirse en "número de mensajes a enviar por el KS". Si todos los miembros tuviesen que interpretar el mismo mensaje, al no considerar el efecto real de la transmisión, todos estarían dispuestos a descifrar con la nueva clave al mismo tiempo y la latencia sería mínima. Cuando los mensajes a interpretar por los miembros son distintos el miembro que recibe e interpreta el primer mensaje tiene que esperarse a que los otros miembros reciban e interpreten sus correspondientes mensajes y la latencia crece.

Las expresiones del número de mensajes necesarios para cada algoritmo ya han sido calculadas en 3.6.2. En la Fig 3.13 se representa gráficamente en función del tamaño del grupo.

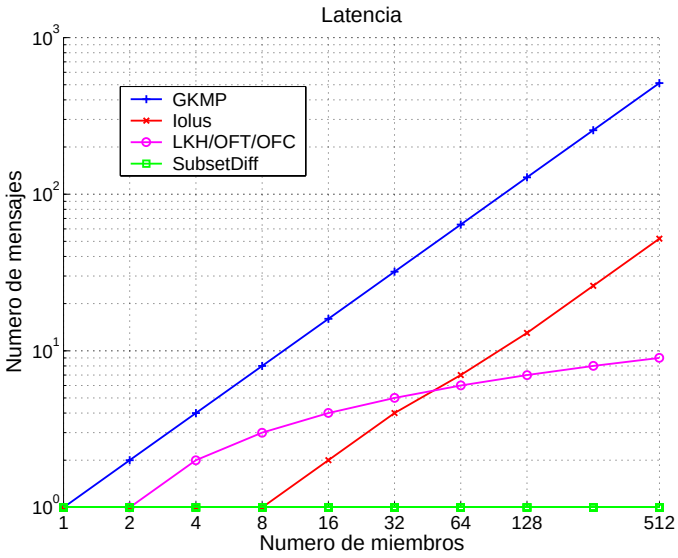


Figura 3.13: Latencia en función del número de miembros del grupo.

GKMP e *Iolus* tienen la misma tendencia que en la Fig 3.12. Por su parte, tal como se observó en el número de mensajes de actualización, los algoritmos que usan árboles lógicos de claves sólo deben enviar $log_2 N$ mensajes. Aquí LKH, OFT y OFC presentan el mismo comportamiento ya que no se tiene en cuenta la longitud del mensaje. *Subset Difference* vuelve a necesitar un único mensaje, y por tanto la latencia que presenta es la mínima.

3.6.4. Cantidad de memoria necesaria

La cantidad de memoria necesaria tanto en el GCKS como en cada uno de los miembros se definía como el número total de bits necesarios para almacenar el material de claves. Si este parámetro se normaliza por la longitud de clave, puede medirse en número de claves a guardar por cada uno de los miembros y GCKS. Al igual que pasaba con los mensajes de inicialización, este valor es mayor en los algoritmos presentados que en GKMP.

Servidor de claves

En el sistema básico, el número de claves que debe mantener el GCKS es igual al número de miembros del grupo[10], N , únicamente una clave individual por cada uno de ellos.

$$Memoria_{KS-GKMP} = N \tag{3.21}$$

En *Iolus*, en cambio, cada gestor de subgrupo (GSI) debe almacenar sólo las claves individuales de los miembros de su subgrupo más la clave de sesión del subgrupo con el que es interfaz. De cualquier forma, si se considera que el gestor global de subgrupos (GSC) debe guardar todas las claves de los miembros globales del grupo más las de los distintos GSI, el número de claves crece según la expresión (3.22).

$$Memoria_{GSC-Iolus} = N + \frac{N}{n} \tag{3.22}$$

[10]No se considera la clave de sesión

En LKH, este parámetro también es mayor ya que se debe mantener una clave por cada nodo del árbol. El número de nodos en un árbol balanceado con N miembros tiene como cota superior la expresion (3.23).

$$Memoria_{KS-LKH} = \sum_{i=0}^{\lceil log_2 N \rceil} 2^i = 2^{\lceil log_2 N \rceil + 1} - 1 = 2N^* - 1 \tag{3.23}$$

donde N* se define como la menor potencia de 2 no inferior a N.

OFT y OFC, al introducir dependencias entre sus nodos, no necesitan guardar todas las KEKs del árbol sino sólo las que les permita calcularlas. Tanto un sistema como otro tienen suficiente con las claves individuales de los miembros, su expresión es, por tanto, (3.24).

$$Memoria_{KS-GKMP} = N \tag{3.24}$$

Finalmente, el GCKS de *Subset Difference* debe guardar una clave por cada uno de los posibles subgrupos diferenciales que se puedan formar. Este valor se corresponde con la expresión en (3.25).

$$Memoria_{KS-SubsetDiff} = \sum_{i=0}^{\lceil log_2 N \rceil} (2^{\lceil log_2 N \rceil - i} \cdot (2^{i+1} - 1)) = 2 \cdot N \cdot log_2 N + 1 \tag{3.25}$$

Miembros

Cada uno de los miembros, por su parte, debía mantener una única clave en GKMP (su clave individual). Esto también ocurría en *Iolus*.

En los algoritmos basados en árboles este parámetro también crece. Cada integrante del grupo debe guardar todas las claves que se encuentran desde su hoja hasta la raíz, o las hermanas a ellas. Como ya se comentó anteriormente, este parámetro se corresponde con la profundidad del árbol, que tiene como cota superior la expresión (3.26).

$$Memoria_{Miembro-LKH/OFT/OFC} = \lceil log_2 N \rceil + 1 \tag{3.26}$$

Finalmente, en *Subset Difference* los miembros deben guardar tantas claves como subgrupos diferenciales puede pertenecer. Este valor ya se calculó para la fase de inicialización y se corresponde con la expresión (3.27).

$$Memoria_{Miembro-SubDiff} = \sum_{a=1}^{\lceil log_2 N \rceil} \sum_{i=1}^{\left\lceil log_2 \frac{N}{2^{(a-1)}} \right\rceil} 2^i - 1 \tag{3.27}$$

Comparativa de cantidad de memoria

Como resumen, en la Tabla 3.7 puede verse la cantidad de memoria necesaria por cada uno de los métodos presentados, tanto para el Gestor de Claves como para los miembros.

Tabla 3.7: Cantidad de Memoria necesaria en el GCKS y miembros.

	GKMP	*Iolus*	LKH	OFT	OFC	*SubsetDiff.*
GCKS	$O(N)$	$O(N)$	$O(2N)$	$O(N)$	$O(N)$	$O(2Nlog_2N)$
Miembro	$O(1)$	$O(1)$	$O(log_2N)$	$O(log_2N)$	$O(log_2N)$	$O(N!)$

La Fig 3.14 muestra la evolución en función del número de miembros del grupo.

Tal como era de esperar, *Subset Difference* es la que presenta peor comportamiento tanto para el KS como para los miembros. Por su parte todos los algoritmos que usan árboles lógicos de claves presentan unos valores aceptables aunque, de nuevo, OFT y OFC consiguen mejores resultados al introducir las dependencias entre nodos.

3.6.5. Coste computacional

Como último parámetro a evaluar hemos dejado el coste computacional. Éste se define como el número de operaciones criptográficas necesarias para realizar cada una de las acciones de inicialización, alta de miembro y baja de miembro. El comportamiento de este parámetro no es tenido muy en cuenta a la hora de diseñar un algoritmo de renegociación de claves. Sin embargo, éste puede ser crítico en escenarios donde los miembros tengan una capacidad de cálculo muy reducida.

La unidad natural con la que se mide este parámetro es MIPS (millones de instrucciones por segundo) o unidades derivadas. De nuevo, este valor dependerá del algoritmo de cifrado o de generación de números aleatorios que se escoja. Por ello, las medidas que aquí se dan se corresponden con las unidades "operaciones de cifrado" y "operaciones de generación" que tendrán su equivalencia en MIPS dependiendo de si se usa un algoritmo criptográfico u otro. De igual forma, la conversión a segundos depende de la potencia de proceso de los dispositivos.

Inicialización

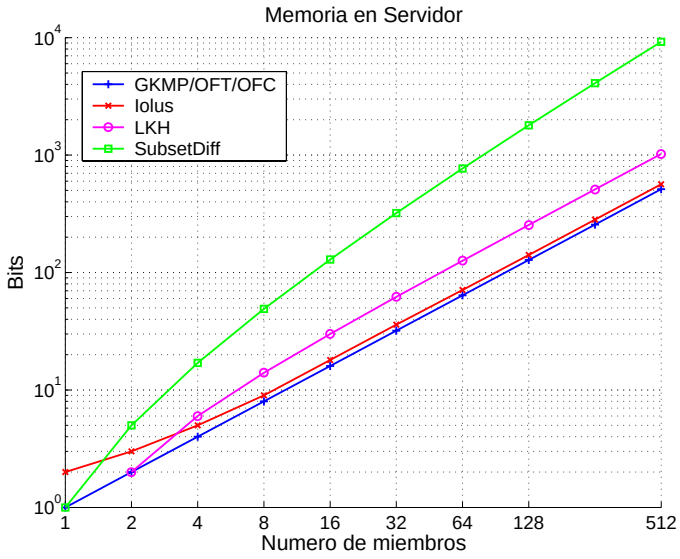
En GKMP, en la fase de inicialización, el KS debe generar de forma aleatoria tantas claves como miembros haya en el grupo. Equivalentemente deberá cifrar las mismas veces para repartirlas. Cada miembro sólo derá realizar 1 descifrado para obtener su clave. De la misma forma se comportan OFT y OFC.

En LKH en cambio, el Gestor de Claves debe generar tantos números aleatorios como nodos, y tantos cifrados como clientes. Por su parte el cliente sólo deberá realizar un descifrado.

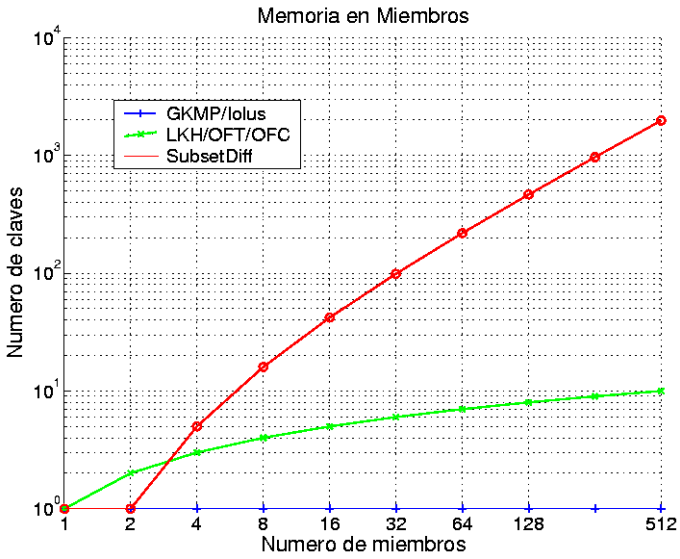
Actualización

Para cambiar la clave de sesión, GKMP requiere por parte del servidor una sola generación aleatoria y tantos cifrados como miembros. En cambio cada cliente deberá realizar sólo una operación de descifrado.

En LKH, cuando se produce una alta/baja, el GCKS deberá generar tantas nuevas claves y realizar tantos cifrados como profundidad del árbol menos 1. Cada cliente sólo deberá realizar



(a) GCKS



(b) Miembros

Figura 3.14: Cantidad de memoria en función del número de miembros del grupo.

un descifrado. Este comportamiento también lo presentan OFT y OFC.

Reactivación

Mención a parte se merece la acción de recuperar toda la información necesaria después de estar *off-line* durante cierto intervalo de tiempo.

En los casos en los que la atención es intermitente, el caso más simple de reactivación consiste en que el KS envíe al miembro todo el material de claves que necesita en ese momento. En ese caso un único cifrado/descifrado y comunicación punto a punto es necesario para este propósito.

3.6.6. Parámetros no aplicables

Aunque citados como parámetros de eficiencia en 2.5, el Control de congestión y *Workoverhead* de paquete de control no tienen sentido cuando se evalúan algoritmos. En ningún momento éstos definen el formato de paquete (con sus correspondientes cabeceras) a usar, por tanto no se puede calcular el *Workoverhead*. Por su parte, saber si existen mecanismos o no de recuperación frente a congestión es también más propio del nivel de protocolo que de algoritmo.

3.7. Contribución a la mejora en cuanto a memoria del Gestor de Claves

A la vista de los parámetros evaluados en el punto 3.6, se observa que al mejorar el ancho de banda de actualización el aspecto más perjudicado es la cantidad de memoria en el gestor de claves. A fin de superar esta pérdida de eficiencia, en [Canetti et al., 1999b] se apunta el uso de funciones pseudo-aleatorias para generar las claves.

3.7.1. Antecedentes

En la Fig 3.15 se ve un ejemplo simple del sistema *Minimal Storage Scheme* propuesto por Canetti. Las funciones pseudo-aleatorias se usan como herramienta para calcular *on-line* la clave de cada uno de los miembros. $F()$ se puede hacer pública o mantener secreto, pero su semilla, $r1$, forzosamente deberá ser conocida únicamente por el GC.

Cuando el Gestor de Claves quiere comunicarse con cualquiera de los miembros $(a, b, ...x)$ sólo debera aplicar la función pseudo-aleatoria a la identidad del miembro con el que quiere comunicarse, usando su parámetro secreto $r1$ y así obtendrá la clave que debe usar para cifrar la comunicación con él.

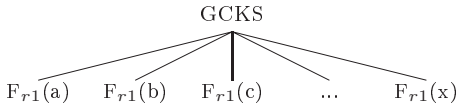


Figura 3.15: Ejemplo de *Minimal Storage Scheme*

Este método, reduce a 1 los párametros que debe guardar el GCKS, en vez de los N que debía guardar antes[11], pero no es útil para renegociaciones de clave, ya que cada vez que cambia la clave de sesión, la única forma de comunicarlo a los miembros es mediante un mensaje a cada uno de ellos.

De todas formas el sistema es válido para reducir la cantidad de memoria en sistemas como *Iolus*. Cada GSI sólo deberá almacenar una semilla (r_i) distinta, y el GSC global sólo tendrá que guardar tantos números aleatorios como subgrupos.

3.7.2. Propuesta de algoritmo: *Optimal Key Storage* (OKS)

El uso de funciones pseudo-aleatorias, sin embargo, puede aplicarse a los árboles lógicos de claves si se tienen en cuenta ciertas restricciones.

A partir del trabajo realizado conjuntamente con [Bin and Hua, 2002] del *Department of Electronic Engineering* de la *Shangai Jiaotong University*, se presentó un algoritmo de renegociación de claves en multicast [Pegueroles and Rico-Novella, 2003b] que usa funciones pseudo-aleatorias y se puede aplicar a algoritmos en árbol. Para facilitar su comprensión, a continuación se detalla un ejemplo sencillo.

[11] No se ha tenido en cuenta la SEK

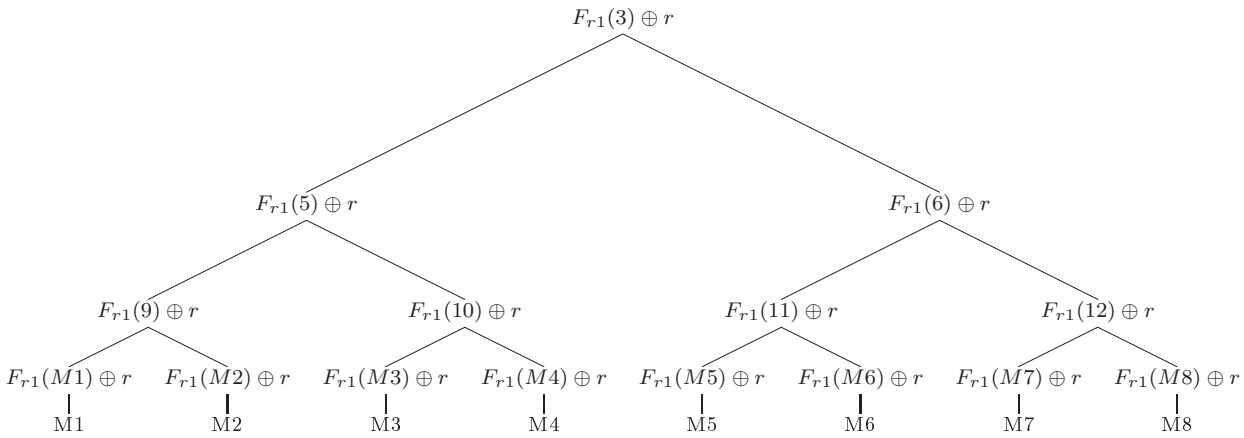


Figura 3.16: Árbol lógico de claves para OKS con 8 miembros

Inicialización

Considere el árbol binario de claves de la Fig 3.16 en el que las claves de las posiciones *(i,j)* se generan siguiendo la expresión (3.28).

$$K_{(i,j)} = F_{r1}(2^i + j) \oplus r \tag{3.28}$$

$F_{r1}()$ es una función pseudoaleatoria con semilla $r1$ y r es otro número aleatorio que servirá para poder actualizar las claves. El símbolo $\oplus$ denota la función XOR.

Nótese que tal como se ha construido el árbol, el GCKS sólo necesita saber las coordenadas *(i,j)* , la función $F_{r1}()$ y r para calcular la clave de cualquier nodo en cualquier momento. Las IKs de los miembros se calculan aplicando la misma función $F_{r1}()$ a algún parámetro relacionado con la identidad del miembro, esto permite que los miembros puedan "subir" o "bajar" de nivel sin tener que cambiar su IK.

Actualización

Cuando un miembro se da de alta o de baja el KS manda a todos los miembros que permanecen en el grupo el parámetro en (3.29).

$$P = r \oplus r' \tag{3.29}$$

Con este parámetro, cada miembro sólo tiene que calcular (3.30) para actualizar sus claves.

$$K'_{(i,j)} = K_{(i,j)} \oplus P = F_{r1}(2^i + j) \oplus r' \tag{3.30}$$

Imaginemos que $M7$ abandona el grupo (ejemplo de la Fig 3.17). La IK del miembro saliente es eliminada. La clave individual del miembro $M8$ pasa a ser la clave del nodo *(3,4)*. Finalmente el KS envía el parámetro de actualización $r \oplus r'$ a los miembros que quedan en el grupo según las reglas LKH.

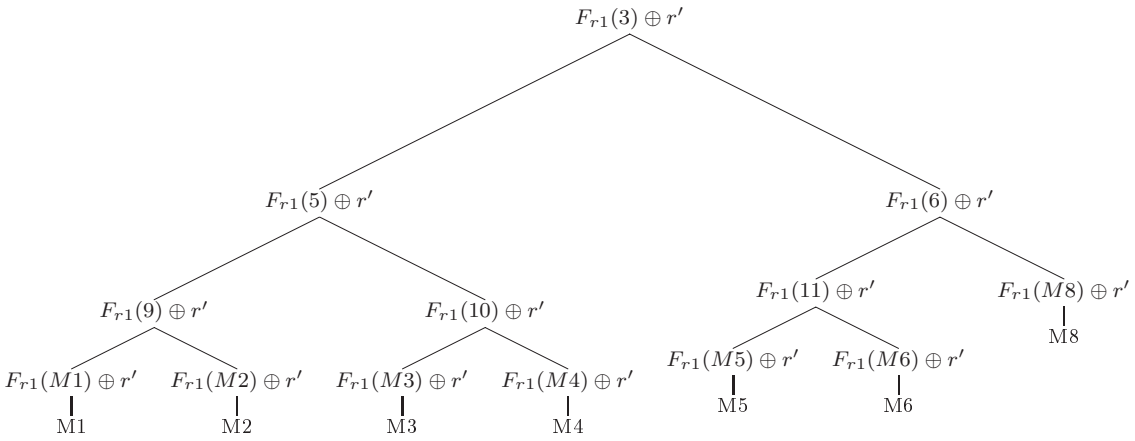


Figura 3.17: Baja de miembro en árbol lógico de claves OKS

Los mensajes que enviará por el canal multicast serán:

$$\{r \oplus r'\}_{F_{r1}(M8) \oplus r'}$$
$$\{r \oplus r'\}_{F_{r1}(11) \oplus r'}$$
$$\{r \oplus r'\}_{F_{r1}(5) \oplus r'}$$

Cada subgrupo del árbol podrá descifrar alguno de los mensajes enviados por el canal multicast y recuperar el parámetro de actualización. Con esa información, cada miembro es responsable de actualizar individualmente todas las claves que existen en el *path* desde su posición hasta la raíz según la expresión (3.30).

Aquí, al igual que en OFT y OFC se ha descargado cierto coste computacional a los miembros para reducir, en este caso, la cantidad de memoria necesaria en el GCKS.

Tal como se observa en la Fig 3.17, debe resaltarse que se actualizan todas las claves del árbol y no sólo las comprometidas. Una vez finalizado el proceso de actualización, el gestor puede borrar de su memoria el número aleatorio r ya que no lo necesitará para generar ninguna de las claves que a partir de ese momento contiene el árbol.

Ventajas e inconvenientes

Con este mecanismo se ha reducido a sólo 2 los números aleatorios que debe guardar el KS ($r1$ y r'^n) pero se ha incrementado el coste computacional de cada uno de los integrantes del grupo.

Por otra parte, un miembro no puede prestar atención intermitente ya que la pérdida de un único mensaje de actualización rompería la cadena de XORs y no podría recuperar las nuevas claves válidas. En ese caso deberá iniciarse un proceso de reactivación que tendrá el mismo coste que en LKH/OFT/OFC.

3.7.3. Análisis de seguridad

Un algoritmo de renegociación de claves se considera roto si a partir de los secretos de un miembro, o de la combinación de secretos de varios miembros, es posible obtener algún secreto no perteneciente a ese miembro o conjunto de miembros.

Existen distintos atacantes susceptibles de vulnerar la seguridad de los algoritmos de renegociación: un miembro autorizado, una confabulación de miembros autorizados, un miembro revocado y una confabulación de miembros revocados.

Los ataques de miembros únicos son casos particulares de confabulaciones en los que el número de confabuladores es 1. Además, debe tenerse en cuenta que usuarios autorizados pueden recopilar información durante sucesivas actualizaciones, este caso se analiza en el apartado "Confabulación de miembros autorizados durante sucesivos *rekeyings*"

Confabulación de miembros revocados

Durante toda la vida del grupo, las claves son de la forma $(F_{r1}(2^i + j) \oplus a)$ donde $F_{r1}()$ y a son sólo conocidos por el KS y, por tanto, los miembros entienden cada una de las claves como números completamente aleatorios. Como nadie a parte del KS conoce ni $F_{r1}(2^i + j)$ ni a, ningún usuario normal podrá calcular ni las claves que se usarán en el futuro ni las pasadas, ya que sólo conocen (i, j)

La información de que dispone un miembro revocado es: el conjunto de claves desde su posición hasta la raíz, por ejemplo si $M1$ fuese revocado del árbol de la Fig 3.17 $\{F_{r1}(9) \oplus r', F_{r1}(5) \oplus r', F_{r1}(3) \oplus r'\}$, su clave individual $\{F_{r1}(M1) \oplus r'\}$ y el parámetro de actualización que le permitió calcular su conjunto de claves $(r'^{(i-1)} \oplus r'^{i})$. Un grupo confabulador tiene esta misma información por cada uno de los confabuladores.

El miembro o grupo de miembros revocados (R) rompería el sistema si con esa información fuese capaz de calcular el parámetro de actualización que le permitiese volver a formar parte del grupo o una clave del grupo autorizado $(A = N - R)$ que le permitiese descifrar el parámetro de actualización.

Según las reglas LKH, los miembros revocados no pueden conseguir el parámetro de actualización ya que se ha enviado cifrado con una clave que no les corresponde.

Con la información de que disponen, los confabuladores sólo pueden intentar combinaciones XOR de sus secretos para conseguir uno que no les pertenezca. Como cada clave en el árbol tiene un índice distinto, la función XOR sobre dos claves del grupo no eliminará nunca el valor de $F_{r1}(i)$, tan sólo "borrará" el parámetro r'^{i} dejando como resultado la función XOR de dos funciones $F_{r1}(i)$, expresión (3.31). Además, por propiedades de las funciones pseudo-aleatorias, la probabilidad de que este resultado se corresponda con el resultado de otro índice sobre la misma función, puede ser tan pequeña como se desee.

$$F_{r1}(x) \oplus r'^{i} \oplus F_{r1}(y) \oplus r'^{i} = F_{r1}(x) \oplus F_{r1}(y) \neq F_{r1}(n)\, \forall n \in N - R\, \forall x, y \in R \qquad (3.31)$$

Confabulación de miembros autorizados

La única forma que un miembro autorizado (o grupo de ellos) tiene de romper el sistema es revelando sus secretos a un miembro no autorizado, de forma que este último pueda acceder al sistema sin realmente estar autorizado.

Este ataque es obvio, común a cualquier sistema de distribución de claves e inevitable (únicamente rastreable o trazable). La literatura no considera este caso como una vulnerabilidad del sistema.

La otra posibilidad es que combinen sus claves para obtener otro secreto que no les pertenezca. Si la información combinada pertenece a un único periodo de actualización nos encontramos en el mismo caso que el punto anterior. Todo lo afirmado anteriormente es igualmente válido aquí. Si la información combinada pertenece a distintos periodos de actualización se trata de confabulación de miembros durante sucesivos *rekeyings*.

Confabulación de miembros autorizados durante sucesivos *rekeyings*

Durante sucesivas actualizaciones, un miembro autorizado (o conjunto de ellos) puede ir recopilando información de subconjuntos de claves y parámetros de actualización (por ejemplo M1 obtendría $\{F_{r1}(9) \oplus r'^{i}$, $F_{r1}(5) \oplus r'^{i}$, $F_{r1}(3) \oplus r'^{i}\}$ y $r'^{i} \oplus r'^{(i+1)}$) para distintos índices *'i*.

En ese caso, a parte de lo considerado para el caso de miembros revocados, debe garantizarse que la función XOR sobre distintos parámetros de actualización obtenidos mientras se era un usuario autorizado del grupo (I_A) no den a lugar un parámetro de actualización correspondiente a un índice del periodo en que se ha sido revocado (I_R) (expresión (3.32))

$$(r'^{i} \oplus r'^{j}) \neq (r'^{i} \oplus r'^{k}) \forall i, j \in I_A, k \in I_R \tag{3.32}$$

3.7.4. Evaluación

A la vista de lo descrito en 3.7.2, la propuesta OKS sólo mejora la eficiencia en cuanto a claves a guardar por el KS. Cada miembro individual debe guardar las mismas claves que debía guardar en los algoritmos basados en árbol. Por su parte, el ancho de banda de actualización es el mismo que presentaban OFC/OFT: tantos mensajes como profundidad del árbol menos 1, con un único parámetro en cada mensaje.

Las expresiones de sus parámetros de eficiencia pueden encontrarse a continuación. La Tabla 3.9 y la Fig 3.18 muestra una comparativa con los algoritmos anteriormente presentados más significativos.

$$BW_{OKS-inic} = N \cdot (L \cdot (1 + \lceil log_2 N \rceil)) \tag{3.33}$$

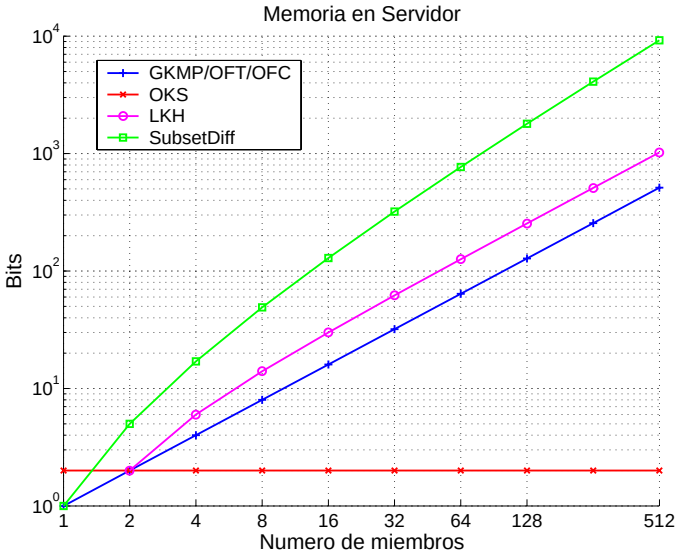
$$BW_{OKS-alta/baja} = L \cdot \lceil log_2 N \rceil \tag{3.34}$$

$$Memoria_{GCKS-OKS} = 2 \tag{3.35}$$

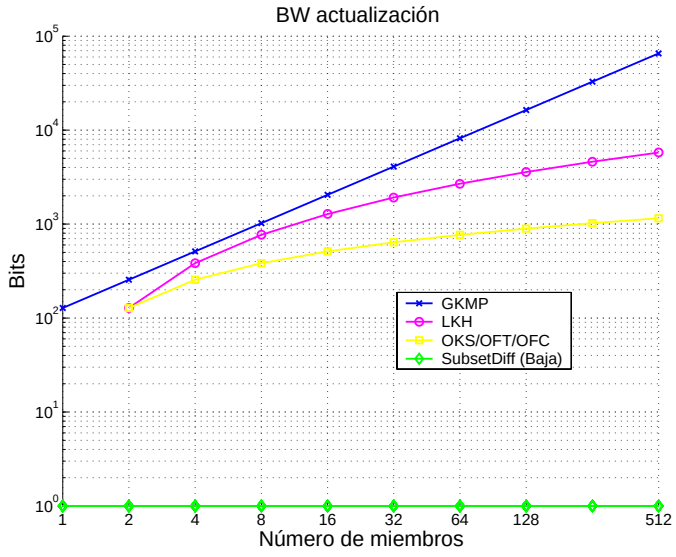
$$Memoria_{Miembro-OKS} = \lceil log_2 N \rceil + 1 \tag{3.36}$$

Tabla 3.9: Comparativa de OKS con otros algoritmos.

	GKMP	LKH	OFT/OFC	*SubsetDiff.*	OKS
Memoria en GCKS	$O(N)$	$O(2N)$	$O(N)$	$O(2Nlog_2N)$	$O(2)$
BW actualización	$O(N)$	$O(log_2^2N)$	$O(log_2N)$	$O(1)$	$O(log_2N)$



(a) Claves a guardar por el GCKS



(b) Ancho de banda de actualización

Figura 3.18: Parámetros de eficiencia de otros algoritmos respecto OKS

3.8. Contribución a la mejora en cuanto a número de mensajes

Otro parámetro de eficiencia susceptible de ser mejorado es el número de mensajes necesario para la actualización de claves. Recordemos que la mejora en este valor incide directamente en una mejora en la latencia de *rekeying*.

Hasta el momento, los algoritmos de renegociación de claves se han basado en mejorar "la forma de agrupar los miembros" y no tanto en estudiar la dependencia entre los nodos o las claves. *Iolus*, LKH y *Subset Difference* son un ejemplo de lo primero, mientras que OFT/OFC utilizan la dependencia entre nodos de una forma muy limitada.

El problema de *rekeying* en multicast es en realidad una ligera modificación del problema clásico denominado cifrado broadcast (en inglés *broadcast encryption*). Sin embargo son pocos los trabajos que abordan el problema de actualización de claves en multicast desde el enfoque clásico del *broadcast encryption*.

Una contribución en este sentido se presentó en [Pegueroles et al., 2003a] donde se consigue una mejora en cuanto a número de mensajes que usa aritmética modular. A continuación se explica brevemente esta propuesta.

3.8.1. Técnicas *broadcast encryption*

Tal como hemos comentado el problema de actualización de claves en multicast es una redefinición del problema *broadcast encryption*. En él se aborda la problemática de permitir a un nodo centralizado el envío de información por un canal broadcast a un conjunto arbitrario de usuarios autorizados. Esta transmisión debe realizarse en un número mínimo de pasos o mensajes [Fiat and Naor, 1993].

La principal diferencia entre los problemas broadcast y multicast *encryption* radica en que en el cifrado broadcast el KS no conoce la identidad de los usuarios que quiere "evitar". En *multicast encryption*, en cambio, el KS sí que conoce esta identidad, ya que, por definición, los usuarios son limitados y conocidos a priori.

Ciertos métodos *broadcast encryption* se basan principalmente en propiedades matemáticas de la teoría de números. Nuestra propuesta aborda la renegociación de claves multicast desde este enfoque. A continuación se detallan las características genéricas que debe tener un sistema de este tipo para adecuarse a nuestro escenario.

Requisitos matemáticos

Sea $M = \{M1,\ M2, ...,\ MN\}$ un grupo de N miembros que se comunican mediante multicast. Considere un conjunto de funciones $F = \{f_1,\ f_2, ...,\ f_N\}$ generadas por el KS. Cada miembro Mi sólo comparte el secreto $f_i()$ con el KS. Sea $A \subset M$ el subconjunto de miembros autorizados a quien se quiere repartir el secreto.

Se desea que las funciones tengan las siguientes propiedades:

- Dado un secreto, K, es fácil obtener el parámetro P tal que $f_i(P) = f_j(P) = k \forall i, j \in A$.
- El conocimiento de P no revela nada de k a ningún miembro perteneciente al grupo no autorizado $(M - A)$.

Si se cumplen estas propiedades, el envío multicast de un único mensaje P bastaría para repartir la nueva clave de sesión k. El algoritmo propuesto se basa en este comportamiento.

3.8.2. Propuesta de algoritmo: *Single Message LKH with OKS* (sm-LKH/OKS)

Inicialización

Considere un grupo multicast dinámico de tamaño N y un KS que comparte un secreto con cada uno de los miembros de este grupo. El KS construye un árbol lógico de claves, según las reglas de OKS. Cada nodo en el árbol es un número aleatorio generado según la expresión (3.28). Como en OKS, cada miembro se sitúa en una hoja del árbol y conoce las claves que hay desde su hoja hasta la raíz. En el ejemplo de la Fig 3.19 M3 conoce las claves enmarcadas en línea discontinua.

Cuando se debe acordar una nueva clave, el KS calcula un parámetro público según la expresión (3.37), donde $rnd_{(i,j)}$ es el resultado de la función pseudo-aleatoria de OKS en el nodo *(i,j)*, expr (3.38).

$$P = r_2 \prod_{(i,j)\subset S} (rnd_{(i,j)}) + (r'^{n} \oplus r'^{(n+1)}) \tag{3.37}$$

$$rnd_{(i,j)} = F_{r1}(2^i + j) \oplus r'^{n} \tag{3.38}$$

r_2 es el resultado de una función pseudo-aleatoria, diferente a $F_{r1}()$, y usada para evitar ataques por confabulación[12]. $\prod_{(i,j)\subset S}(rnd_{(i,j)})$ es el producto de todos los números aleatorios del subconjunto S, siendo S el subconjunto de nodos hermanos (*siblings*) al camino de claves comprometido. $(r'^{n} \oplus r'^{(n+1)})$ es el parámetro que necesitan los miembros del grupo para actualizar el árbol. Diremos que este parámetro está cegado (*blinded*) u ofuscado por el productorio de la expresión (3.37).

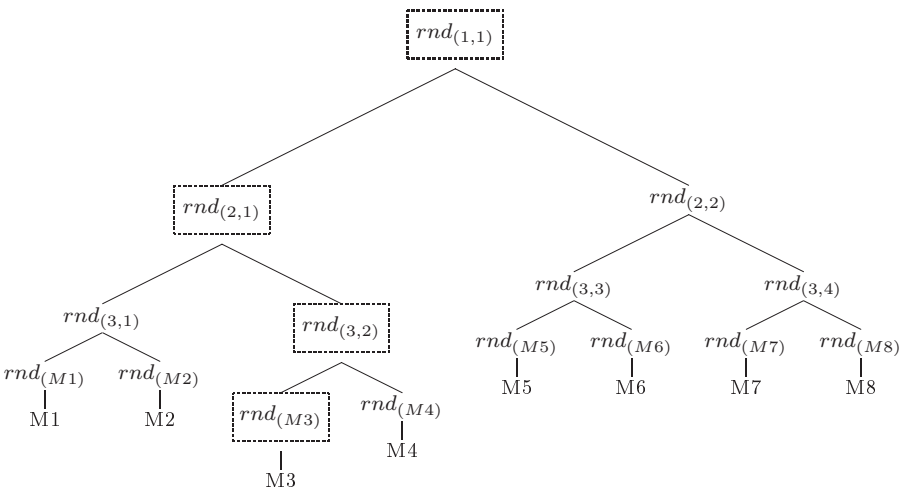


Figura 3.19: Árbol lógico de claves para sm-LKH/OKS con 8 miembros

[12]Los ataques por confabulación se discutirán más adelante en esta sección

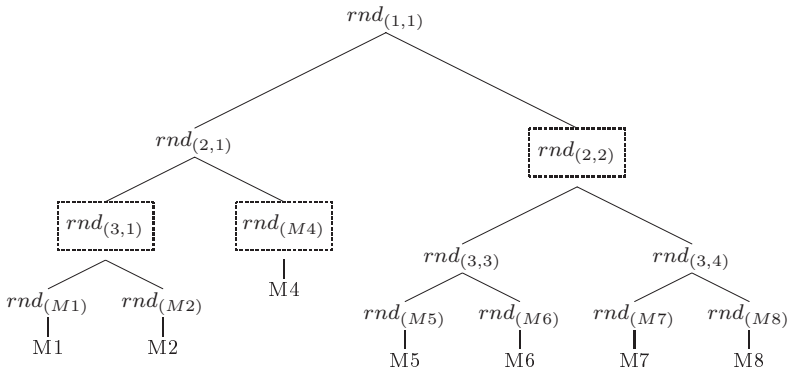


Figura 3.20: Baja de miembro en árbol lógico de claves sm-LKH/OKS

Actualización

Como ejemplo, considere que el miembro $M3$ de la Fig. 3.19 abandona el grupo. Las claves comprometidas se encuentran marcadas mientras que las claves incluidas en S están dentro de cajas cuadradas en la Fig 3.20.

El parámetro P se calculará según la expresión (3.39) y se enviará por el canal multicast.

$$P = r_2 \cdot rnd_{(2,2)} \cdot rnd_{(3,1)} \cdot rnd_{(M4)} + (r \oplus r') \tag{3.39}$$

Para recuperar el parámetro cegado, cada miembro realizará una reducción modular de P módulo alguno de los números aleatorios que conoce, correspondientes al camino desde su hoja hasta la raíz. Como este número aleatorio estará incluido en el producto de números usados para generar P, la reducción modular tendrá como resultado $r \oplus r'$. Observe la expresión (3.40).

$$P = [r_2 \cdot rnd_{(2,2)} \cdot rnd_{(3,1)} \cdot rnd_{(M4)} + (r \oplus r')] mod\, rnd_{(2,2)} = (r \oplus r') \tag{3.40}$$

Cada usuario puede actualizar sus claves, al igual que en OKS siguiendo la expresión (3.30).

Ventajas e inconvenientes

sm-LKH/OKS aprovecha las ventajas de OKS en cuanto a reducción de memoria en el KS. Al igual que éste, la cantidad de memoria en el KS no depende de N y fija en 3 los parámetros que debe guardar el KS ($r1$, $r2$, y r'^n). Este valor se incrementa en una unidad respecto OKS por ser necesario $r2$ para evitar ataques de confabulación.

Tal como ocurría con OKS, se incrementa el coste computacional de cada uno de los integrantes del grupo, pero ahora los miembros ya no realizan operaciones de descifrado sino únicamente reducción modular, mucho menos costosas que las anteriores.

Por otra parte, un miembro no puede prestar atención intermitente ya que la pérdida de un único mensaje de actualización rompería la cadena de XORs y no podría recuperar las nuevas claves válidas. En ese caso deberá iniciarse un proceso de reactivación que tendrá el mismo coste que en LKH/OFT/OFC.

La latencia se reduce al mínimo, como en el caso de *Subset Difference* con un sólo mensaje

de actualización, aunque este mensaje tiene una longitud mayor[13] con lo que el orden de magnitud del ancho de banda se mantiene.

Tal como se verá en el Capítulo 4, la propia mecánica del algoritmo lo hace adecuado para proceso por lotes.

3.8.3. Análisis de Seguridad

Al igual que en OKS, existen cuatro posibles atacantes al sistema sm-LKH/OKS: un miembro autorizado, una confabulación de miembros autorizados, un miembro revocado y una confabulación de miembros revocados. Tal como se obró en 3.7.3 a continuación se presentan los casos de confabulación, por ser los más genéricos, en los supuestos de único *rekeying* y sucesivos *rekeyings*.

Confabulación de miembros revocados

La única forma de romper el sistema consiste en encontrar alguno de los factores de $\prod_{(i,j)\subset S} rnd_{(i,j)}$. Si algún atacante llega a saber cualquiera de esos $rnd_{(i,j)}$ será capaz de realizar la reducción modular y obtener el parámetro de actualización.

Un único miembro revocado del sistema no conoce ninguno de los factores ya que, por definición, las claves que conoce no se incluyen en el mensaje de *rekeying*. Además, los secretos que él conocía mientras era miembro autorizado del grupo no se usarán en el futuro ya que son actualizados al abandonar el grupo

Al igual que en los miembros individuales, una confabulación de miembros revocados no puede obtener ninguna información sobre los secretos usados, ya que todos los parámetros que ellos conocían no serán de nuevo usados en el mensaje de *rekeying*, y todos fueron actualizados.

Confabulación de miembros autorizados

Si no se usara el parámetro $r2$ en la expresión 3.37, dos atacantes autorizados podrían obtener algún factor que no les corresponde mediante el análisis de mensajes de *rekeying* junto con sus propios secretos. A continuación se presenta un ejemplo.

De nuevo, considere que $M3$ abandona el grupo. Suponga que $r2$ no se incluye en la expresión (3.39). Dos miembros autorizados confabuladores, por ejemplo $M1$ y $M4$, podrían colaborar para conocer el secreto $rnd_{(2,2)}$ mediante el análisis del mensaje de actualización, según los siguientes pasos:

1. $M1$ obtiene el secreto $(r \oplus r')$ mediante la reducción modular con su secreto $(rnd_{(3,1)})$
2. $M1$ obtiene $\prod_{(i,j)\subset S} rnd_{(i,j)}$ restando $(r \oplus r')$ de $P = rnd_{(2,2)} \cdot rnd_{(3,1)} \cdot rnd_{(M4)} + (r \oplus r')$.
3. $M1$ divide $rnd_{(3,1)}$ de $\prod_{(i,j)\subset S} rnd_{(i,,j)}$ y le da el resultado a $M4$.
4. Finalmente $M4$ divide $rnd_{(M4)}$ de los datos que le ha dado $M1$ y obtiene $rnd_{(2,2)}$. Que es un secreto que no pertenecía ni a $M4$ ni a $M1$.

[13] Este aspecto se discutirá con más detalle en la sección de Evaluación

Tanto $M1$ como $M4$ conocen el secreto $(r \oplus r')$. Por tanto, podrán actualizar $rnd_{(2,2)}$ y así descifrar la nueva clave de sesión cuando abandonen el grupo.

Este agujero de seguridad se supera con la introducción del número aleatorio $r2$ a la expresión (3.37). De esta forma, la única manera que tiene $M4$ de obtener $rnd_{(2,2)}$ es factorizando $rnd_{(2,2)} \cdot r2$. Este problema es considerado de difícil resolución si el resultado del producto está compuesto por factores primos suficientemente grandes. Más adelante en esta sección discutiremos cómo se puede asegurar que los $rnd_{(i,j)}$ estén compuestos por primos grandes.

De igual forma, se debe notar que este ataque sólo es posible si se confabulan un mínimo de $log_2 N - 1$ miembros, y además, éstos deben estar situados en posiciones concretas del árbol. Cada uno debe pertenecer a un subconjunto que le dé acceso a uno de los rnd_i usados en P. Por tanto, para los escenarios donde es útil nuestra propuesta ($N \uparrow\uparrow$) los ataques por confabulación aún son más difíciles.

Miembros autorizados durante sucesivos rekeyings

Otra amenaza importante para nuestro sistema es lo que un usuario autorizado puede averiguar analizando la información de la que dispone durante sucesivos *rekeyings*. Si los números aleatorios fueran reutilizados para distintos mensajes de actualización un miembro del grupo podría fácilmente encontrar un factor con sólo aplicar al algoritmo de máximo común divisor. Se puede ver un ejemplo en la expresión (3.41).

Mensaje 1 ($M3$ de la Fig. 3.20, revocado):

$$P = r2 \cdot rnd_{(2,2)} \cdot rnd_{(3,1)} \cdot rnd_{(M4)} + (r \oplus r') \tag{3.41}$$

Mensaje 2 ($M1$ de la Fig. 3.20, revocado):

$$P' = r2' \cdot rnd_{(M2)} \cdot rnd_{(M4)} \cdot rnd_{(2,2)} + (r' \oplus r'') \tag{3.42}$$

Con estos dos mensajes, $M4$ que es un miembro autorizado puede calcular fácilmente $r2 \cdot rnd_{(2,2)} \cdot rnd_{(3,1)}$ y $r2' \cdot rnd_{(2,2)} \cdot rnd_{(M2)}$ con sólo encontrar el secreto y restándoselo de P. Con estos dos productos, es fácil encontrar $rnd_{(2,2)}$ como el $mcd(r2' \cdot rnd_{(2,2)} \cdot rnd_{(M2)}, r2 \cdot rnd_{(2,2)} \cdot rnd_{(3,1)})$.

Para evitar este ataque, todos los números aleatorios en el árbol de claves deben ser actualizados cada vez que se realiza un *rekeying*, y así se evita su reuso.

Si sólo se usan mecanismos LKH para actualizar todos los aleatorios se produce un incremento considerable del número de mensajes y por tanto del ancho de banda necesario. Para evitar este efecto al diseñar el algoritmo propuesto se combinó con el sistema de mínima memoria en KS descrito anteriormente (OKS).

Como se hizo notar en 3.7.2, en cada *rekeying* se actualiza todo el árbol y se recalculan todos los números aleatorios. De esta forma distintos mensajes de actualización correspondientes a distintas renegociaciones de clave tendrán distintos factores y el algoritmo de máximo común divisor no revelará nada sobre esos factores.

Además, no se requiere un incremento de mensajes para actualizar todos los aleatorios ya que el mecanismo de *rekeying* de OKS delega las funciones de actualización del árbol a los usuarios individuales.

3.8.4. Restricciones

Por descontado, para que al algoritmo propuesto funcione correctamente se requiere tener en cuenta algunas restricciones sobre los parámetros a usar.

En primer lugar, $(r \oplus r')mod\, rnd_{(i,j)} = (r \oplus r')$ implica que $(r \oplus r') < rnd_{(i,j)}$ para todo *(i,j)*. Esta condición es de fácil cumplimiento ya que por diseño se pueden fijar, por ejemplo, las longitudes en número de bits de cada uno de estos parámetros.

Por otra parte, como se comentó anteriormente, los números $rnd_{(i,j)}$ deberán estar formados como mínimo por un primo grande. Esta condición hace que la factorización sea un problema considerado difícil en criptografía, y sobre el cual se basan numerosos algoritmos robustos como el RSA.

Es importante remarcar que el método propuesto no requiere que los números $rnd_{(i,j)}$ sean primos ellos mismos. El hecho de no ser un único número primo puede incluso hacer más seguro el sistema ya que la factorización de $\prod_{(i,j)\subset S} rnd_{(i,j)}$ no revelará nada sobre los $rnd_{(i,j)}$ sino que dará como resultado sus factores, y encontrar un $rnd_{(i,j)}$ válido sólo será posible mediante combinaciones de esos factores.

En cualquier caso en [Menezes et al., 2001] se puede encontrar la información necesaria de cómo se puede asegurar que un número aleatorio con una longitud mínima de bits contenga como mínimo un primo de otra determinada longitud.

3.8.5. Evaluación

Recordemos que el objetivo de sm-LKH/OKS era reducir el número de mensajes de actualización. Esta mejora se ve reflejada en el parámetro de latencia. Además, era preciso no empeorar ninguno de los demás parámetros de eficiencia.

En sm-LKH/OKS, el KS debe guardar tan solo 3 parámetros para mantener el árbol lógico de claves. Esta característica viene heredada del sistema OKS (expr. (3.43))

$$Memoria_{GCKS-sm-LKH/OKS} = 3 \tag{3.43}$$

Además, tal como sucedía en los otros algoritmos basados en árboles lógicos de claves (LKH/OFT/OFC), cada miembro sólo debe guardar las claves que se encuentran en el camino desde su hoja hasta la raíz del árbol (expr. (3.44)).

$$Memoria_{Miembro-sm-LKH/OKS} = \lceil log_2 N \rceil + 1 \tag{3.44}$$

En cuanto a coste computacional, cada vez que se produce un *rekeying* todos los miembros deben actualizar todas las claves y, por tanto, se incrementa el coste computacional, pero éste se ve repartido entre todos los miembros y, además, las operaciones a realizar por los clientes son reducciones modulares, mucho menos costosas computacionalmente que las operaciones de descifrado.

La latencia, tal como ocurría con el caso de *Subset Difference*, es mínima, al sólo necesitarse un mensaje para la actualización (expr. (3.45))

$$Numero\, mensajes\, actualizacion_{sm-LKH/OKS} = 1 \tag{3.45}$$

En cuanto a ancho de banda de inicialización, a cada miembro se le deben notificar tantos números aleatorios como profundidad del árbol, estos se reparten de forma independiente a cada uno de los integrantes del grupo mediante canal unicast, por tanto tendrá el mismo comportamiento que los demás algoritmos basados en árboles, expr (3.46).

$$BW_{sm-LKH/OKS-inic} = N \cdot (L \cdot (1 + \lceil log_2 N \rceil)) \tag{3.46}$$

Mención a parte merece el ancho de banda de actualización. Éste resulta del mismo orden que el presentado por OFC/OFT ya que, aunque sólo se envía un mensaje, éste es el resultado del producto de $log_2 N$ factores, y si cada uno de ellos es de longitud L (misma longitud que las claves) el ancho de banda de actualización se mantiene (expr.(3.47)).

Es preciso matizar que si se mantiene la misma longitud de clave para LKH y sm-LKH el nivel de seguridad se reduce considerablemente. En el primer caso, al tratarse sólo de la longitud de la clave de cifrado de un algoritmo simétrico, una longitud de 128 bits proporciona un nivel de seguridad suficiente. La propuesta sm-LKH, sin embargo, basa su seguridad en la dificultad de factorizar números compuestos por primos grandes, en este supuesto, los factores deberían tener del orden de 512 bits para proporcionar un nivel de seguridad equivalente al de las técnicas simétricas. En las Fig 3.21 y 3.23 se han considerado los valores para distintas longitudes de factores en el caso sm-LKH (256,512 y 1024 bits), el caso de 128 bits coincide con el de LKH.

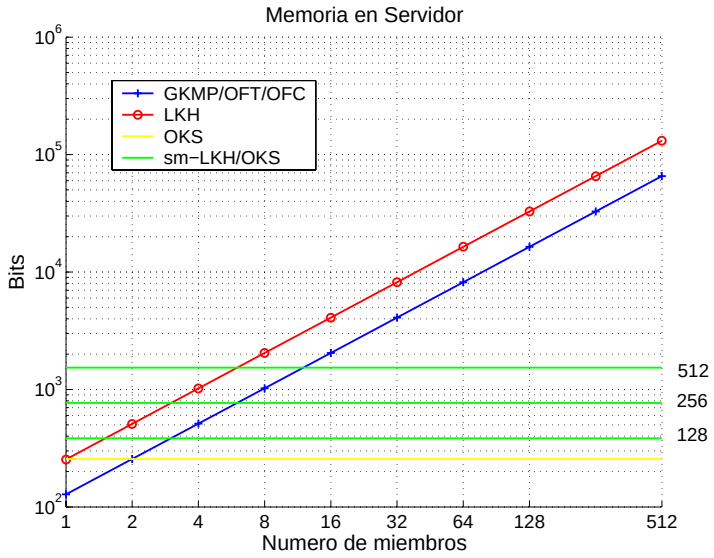
De todos modos, aunque en el caso de una renegociación individual este valor no se reduce, en el Capítulo 4 se verá como la forma en que se construye el mensaje de actualización lo hace mucho más adecuado que otros algoritmos para renegociaciones por lotes.

$$BW_{sm-LKH/OKS-alta/baja} = L \cdot \lceil log_2 N \rceil \tag{3.47}$$

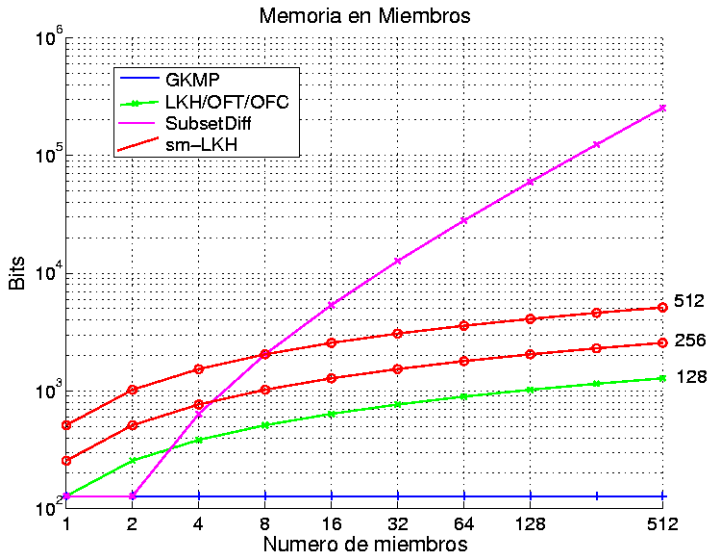
La Tabla 3.11 y las Fig 3.21 3.22 y 3.23 muestra una comparativa de los parámetros de eficiencia más significativos con los algoritmos anteriormente presentados.

Tabla 3.11: Comparativa de sm-LKH/OKS con otros algoritmos.

	GKMP	LKH	OFT/OFC	*SubsetDiff.*	OKS	sm-LKH/OKS
Memoria en GCKS	$O(N)$	$O(2N)$	$O(N)$	$O(2Nlog_2N)$	$O(2)$	$O(3)$
Memoria en miembro	$O(1)$	$O(log_2N)$	$O(log_2N)$	$O(N!)$	$O(log_2N)$	$O(log_2N)$
Latencia	$O(N)$	$O(log_2N)$	$O(log_2N)$	$O(1)$	$O(log_2N)$	$O(1)$
BW inicialización	$O(N)$	$O(Nlog_2N)$	$O(Nlog_2N)$	$O(N!)$	$O(Nlog_2N)$	$O(Nlog_2N)$
BW actualización	$O(N)$	$O(log_2^2N)$	$O(log_2N)$	$O(1)$	$O(log_2N)$	$O(log_2N)$

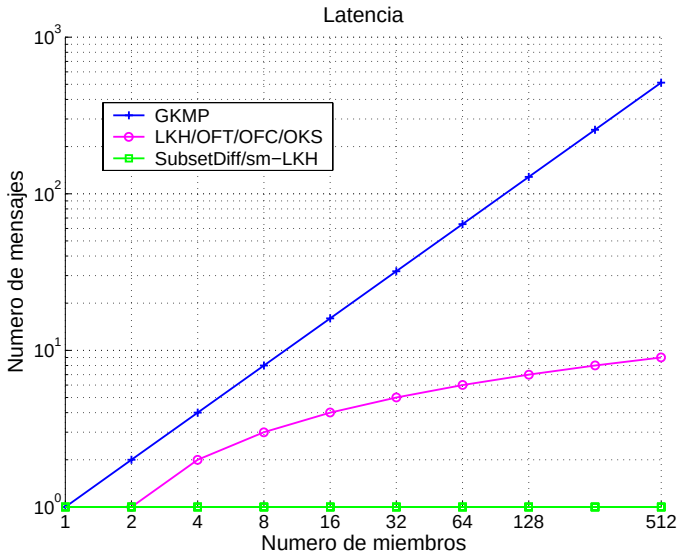


(a) GCKS



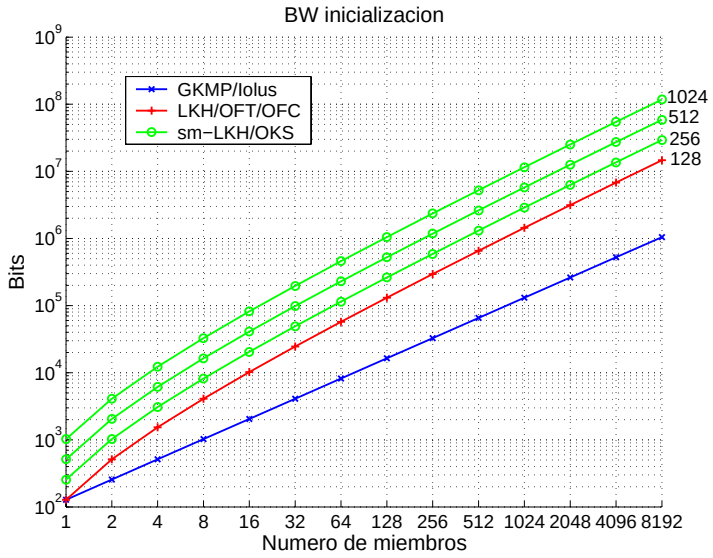
(b) Miembro

Figura 3.21: Memoria necesaria de otros algoritmos respecto sm-LKH/OKS

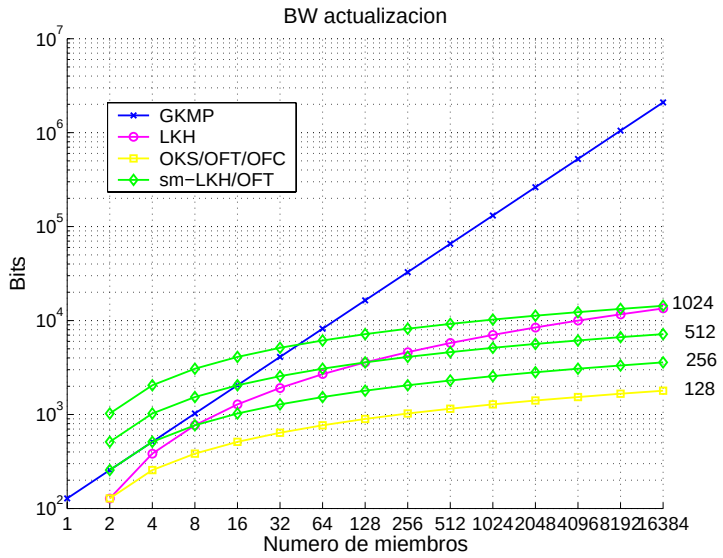


(a) Claves a guardar por el GCKS

Figura 3.22: Latencia de otros algoritmos respecto sm-LKH/OKS



(a) Inicialización



(b) Actualización

Figura 3.23: Ancho de banda de otros algoritmos respecto sm-LKH/OKS

3.9. Conclusiones

Una de las áreas problemáticas a abordar por la seguridad en multicast es la gestión eficiente del material de claves. Mediante el uso de una única clave de sesión, compartida por todos los miembros de un grupo, se consigue confidencialidad y autenticación de grupo. Para que el nivel de confidencialidad sea máximo (confidencialidad estricta hacia adelante y hacia atrás) debe cambiarse esta clave de sesión cada vez que un miembro se de de alta o de baja en el grupo. Esta condición hace necesaria la búsqueda de algoritmos de gestión de claves escalables y adecuados para grupos dinámicos.

Los parámetros más usados para evaluar la eficiencia de los algoritmos de gestión de claves son: ancho de banda necesario (tanto para la inicialización como para la actualización de las claves), cantidad de memoria requerida en el servidor y en cada uno de los miembros y latencia del proceso de actualización. Los algoritmos de más éxito son los que consiguen romper la dependencia de estos parámetros con el número de integrantes del grupo multicast.

La primera propuesta para gestionar las claves de grupos se denominó GKMP. Éste presenta un mecanismo muy sencillo basado en la interacción punto a punto del servidor de claves con cada uno de los miembros del grupo cada vez que se requiere una actualización de clave. A pesar de su sencillez, como los valores de sus parámetros de eficiencia dependen todos del número de integrantes del grupo, no se considera adecuado para grupos grandes y dinámicos. Con el fin de superar los inconvenientes que presenta GKMP existen en la literatura propuestas de algoritmos de gestión de claves en grupo alternativas a GKMP.

Iolus, supuso una primera mejora a GKMP. Se basa en la división del grupo multicast global en subgrupos afines geográficamente, de forma que al darse de alta o de baja un miembro, éste sólo afecta a sus compañeros de subgrupo, en vez de a todos los integrantes del conjunto, como era el caso de GKMP. Esta solución sí que reduce en valor absoluto los parámetros de eficiencia pero no rompe su dependencia con el número de miembros del grupo, ya que sus parámetros de eficiencia siguen ligados al número de integrantes (esta vez) de subgrupo.

Las propuestas más ampliamente aceptadas han sido las basadas en árboles lógicos de claves. Éstas reducen el número de mensajes de *rekeying* (y por tanto el ancho de banda de actualización) a valores no dependientes del número de integrantes del grupo sino del logaritmo de número de miembros. Dentro de este grupo de algoritmos se encuentran LKH, OFT y OFC. Una generalización de estos algoritmos es la propuesta de *Subset Difference*. La mejora en cuanto a ancho de banda es considerable pero la evaluación de los mismos pone de manifiesto la necesidad de mejoras en cuanto a número de mensajes (latencia) y memoria necesaria en el Gestor de Claves.

La primera contribución de este capítulo es el algoritmo *Optimal Key Storage* (OKS) que reduce la cantidad de memoria necesaria en el KS a sólo 2 números aleatorios. Se rompe la dependencia con el tamaño del grupo y se consigue mantener el resto de parámetros de eficiencia. La propuesta está basada en el uso de funciones pseudo-aleatorias. Éstas se utilizan como "reglas nemotécnicas" por el Gestor de Claves para poder generar en cualquier instante de tiempo, de forma *on-line*, cualquier clave que éste necesite saber.

Como segunda contribución se ha presentado *single-message LKH with Optimal Key Storage* (sm-LKH/OHS). Ésta aprovecha las ventajas de OKS, mantiene todos sus parámetros

de eficiencia y además reduce el número de mensajes de actualización a 1. Con esto se consigue reducir la latencia aunque, al tratarse de un mensaje mayor, el ancho de banda de actualización no disminuye.

Este algoritmo une a las funciones pseudo-aleatorias, propiedades de la reducción modular. El nivel de seguridad y la evaluación de las dos propuestas realizadas también se ha discutido en este capítulo.

Capítulo 4

Gestión de claves por lotes en grupos multicast

4.1. Introducción

En comunicaciones de grupo, la confidencialidad estricta hacia adelante y hacia atrás sólo se puede conseguir con sistemas como los descritos en el Capítulo 3, pero el alto nivel de seguridad se paga con un elevado coste computacional y de ancho de banda.

En numerosos casos, la seguridad que requiere un determinado servicio multimedia no justifica el gasto de tantos recursos. Una forma de mejorar las prestaciones de los algoritmos de gestión de claves consiste en realizar las operaciones de actualización de forma periódica, esta mejora, sin embargo, va en detrimento del nivel de seguridad.

Los algoritmos de gestión de claves que operan de esta manera se denominan algoritmos con proceso por lotes o en *batch*.

Cuando las claves se actualizan en instantes determinados de tiempo, se deben posponer y acumular las altas y bajas solicitadas entre dos actualizaciones hasta el instante en el que se decide realizar la actualización. De alguna forma, el hecho de posponer el procesado de peticiones independiza la frecuencia de *rekeying* de las dimensiones del grupo y de su dinamismo. Este mecanismo mejora la escalabilidad de los algoritmos del Capítulo 3 pero introduce un nuevo campo de estudio en la gestión de claves multicast.

En este capítulo se analizan los algoritmos de gestión de claves que operan por lotes, se discuten sus características y requisitos y se realizan distintas propuestas para su mejora.

En primer lugar se presenta el sistema Lam-Gouda. Se trata de la primera propuesta de gestión por lotes realizada en la literatura. Consta de un principio de funcionamiento muy simple pero con un inconveniente: cuando se realiza en escenarios reales da lugar a árboles lógicos de claves desbalanceados y, por tanto, decrece su eficiencia. Para solucionar este problema los mismos autores introdujeron una variación de funcionamiento (ajuste de balanceo) que también será presentada en este capítulo. Como se verá, esta solución incrementa el coste computacional y de gestión global.

El sistema Lam-Gouda mantiene a los miembros del grupo en la misma posición del árbol que se les asignó inicialmente. Esto facilita al KS la gestión del árbol, pero el desbalanceo de árboles supone un coste que no justifica su elección.

Como segundo punto, en este capítulo se presenta la propuesta de mejora consistente en un algoritmo de gestión por lotes con miembros dinámicos. Es decir, los miembros, a fin de mantener el árbol balanceado, pueden cambiar de posición en el árbol de un *batch* a otro. Este sistema se evalúa usando patrones de tráfico sintético siguiendo un comportamiento real (Anexo A).

Como los métodos de proceso por lotes son ortogonales con los algoritmos de renegociación de claves, en tercer lugar se aplican las técnicas en *batch* a las propuestas realizadas en el Capítulo 3.

4.2. Compromiso seguridad-ancho de banda

Con los algoritmos basados en árboles, la renegociación individual de claves llega a su cota mínima en cuanto a número de mensajes [Snoeyink et al., 2001]. De cualquier forma las técnicas de proceso por lotes se presentan como una alternativa viable para superar dicha cota.

En los algoritmos de *batch rekeying* las peticiones de *join* y *leave* que se solicitan durante un intervalo de tiempo se agrupan y se procesan de forma conjunta al final de este periodo. Este sencillo mecanismo puede llegar a disminuir enormemente el ancho de banda requerido para la renegociación de claves pero, por otra parte, reduce considerablemente el nivel de seguridad del sistema.

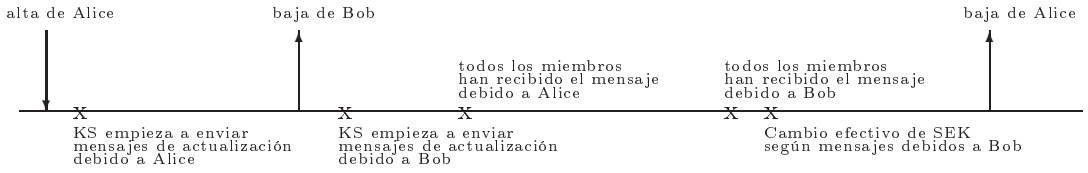
Ancho de banda

Existen dos casos en los que los algoritmos de proceso por lotes reducen el ancho de banda. El primero, consiste en eliminar las actualizaciones innecesarias inherentes en los métodos de proceso individual. En este caso se consigue una mejora en ancho de banda sin perjudicar a la seguridad. El segundo caso no sólo elimina las renegociaciones innecesarias sino que también reduce las que mantienen la confidencialidad estricta. Esta reducción afecta directamente el nivel de seguridad ya que será imposible ofrecer confidencialidad estricta hacia adelante y en algunos casos hacia atrás.

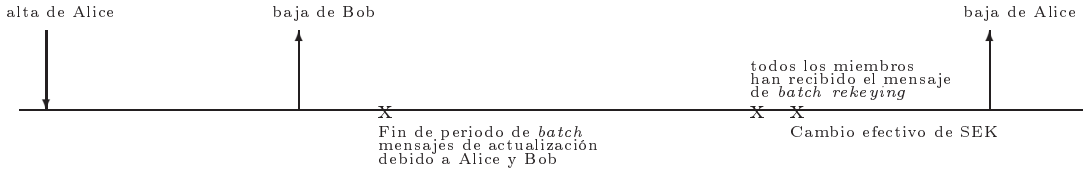
Eliminación de renegociaciones innecesarias

En la Fig 4.1a puede observarse un sencillo ejemplo de *rekeying* innecesario. Cuando el tiempo entre solicitudes es menor a la latencia, todos los mensajes que el KS envía debido a la primera solicitud no llegan a ser útiles pues deben enviarse nuevos mensajes de actualización con nueva clave antes de que el servidor pueda hacer efectivo el cambio de SEK. Una primera mejora en uso de ancho de banda consiste en eliminar los mensajes debidos a las actualizaciones innecesarias.

Determinar qué peticiones van a ser necesarias o innecesarias y decidir en qué momento exacto realizar el *rekeying* es una tarea muy difícil ya que el patrón de comportamiento de usuarios es aleatorio. Sólo se puede determinar la necesidad de una solicitud de forma probabilística mediante técnicas predictivas. La forma habitual de operar es fijando un determinado periodo de renegociación en función de la latencia del sistema (que también es



(a) Proceso individual



(b) Proceso por lotes

Figura 4.1: Ejemplo de diagrama temporal de algoritmos de gestión de claves

variable) y procesando los *rekeyings* al final de ese periodo. Obviamente en este caso también se eliminan renegociaciones necesarias.

Reducción de renegociaciones necesarias

Por renegociaciones necesarias, entendemos aquellas que proporcionan confidencialidad estricta hacia adelante y hacia atrás y que no presentan solapamientos en cuanto a mensajes de actualización.

En la subfigura (b) de Fig 4.1, el periodo de *batch*, por diseño, siempre será superior a la latencia, y como todas las peticiones se procesan conjuntamente, es imposible que se den casos de renegociaciones innecesarias. Pero, imaginemos que después de la "baja de Alice" no se produjese ninguna solicitud que se solapase con ella. En ese caso, la renegociación correspondiente sí que sería necesaria, sin embargo, el cambio efectivo de SEK no se produciría hasta cierto tiempo después de que expirara el correspondiente periodo de *batch* y, por consiguiente, se pierde la confidencialidad estricta hacia adelante.

Seguridad

En el mismo instante en que Alice manifiesta su deseo de abandonar el grupo debería ser incapaz de acceder a la información cifrada. Si el cambio real de SEK no se produce en ese momento, el miembro está accediendo a información posteriormente a su "baja" y se pierde la *Perfect Forward Secrecy*.

Por otra parte, cuando un miembro quiere acceder al grupo, existen dos opciones: darlo de alta inmediatamente revelándole el material de claves en uso al momento o hacerle esperar hasta el siguiente instante de *rekeying*.

En el segundo caso, la seguridad no se ve comprometida ya que las claves de que dispondrá sólo serán válidas para un determinado periodo. Sin embargo, estamos retardando el acceso al servicio del usuario, con la pérdida de calidad de servicio subjetiva que eso supone.

Si, en cambio, se decide incorporar al miembro inmediatamente se pierde la confidencialidad estricta hacia atrás. El nuevo miembro será capaz de descifrar toda la información

intercambiada desde el último instante de *rekeying* hasta el momento de su alta sin estar autorizado para ello.

Es fácil deducir que cuanto menor sea el periodo de *rekeying* menor será la cantidad de información a la que se pueda acceder de forma no lícita. Igualmente, a menor tiempo de *batch* menor será la cantidad de renegociaciones necesarias que se eliminarán y, por tanto, el número de mensajes y ancho de banda crecerán.

El compromiso entre *periodo de renegociación/ancho de banda* y *seguridad* dependerá de cada caso particular.

4.3. Proceso por lotes con miembros estáticos: Sistema *Lam-Gouda*

El primer algoritmo de renegociación de claves por lotes lo presentaron [Yang et al., 2001]. El sistema consiste en aplicar periódicamente un método muy simple de marcado de los nodos del árbol, dependiendo de si el nodo se debe actualizar o no. Después de este proceso, se envían los mensajes de actualización necesarios a los miembros que quedan en el grupo siguiendo las reglas LKH.

En este sistema, al igual que en LKH, los miembros siempre ocupan una posición determinada en el árbol. Es decir, pueden subir o bajar en el camino que existe desde su hoja hasta la raíz, pero nunca abandonarán el camino determinado inicialmente. Por este motivo diremos que se trata de un sistema con miembros estáticos.

Seguidamente, se explica este método con más detalle.

4.3.1. Lam-Gouda *Batch Rekeying*

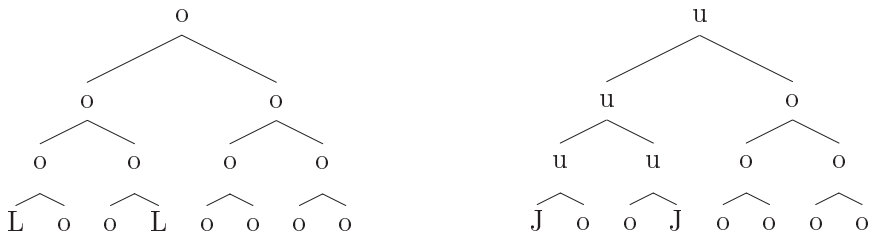
Sea J el número de peticiones de alta o *joinings* y L el número de abandonos del grupo o *leavings*. Sea T_{batch} el intervalo de tiempo en que se dan las $J + L$ solicitudes. Se asume que en un periodo de *batch* un mismo usuario no se une y abandona el grupo ni viceversa. La principal tarea del Servidor de Claves, después de recoger todas las solicitudes, es determinar qué KEKs del árbol deben ser actualizadas, añadidas o eliminadas.

Recordemos que en el *rekeying* individual, todas las claves del camino desde la hoja del miembro implicado hasta la raíz debían ser actualizadas. En el proceso por lotes, en cambio, habrán distintos miembros implicados y por consiguiente distintos *paths* a actualizar. La unión de estos *paths* forma un subárbol que se denominará subárbol de actualización (*rekey subtree*) que incluye todas las claves a añadir o cambiar. Por convenio, en el subárbol de renegociación no se incluyen las IKs.

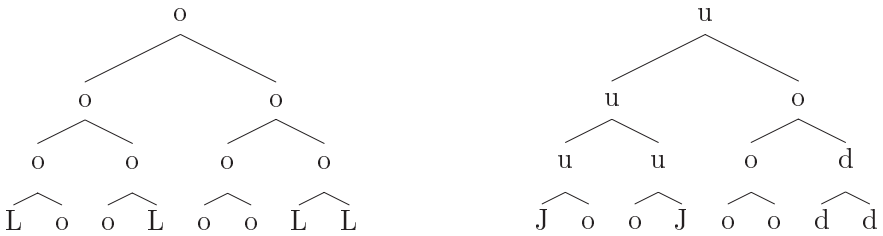
El KS no tiene control sobre los miembros que abandonan el grupo pero sí puede decidir en qué posición coloca a los nuevos miembros que se añaden al árbol. Así, el KS debe prestar especial atención en cómo realiza esa distribución, de forma que dé lugar a un número mínimo de cifrados y mensajes de actualización.

De forma heurística, puede resumirse el algoritmo de Lam-Gouda según los siguientes casos esquematizados en la Fig 4.2.

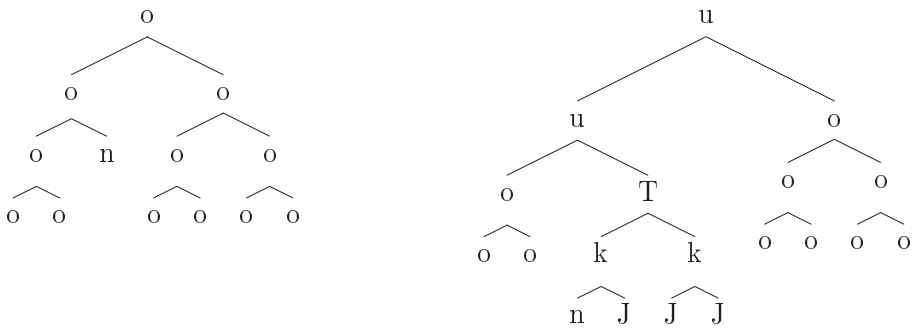
- Caso 1: J=L (subfigura (a))



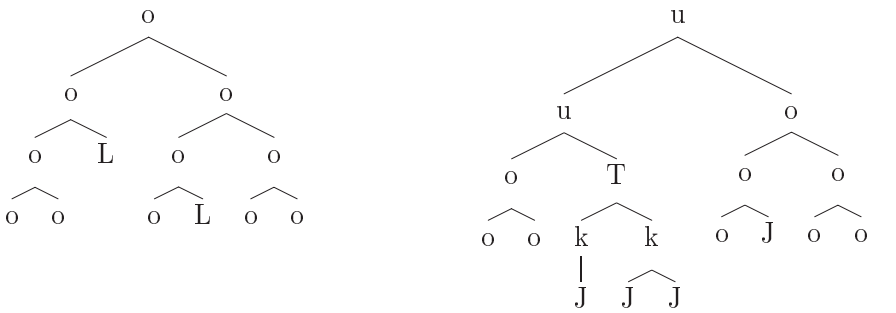
(a) J=L



(b) J<L



(c) J>L, L=0



(d) J>L, L>0

Figura 4.2: Casos del algoritmo de marcado en el método Lam-Gouda

1. Sustituir *leavings* por *joinings*
2. Nodos desde las posiciones reemplazadas hasta la raíz: UPDATE (u)

- Caso 2: $J<L$

1. De los L *leavings* seleccionar los J de menor profundidad
2. Sustituir los nodos seleccionados por J *joinings*
3. Nodos desde posiciones reemplazadas hasta la raíz: UPDATE (u)
4. Nodos de *leavings* sin reemplazar: DELETE (d)
5. Un nodo no hoja es DELETE (d) *sii* todos sus hijos son DELETE (d)

- Caso 3: $J>L$ y $L=0$

1. Seleccionar el nodo hoja de menor profundidad (n)
2. Eliminar (n) del árbol
3. Construir un árbol T con todos los *joinings* y n
4. Adjuntar T a la antigua posición de n
5. Todos los nodos no-hoja de T son claves nuevas NEW (k)
6. Nodos desde posición de T hasta raíz: UPDATE (u)

- Caso 4: $J>L$ y $L>0$

1. Sustituir todos los *leavings* por *joinings*
2. Seleccionar la posición sustituida de menor profundidad (n)
2. Eliminar (n) del árbol
3. Construir un árbol T con todos los extra *joinings* y n
4. Adjuntar T a la antigua posición de n
5. Todos los nodos no-hoja de T son claves nuevas NEW (k)
6. Nodos desde posición de T hasta raíz: UPDATE (u)
7. Nodos desde posición de *leavings* hasta la raíz: UPDATE (u)

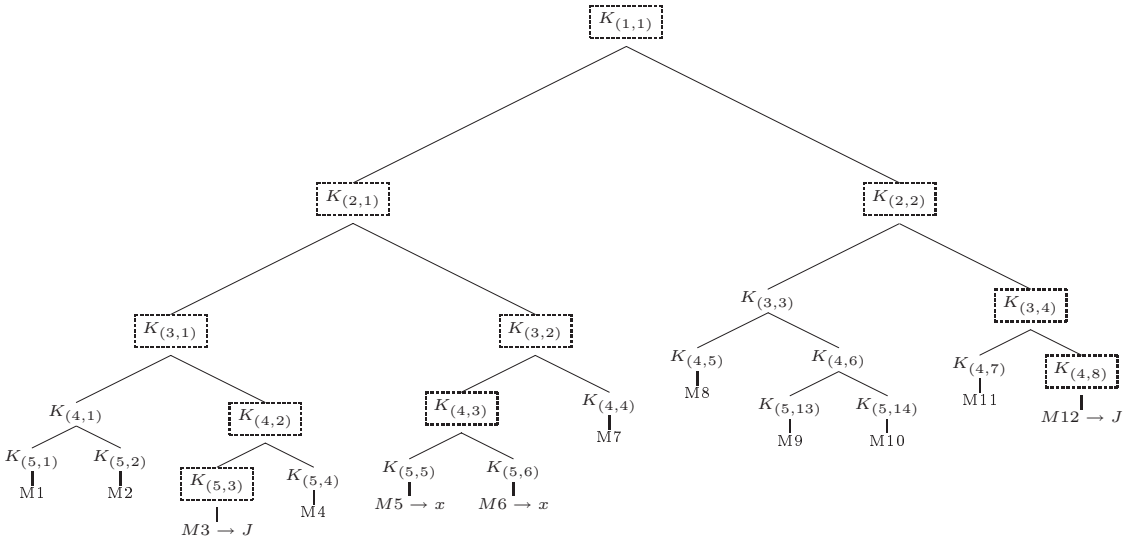
Después de aplicar el algoritmo de marcado, el KS elimina todos los nodos que contienen la marca DELETE y genera nuevas claves de cifrado para todos los nodos marcados como UPDATE o NEW. Finalmente envía a los miembros las claves correspondientes usando el método LKH de construcción de mensajes multicast.

Para entender mejor su funcionamiento, a continuación se presenta un ejemplo sencillo. Fig 4.3.

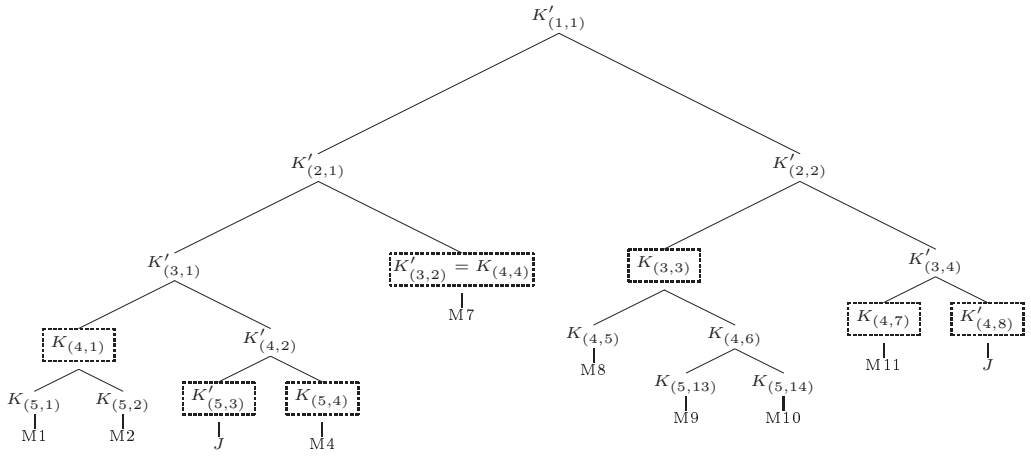
Supongamos que durante un periodo de *batch* los miembros $M3$, $M5$, $M6$ y $M13$ deciden darse de baja del grupo y además se producen 2 solicitudes de alta. En primer lugar se seleccionan las posiciones de *leavings* de menor profundidad *(4,8)* y *(5,3)* y se sustituyen por *joinings*. A continuación se marcan todos los nodos desde las posiciones cambiantes hasta la raíz como UPDATE. En la Fig 4.3a aparecen enmarcados con línea discontinua.

Los nodos *(5,5)*, *(5,6)* y *(4,3)* se eliminan ya que se corresponden con nodos de *leavings* y nodos sin hijos. El nodo *(3,2)* pasa a albergar el antiguo nodo *(4,4)*. El árbol queda como en la Fig 4.3b. Todas las claves que debían actualizarse están ahora marcadas como K'. Las

claves que servirán para enviar los mensajes de actualización se han enmarcado con línea discontinua.



(a) Árbol Lam Gouda durante el marcado de nodos



(b) Árbol Lam Gouda después del marcado de nodos

Figura 4.3: Ejemplo de algoritmo Lam-Gouda simple

Los mensajes que deben enviarse a fin de notificar a los miembros restantes los nuevos valores de claves son los siguientes:

Por el canal multicast:
$\{K'_{(1,1)} || K'_{(2,1)} || K'_{(3,1)}\}_{K_{(4,1)}}$
$\{K'_{(1,1)} || K'_{(2,1)}\}_{K'_{(3,2)}}$
$\{K'_{(1,1)} || K'_{(2,2)}\}_{K_{(3,3)}}$
Por canales unicast:
$\{K'_{(1,1)} || K'_{(2,1)} || K'_{(3,1)} || K'_{(4,2)}\}_{K'_{(5,3)}}$
$\{K'_{(1,1)} || K'_{(2,1)} || K'_{(3,1)} || K'_{(4,2)}\}_{K_{(5,4)}}$
$\{K'_{(1,1)} || K'_{(2,2)} || K'_{(3,4)}\}_{K_{(4,7)}}$

$$\{K'_{(1,1)}||\, K'_{(2,2)}||\, K'_{(3,4)}\}_{K'_{(4,8)}}$$

Obsérvese que de no haberse realizado el proceso por lotes, como se han producido 6 peticiones (4 bajas y 2 altas) se hubieran podido llegar a enviar 24 mensajes ($6 \cdot \lceil log_2 13 \rceil$).

Ventajas e inconvenientes

Es evidente que la cantidad de mensajes necesaria es mucho menor en el caso de *rekeying* en *batch* que en el proceso individual. Este hecho se podrá comprobar en la sección de evaluación (4.3.3). Sin embargo, el algoritmo de Lam-Gouda presenta un inconveniente a resaltar.

Para que los algoritmos de renegociación de claves basados en árbol alcancen su máxima eficiencia, el árbol debe mantenerse balanceado durante toda la vida del grupo. El algoritmo expuesto anteriormente se propone mantener balanceado el árbol únicamente situando los nuevos miembros en los nodos menos profundos del árbol. Este mecanismo solamente es válido cuando el número de miembros entrantes y salientes es muy similar, en caso contrario, es necesario aplicar un algoritmo de rebalanceo adicional.

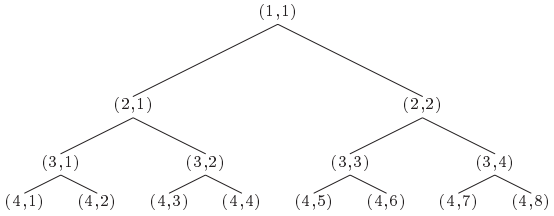
En la Fig 4.4 es fácil observar como el árbol de claves crece desbalanceado durante sucesivos *batches*.

En el ejemplo, en el primer *batch* se piden el mismo número de *joinings* que de *leavings*, y así, el árbol sí que se mantiene balanceado. En el segundo *batch* (subfigura (b)) los miembros en los nodos *(4,4)* y *(4,8)* abandonan el grupo, en el mismo periodo no se solicita ningún *joining*; así los nodos *(3,2)* y *(3,4)* pasan a ser hojas con IKs de los miembros que quedan en el grupo. En el tercer periodo, Fig 4.4c, se piden 3 *joinings* sin haberse solicitado ningún *leaving*. Esta vez el subárbol con los nuevos miembros se sitúa bajo el nodo *(3,2)*, que es el de menor profundidad. Finalmente, en Fig 4.4d dos de los miembros hijos *(2,2)* abandonan el grupo. Después de 4 *batches* sólo quedan 6 miembros y sin embargo el árbol de claves resultante es de orden 5 en vez de 4, como debería corresponderse con el caso de 6 miembros.

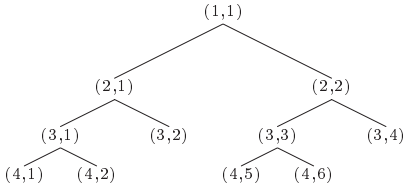
Nótese que en la Fig 4.4d, si el miembro en el nodo *(5,5)* abandonara el grupo, el número de mensajes a enviar por el KS serían 4, tantos como hermanos tiene el camino desde *(5,5)* hasta la raíz. En el caso de árbol balanceado, la profundidad máxima del árbol es 4 ($\lceil log_2 N \rceil + 1$), con lo cual el número de mensajes hubieran sido sólo 3.

4.3.2. Lam-Gouda con ajuste de balanceo

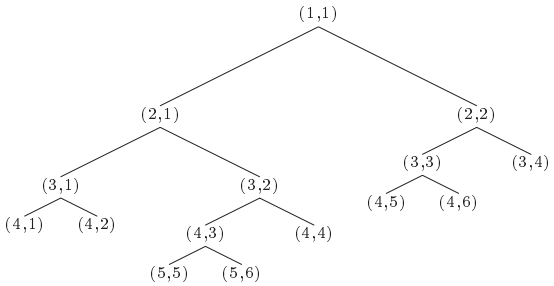
Para evitar la pérdida de eficiencia vista en la sección anterior, [Yang et al., 2001] en su mismo artículo recomiendan la introducción periódica de un algoritmo de rebalanceo. En esta línea se presentó el trabajo de [Moyer et al., 2001]. Se trata de un algoritmo completamente independiente al mecanismo de renegociación de claves utilizado. Puede realizarse en cualquier momento que se detecte que un árbol está desbalanceado. Tiene un modo de funcionamiento muy sencillo que se detalla continuación.



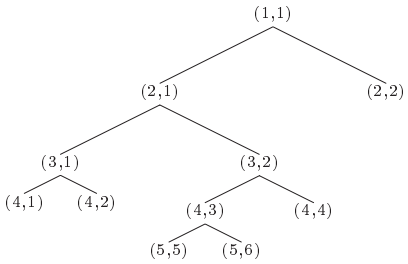
(a) Mismo número de altas que de bajas



(b) Miembros M4 y M8 abandonan el grupo



(c) 3 altas y ninguna baja



(d) 2 bajas en nodo (2,2) y ninguna alta

Figura 4.4: Ejemplo de desbalanceo en el algoritmo Lam-Gouda

Rebalanceo periódico

Básicamente el algoritmo recorre las hojas de más profundidad, las elimina y las inserta en los nodos menos profundos del árbol. Por cada cambio de posición deben actualizarse 2 *paths* hasta la raíz: los correspondientes caminos desde las posiciones antigua y nueva hasta la raíz. Esto lo hace realmente costoso para según que casos. Un posible pseudocódigo del algoritmo es el que se presenta a continuación.

```
Mientras (Profundidad_max - Profundidad_min > 1)
{
       Seleccionar_hoja_más_profunda_del_árbol
          Marcarla_como_DELETE (d)
          UPDATE_(u)_nodos_desde_su_posición_hasta_la_raíz
       Seleccionar_hoja_menos_profunda_del_árbol.
       Situar_hoja_eliminada_debajo_de_ese_nodo.
       UPDATE(u)nodos_desde_esa_posición_hasta_la_raíz
}

Enviar_mensajes_actualización_claves
```

Ventajas e inconvenientes

La ventaja principal del sistema Lam-Gouda con ajuste de balanceo es que supera el inconveniente de su predecesor. De forma periódica (o al detectar desbalanceo en el árbol) se aplica el algoritmo que deja el árbol completamente balanceado, volviendo a conseguir máxima eficiencia.

Este sistema, en cambio, tiene como inconveniente el coste computacional y de ancho de

banda adicional que supone el algoritmo. Un rebalanceo siempre supone una nueva actualización de claves que no se hubiera dado si los miembros se mantuvieran en su posición.

Además, se pierde la condición estática de los miembros. Cuando se produce rebalanceo, algunos miembros cambian de nodo y por tanto se les debe notificar su nueva posición, con las dificultades añadidas que esto comporta. Se dice, entonces, que éste es un sistema híbrido estático/dinámico ya que aunque sí que se puede escoger en qué momento los miembros podrán cambiar de posición, este cambio es inevitable cada vez que se rebalancea.

4.3.3. Evaluación

Los algoritmos de proceso en *batch* deben compararse usando los mismos parámetros de eficiencia que los algoritmos individuales. De todos modos, la forma de obtener los parámetros de eficiencia será radicalmente diferente. En los algoritmos individuales sólo podían darse tres posibles acciones en un algoritmo: petición de *joining*, petición de *leaving* o ninguna petición. Del estudio de cada uno de los casos, se deducía que el efecto de una alta era muy similar al de una baja, y además, sólo dependía de la profundidad del árbol en ese momento, con lo que, considerando el árbol balanceado, siempre se daba el mismo comportamiento, y la única variable posible era el tamaño del árbol (número de elementos en el grupo).

En el caso que ahora nos ocupa, los grados de libertad son muchos más. Al agrupar peticiones de proceso y poder variar el tamaño del periodo de *batch*, los casos a contemplar son ilimitados. La forma más conveniente de evaluar estos algoritmos es mediante simulaciones que contemplen la mayoría de casos posibles en la realidad.

Para la evaluación de todos los algoritmos de proceso por lotes expuestos en este capítulo, se han escogido tres escenarios de referencia que siguen los comportamientos descritos en 2.4.

4.3.3.1. Patrones de referencia usados

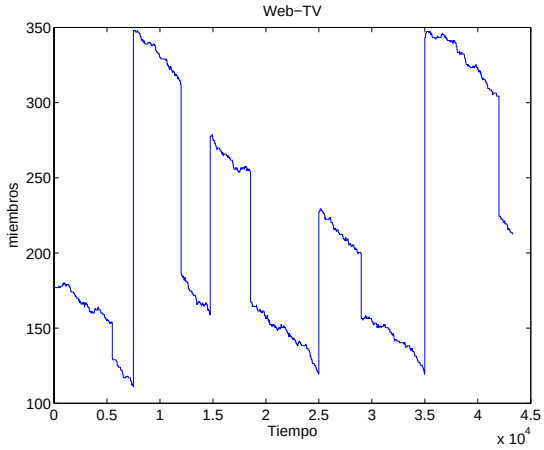
En la Fig 4.5 pueden observarse los perfiles de "número de miembros" de los tres escenarios: Web-TV, videoconferencia y juegos en red. Como ya se comentó, las diferencias principales entre estos escenarios se basan en el número de miembros implicados, el dinamismo de los mismos y el comportamiento de rafagueo. En el Anexo A puede consultarse con más detalle la forma en que se han generado los patrones mencionados.

Web-TV

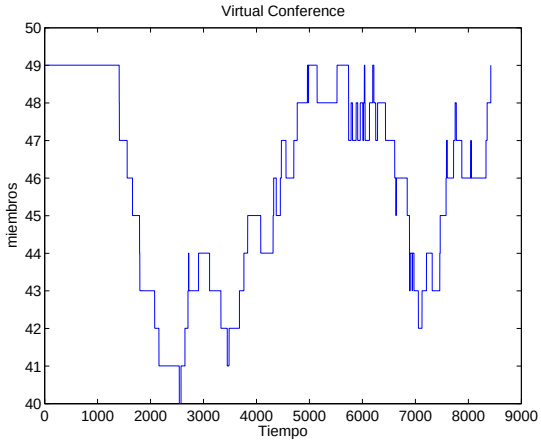
Para la simulación del escenario con difusor único se ha generado un patrón de 43200 muestras, considerando una muestra por segundo (12 horas). En él se reflejan 4 picos de programación (eventos de gran audiencia) distribuidos uniformemente durante el tiempo de evaluación, esto dota de rafagueo al patrón.

Durante la emisión de cada uno de los eventos (al igual que en el tiempo entre eventos) se considera una tasa de altas y bajas (desánimos) con tiempo entre solicitudes exponencial y valor medio igual a 200 muestras.

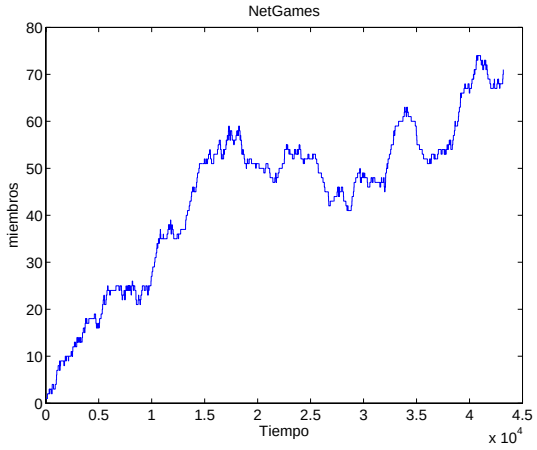
Según lo establecido para este tipo de escenarios, el número de miembros siempre es del orden de centenas.



(a) Patrón de referencia para web-tv



(b) Patrón de referencia para video conferencia



(c) Patrón de referencia de networked games

Figura 4.5: Patrones de referencia usados en las simulaciones

Videoconferencia

El patrón de videoconferencias presenta las dos fases definidas en 2.4: exposición y debate. El patrón generado consta de 8400 muestras, considerando un segundo por muestra (140 minutos). La fase de exposición se ha fijado en 25 minutos (1500 muetras) y en la de debate se ha permitido el desánimo de miembros y la reagrupación para poder estudiar como afecta la recomposición del grupo.

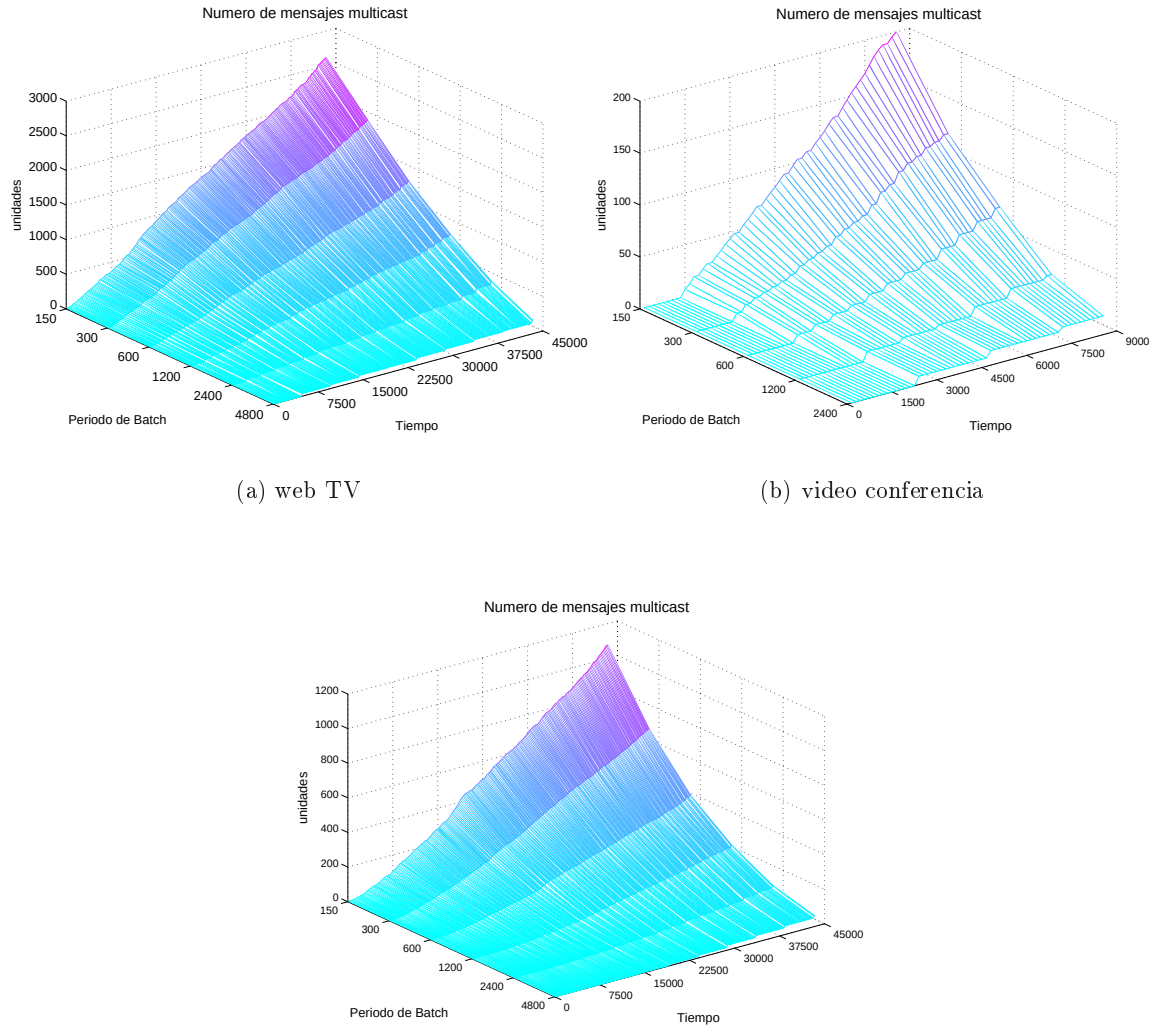
Todos los tiempos entre peticiones a nivel de sesión se han considerado con estadística exponencial. Tal como corresponde a este tipo de escenarios su dinámica no es muy elevada. El número de miembros está siempre en el orden de las decenas.

Juegos en red

El tercer escenario estudiado se ha realizado mediante la generación de un perfil con 43200 muestras, al igual que en Web-TV, pero sin ráfagas o picos de llegadas y salidas. Las peticiones de alta de miembros están correladas entre sí. El tiempo entre llegadas se calcula siguiendo una estadística gamma-pareto que modela las dependencias a largo plazo. El número de usuarios crece desde 1 hasta varias decenas. No se han considerado efectos de hora cargada.

4.3.3.2. Número de mensajes para *rekeying*

Tal como se hizo con los algoritmos de renegociación individual, el parámetro más importante a estudiar es el ancho de banda necesario para el *rekeying*. Una forma indirecta de tomar esta medida es mediante la observación del número de mensajes enviados (normalizado al tamaño de clave). Fig 4.6.



(a) web TV

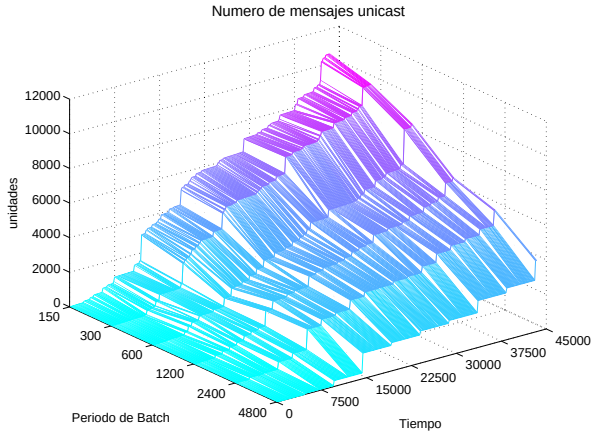
(b) video conferencia

(c) networked games

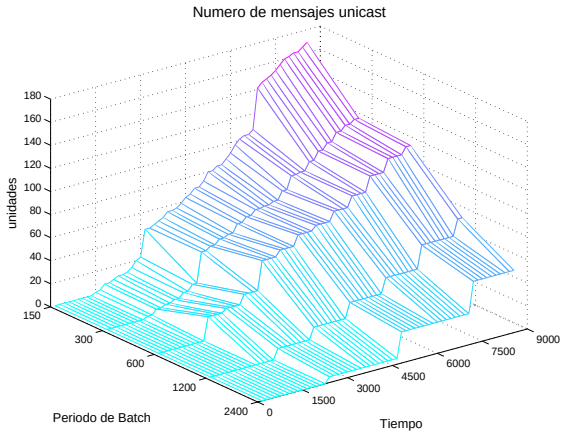
Figura 4.6: Número de mensajes multicast caso *batch-LKH*

En esta figura se observa el número de mensajes multicast acumulado debido a las renegociaciones de clave producidas durante todo el tiempo de observación. Los valores se han simulado para los tres escenarios a estudio y distintos periodos de *batch*. En la Fig 4.7 se presenta la misma gráfica pero teniendo en cuenta sólo los mensajes unicast. En las dos figuras, el eje *'x'* se corresponde con la evolución temporal y el eje *'y'* representa distintos valores de periodo de *batch* para poder analizar el comportamiento del parámetro en relación a su duración.

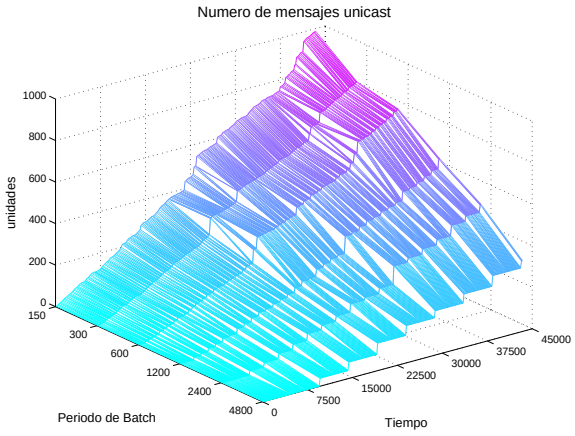
El primer comportamiento observado es la mejora de ancho de banda para tiempos de *batch* mayores en los tres escenarios. A mayor perido de *batch*, menor número de mensajes multicast totales acumulados. Este comportamiento se deriva directamente de lo esperado por el diseño del algoritmo.



(a) web TV



(b) video conferencia



(c) networked games

Figura 4.7: Número de mensajes unicast caso *batch-LKH*

Los periodos de *batch* escogidos han sido 150, 300, 600, 1200, 2400 y 4800 muestras (segundos); que se corresponden con 2.5 , 5, 10, 20, 40, 80 minutos respectivamente[1]. El primero de los periodos es un tiempo inferior a la media del tiempo entre llegadas, caso más parecido al *rekeying* individual. En el caso límite (tiempo de *batch* igual a una sola muestra) coincidiría con el *rekeying* individual.

Cuanto mayor es este tiempo, más peticiones se producen en un periodo y mayor es el aprovechamiento de mensajes de *rekeying*. El nivel de seguridad, en cambio, disminuirá. Éste es un parámetro que no puede verse reflejado en las gráficas pero considérese que para el mayor periodo de *batch*, en el caso de Web-TV, la ventana de vulnerabilidad se correspondería aproximadamente con la duración de una película. El comportamiento descrito también aplica al caso de mensajes unicast Fig 4.7.

La diferencia fundamental entre el número de mensajes multicast y unicast es su sensibilidad al rafagueo. En todos los escenarios, el incremento en función del tiempo para mensajes multicast es lento, y no depende de si el patrón de entrada presenta rafagueo o no. En cambio, en las gráficas del número de mensajes unicast se detectan perfectamente los instantes en los que se producen picos de audiencia (caso Web-TV) , o cambios bruscos en la composición del grupo (caso videoconferencia).

Este efecto es debido también al propio diseño del algoritmo. Una alta produce más mensajes unicast (y de mayor longitud) que una baja. La ausencia de picos en el menor periodo de *batch* para el caso de juegos en red corrobora la hipótesis de caso más parecido al *rekeying* individual. Los incrementos bruscos en los periodos mayores son debidos a las "falsas" ráfagas que provoca el mismo *batch*.

Finalmente se debe notar la diferencia en cuanto a proporción de mensajes unicast/multicast que se produce para escenarios con rafagueo distinto. Cambios bruscos en la composición del grupo (rafagueo alto, caso Web-TV) produce muchos más mensajes unicast que multicast ya que deben enviarse mensajes para reconstruir árboles enteros a miembros que no pueden aprovechar ninguna información del árbol anterior. Cambios suaves en la composición del grupo producen aproximadamente los mismos mensajes multicast que unicast ya que la composición del árbol se mantiene relativamente y los miembros pueden aprovechar la información que conocían de árboles anteriores.

4.3.3.3. Desbalanceo del árbol

El segundo aspecto a tener en cuenta en los agoritmos de proceso por lotes es la posibilidad de que los árboles crezcan desbalanceados.

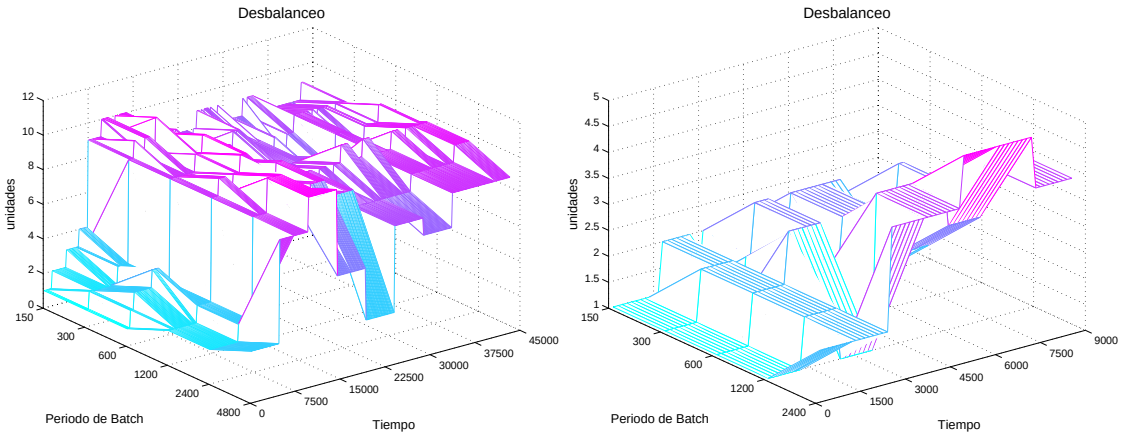
Recordemos que para que la eficiencia de los algoritmos de *rekeying* sea máxima el árbol de claves debe permanecer balanceado para todo instante de tiempo. En caso contrario, el exceso de profundidad en las ramas del árbol producirá mensajes de renegociación más largos (los caminos hasta la raíz serán más largos de lo necesario) y por tanto mayor uso de ancho de banda.

En la Fig 4.8 puede observarse la evolución de la diferencia entre la profundidad máxima y mínima en el árbol de claves, para los mismos casos estudiados en el apartado anterior.

[1] En el caso videoconferencia, debido al menor tiempo de observación, no se ha considerado el mayor periodo (4800).

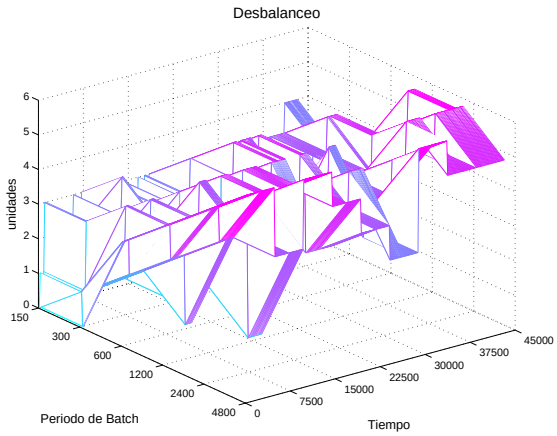
Tal como se había demostrado en 4.3.1, los árboles crecen desbalanceados en todos los escenarios estudiados. De cualquier forma, esta diferencia es tanto mayor cuanto más rafagueo presente el patrón de entrada. Las oscilaciones en el desbalanceo a lo largo de la vida del grupo demuestran que el mecanismo de rebalanceo intrínseco explicado por Lam-Gouda no es suficiente.

Además, también se observa cómo el "falso rafagueo" provocado por los *batches* provoca un crecimiento del desbalanceo, es decir, a mayor periodo de batch, mayor rafagueo provocado y por tanto mayor desbalanceo del árbol. Este comportamiento es válido para todos los escenarios excepto Web-TV en que el rafagueo propio del patrón enmascara el "falso rafagueo".



(a) web-tv

(b) video conferencia



(c) networked games

Figura 4.8: Distancia entre profundidad máxima y mínima del árbol caso *batch-LKH*

4.4. Contribución al proceso por lotes con miembros dinámicos.

Los inconvenientes que presentaba el sistema híbrido de Lam-Gouda eran dos. Por una parte es inevitable que los miembros se vuelvan dinámicos, por otra se produce un incremento de ancho de banda y coste computacional debido únicamente al algoritmo de rebalanceo, ya que si los miembros se mantuvieran en sus posiciones no sería necesario un *rekeying*.

A fin de superar estas desventajas, en [Pegueroles and Rico-Novella, 2003a] se propuso introducir el mecanismo de rebalanceo en el mismo algoritmo de proceso por lotes. Esta propuesta consigue mantener el árbol balanceado en todo instante de tiempo, superando así al Lam-Gouda simple. Por otra parte no se produce un coste adicional en cuanto a actualizaciones de clave. Como contrapartida se tiene que los miembros son siempre dinámicos y no se puede forzar el mantenimiento de miembros desbalanceados en el mismo nodo.

4.4.1. Propuesta de algoritmo: *Balanced Batch* LKH

El sistema Lam-Gouda se basa en recolocar los nuevos miembros de forma que el número de claves a actualizar sea mínimo. La propuesta de [Pegueroles and Rico-Novella, 2003a], en cambio, se focaliza en maximizar las claves a reaprovechar. Aunque a priori pueda parecer lo mismo, en árboles binarios los dos enfoques son completamente distintos.

Considere el hermano de un miembro saliente ($M3$ en la Fig 4.9), el hecho de que $M4$ abandone el grupo lo fuerza a actualizar todas las claves desde su nodo hasta la raíz (excepto su propia IK), y además, por estar en una hoja, los mensajes de actualización se le deben enviar de forma individual, es decir, el mensaje que se le envía a él no es aprovechable por nadie más.

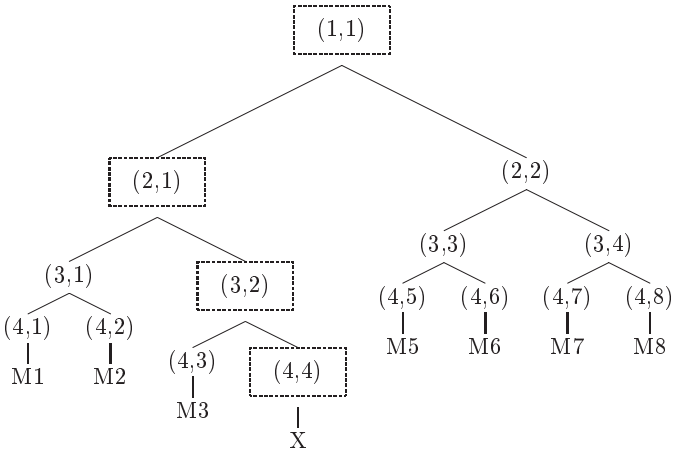


Figura 4.9: Efecto de miembro saliente en el nodo hermano

Con todo esto es fácil deducir que cualquier hermano de miembro saliente tiene los mismos efectos (en cuanto a ancho de banda, número de mensajes, etc.) que un miembro entrante. Lam-Gouda no tiene en cuenta este hecho y sustituye los miembros salientes por miembros entrantes, con lo que se deben enviar mensajes individuales tanto al entrante como a su hermano, y además los árboles pueden crecer desbalanceados.

El algoritmo propuesto tiene en cuenta este hecho, no aprovecha las posiciones hermanas de los miembros salientes y trata a esos nodos como si fueran *joinings* con lo cual se consiguen

árboles balanceados.

A continuación se describen las acciones que deben realizar tanto el KS como los miembros para llevar a cabo el algoritmo.

4.4.1.1. Gestor de Claves

El algoritmo de renegociación de claves consta de cuatro acciones que el KS debe realizar en cada *batch*: marcado de los nodos a actualizar (*mark the rekeying nodes*), poda del árbol (*prune the tree*), construcción del nuevo árbol de claves (*make new rekey tree*) y envío de los mensajes multicast (*construct and send the multicast rekeying messages*).

Marcado de nodos a actualizar

Como primer paso se deben marcar los nodos a eliminar. Durante el periodo de *batch* anterior se guardaron todas las peticiones de alta y baja a procesar en el siguiente instante de *rekeying*. Todos los nodos que se encuentren en el camino desde cualquier miembro saliente hasta la raíz se deben marcar para posteriormente ser eliminados.

Nótese que, contrariamente a lo que ocurría en Lam-Gouda, no se sustituyen los miembros salientes por miembros entrantes. El motivo de esto, como se ha comentado anteriormente, es que el parámetro importante de este algoritmo son los nodos a reaprovechar y no los nodos a actualizar. Recuérdese que los hermanos de los miembros a borrar también debían ser actualizados, por tanto, a efectos del algoritmo, tienen el mismo coste que un miembro entrante, con la particularidad de que ya disponen de su IK.

```
mark_rekeying_nodes {
    for every_registered_leaving (i,j){
        n=j

            for (m=i : -1 : 1) {
                    mark_delete_node (m,n)
                    n=ceil(n/2)
                    }
             }
     }
```

Poda del árbol

La acción de poda es muy simple, consiste en eliminar los nodos marcados y mantener las estructuras de subárbol que resultan de la poda.

Después de esta acción el KS tiene que operar con tres tipos distintos de elementos:

- Subárboles: estructuras con más de un miembro que mantienen alguna KEK en común
- Hermanos de miembros salientes: sólo pueden reaprovechar su IK, todas las demás claves hasta la ráiz se le deben notificar de nuevo y de forma individual. A efectos prácticos se comporta como un miembro entrante.
- Miembros entrantes

En la Fig 4.10 pueden verse los subárboles resultantes de aplicarle el algoritmo *balanced batch* LKH al ejemplo de altas y bajas discutido en la Fig 4.3 para el *Batch* LKH Simple.

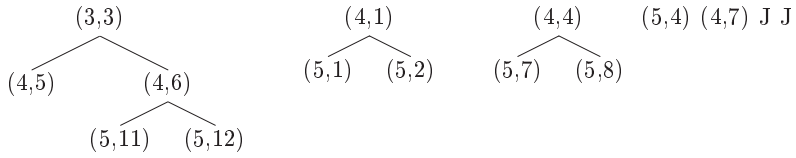


Figura 4.10: Ejemplo de subárboles en *balanced batch* LKH

Una posible implementación en pseudo-código es la que sigue:

```
prune_tree  {
    for every_marked_as_deleted_node (i,j){
        if (node(i+1,2j-1)<>null OR node(i+1,2j)<>null ){
                    marked_as_subtree_root(i,j)
                     }
              }
      }
```

Construcción del nuevo árbol

Llegados a este punto, el KS debe construir el nuevo árbol de forma balanceada usando los elementos resultantes de la poda. Para ello seguirá el siguiente algoritmo de forma recursiva.

1. Agrupar todos los subárboles de profundidad j de dos en dos. Se obtendrán como resultado subárboles de profundidad $j+1$

2. Si el número de elementos de profundidad j es impar, agruparlo con un subárbol de profundidad $j+1$ y tratarlo como un subárbol de profundidad $j+2$.

Este proceso debe empezar con árboles de profundidad mínima (elementos simples) y repetirse hasta que sólo quede un árbol resultante. A continuación se muestra en forma de pseudocódigo aunque es mucho más fácil de entender gráficamente. En la Fig 4.11 pueden verse cómo se agruparían los subárboles de la Fig 4.10.

```
update_tree {
   while number_of_subroot_trees > 1 {
       find deepest_subroot_tree
       find deepest_subroot_tree
       group_subroots_and_mark_as_new_subroot_tree
       update naming
            }
      }
```

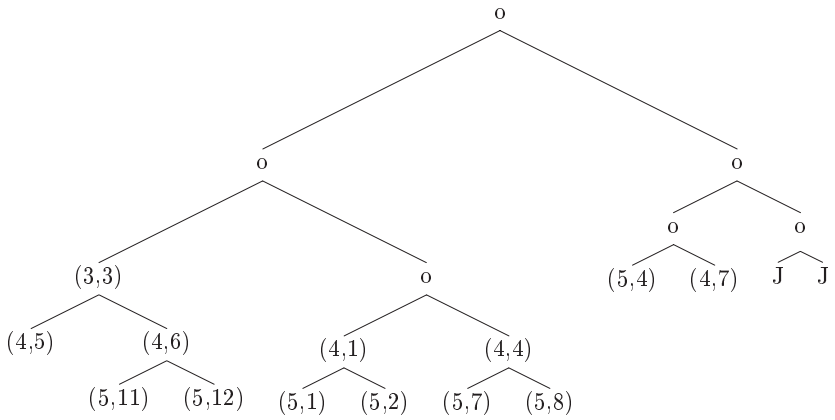


Figura 4.11: Reconstrucción del árbol en *balanced batch* LKH

Generación y envío de nuevos mensajes de *rekeying*

Finalmente, deben enviarse los mensajes de *rekeying*. Estos mensajes deberían incluir en sus campos la siguiente información: nodo destinatario, nueva posición del nodo destinatario y material de actualización de claves.

- *(i,j):* El nodo destinatario es el nodo a los descendientes del cual va dirigido el mensaje. Este campo lo utilizan los miembros receptores para decidir si un mensaje multicast va dirigido a ellos o no.
- *(p,q)*: La nueva posición del nodo destinatario es el campo que se usa para renombrar los nodos. Con esta información, los usuarios pueden renombrarse a ellos mismos y sus correspondientes claves. Las reglas usadas para el proceso de actualización de nombres se explican en la siguiente subsección.
- El campo de material de claves son las propias claves actualizadas, calculadas, por ejemplo, siguiendo las reglas LKH.

4.4.1.2. Miembros del grupo multicast

Básicamente, los miembros del grupo multicast sólo deben decidir si un determinado mensaje les concierne y, dado el caso, descifrarlo y actualizar con esa información tanto sus claves como las respectivas posiciones.

Recibir mensaje

Un miembro individual (situado en el nodo (m,n)) sólo considerará el contenido de un mensaje si las coordenadas del campo "nodo de destino" (i,j) cumplen con las siguientes condiciones.

$$m \geq i \tag{4.1}$$

$$(j \cdot 2^{m-i}) - (2^{m-i} - 1) \leq n \leq j \cdot 2^{m-i} \tag{4.2}$$

Actualizar posición y claves

Después de decidir si un determinado mensaje le incumbe o no, el nodo (m,n) y las claves se renombran a la nueva posición (m',n') usando la información del campo "nueva posición" (p,q) . La actualización de la posición sigue las siguientes expresiones.

$$m' = p + (m - i) \tag{4.3}$$

$$n' = q \cdot 2^{m-i} - (j \cdot 2^{m-i} - n) \tag{4.4}$$

Las claves reusadas (del nodo renombrado hacia las hojas) y las nuevas (del nodo renombrado hacia la raíz) también se deben renombrar de acuerdo a la posición relativa con el nuevo nombre de nodo.

```
Multicast_Key_Client{
    while session_on{
        listen_to_multicast_channel( )
        }
    listen_to_multicast_channel( ) {
           if rekeying_message {
               check_if_I_am_son_of_the_destination_node_field
               update_my_position_and_key_names
             }
         }
 }
```

4.4.2. Evaluación

Al igual que se hizo para el proceso por lotes simple, a continuación se muestran los resultados de las simulaciones para la propuesta de *batch* balanceado según los tres patrones de referencia. En primer lugar se presentan los resultados en cuanto a evolución de la profundidad del árbol y desbalanceo. A continuación se verá como afecta la mejora en balanceo al número de mensajes necesario.

Para poder comparar los resultados con el caso no balanceado, los resultados del *balanced batch* (gráficas a la derecha) se muestran en tonos rojizos mientras que los valores para el *batch* simple (izquierda) se dibujan en azul.

4.4.2.1. Evolución de la profundidad del árbol

El efecto inmediato que se espera de la ejecución del algoritmo balanceado respecto el no balanceado es el ajuste del valor de la profundidad del árbol así como la distancia entre profundidad máxima y mínima de ramas en el árbol (desbalanceo).

En la Fig 4.12 se observa la mejora en la estabilidad de la profundidad del árbol. Los resultados para el caso balanceado se muestran a la derecha. Obsérvese que el comportamiento de la profundidad del árbol para todos los *batches* y todo instante de tiempo es aproximadamente plano.

Este comportamiento se corresponde con la hipótesis realizada. Nótese que el rango en el número de miembros de los patrones de entrada de Web-TV y videoconferencia está com-

prendido entre dos potencias de 2 consecutivas (Web-TV: 128-512; videoconferencia: 32-64). Por tanto, los valores de la profundidad de dichos árboles deberían oscilar entre 8 y 9 en el caso de Web-TV y mantenerse constante con valor 6 para el caso de videoconferencia. Las oscilaciones respecto estos valores están provocadas por el desbalanceo.

El caso de juegos en red merece mención especial. Como el número inicial de miembros es 1 y su valor va creciendo, el árbol pasa por todos los posibles valores. Sin embargo, en el caso no balanceado éste no se estabiliza en un valor como en el caso balanceado, ya que las constantes altas y bajas siguen provocando desbalanceo.

Los valores de la profundidad del árbol son similares para todos los *batches*. Este comportamiento es normal si se tiene en cuenta que este parámetro sólo depende del número de miembros en cada instante de tiempo. A igual número de miembros, profundidad similar del árbol (exceptuando desbalanceos) sea cual sea el periodo de *batch* considerado.

Este comportamiento es válido para escenarios donde el rafagueo es muy fuerte, en cuyo caso las ráfagas del patrón enmascaran el incremento de profundidad debido al falso rafagueo (caso Web-TV) y donde el rango en el número de miembros es muy pequeño (caso videoconferencia) en cuyo caso, las falsas ráfagas provocadas por los *batches* no son significativas.

En el caso de juegos en red, en que las ráfagas intrínsecas al patrón son muy débiles, las falsas ráfagas de los distintos *batches* provocan un incremento de la profundidad del árbol para mayores periodos de renegociación.

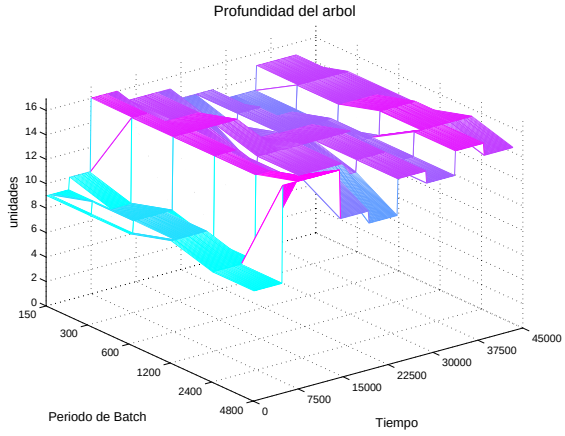
Este comportamiento se observa perfectamente en las gráficas de la izquierda (caso no balanceado) en que las oscilaciones se corresponden con las de la Fig 4.13.

En cuanto al desbalanceo (Fig 4.13), el Lam-Gouda simple ya se comentó, y sólo debe remarcarse que sigue un comportamiento muy similar a la profundidad del árbol, excepto un cierto *offset* que se corresponde con la profundidad mínima necesaria en cada instante de tiempo.

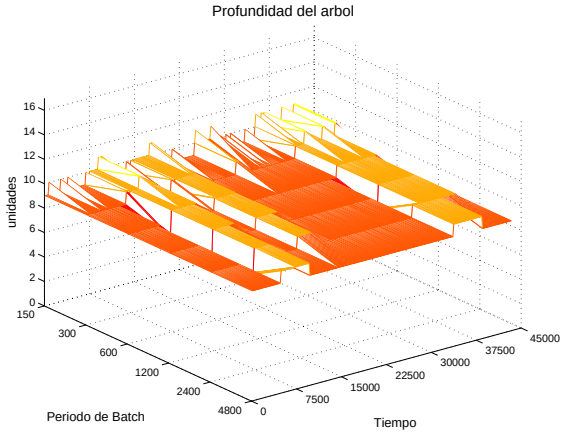
El caso balanceado (gráficas de la derecha) muestra un comportamiento que debe comentarse. Por diseño del algoritmo se podría esperar que el desbalanceo máximo permitido fuera 1, en los resultados obtenidos, sin embargo, se observa que el valor se extiende en el rango [0-2].

Este desajuste en el desbalanceo es debido a un efecto no tenido en cuenta a la hora de diseñar el algoritmo. En el *balanced batch* cuando se agrupan los subárboles aprovechables, puede haber un número impar de árboles de mismo nivel. En esos casos, si se deja sin agrupar algún árbol no completo (desbalanceo 1) , se incrementa en 1 nivel el desbalanceo global que presentará el árbol final.

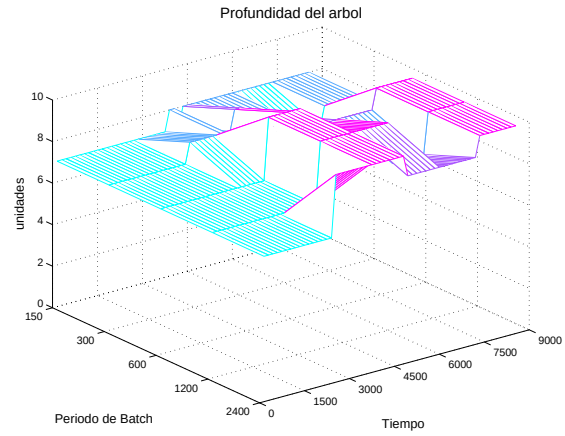
El efecto podría eliminarse modificando el algoritmo de forma que se reagrupen los subárboles con prioridades según su completitud o no.



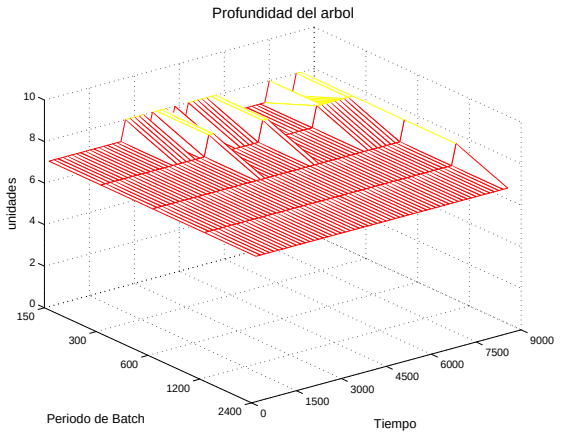
(a) web-tv caso no balanceado



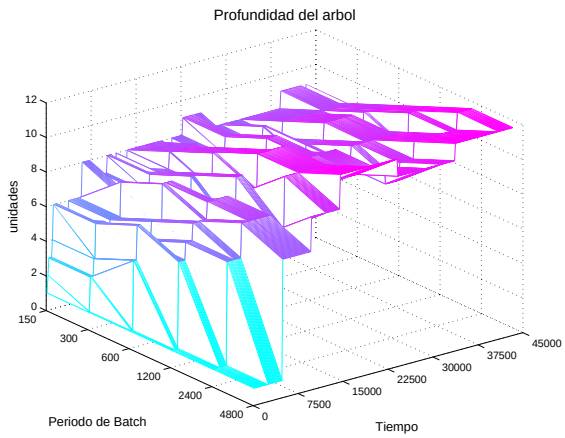
(b) web-tv caso balanceado



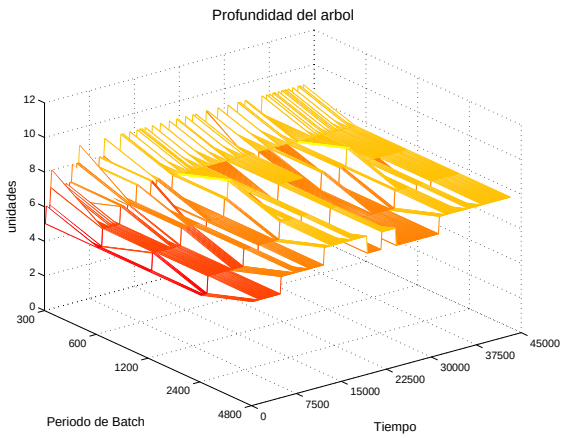
(c) video conferencia caso no balanceado



(d) video conferencia caso balanceado

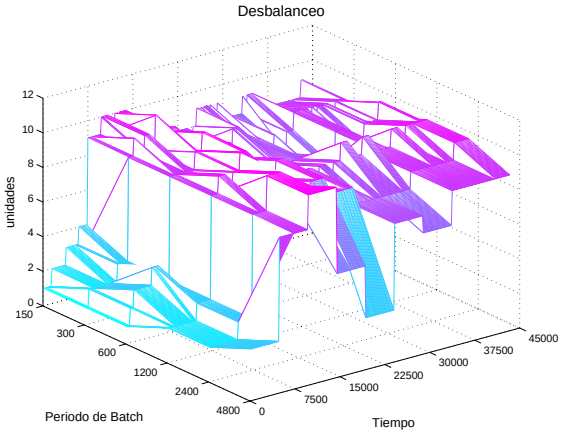


(e) networked games caso no balanceado

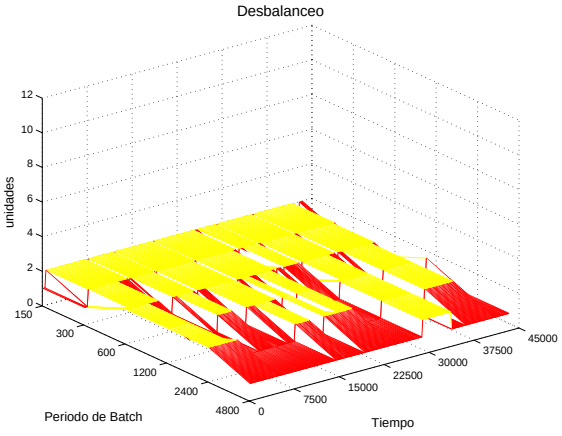


(f) networked games caso balanceado

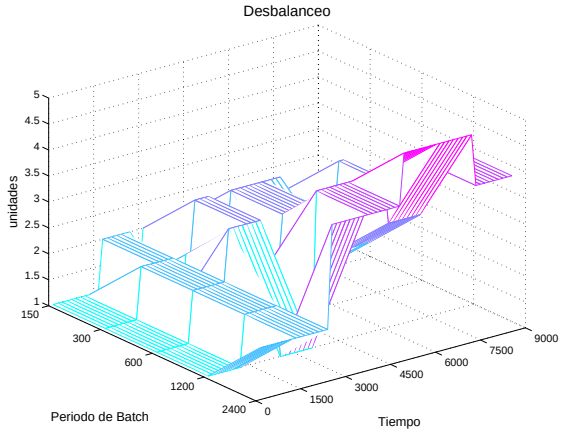
Figura 4.12: Profundidad del árbol caso *Balanced batch-LKH*



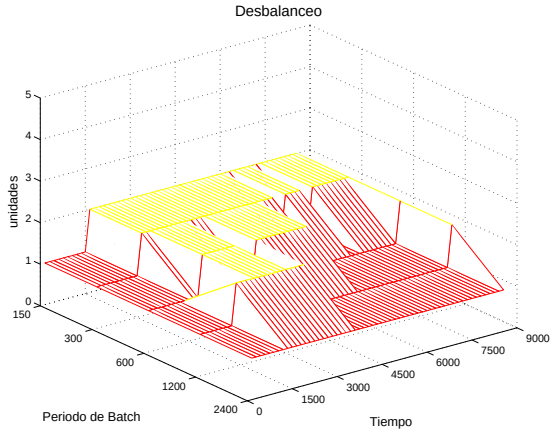
(a) web-tv



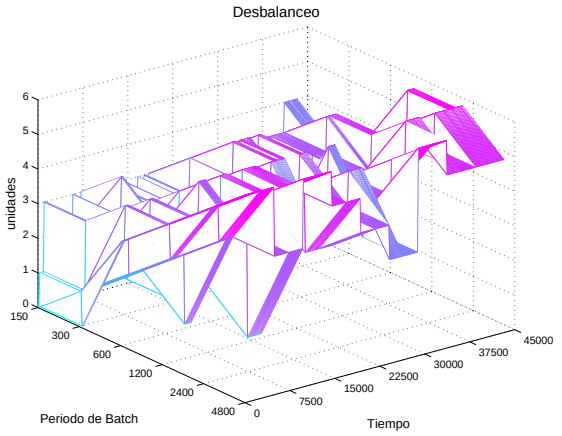
(b) web-tv caso balanceado



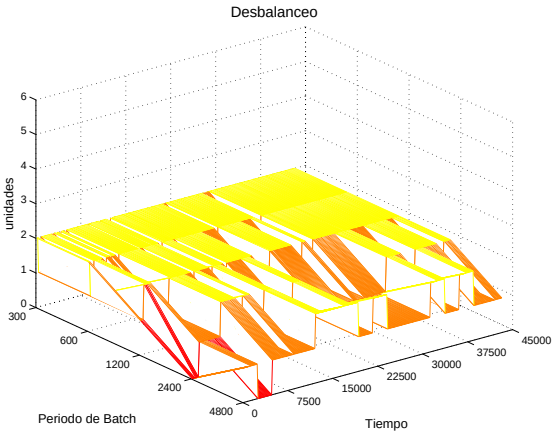
(c) video conferencia



(d) video conferencia caso balanceado



(e) networked games



(f) networked games caso balanceado

Figura 4.13: Distancia entre profundidad máxima y mínima del árbol caso *Balanced batch-LKH*

4.4.2.2. Número de mensajes para *rekeying*

Por lo explicado en las secciones 4.3.1 y 4.4.1, el decremento en desbalanceo y profundidad del árbol se deriva directamente del diseño del algoritmo. Esta mejora en comportamiento, sin embargo, de poco serviría si no se tradujese en una mejora en cuanto a ancho de banda.

En las Fig 4.14 y 4.15 se muestran las comparaciones de número de mensajes multicast y unicast para los casos desbalanceados y balanceados en los tres escenarios de referencia.

El comportamiento del número de mensajes multicast es muy similar para el caso balanceado y desbalanceado en los tres escenarios estudiados. El tanto por ciento en decremento en número de mensajes producido por el aumento del periodo de *batch* es muy similar, al igual que el efecto del rafagueo.

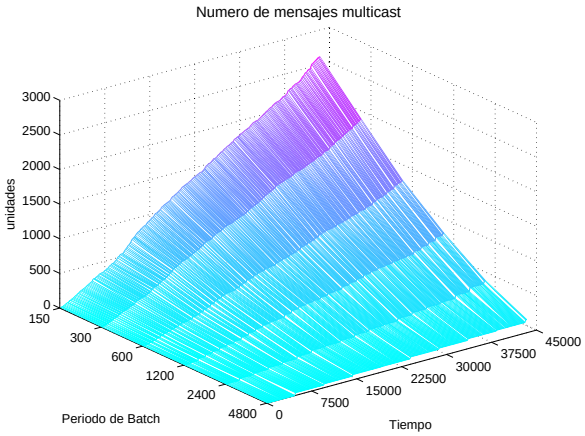
La única diferencia apreciable entre el caso balanceado y desbalanceado es una ligera disminución del número global de mensajes multicast enviados. Esta diferencia es debida al ajuste de la profundidad que provoca mensajes menores en longitud. De cualquier forma, aunque siempre se obtienen valores menores para el caso balanceado, la mejora no es muy significativa.

El número de mensajes unicast, en cambio, se reduce enormemente cuando se aplica el algoritmo balanceado en vez del Lam-Gouda simple. Fig 4.15. Este comportamiento se explica por dos motivos diferentes.

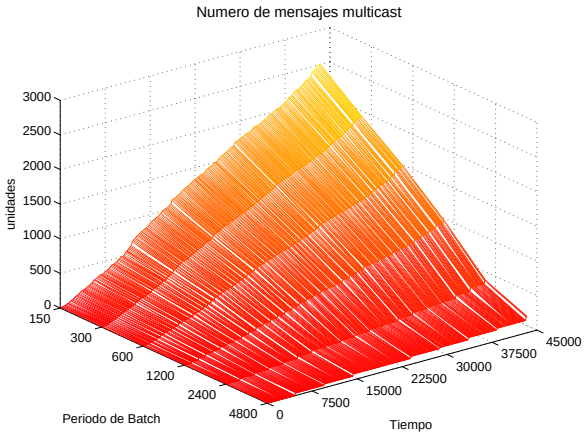
En primer lugar, por diseño del algoritmo, los mensajes unicast contienen claves que empiezan en la raíz, y nunca deben aprovechar claves anteriores de árboles. La longitud de estos mensajes, por tanto, siempre será menor, tanto si se corresponden con miembros nuevos como de hermanos de miembros salientes.

En segundo lugar, para los nuevos miembros, al incluirse los mensajes unicast en la fase 1 del GDOI, estos no se contabilizan como mensajes de renegociación y por tanto, el decremento en cuanto a ancho de banda es mucho mayor.

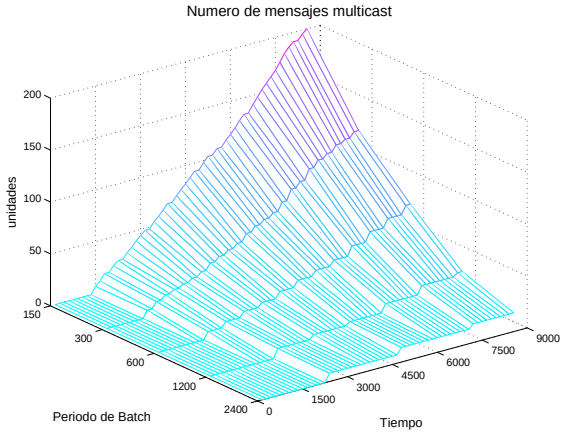
Finalmente, debe notarse que el número de mensajes unicast acumulados no presenta una fuerte dependencia con el tamaño de periodo de *batch*. El decremento es mucho mayor para los mensajes multicast que para los mensajes unicast. Esto es debido a que los mensajes multicast afectan a los miembros del grupo no implicados en un cambio de composición y los unicast a los miembros sí implicados (altas, bajas o hermanos de altas o bajas). En este último caso, esperar más o menos tiempo a procesar no afecta al número de miembros implicados en un cambio de composición, ya que si no se procesan en un *batch* se procesan en otro, y consecuentemente, el número de mensajes acumulado tenderá a ser similar.



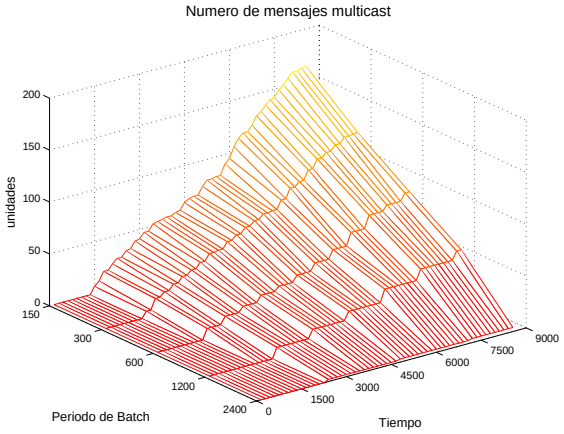
(a) web TV caso no balanceado



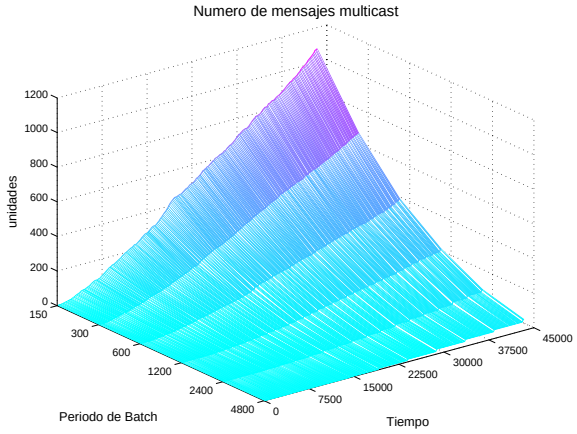
(b) web TV caso balanceado



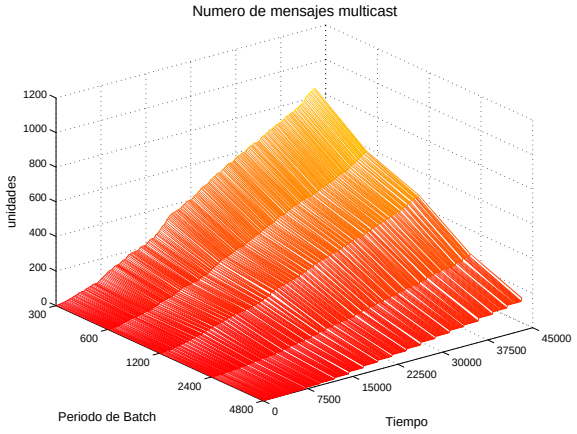
(c) video conferencia caso no balanceado



(d) video conferencia caso balanceado

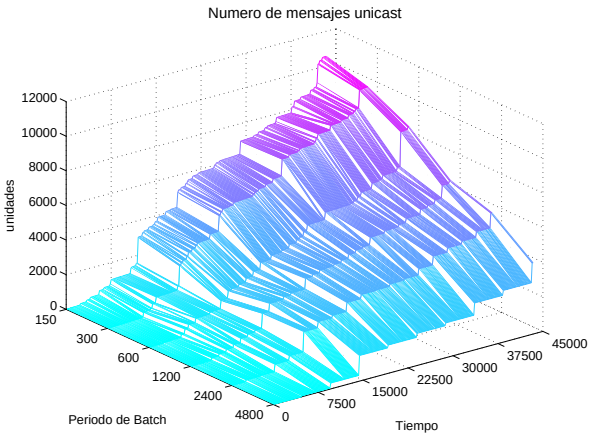


(e) networked games caso no balanceado

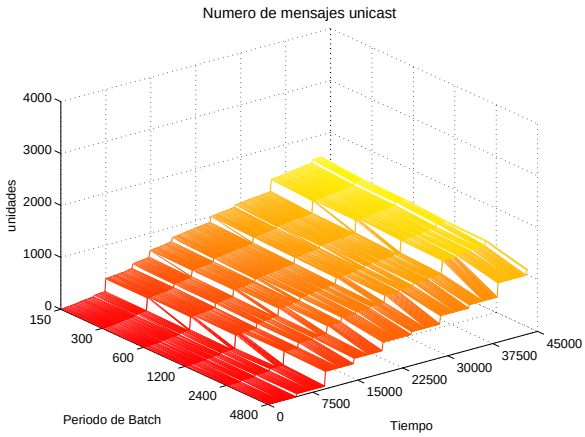


(f) networked games caso balanceado

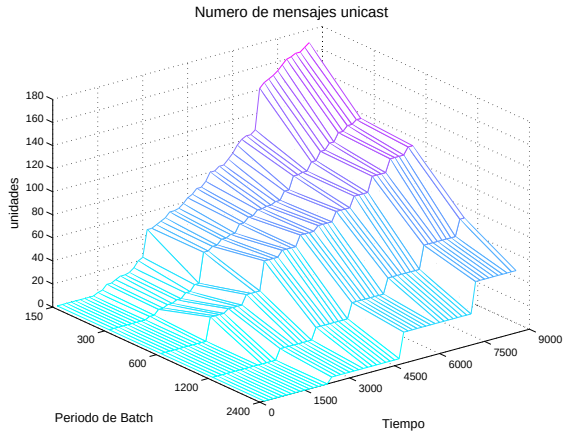
Figura 4.14: Número de mensajes multicast caso *Balanced batch-LKH*



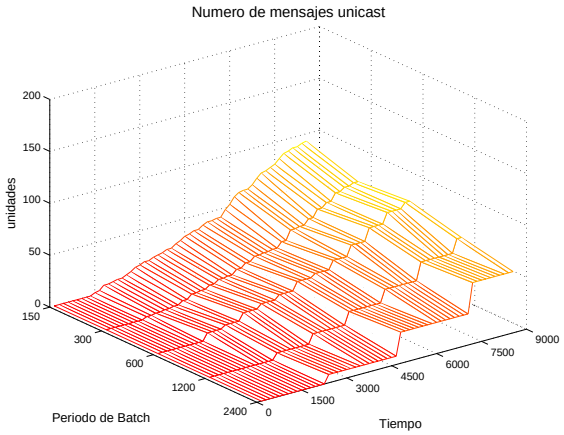
(a) web-tv caso no balanceado



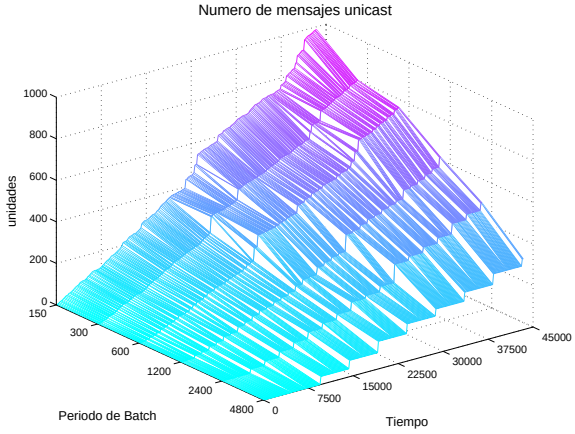
(b) web-tv caso balanceado



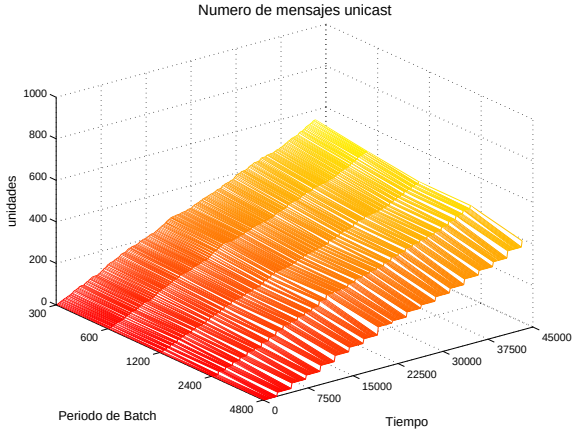
(c) video conferencia caso no balanceado



(d) video conferencia caso balanceado



(e) networked games caso no balanceado



(f) juegos en red caso balanceado

Figura 4.15: Número de mensajes unicast caso *Balanced batch-LKH*

4.5. Contribución a la integración de la gestión por lotes en el algoritmo de gestión de claves

En las secciones anteriores se ha observado como, penalizando el nivel de seguridad, se puede conseguir una reducción considerable en ancho de banda. En muchos escenarios, la reducción de seguridad no es crítica, y los algoritmos de proceso por lotes son una herramienta extremadamente útil para reducir el uso de recursos.

Todo lo discutido respecto los algoritmos de proceso por lotes ha sido considerando que la construcción de mensajes de *rekeying* se realiza usando las reglas del LKH. Las claves se han considerado aleatorias e independientes entre sí. Sin embargo, el proceso en *batch* es ortogonal a los algoritmos expuestos en el Capítulo 3 y nada impide combinar las técnicas de proceso por lotes con los otros algoritmos de renegociación individual de claves.

De todas formas, dependiendo de cómo se haya diseñado la generación de claves y sus dependencias, aplicar técnicas en *batch* puede resultar más o menos directo. OFT y OFC consiguen reducir el ancho de banda en actualizaciones individuales introduciendo dependencias entre los nodos, pero estas mismas dependencias dificultan enormemente la generación de mensajes de *rekeying* cuando debe procesarse más de una alta o baja. De hecho, en la literatura no se ha encontrado ninguna propuesta de proceso por lotes para estos algoritmos.

Sin embargo, la propuesta de algoritmo realizada en 3.8 construye los mensajes de actualización de tal forma que el proceso por lotes se deriva de forma directa. A continuación se presenta la combinación de la propuesta sm-LKH/OKS para proceso por lotes tanto mediante miembros estáticos como dinámicos [Pegueroles et al., 2003b].

4.5.1. Propuesta de algoritmo: sm-LKH/OKS con *batch* estático

Recordemos que en el *batch rekeying* de Lam-Gouda los miembros permanecían en el mismo *path* hasta la raíz durante toda la vida del grupo. Los únicos cambios que se permitían eran cambios de nivel cuando los miembros hermanos abandonaban el grupo. En cualquier caso, el conjunto de claves que un miembro debía mantener era siempre el mismo.

En un escenario como el descrito, sm-LKH/OKS puede aplicarse de forma directa para proceso en lotes. La información que requieren los miembros del grupo para actualizar las claves es $(r \oplus r')$. Ésta se hace llegar a todos los miembros que permanecen en el grupo después de un periodo de *batch* de la misma forma que se hacía en el caso de proceso individual. Únicamente debe considerarse que ahora, el número de *randoms* que contendrá el productorio dependerá del número de *leavings* que se hayan producido y de su posición en el árbol. Los *joinings* no afectarán a la construcción del mensaje ya que a ellos se les deberá notificar el conjunto entero de claves, y usualmente, se realizará a través de un canal unicast.

Para entender mejor su modo de funcionamiento, a continuación se presenta un ejemplo sencillo de *rekeying*. Considere el árbol con 11 miembros de la Fig 4.16. Durante un periodo de *rekeying* los miembros $M3$ y $M11$ abandonan el grupo y no se produce ningún *joining*. Todas las claves que se han visto comprometidas deben ser actualizadas.

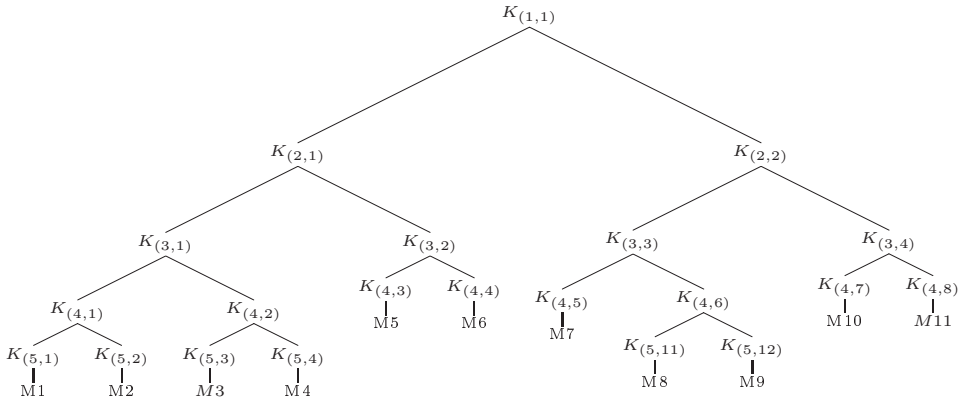


Figura 4.16: Ejemplo de algoritmo sm-LKH/OKS con Lam-Gouda simple

Recodemos que mediante el sistema Lam-Gouda los mensajes que debían enviarse eran los siguientes:

$$\{K'_{(1,1)}||\, K'_{(2,1)}||\, K'_{(3,1)}\}_{K_{(4,1)}}$$
$$\{K'_{(1,1)}||\, K'_{(2,1)}||\, K'_{(3,1)}||\, K'_{(4,2)}\}_{K_{(5,4)}}$$
$$\{K'_{(1,1)}||\, K'_{(2,1)}\}_{K_{(3,2)}}$$
$$\{K'_{(1,1)}||\, K'_{(2,2)}\}_{K_{(3,3)}}$$
$$\{K'_{(1,1)}||\, K'_{(2,2)}\}_{K_{(4,7)}}$$

sm-LKH/OKS, en cambio sólo requerirá el siguiente mensaje:

$$r \oplus r' + K_{(4,1)} \cdot K_{(5,4)} \cdot K_{(3,2)} \cdot K_{(3,3)} \cdot K_{(4,7)}$$

4.6. Conclusiones

El proceso por lotes de las peticiones de *joining* y *leaving* independiza el ancho de banda de *rekeying* de la dinámica del grupo. Como contrapartida, esta reducción afecta a la seguridad global del sistema ya que no se podrá proporcionar confidencialidad estricta hacia adelante y hacia atrás. De cualquier forma, la mayoría de escenarios reales pueden soportar esta disminución de seguridad.

El primero de los algoritmos de proceso por lotes presentado en la literatura (sistema Lam-Gouda) conseguía su objetivo en cuanto a reducción de ancho de banda, pero no tenía en cuenta el balanceo de los árboles lógicos de claves. Para que los algoritmos de renegociación de claves basados en árboles lógicos de claves consigan su máxima eficiencia se debe mantener el árbol balancedo en todo instante de tiempo.

En este capítulo se ha demostrado mediante simulación que cuando el *batch* LKH se expone a patrones de tráfico reales da lugar a árboles desbalanceados. Se ha comprobado como el desbalanceo provoca una disminución en eficiencia global.

Como contribución de esta tesis se ha presentado un algoritmo de proceso por lotes que da lugar a árboles balanceados para todo instante de tiempo. El algoritmo se basa en reconstruir los árboles de claves teniendo en cuenta sólo las claves que se pueden reaprovechar y no los miembros que se dan de alta o de baja en el grupo. A diferencia que en el *batch* LKH, el *balanced*

batch LKH permite que los miembros cambien de posición de un *batch* a otro. Este mecanismo posibilita el balanceo pero, a su vez, incrementa ligeramente el coste computacional, tanto del servidor de claves como de los miembros.

Finalmente, se ha propuesto la integración de los algoritmos propuestos en el Capítulo 3 para su proceso por lotes.

Capítulo 5

Contribución a la implementación y desarrollo del *Testbed*

5.1. Introducción

Los algoritmos de gestión de claves descritos y propuestos en los capítulos anteriores no disponían de ninguna implementación real que permitiese la validación de su funcionamiento en un entorno de pruebas. Los estudios y comparativas realizados hasta la fecha de realización de este trabajo se basaban en resultados analíticos.

Como se comentó en el Capítulo 4, la evaluación de los algoritmos de proceso por lotes presenta un elenco ilimitado de casos posibles, sólo analizable mediante simulación. Para poder realizar el estudio del Capítulo 4 se requirió la implementación de un simulador de algoritmos de gestión de claves. La herramienta diseñada y descrita en este capítulo posee las características de simulador.

A su vez, y para aprovechar el código programado, la misma herramienta consta de la opción de *testbed* real, con su parte servidor y cliente que generan y se intercambian mensajes de gestión de claves de forma real. El desarrollo del entorno real permitió descubrir y refinar aspectos no tenidos en cuenta en la definición del algotimo GDOI, del cual, hasta la fecha, tampoco existía ninguna versión que soportase algoritmos de gestión basados en árboles.

En este capítulo en primer lugar se describe la estructura del *testbed* real y se presenta la forma en que debe integrarse en aplicaciones multicast.

Finalmente, y debido a que el estándar de comunicaciones seguras de grupo del IETF no soporta las propuestas realizadas en esta tesis, se propone una modificación del GDOI para que sí se incluyan en él.

5.2. Desarrollo Java

Como contribución a la implementación de los algoritmos de gestión de claves se desarrolló un *package* Java que implementa los algoritmos de *rekeying* basados en árboles lógicos. En concreto se implementó el LKH y las dos propuestas de mejora explicadas en los Capítulos 3 y 4. Estos *package* permiten la integración sencilla de las funcionalidades de gestión de claves en cualquier aplicación Java multicast.

Como complemento a estas librerías se desarrollaron dos aplicaciones que hacen uso de este nuevo package. Una aplicación simula el funcionamiento real del *Key Server* (KS), pudiendo generar árboles lógicos que permiten gestionar claves con los algoritmos de *rekeying* del *package*, y usando una interfaz gráfica que facilita el entendimiento de su funcionamiento. La otra, corresponde a la simulación de un miembro del grupo, y permite realizar peticiones de *joining* o *leaving* a un KS que escuche en la dirección y puerto especificados. La descripción del trabajo y sus resultados se presentaron en [Pegueroles and Rico-Novella, 2003c].

5.2.1. Contenidos del package

En este apartado se presenta de forma global cómo se ha realizado y estructurado el *package* generado y los aspectos más destacables en algunas de las decisiones tomadas en su creación.

El propósito del *package* es ofrecer una implementación práctica de todo el mecanismo de renegociación de claves descrito en los capítulos anteriores. Esta implementación debe ser sencilla y a su vez reutilizable para su posterior adaptación a las nuevas soluciones y propuestas que irán apareciendo. Como se ha comentado anteriormente, antes de la realización del presente trabajo no se conocía ninguna implementación en Java de los algoritmos mencionados. Además, el *package* desarrollado ofrece una solución para todas aquellas aplicaciones orientadas a grupos multicast realizadas en Java y que quieran dotar de seguridad a sus comunicaciones. Lo único que requerirá la nueva aplicación es crear una instancia de la clase que implementa el gestor de claves, y añadir los mecanismos necesarios para cambiar la clave de cifrado de información ante avisos producidos por cambios en la naturaleza del grupo.

5.2.1.1. Estructuración y aspectos a destacar

El *package* consta de dieciocho clases, que se han dividido en ocho *subpackages* distintos. Esta división se ha realizado por la funcionalidad asociada a cada una de ellas. Así tenemos:

secureMulticast.binaryTree Todas las clases necesarias para la construcción de una jerarquía LKH basada en árboles lógicos. Cronológicamente, fue el primer subpackage desarrollado, y consta de tres clases. Éstas, representan de forma abstracta lo que sería un nodo del árbol, el propio árbol binario uniendo nodos, y una clase que se encargaría de guardar en los nodos hoja, la información de los miembros del grupo a los que están asociados.

secureMulticast.client Contiene tan sólo una única clase. Es la clase necesaria para que una aplicación cliente sea capaz de tratar el tráfico multicast recibido, analizarlo y guardar todas las claves necesarias para el posterior descifrado de información.

secureMulticast.keyDistribution.algorithm Es el subpackage que contiene las clases que implementan el funcionamiento de los algoritmos de rekeying incluidos en el trabajo. Consta de cuatro clases. Una clase abstracta que contiene los métodos que cualquier nuevo algoritmo debería implementar para adecuarse al funcionamiento del servidor de claves, y tres clases más que son las implementaciones realizadas del algoritmo LKH,

la primera modificación propuesta y conocida como el Batch LKH, y por último, la segunda propuesta, el Balanced Batch LKH.

secureMulticast.keyDistribution.cipher Contiene una única clase, que es la encargada de realizar la totalidad de las operaciones criptográficas basadas en algoritmos de clave simétrica que tienen lugar en el package. Tan sólo queda excluida la implementación del algoritmo de Diffie-Hellman, que es de clave pública, y que se realiza en el momento de la admisión de un nuevo miembro dentro de la implementación del servidor.

secureMulticast.keyDistribution.net En este subpackage se concentran todas las operaciones relacionadas con la construcción y transmisión de paquetes, tanto en modo unicast como multicast. Por lo tanto tenemos dos modelos, el modelo TCP, que usaremos para el modo unicast, y el modo UDP, para el multicast, que se traducen en dos clases distintas.

secureMulticast.keyDistribution.symKeyGen Este subpackage contiene dos clases. Una, es la representación abstracta del modelo de clave KEK utilizado por el servidor de claves en esta implementación. La segunda, es la clase que se encargará de generarlas cuando sea instanciada. Soporta tres algoritmos de creación de claves simétricas, el DES, el triple DES o DESede, y el Blowfish.

secureMulticast.server Contiene dos clases. Ambas son necesarias para la creación de un Group Key Server. En su creación se deben especificar datos como el algoritmo de rekeying requerido, el periodo de batch si éste existe, el algoritmo de generación de claves, etc.

secureMulticast.util Este es, quizás, el subpackage más especial de todos. Contiene tres clases, que nos van a servir para realizar la comunicación entre clases. Es decir, mediante su utilización nos permitirán avisar desde una clase a otra, que un evento determinado ha ocurrido. Por ejemplo, si nosotros instanciamos un servidor de claves en nuestra aplicación, seremos capaces de detectar y recibir la nueva clave de cifrado de información, SEK, cada vez que esta cambie si utilizamos correctamente las clases de este subpackage.

Además de la estructuración del package, se ha creído conveniente destacar algunas de las decisiones tomadas en su realización, pues son decisiones en algunos casos críticas y que permiten el correcto funcionamiento del software.

Lo primero que hay que destacar, es que tanto el servidor como el cliente y a su vez los algoritmos, son threads (hilos de ejecución) independientes. Con esto, junto con las clases del último subpackage comentado, conseguimos no bloquear nunca los procesos internos de cada una de las clases al comunicarse entre ellas.

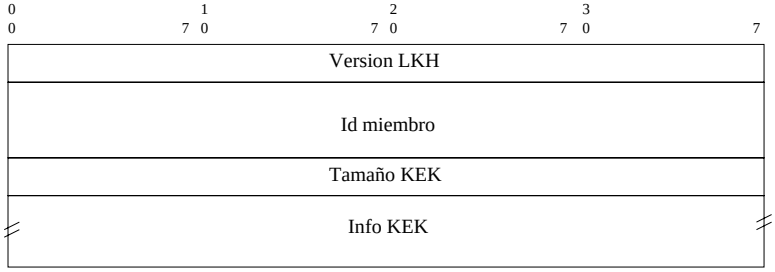
Otro de los aspectos a destacar, es que todos los métodos de las clases que implementan a los algoritmos de rekeying, son métodos sincronizados, con lo que garantizamos la exclusión mutua en ellos. Con esto se consigue evitar perder alguna petición por la sobreescritura de alguno de los atributos que en ellos se modifica.

También es importante destacar el proceso de joining por parte de un miembro. Para este propósito se ha escogido el siguiente método. En primer lugar, el miembro debe hacer una petición a la dirección y puerto donde el servidor esté escuchando. En el momento en que la recibe, el servidor crea un nuevo nodo con el identificador y la KEK asociada al miembro.

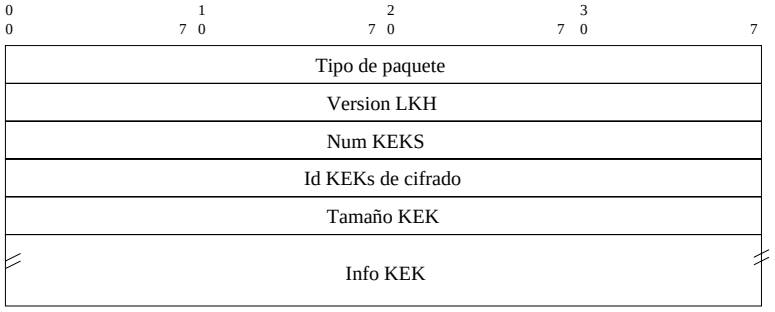
Llegados a este punto, se inicia el algoritmo de Diffie-Hellman para conseguir privacidad en el canal de comunicaciones. La particularidad de este algoritmo basado en clave pública, es que la clave no se va a transmitir por el canal y, por lo tanto, no se van a comprometer ni el identificador ni la KEK del miembro, ambas necesarias para el joining correcto del miembro, de ahí su elección.

Una vez disponemos de esta clave, se puede proceder a enviar los datos necesarios ya comentados. El resto, se enviarán una vez empiece el proceso de rekeying para todos los miembros afectados por este joining.

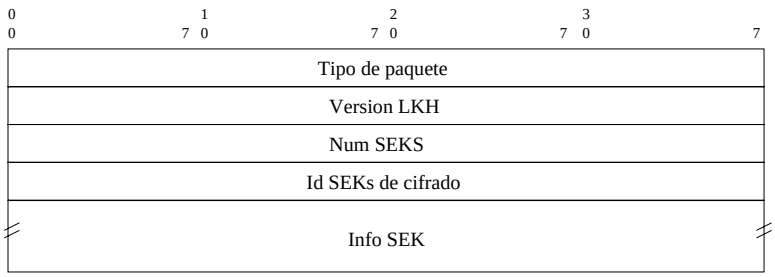
El último de los aspectos a destacar, es cómo se construyen los paquetes a transmitir tanto en el proceso de rekeying como en el proceso de joining. La estructura utilizada varía según el tipo de paquete; así tenemos tres estructuras distintas:



(a) Paquete de datos en el joining (TCP)



(b) Paquete de KEKs en el rekeying (UDP)



(c) Paquete con nueva SEK en el rekeying (UDP)

Figura 5.1: Estructuras de paquetes en el *testbed*

Siguiendo el formato de los arrays de claves LKH en GDOI, el campo versión LKH se refiere al algoritmo de rekeying utilizado, el ID del miembro es el identificador único asociado a éste, el tamaño de la KEK es el tamaño en bytes de ésta ó el tipo de paquete se refiere a la finalidad de éste, es decir, si es un paquete de rekeying unicast que contiene KEKs, si se trata de un paquete de rekeying multicast o si se trata de un paquete que contiene la nueva SEK a utilizar por el grupo.

5.2.2. Integración en aplicaciónes multicast

Para la utilización de este package, tan sólo vamos a necesitar utilizar tres clases: el LKHclient, el LKHserver, además del LKHListener.

Parte del servidor

Éste es el código necesario para poder tener un servidor de claves funcionando en nuestra aplicación:

```
import secureMulticast.server.*;
public class
{
    LKHserver server = new LKHserver(int LKHalgorithm, int sleepTime, int
                                     keyAlgorithm, String groupAddress, int
                                     UDPport, int TCPport)
    SecretKey SEK = server.start();
 }
```

La clase server es un *thread* que escucha posibles nuevas peticiones de *joining* o *leaving*. Cuando se llama al método start, se arranca el servidor; además, esta llamada devuelve la primera clave necesaria para poder empezar a cifrar la información a transmitir por la aplicación.

Los atributos necesarios en la instanciación son los siguiente:

LKHalgorithm Un entero que representa el algoritmo de rekeying escogido; 0 LKH, 1 Batch LKH, 2 Balanced Batch LKH. Todos ellos son constantes definidas en la clase.

sleepTime Entero con el valor en segundos del intervalo entre rekeyings en el caso de estar utilizando un algoritmo con periodo de Batch.

keyAlgorithm Es un entero que representa el algoritmo de generación de claves simétricas; 0 DES, 1 Blowfish, 2 DESede. Todas ellas son constantes definidas en la clase.

groupAddress El String con la dirección IP multicast que corresponde al grupo y que se utilizará para enviar el tráfico de rekeying.

UDPport Entero con el valor del puerto destino de los paquetes de rekeying. Por tanto, el puerto donde los clientes deben estar escuchando a la espera de recepción de mensajes multicast de rekeying.

TCPport Es un entero con el valor del puerto que utilizará el servidor para escuchar nuevas peticiones tanto de joining como de leaving.

Además, se deberá implementar el interfaz LKHListener, pues es la clase encargada de recepcionar avisos ante cambios en la naturaleza del grupo y generación de nuevas claves. De esta forma, conseguiremos actualizar la clave SEK cada vez que ésta cambie. Aquí tenemos un ejemplo de cómo debería ser el código teniendo en cuenta esto último:

```
import secureMulticast.server.*;
import secureMulticast.util.*;
public class implements LKHListener

    {
        LKHserver server = new LKHserver(int LKHalgorithm,
                                         int sleepTime, int keyAlgorithm,
                                         String groupAddress, int UDPport,
                                         int TCPport)
        server. addLKHListener(this);
        SecretKey SEK = server.start();
        public void LKHEventPerformed(LKHEvent e)
            {
                if(e.getEvent() == LKHEvent.NEW_SEK)
                e.getNewSEK();
            }
     }
```

Parte del cliente

La parte del cliente es todavía mucho más sencilla de poner en funcionamiento. Esto sería lo que deberíamos hacer:

```
import secureMulticast.client.*;
    public class
   {
        LKHclient client = new LKHclient(String KeyServer, int TCPport);
        int ID = client.joining();
        client.leaving();
    }
```

La clase LKHclient, igual que el servidor, es un thread, pero en esta ocasión el método start se llama internamente cuando el método joining ha funcionado correctamente. Con esto se consigue evitar estar escuchando ininterrumpidamente en el puerto UDP a la espera de tráfico multicast si el proceso de joining al grupo ha sido infructuoso.

Estos son los atributos necesarios para la instanciación:

keyServer Es el String con la dirección IP del Group Key Server.

TCPport Es un entero con el valor del puerto TCP donde el Group Key Server está escuchando las posibles nuevas peticiones de joining y leaving.

Como se observa en el ejemplo, para unirse al grupo se debe llamar al método joining(), y para abandonarlo al método leaving(). Con el primero, se arranca el thread y se empieza

a escuchar en el puerto especificado tráfico multicast de rekeying. Con el segundo, se para el thread. Igual como sucedía en el servidor, también debemos de implementar el interfaz LKHListener si deseamos detectar cambios en la clave de cifrado, SEK:

```
import secureMulticast.client.*;
import secureMulticast.util.*;
public class implements LKHListener
   {
       LKHclient client = new LKHclient(String KeyServer, int TCPport);
       client. addLKHListener(this);
       int ID = client.joining();
       client.leaving();
       public void LKHEventPerformed(LKHEvent e)
          {
              if(e.getEvent() == LKHEvent.NEW_SEK)
              e.getNewSEK();
          }
    }
```

5.3. Contribución a la adecuación del GDOI

Todas las propuestas de algoritmos de renegociación presentadas en los Capítulos 3 y 4 deben incluirse en la definición de un protocolo de comunicaciones para llevarse a cabo. El GDOI es el protocolo estándar del IETF para las comunicaciones seguras de grupo. La forma más sencilla se definir un protocolo para los citados algoritmos es incluirlos en el estándar GDOI.

5.3.1. *Group Domain of Interpretation*: GDOI

El GDOI *Group Domain of Interpretation* es un protocolo definido por el IETF en el RFC3547 [Baugher et al., 2003]. Permite la gestión y distribución de claves de cifrado en comunicaciones seguras en grupo. Se trata de un protocolo extremo a extremo de nivel de aplicación que utiliza UDP como nivel de transporte.

Dentro del marco de referencia de la seguridad en multicast, descrito en 2.3.2, se trata del protocolo de comunicación entre los miembros emisores o receptores y el Gestor de Claves, el cual se encarga de calcular las claves y distribuirlas y establecer una asociación de seguridad (SA o *security association*) que incluya a todos los miembros autorizados.

Básicamente se trata de una extensión al caso multicast del protocolo IKE (*Internet Key Exchange*) [Harkins and Carrel, 1998], del cual aprovecha la fase inicial para establecer un canal seguro entre un miembro y el GCKS. Además define una nueva fase que sirve para la transmisión de la clave de grupo y de los mensajes multicast necesarios para poder distribuir esta clave a todos los miembros.

Con la primera fase del protocolo IKE se establece un canal seguro mediante el cual el GCKS y cada uno de los miembros mantienen lo que se denomina una Asociación de Seguridad de tipo 1 (*Category 1 SA*) unicast que sirve para transmitir las claves al miembro en modo PULL. El modo PULL se activa siempre que el miembro solicita información al GCKS de forma unicast.

El segundo tipo de Asociación de Seguridad (*Category 2 SA*) sirve exclusivamente para la protección de los mensajes de gestión de claves. Esta SA la establece el KS y se envía a los miembros en modo PUSH. En este modo la comunicación viene forzada por el KS que envía un mensaje a todos los miembros del grupo multicast.

Finalmente existe una tercera categoría de asociación de seguridad (*Category 3 SA*), también de tipo multicast, que sirve para la protección del tráfico de datos. Todo este tipo de relaciones pueden observarse en la Fig 5.2.

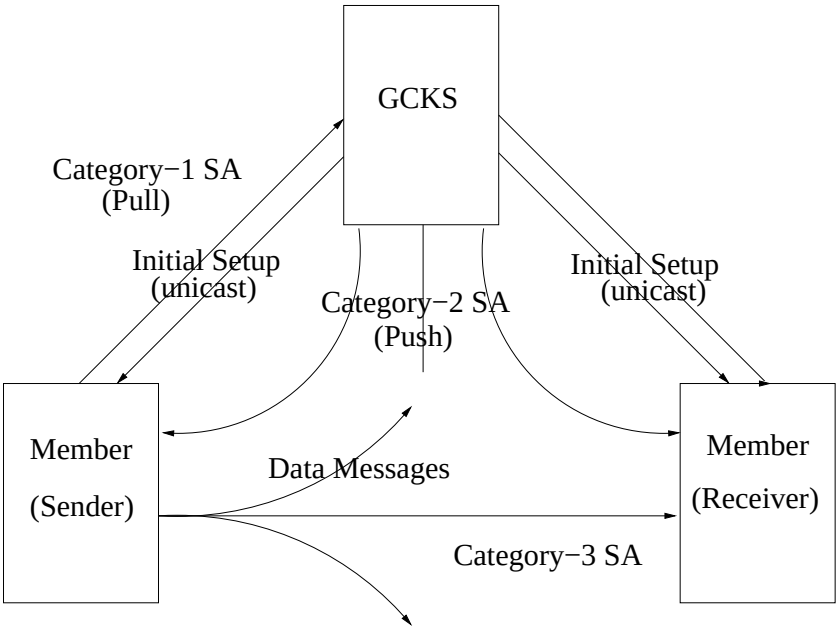


Figura 5.2: Asociaciones de Seguridad en GDOI

Por compatibilidad con los requisitos de seguridad descritos en el Capítulo 4, la SA de tipo 3 se corresponde básicamente con la TEK o SEK, las asociaciones de seguridad tipo 1 (entre GCKS y miembros) son las IK establecidas entre ellos, y las SA tipo 2 se corresponden con las KEKs.

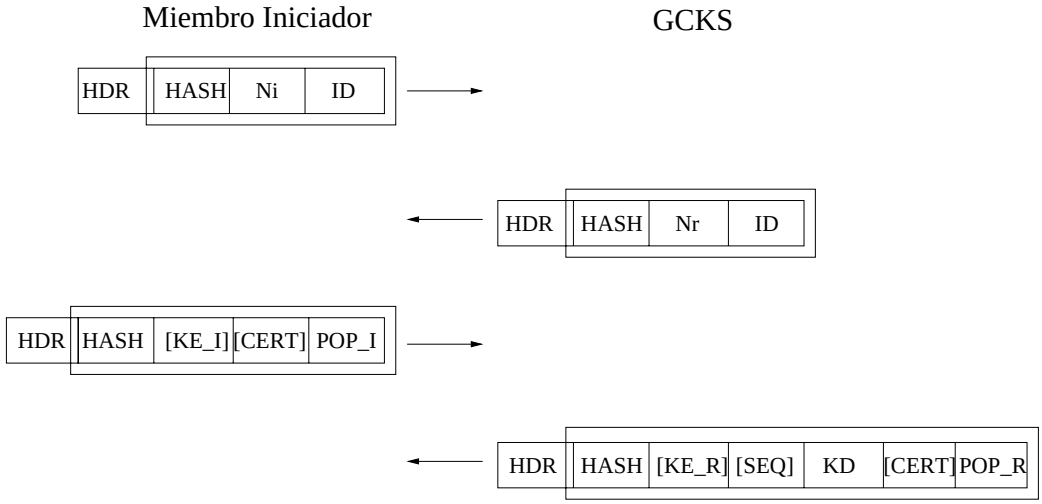
El protocolo está descrito con mucho más detalle en el Anexo B. De cualquier forma, a continuación describiremos brevemente las dos fases de funcionamiento para entender cómo deben adaptarse los mensajes de *rekeying* a las propuestas de esta tesis.

El protocolo define dos fases principales:

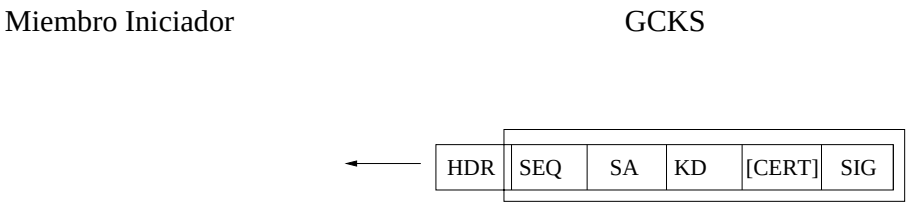
1. Una fase *GroupKey-Pull* que sigue a la fase uno de IKE y que se compone de cuatro mensajes entre el miembro y el GCKS. En esta es el miembro el que inicia la fase para solicitar el material de claves necesario para obtener la SEK (SA3). El intercambio de mensajes se protege mediante la IK (SA1). La fase tiene comportamiento PULL en tanto que es el miembro el que fuerza el inicio del intercambio de mensajes.

2. Una fase *GroupKey-Push* en la cual el Servidor de Claves envía al grupo una nueva clave, o bien porque la clave en uso ha caducado o bien porque se ha producido un cambio en el grupo (alta/baja de miembro). Obviamente este mensaje debe enviarse cifrado para proteger la nueva clave. La fase tiene un comportamiento tipo PUSH en tanto que es el KS el que fuerza el inicio del intercambio de mensajes.

En la Fig 5.3 pueden verse los mensajes de forma esquemática. A continuación se presenta una breve descripción de los mensajes que componen las dos fases.



(a) Fase GroupKey-Pull



(b) Fase GroupKey-Push

Figura 5.3: Mensajes del protocolo GDOI

5.3.2. Propuesta de adaptación del protocolo GDOI a *batch* con miembros dinámicos.

Básicamente, la inclusión de los algoritmos descritos, sólo afecta a los paquetes que se envían en la PHASE 2 del GDOI, cuando los mensajes tipo PUSH y PULL se intercambian. De todos los campos expuestos contenidos en los distintos paquetes, el único que afecta a la forma en que se genera el mensaje de actualización es el KD (*Key Download array*). El formato de paquete para un *array* de actualización LKH es el que se presenta en la Fig 5.4.

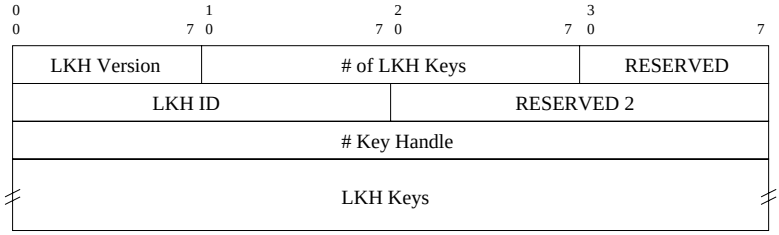


Figura 5.4: Array de actualización LKH

Se usa para enviar a los miembros las claves que deben actualizar. Los campos están mejor explicados en el Anexo B pero consiste mayormente en un identificador de la versión del LKH seguido de un conjunto de claves LKH. A su vez, cada clave LKH sigue una estructura de paquete según el siguiente formato. Fig B.12.

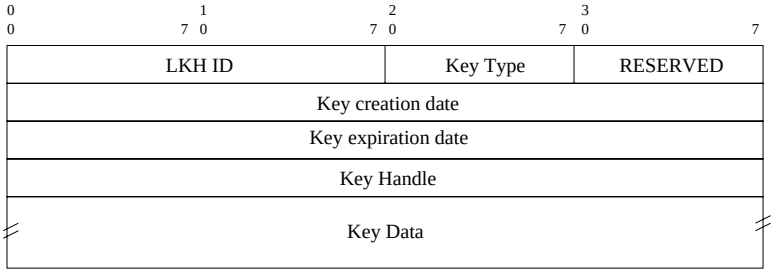


Figura 5.5: Formato de clave LKH

Donde LKH ID es la posición de la clave en el árbol, *key handle* es un identificador único que sirve para localizar la clave y *Key Data* es el valor real de la clave.

Tanto para el caso de OKS como sm-LKH/OKS en proceso individual, el formato de paquetes descritos ya es válido para transortar sus *arrays* de actualización. Únicamente debería definirse un valor estándar para cada uno de ellos de forma que el receptor supiera que se trata de vectores de actualización de estos algoritmos.

En el caso en que se requiera proceso por lotes, si se realiza mediante Lam-Gouda simple, la posición de los miembros es siempre la misma, o como mínimo sigue las reglas definidas por el camino inicial, con lo cual, la posición de las claves no debe definirse de forma explícita.

Si, en cambio, se pretende realizar proceso por lotes mediante algoritmo balanceado (*balanced batch*) debe notificarse la nueva posición de los nodos raíz de los subárboles aprovechables en el nuevo árbol de claves. Ello implica que LKH ID deberá cambiar de un *batch* a otro. Si se envía esta información al grupo, se requerirá un cambio en el formato de paquete KD, además, se producirá un incremento de ancho de banda debido a la nueva información transmitida.

Con el fin de evitar esto, en la implementación del GDOI realizada no se usa el valor LKH ID para posicionar los nodos en el árbol sino únicamente el *Key Handle*, que se usará para localizar la clave a actualizar en el árbol entero de claves (en el *Server*) o en el vector de claves (en cada uno de los miembros).

De hecho, cuando se utilizó el LKH ID de los paquetes descritos anteriormente su valor se actualizaba de forma incorrecta cuando habían transcurrido diversos *rekeyings*. Así, se consideró que dicho parámetro no sólo era innecesario sino que entorpecía el proceso de actualización de claves.

La implementación del GDOI que hemos comentado sigue todas las recomendaciones del I-D, y por consiguiente, genera paquetes KD con el campo LKH ID correspondiente, pero este es ignorado a la hora de generar e interpretar el mensaje de rekeying.

Todas las variaciones del LKH explicadas pueden implementarse de forma sencilla usando tan sólo el *Key Handle* como identificador de la clave para localizarla dentro del vector de claves. El cambio de posición no afecta realmente a los miembros ya que ellos no necesitan saber donde están situados en los árboles sino únicamente tener la información suficiente para

mantener su vector. De hecho, cuanta menos información se les proporcione a los miembros menos susceptibles a ataques de confabulación seremos.

De acuerdo a estas consideraciones, y a fin de simplificar el formato de paquetes LKH, se propone cambiar el *array* de actualización del LKH al formato dibujado en la Fig 5.6.

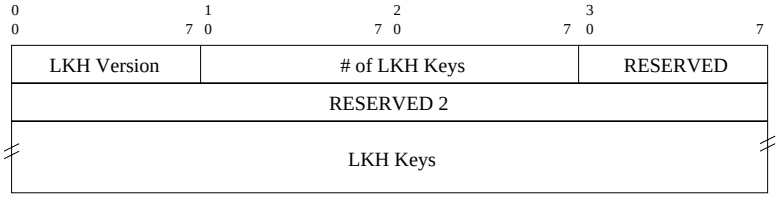


Figura 5.6: Propuesta de Array de actualización LKH

Y construir cada clave LKH de la siguiente forma. Fig 5.7.

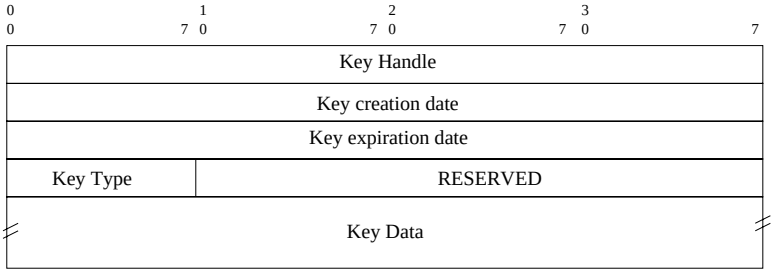


Figura 5.7: Propuesta de Clave LKH

Capítulo 6

Conclusiones y líneas futuras

6.1. Conclusiones

El uso comercial de los servicios multimedia en red hace imprescindible la introducción de aspectos de seguridad. Típicamente, un servicio multimedia en red puede dividirse en dos fases de funcionamiento: acceso y distribución. Mientras que la seguridad en la fase de acceso (autenticación y pago) ha sido ampliamente estudiada en la literatura, la seguridad en la fase de distribución ha sido objeto de menor atención (cifrado y gestión de claves).

Una de las tecnologías usadas en la fase de distribución de los servicios multimedia es el multicast. Éste se ha consolidado como la forma de usar de forma eficiente el ancho de banda de la red cuando las comunicaciones implicadas son de grupo. Añadir seguridad a los servicios multimedia requerirá el estudio de los requisitos de seguridad en este tipo de comunicaciones.

Según el IETF, la integración de la seguridad en las comunicaciones de grupo, y concretamente en la tecnología multicast, contempla la confidencialidad, autenticación de fuente y de grupo y gestión de políticas. Tanto la confidencialidad como la autenticación de grupo quedan resueltas con el uso de una clave de cifrado usada y compartida por todos los miembros del grupo. Sin embargo, esta clave puede verse comprometida de forma muy fácil debido a la propia dinámica de los grupos multicast.

Para garantizar secreto estricto hacia adelante y hacia atrás, la clave de sesión debe actualizarse cada vez que un usuario se dé de alta o de baja en el grupo. Este hecho introduce la gestión de claves de grupo como uno de los ejes principales a estudiar por la seguridad en multicast. Debe hallarse un método simple y eficiente de repartir y redistribuir las claves dentro del grupo. Este es el denominado problema de *rekeying*.

La forma más sencilla de redistribuir la clave de sesión, cuando ésta se ha visto comprometida, es a través de conexiones unicast con cada uno de los miembros. Evidentemente este método es muy ineficiente en cuanto a ancho de banda, no aprovecha el canal multicast y además no es escalable en N. Entre otros, los parámetros que se usan para la evaluación y comparación de este tipo de algoritmos son: ancho de banda necesario para actualizar la clave de sesión, cantidad de memoria a mantener por el servidor y los miembros, número de mensajes implicados en la renegociación de claves y latencia.

Las propuestas más exitosas a la hora de renegociar claves en grupos se basan en el uso de árboles lógicos de claves. Estos sistemas utilizan dos tipos de claves: claves de sesión o SEKs y claves que cifran claves o KEKs. Una entidad centralizada de confianza (Gestor de Claves)

es el encargado de mantener, actualizar y distribuir una estructura lógica de claves que sirve a cada uno de los miembros para poder descifrar la clave de sesión en uso.

El uso de la estructura lógica del árbol permite reducir el ancho de banda necesario y el número de mensajes de forma logarítmica respecto el caso trivial, a cambio se incrementa la cantidad de memoria a mantener por el servidor de claves y los miembros y crece el coste computacional.

La evaluación y comparación de distintas propuestas existentes en la literatura se ha realizado en el Capítulo 3. Se ha concluido que dependiendo de cuál sea el parámetro más importante a economizar deberá utilizarse un mecanismo u otro. Todos ellos se basan en árboles lógicos de claves.

Como contribución de este trabajo de tesis se han propuesto dos algoritmos de gestión de claves en grupos multicast que minimizan la cantidad de memoria necesaria en el gestor de claves y el número de mensajes respectivamente. La primera de las contribuciones (OKS) se basa en el uso de funciones pseudoaleatorias que facilitan al servidor de claves la generación de las claves asociadas a una determinada posición, evitando así que deba guardar todas las claves del árbol.

Del análisis de seguridad descrito se desprende que la clave asociada a un determinado nodo debe cambiar en cada proceso de *rekeying*. La forma en que se actualizan todas las claves del árbol en cada renegociación es a través de la distribución a todos los miembros de un parámetro de actualización que debe operarse *xor* con el valor anterior. Como sistema de distribución se utiliza un mecanismo similar al LKH, por este motivo no se consigue mejora en cuanto a ancho de banda y número de mensajes.

La segunda contribución de esta tesis en cuanto a algoritmos de gestión de claves (sm-LKH) consigue reducir el ancho de banda y el número de mensajes de OKS. El mecanismo se basa en técnicas de *broadcast encryption* que usan aritmética modular. Se fundamenta en la propuesta anterior pero reparte el parámetro de actualización mediante un sólo mensaje construido de forma que sólo los receptores deseados podrán recuperar (mediante reducción modular) el mensaje enviado a todo el grupo. Con este método se consigue reducir el ancho de banda y número de mensajes (latencia) de las propuestas anteriores.

Aunque formalmente las técnicas con proceso individual son las únicas que pueden garantizar secreto hacia adelante y hacia atrás de forma estricta, la mayoría de aplicaciones reales no requieren de un nivel de seguridad tan elevado. En estos casos el proceso por lotes consigue reducir el ancho de banda necesario a costa de decrementar el nivel de seguridad del sistema: se introduce una ventana de vulnerabilidad en que se podrá acceder a los datos del sistema sin ser autorizado.

Las técnicas de proceso por lotes, para garantizar un desempeño óptimo, deben mantener el árbol lógico de claves balanceado para todo instante de tiempo. La única propuesta existente en la literatura de proceso en *batch* no garantiza el balanceo si se aplica a escenarios reales con llegadas y bajas de usuarios a ráfagas.

Otra contribución de esta tesis ha consistido en la definición, implementación y simulación de un algoritmo de renegociación de claves por lotes que da lugar a árboles balanceados. La contribución se basa en permitir que los miembros cambien de posición en el árbol, de manera que, aunque se diesen de baja ráfagas de miembros, los restantes puedan recolocarse para

mantener un árbol balanceado. Se comprobó como el balanceo se traduce en un decremento en uso de ancho de banda. Asimismo, se han combinado las propuestas de minimización de memoria y mensajes con el proceso por lotes para conseguir incrementar la eficiencia de estos últimos.

Como contribución final, y resultado de la implementación de los algoritmos y el diseño del simulador, se propusieron mejoras al estándar de comunicaciones de grupo del IETF (GDOI) para que puedan soportar los algoritmos descritos.

6.2. Líneas futuras

El área de investigación de la seguridad en multicast es incipiente y muchos de sus campos quedan aún inexplorados. Además de los temas que quedan fuera del propósito de esta tesis, el estudio de los temas abordados ha dado lugar a flancos abiertos susceptibles de ser tratados con más profundidad.

En primer lugar, del propio Marco de Referencia presentado, se desprende la necesidad de generalizar los algoritmos descritos al modelo descentralizado. Todos los mecanismos de gestión de claves descritos se basan en la existencia de una única entidad central de confianza. Cómo debe adaptarse este Marco de Referencia a entornos descentralizados actuales como las redes ad-hoc o la Grid es la primera línea futura a abordar.

Un tema lateral, pero imprescindible para lograr los objetivos de esta tesis, es la generación de patrones de comportamiento sintético de usuarios multicast. De los tres escenarios considerados, Web-TV y videoconferencia están bastante estudiados y caracterizados. Los juegos en red, sin embargo, no se han explorado con detalle. Dentro del trabajo de la tesis, se ha realizado una caracterización del comportamiento de los usuarios de juegos en red basado en un estudio existente. De todos modos, queda por realizar la validación empírica mediante muestras reales de las características asumidas del estudio de Henderson.

Un aspecto surgido de la propuesta de algoritmo de gestión de claves basado en *broadcast encryption* es la imposibilidad de evitar que un usuario malintencionado revele su *set* de claves (recuérdese que no se consideró ataque de confabulación). Una posible línea futura es el estudio de las técnicas de *fingerprinting* aplicadas al sistema de claves de forma que revelar un conjunto de claves apunte al miembro deshonesto.

La introducción de propiedades matemáticas de la teoría de números en la generación del mensaje de actualización, ya por sí sola, abre todo un abanico de líneas futuras. Las técnicas *broadcast encryption* discutidas en esta tesis no son únicas, y existen otros mecanismos susceptibles de ser combinados con los métodos basados en árboles lógicos de claves.

Un aspecto no mencionado en esta tesis, pero que se ha dado por supuesto a la hora de realizar sus contribuciones, es el problema de la fiabilidad de los mensajes multicast. Se ha supuesto que tenemos la garantía que los mensajes de *rekeying* llegan a su destino, pero en realidad no se tiene seguridad de ello, ya que por definición, multicast utiliza transporte no fiable. Al introducir fiabilidad en los protocolos de transporte multicast aparece el problema de la implosión de mensajes de reconocimiento. La forma de superar los inconvenientes que causan estas implosiones es mediante un mecanismo denominado *concast* y que viene a ser el proceso inverso al multicast.

Si ya la seguridad en multicast es un campo relativamente joven, el estudio de las amenazas de seguridad en entornos *concast* es un campo todavía más abierto. Cómo deben aprovecharse los servicios propuestos en multicast para garantizar la segurida en *concast* es otra línea futura que debe apuntarse.

En cuanto a algoritmos de proceso por lotes, se ha visto como la mejora de ancho de banda respecto al caso individual se debe tanto a la supresión de *rekeyings* innecesarios como necesarios. Los segundos son los que hacen disminuir la seguridad del sistema mientras que la supresión únicamente de los primeros daría lugar al caso ideal de reducción máxima de ancho de banda sin comprometer la seguridad. Como se vió, esto no es posible ya que no puede saberse cuando van a producirse las altas o las bajas del patrón en un escenario real, y por tanto no puede adaptarse el periodo de *batch* a estas características. La inclusión de técnicas predictivas en el comportamiento del patrón de usuario es otra posible línea de estudio. Así se podría adaptar el periodo de renegociación para reducir el ancho de banda comprometiendo de forma mínima la seguridad del sistema

Se ha observado que no todos los periodos de *batch* producen la misma mejora en parámetros de eficiencia. Debe determinarse, para cada escenario, con estadística conocida, cuál será el periodo de renegociación óptimo que dará lugar a unos parámetros de eficiencia mejores respecto el tamaño de ventana de vulnerabilidad permitido.

Como punto final, aunque más de carácter burocrático, debe comentarse que el trabajo futuro a realizar para la inclusión de las contribuciones de esta tesis en un *standards track* que recoja todos los algoritmos de gestión de claves estandarizados por el IETF.

Apéndice A

Caracterización de grupos multicast

Los algoritmos de renegociación de claves por lotes, debido a la complejidad de su casuística, sólo pueden evaluarse mediante simulaciones. Para ello, se debe caracterizar el comportamiento de los usuarios de forma que se obtengan patrones de peticiones de alta y de baja de acuerdo a los distintos escenarios de referencia presentados en 2.4. En este anexo se presenta el trabajo realizado para la obtención de los modelos estadísticos de fuente en los escenarios WebTV, juegos en red multi-jugador y servicios de videoconferencia.

A.1. Web TV

WebTV es un escenario en el que hay una única fuente que transmite información hacia varios (muchos) usuarios o receptores pasivos. Sus características fundamentales son las siguientes [Canetti et al., 1999a]:

- El número de receptores pasivos es muy elevado, digamos que hasta órdenes de cientos e incluso de miles.
- La fuente de información se supone con un ancho de banda y unos recursos suficientes para satisfacer los requerimientos de los usuarios, mientras que se debe tener en cuenta que los recursos de los usuarios o receptores son, al menos en su mayoría, limitados.
- El tiempo de vida de un grupo normalmente es largo, aunque la entrada y salida de miembros en ese grupo sea bastante frecuente. Además, en horas punta, el volumen de altas y bajas de usuarios subscritos a una determinada emisión es muy alto, aunque para facilitar el estudio se supone que los miembros permanecen vinculados a su grupo por un largo periodo de tiempo.
- El volumen de datos es elevado y mantener una latencia pequeña y constante es fundamental.
- La revocación o baja de miembros debe realizarse en términos de segundos e incluso minutos, pero no parece necesario que se realice en términos de fracciones de segundo.

A.1.1. Modelo utilizado

El comportamiento de los usuarios en un servicio de Web TV es similar al de los telespectadores de televisión terrestre, donde sí existen amplios estudios de su comportamiento. El modelo para las altas y bajas de usuarios es un sistema de llegadas de Poisson exactamente igual al usado, por ejemplo, para contar las llamadas recibidas en una centralita telefónica en un determinado periodo de tiempo, en la que a priori no existe correlación alguna entre el hecho de que se produzca una llamada y el hecho de que se haya producido una llamada en un instante anterior (sistema sin memoria). De esta forma y partiendo de las características descritas anteriormente, se puede observar que, tal y como explica [Almeroth and Ammar, 1996] podemos modelar las altas y baja de usuarios en el sistema como un proceso sin memoria de Poisson.

Es sabido que el tiempo entre llegadas sigue una distribución exponencial muy sencilla de modelar.

Función de distribución exponencial

Tal y como hemos visto en el apartado anterior el tiempo entre llegadas en un escenario de WebTV se modela con una función de distribución exponencial, ésta viene definida por:

$$f(x) = \begin{cases} 0 \; si \; x < 0 \\ \lambda e^{-\lambda x} \; si \; x \geq 0 \end{cases}$$

$$F(x) = \begin{cases} 0 \; si \; x < 0 \\ 1 - e^{-\lambda x} \; si \; x \geq 0 \end{cases}$$

Si pensamos en X como el tiempo que pasa entre sucesos de una distribución de Poisson, entonces $\lambda = \frac{\alpha}{T}$. Siendo el tiempo medio entre llegadas $\frac{1}{\lambda} = \frac{T}{\alpha}$.

Generación de la función de distribución exponencial Es sabido que es computacionalmente sencillo la generación de números (pseudo) aleatorios siguiendo una distribución uniforme entre dos números dados, que por simplificación son en la mayoría de casos 0 y 1. Por lo tanto, el problema es cómo pasar de una distribución uniforme a una exponencial.

$$si \; X \; es \; U(0,1) \; \Rightarrow \; Y = \frac{-lnX}{\lambda} \; es \; Exp(\lambda)$$

Es decir, que si X sigue una función de distribución uniforme entre 0 y 1, Y=(-lnX) sigue una distribución exponencial de media 1

Parámetros

Almeroth y Ammar en [Almeroth and Ammar, 1997], obtuvieron el modelo de comportamiento de usuarios en entornos tipo WebTV por comparación con estadísticas obtenidas a partir de trazas de tráfico reales de distintas sesiones de WebTV y radio "on-line" (que por otra parte tiene un comportamiento muy similar). En concreto encontraron el parámetro que define unívocamente la distribución exponencial. Hemos visto en el apartado anterior que para

modelar este comportamiento solamente es necesario éste parámetro, así como (en simulación) el número de muestras necesario para estar en un rango de error aceptable.

De los escenarios escogidos por Almeroth y Ammar éste es el escogido para su análisis en este trabajo:

- NASA STS-63 audio session del 17 de Febrero de 1995, que tuvo una duración de un par de días con una media de tiempo entre llegadas de 2,5 minutos, es decir que se puede modelar con una distribución de tiempo entre llegadas exponencial con parámetro $\lambda = \frac{1}{2{,}5 \cdot 60} = \frac{1}{150}$.

Si tenemos en cuenta que la suma de dos procesos de Poisson con λ_1 y λ_2 es otro proceso de Poisson de $\lambda = \lambda_1 + \lambda_2$, y si partimos del hecho de que al fin y al cabo toda llegada ha de producir una salida podemos suponer .

$$\lambda = \lambda_{llegadas} + \lambda_{salidas} = 2\ x\ \lambda_{llegadas} = 2\frac{1}{2{,}5 \cdot 60} = \frac{1}{75}$$

A.1.2. Generación del patrón con matlab

Partimos de que ya estamos en un régimen estacionario, puede suponerse que la probabilidad de petición de alta sea igual que la de petición de baja, es decir, $p = \frac{1}{2}$. Dividimos el tiempo total en subintervalos del orden de segundos, esto nos obliga en simulación a utilizar 3600 muestras por hora simulada. Por cada muestra simulada guardamos un valor que puede ser:

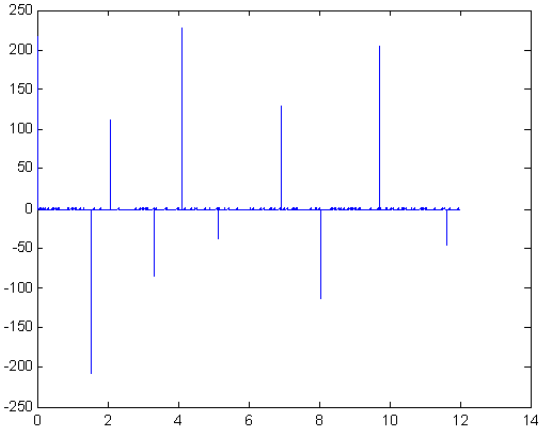
- 0 - si no ha habido ni altas ni bajas
- 1 - si ha habido una petición de alta
- -1 - si ha habido una petición de baja

Es decir, que la suma total de muestras debe tender a 0.

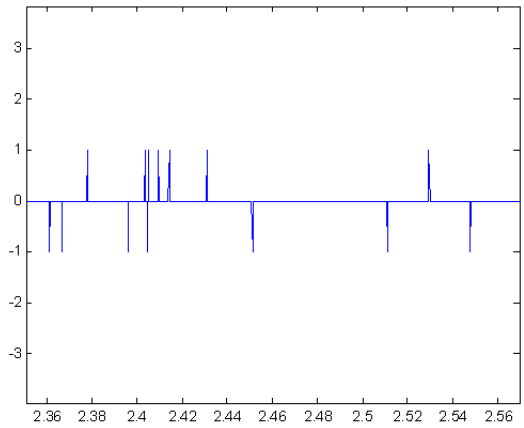
Para la obtención de valores aleatorios siguiendo una distribución exponencial en Matlab utilizamos la función ya existente $exprnd(\lambda)$, que nos devuelve un valor según una distribución exponencial de media $\frac{1}{\lambda}$.

Además hemos añadido de forma determinista al modelo una serie de picos de altas y bajas, que se podrían corresponder con programas con una audiencia mayor y fiel.

A continuación (Fig A.1) se muestran resultados de simulaciones para un período de 12 horas. Se muestra el patrón de tráfico sintético generado mediante Matlab, en la gráfica se toma cada petición de alta en un instante t_o como una $\delta(t - t_o)$ y cada petición de baja como una $-\delta(t - t_o)$. Al haberse tomado un intervalo de tiempo muy largo, lo que se observa en esta ilustración es sobre todo los picos de altas y bajas que se han introducido de forma determinista en el modelo. Si observamos un intervalo más pequeño se observa, en cambio, altas y bajas que siguen una distribución exponencial con $\lambda = \frac{1}{75}$, o lo que es lo mismo con un tiempo medio entre peticiones de 75 segundos.



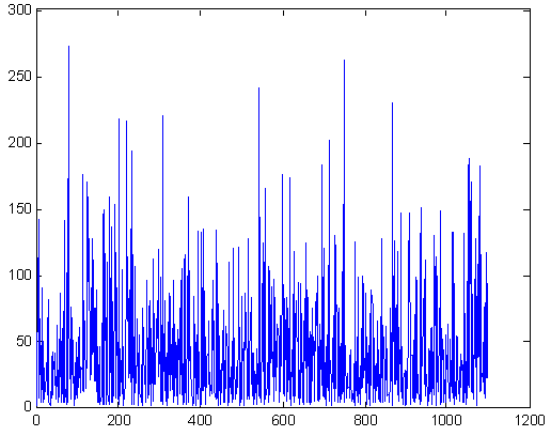
(a) altas (+1) y bajas (-1) de usuarios



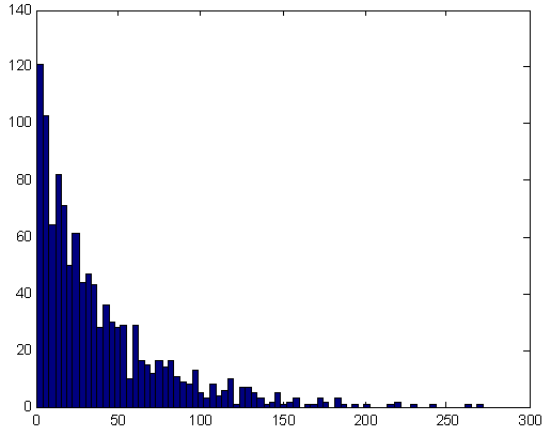
(b) detalle de tráfico generado

Figura A.1: Patrones y tiempo entre llegadas para Web-TV

La Fig A.2 muestra el histograma del tiempo entre peticiones, tanto de altas como de bajas. Se ha prescindido a la hora de generar el histograma de los picos de altas y bajas con el objetivo de reflejar más claramente la función densidad exponencial para el tiempo entre peticiones (de altas o de bajas).



(a) Tiempo entre llegadas



(b) histograma

Figura A.2: Tiempo entre llegadas e histograma para Web-TV

El código matlab usado para generar los patrones es el que sigue

```
% trafic users
% web-tv
clear;
users=zeros(1,43200);
peak_times_start=[1 7500 14750 25000 35000];
peak_times_stop=[5500 12000 18550 29000 42000];
%joinings
indice=1;
while indice < 43200
    users(indice)=1;
    indice=indice+ceil(exprnd(300));
end
```

```
%picos de trafico de la sesion
%causados por inicio de programacion
for j=1:length(peak_times_start)
    indice=peak_times_start(j);
    num_join_sessio=ceil((200*(rand(1,1)))+100);
    users(indice)=num_join_sessio;
end
% leavings
indice=1;
while indice < 43200
    indice=indice+ceil(exprnd(100));
    users(indice)=-1;
end

%picos de leavings de la sesion
for j=1:length(peak_times_stop)
    indice=peak_times_stop(j);
    num_leaving_sessio=users(peak_times_start(j))*rand(1,1);
    users(indice)=-num_leaving_sessio;
end
```

A.2. Servicios de Videoconferencia

Se entienden escenarios de videoconferencia como escenarios con un número reducido de usuarios en los que todos pueden transmitir y recibir, aunque sólo unos pocos acaparen la mayoría del ancho de banda de transmisión. Digamos que se corresponden a escenarios de videoconferencia entre ejecutivos de una empresa o a escenarios en los que se imparten clases "on-line".

A partir de las descripciones de [Canetti et al., 1999a] y [varios autores, 1999] se puede extraer que las características principales de este tipo de escenarios son las siguientes:

- Los grupos se forman generalmente para un evento específico y son por lo tanto de corta duración, digamos minutos u horas. El número de miembros del grupo puede considerarse estático, ya que, en general cada usuario se inscribe al principio para participar en la videoconferencia y permanece hasta que ésta acaba. Además, si un miembro abandona la videoconferencia no se considera crucial revocarle criptográficamente (renegociación de claves) y en la mayoría de casos no se realiza.

- Aunque los requerimientos de ancho de banda y de latencia pueden variar sustancialmente dependiendo del tipo de aplicaciones a usar en la videoconferencia, se supone para el modelo que se ha de garantizar una latencia pequeña ($<$200ms) que garantice la simultaneidad e interactividad entre de los usuarios.

- La parte más crucial en términos de seguridad es la autenticación de los datos y de los usuarios que los intercambian. Se ha de garantizar el secreto de los datos, así como garantizar la validez de un usuario como receptor o emisor, ya sea de forma anónima o revelando su identidad. Generalmente debe haber un "mediador" que debe servir como

centro de confianza en iniciar la videoconferencia, aunque en muchos casos esta función se distribuya entre algunos de los miembros del grupo.

- Debido a la poca importancia que se ha dado a este escenario y a su carácter determinista, no hay estudios anteriores a este trabajo. Por lo tanto no hay referencia alguna en lo que a este modelo se refiere.

Para modelar el comportamiento de usuarios en este escenario se parte de un modelo determinista en el cual todas las altas se producen en un intervalo de tiempo anterior al comienzo de la videoconferencia y todas las bajas en un intervalo de tiempo posterior, puesto que se supone que el número de integrantes del grupo en conferencia es fijo y se calcula previamente a la sesión. Hay que tener en cuenta que en el modelo más común de videoconferencia, un grupo de usuarios fijo y reducido queda a una determinada hora para realizar una videoconferencia, y por lo tanto, salvo raras excepciones, no habrá peticiones de baja durante el transcurso de ésta.

De todas formas a la hora de generar el modelo se ha establecido una fase de debate en la cual las bajas se producen de forma exponencial una vez ha terminado la videoconferencia, es decir, se considera que no hay bajas durante la videoconferencia y que éstas se producen de forma exponencial cuando ésta termina. Comportamiento que se puede justificar como mini debates o conversaciones entre integrantes posteriormente al grueso del debate o videoconferencia en sí.

A.2.1. Modelo utilizado

Como se ha explicado en el apartado anterior se divide temporalmente la videoconferencia en dos fases, una determinista y una probabilística con distribución exponencial:

- Fase determinista: incluye las altas de todos los usuarios y el período que dura el grueso de la videoconferencia. Se parte de la hipótesis de que durante este periodo no se producen peticiones de bajas, y que todas las altas se producen antes de que comience y de un número fijo y sabido de usuarios.

- Fase exponencial o Fase de debate: una vez acabada la videoconferencia se supone que los usuarios la van abandonando de forma aleatoria según un proceso de Poisson, es decir con una media de tiempo entre abandonos distribuida de forma exponencial.

En realidad la única fase que requiere de un modelo de estudio sería la fase de debate. Para realizar este modelo se ha seguido un proceso de Poisson, que bien podría mejorarse teniendo en cuenta las posibles interacciones entre subgrupos, lo que generaría que se abandonara la sesión por grupos de más o menos usuarios en post-debate. Como se ha optado por el modelo de Poisson el dato más importante para el análisis es el tiempo medio entre peticiones de baja.

Parámetros

Se ha partido del supuesto que el grupo inicial de usuarios dados de alta en la videoconferencia es un número aleatorio uniforme de entre 5 y 100 usuarios. La duración del grueso

de la videoconferencia es uniforme entre 30 y 90 minutos. El parámetro se debería obtener a partir de estadísticas reales, aunque, como éste no puede ser el caso, la hemos estimado en $\frac{1}{\lambda} = X$, siendo X una variable aleatoria uniforme entre 15 y 120 segundos.

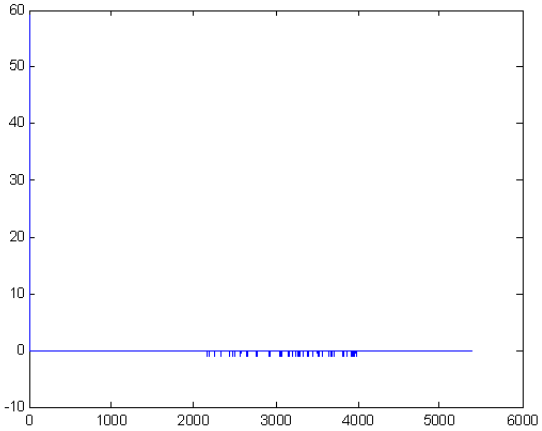
A.2.2. Generación del patrón con matlab

Duración del grueso de la videoconferencia = U(30 minutos, 60 minutos). Tiempo medio entre peticiones de baja = U(15 segundos, 120 segundos).

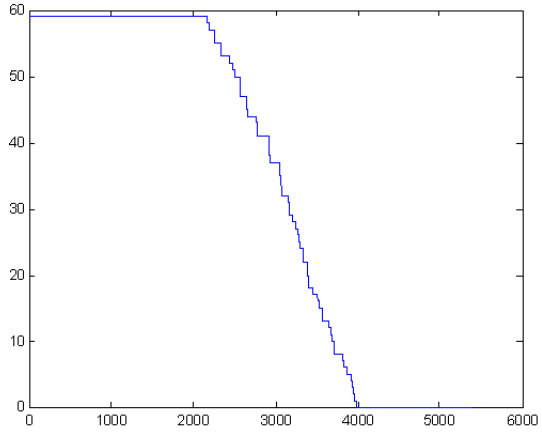
En la Fig A.3 se observa claramente las dos fases del modelo:

- Fase determinista: todas las altas (cada alta es un +1) se producen en el instante inicial, en concreto se producen 59 altas (número de usuarios=59). Durante el período de tiempo que dura el grueso de la videoconferencia (2160 segundos) no se produce ninguna baja, tal y como indica el modelo.

- Fase probabilística o de debate: se producen las bajas de los usuarios del sistema de forma exponencial a partir del fin del grueso de la videoconferencia. Se ve claramente cada baja como un -1. Además tal y como está diseñado el modelo es fácil observar que no se produce nunca más de una baja en la unidad de tiempo (segundo).



(a) Patrón de tráfico

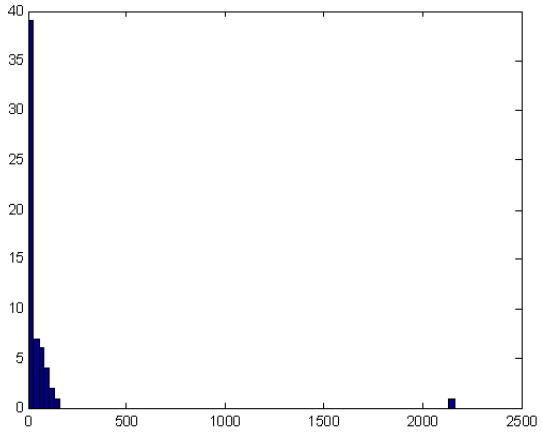


(b) numero de usuarios en cada t

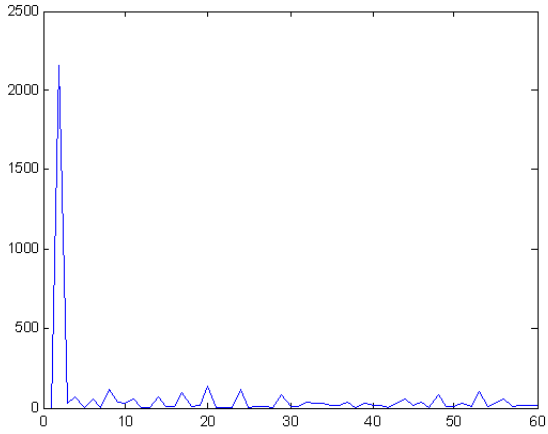
Figura A.3: Patrones y número de miembros para videoconferencia.

De la Fig A.4a cabe resaltar como para tiempos pequeños se intuye la exponencial del tiempo entre bajas, además se observa un pico alrededor de 2150 seg, es decir al final de la videoconferencia, que los que nos viene a mostrar es el tiempo entre las altas y bajas (duración de la video-conferencia).

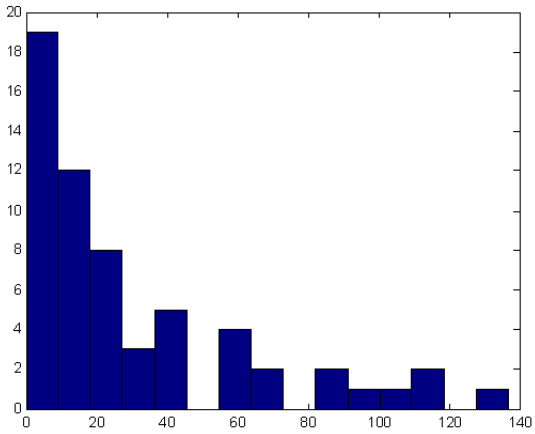
Obviamente si prescindimos de la fase que dura el grueso de la videoconferencia y analizamos el tiempo entre llegadas se observa mejor la distribución exponencial que siguen las bajas de la videoconferencia (véase Fig A.4b y c), aunque debido al número reducido de usuarios que se usa en este modelo (entre 5 y 100) no se aprecia de forma muy clara.



(a) Histograma dos fases



(b) Tiempo entre llegadas



(c) histograma fase de debate

Figura A.4: Tiempo entre llegadas e histograma para videoconferencia

El código matlab usado para generar los patrones es el que sigue

```
% trafic users
% virtual_conference
clear;
users=zeros(1,4800);
% joinings
indice=1;
num_users=200*rand(1,1);
users(indice)=num_users;
%num users inicial
leavings=users(1)*(rand(1,1))
%users leavings
```

```
indice=ceil(2400*rand(1,1))
for i=1:ceil(leavings)
    indice=indice+ceil(exprnd(200));
    users(indice)=-1;
end
%reagrupacion

indice=ceil(2400*rand(1,1))
for i=1:ceil(leavings)
    indice=indice+ceil(exprnd(200));
    users(indice)=1;
end
```

A.3. Juegos en red multi-jugador

Los juegos en red para múltiples jugadores representan actualmente una de las formas más populares de comunicación de grupos en Internet. Tristan Henderson realizó un modelo de comportamiento de usuarios multicast en este tipo de escenarios. De [Henderson and Bhatti, 2001] podemos obtener resumidamente las siguientes características para juegos multi-jugador:

- Los grupos se forman para una partida específica, gestionada siempre por un servidor de juegos que hace las veces de "moderador", y son por lo tanto de corta duración. Ahora bien, ésta corta duración no impide que el número de miembros que se inscriben y salen de cada grupo no sea elevado, así como el número de usuarios jugando simultáneamente. Los grupos en todo caso no suelen pasar en cantidad el orden de decenas.

- La naturaleza de tiempo-real de los juegos en red, especialmente para los juegos de acción o "arcade", provoca la necesidad de garantizar que tanto los retardos como los tiempos de respuesta sean mínimos. Es comprensible que un tiempo de respuesta lento alente a los jugadores a abandonar la partida, puesto que la jugabilidad se ve altamente disminuida. También es posible entender que los límites de jugabilidad pueden ser "repuestos" con la propia habilidad del jugador, es decir, que jugadores más hábiles permanecerán durante más tiempo en un grupo cuando la jugabilidad se vea afectada.

A.3.1. Modelo utilizado

El modelo utilizado es el obtenido por Tristan Henderson en [Henderson and Bhatti, 2001]. Este modelo fue obtenido a partir de las trazas generadas por usuarios reales en un servidor del juego Half-Life, muy popular en Internet.

En [Feldmann et al., 1998] y [Paxson and Floyd, 1995] se muestra como el tiempo entre llegadas de usuarios para aplicaciones de un único usuario sigue una distribución de Poisson. En cambio, para aplicaciones multi-usuario, y especialmente en aquellas en las que hay interacción entre los usuarios, como en los juegos multi-jugador, se ha llegado a la conclusión [Borella, 2000, Henderson and Bhatti, 2001] de que los tiempos entre llegadas de usuarios están muy correlados entre sí y que además presentan dependencias a largo plazo.

A.3.2. Modelo utilizado

La función Gamma-Pareto es, como su propio nombre indica, una híbrida entre las funciones Gamma y Pareto. Por lo tanto se definirá por cuatro parámetros α y p, c y α; aunque podemos eliminar un parámetro igualando la pendiente (log-constante) de la cola de la Pareto con la pendiente (variable) de la Gamma en un punto umbral x_{umbral}.

Para simplificar y con vistas a su aplicación suponemos la cola puede aproximarse con la función Zipf o Pareto ley 1 (c=1). Es decir, tenemos una función Gamma(a, p) 0 <x <x_{umbral} y una función Zipf(a) para $x_{umbral} \leq x < \infty$.

A continuación se plantea el modelo definido a trozos pensando en su uso para generar tráfico sintético.

Modelo definido a trozos

En la práctica no resulta sencillo generar tráfico gamma-pareto (o gamma-zipf) a partir de un modelo por aproximación continua, y no parece necesario para el propósito de este trabajo. Por lo tanto, se propone otro diseño de la función Gamma-Pareto visualmente menos atractivo pero mucho más fácil de simular. Este nuevo modelo, no es en principio continuo en la derivada, y se basa únicamente en las dos siguientes condiciones:

$$f_{gamma}(x_{umbral}) = f_{zipf}(x_{umbral})$$

$$\int_{-\infty}^{\infty} f_{gamma-zipf}(x)dx = 1 \Rightarrow \int_{0}^{x_{umbral}} f_{gamma}(x)dx + \int_{x_{umbral}}^{\infty} f_{zipf}(x)dx = 1$$

$$\Rightarrow \int_{0}^{x_{umbral}} f_{gamma}(x)dx - \int_{1}^{x_{umbral}} f_{zipf}(x)dx = 0$$

De donde fijando α y p, podemos obtener valores para x_{umbral} y a mediante MAPLE.

Parámetros

Tristan Henderson define en [Henderson and Bhatti, 2001] para el tiempo entre llegadas una distribución "heavy tailed". Este tipo de distribución puede modelarse mediante una función gamma-pareto.

Henderson estimó que la cola se modelaría concretamente con una Pareto ley 1 (c=1) o Zipf, e hizo uso del estimador de Hill para obtener una estimación del parámetro α que definía unívocamente esta función. Basándose en un conjunto de trazas préviamente obtenido, Henderson obtuvo una $\alpha = 1,15$.

Aplicando el modelo definido a trozos, podemos obtener la siguiente tabla para distintos valores de p:

Tabla A.1: Valores de p y a para α=1.15

α	1.15	1.15	1.15	1.15	1.15	1.15
P	2	3	4	5	6	7
A	.9211841260	1.472239147	2.060181161	2.675897600	3.313614220	3.969381890
x_{umbral}	1.945115316	1.672945805	1.531759390	1.444488333	1.384792542	1.341163384

Y de forma gráfica:

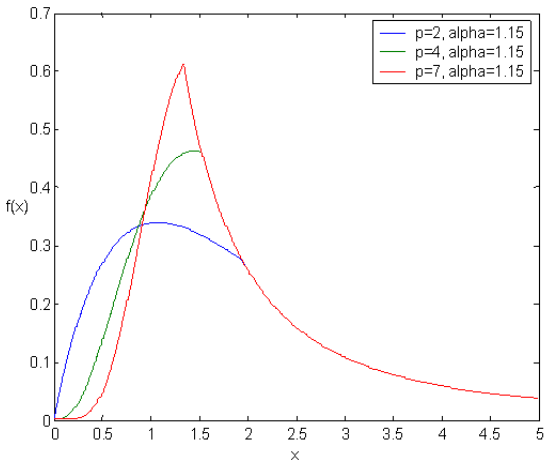


Figura A.5: Densidad Gamma-Zipf (modelo def a trozos) para $\alpha = 1{,}15$ y p variable

Se observa claramente que la Fig A.5 no es continua en la derivada, pero a la hora de generar tráfico tenemos en cambio una parte Gamma pura y otra Zipf pura, para las cuales sabemos generarlo, además se observa que para p=2 la discontinuidad en la derivada no es tan importante.

A.3.3. Generación del patrón con matlab

Para poder generar tráfico de forma sencilla se utiliza el modelo 2 de Gamma-Zipf propuesto.

Para simplificar el análisis vamos a utilizar solamente p=2, así los parámetros que definen nuestra Gamma-Zipf son:

- $\alpha = 1{,}15$
- a=.9211841260
- p=2
- x_{umbral}=1.945115316

que gráficamente se presenta en Fig A.6.

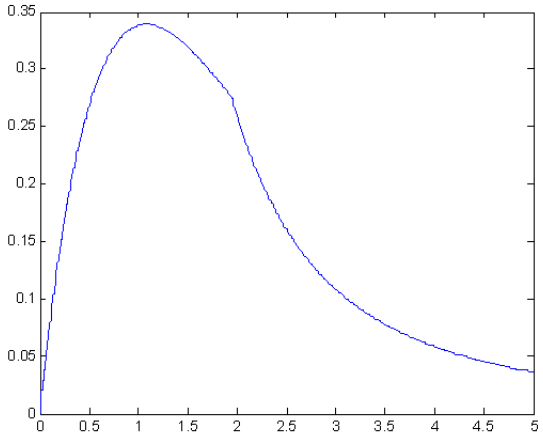


Figura A.6: Modelo definido a trozos para $\alpha = 1{,}15$ y p variable

Para generar tráfico se divide la gráfica en dos zonas y calculamos la probabilidad de estar en cada una de ellas:

- Zona1 = 0<x <x_{umbral}

$$P(zona1) = \int_0^{x_{umbral}} f_{gamma-zipf}(x)dx = 0{,}53472165$$

- Zona 2 x_{umbral}<x <∞

$$P(zona2) = \int_{x_{umbral}}^{\infty} f_{gamma-zipf}(x)dx = 0{,}46527835$$

Vamos a generar tráfico con una distribución Gamma(a,p) con P(Zona1) y tráfico con una distribución Zipf() con probabilidad P(Zona2). Además vamos a despreciar aquellos valores que devueltos por la distribución Gamma sean mayores que x_{umbral} y aquellos que devueltos por la Zipf sean menores que x_{umbral}.

El programa generado:

```
clear;
pzona1=.4926928425;
pzona2=1-pzona1;
a=.2296898071;
p=2.422038512;
c=5;
alpha=1.15;
xumbral=9.021029440;
n_muestras=5000;
trafico=zeros(1,n_muestras);
for i=1:n_muestras
   trafico(i)=rndgampareto(pzona1,pzona2,a,p,c,alpha,xumbral);
end
```

donde la función gama pareto se genera como:

```
function rand_number=rndgampareto(pzona1,pzona2,a,p,c,alpha,xumbral)
pzona=rand(1);
if pzona<=pzona1
rand_number=gamrnd(p,1/a);
while rand_number > xumbral
    rand_number=gamrnd(p,1/a);
end
else
rand_number=c/(rand(1))^(1/alpha);
while rand_number <= xumbral
    rand_number=c/(rand(1))^(1/alpha);
end
end
```

Apéndice B

GDOI

Group Domain Of Interpretation es un protocolo ISAKMP *Domain Of Interpretation* (DOI) para la gestión de claves de grupo. Gestiona asociaciones de seguridad de grupo (SA) que se usan en IPSec y otros posibles protocolos seguros sobre IP o sobre la capa de aplicación. Estas asociaciones de seguridad protegen una o más claves que cifran claves (KEKs), claves que cifran tráfico (SEKs) y datos compartidos por los miembros del grupo.

En este modelo de gestión de claves de grupo, el protocolo GDOI tiene lugar entre un miembro del grupo y el Controlador de Grupo/Servidor de claves (GCKS), que establece asociaciones de seguridad entre los participantes del grupo [RFC2401].

ISAKMP define dos fases de negociación [RFC2408]. El GDOI es un protocolo de la fase 2 protegido por un protocolo de fase 1. Éste puede ser cualquiera siempre que proporcione las siguientes protecciones:

- Autentificación punto-a-punto.
- Confidencialidad
- Integridad en los mensajes

Partiendo de la protección proporcionada por la SA de fase1, el protocolo GDOI define el intercambio de la fase 2 y propone nuevos *payloads* e intercambios de acuerdo al standard ISAKMP.

En GDOI se definen 6 nuevos *payloads*:

- GDOI SA.
- SA KEK (SAK) que sigue al payload SA.
- SA TEK (SAT) que sigue al payload SA.
- Vector de descarga (*download*) de claves (KD).
- Número de Secuencia (SEQ).
- Prueba de Posesión (POP).

Y dos nuevos intercambios:

- Un intercambio de la fase 2 que crea SAs de protocolos de *rekey* y de datos seguros.
- Un datagrama posterior que establece SAs de protocolos adicionales de Rekey y de datos seguros.

El nuevo intercambio de la fase 2, denominado GROUPKEY-PULL descarga claves para la SA *rekey* de grupos y SA de datos seguros. La SA *rekey* incluye las claves de cifrado de claves, KEKs; una SA de datos seguros incluye una clave de cifrado de datos, TEK, usada por protocolos de datos seguros para encriptar y desencriptar trafico de datos [RFC2407]. La SA para KEK o TEK incluye autentificación de claves, claves de cifrado, política criptogràfica y atributos. El intercambio GROUPKEY-PULL tiene un comportamiento *pull* ya que es el miembro el que inicia la obtención de las SAs del GCKS.

El datagrama GROUPKEY-PUSH se transmite des del GCKS hacia los integrantes del grupo para crear o actualizar SAs de *rekey* o de datos seguros. Una SA de *rekey* protege los mensajes GROUPKEY-PUSH. Por tanto, un GROUPKEY-PULL es necesario para establecer almenos una SA de *rekey* para proteger mensajes posteriores GROUPKEY-PUSH. El GCKS encripta los mensajes GROUPKEY-PUSH usando la SA de *rekey* KEK. El GDOI permite el uso de vectores de KEKs para algoritmos de gestión de claves de grupo usando LKH.

Aunque el GROUPKEY-PUSH se puede usar para refrescar la SA de *rekey*, el uso más común es el de establecer SAs de datos seguros para protocolos de seguridad de datos. El GDOI permite futuras extensiones para soportar varios protocolos de seguridad de datos, aunque actualmente sólo suporta IPSec ESP. Un protocolo de seguridad de datos utilizará el TEK y tiene una SA de datos seguras, por tanto, en el GDOI, IPSec ESP usará el TEK.

No todas las aplicaciones de multicast seguro o multimedia pueden usar IPSec ESP. Muchas aplicacions de Real Time Transport Protocol, por ejemplo, requieren seguridad por encima de la capa IP para preservar la eficiencia de la compresión en la cabecera RTP y independencia en el transporte [RFC1889]. El futuro protocolo de seguridad RTP se podrá beneficiar del uso del GDOI para establecer las SA de grupo para los servicios de seguridad en multicast y unicast.

Así, el GDOI es un protocolo de gestión de asociaciones de seguridad de grupo. Todos los mensajes son para crear, mantener, o revocar asociaciones de seguridad para un grupo.

B.1. Intercambio *GroupKey-Pull*

El objectivo del GROUPKEY-PULL es establecer unas SAs Rekey y de Datos Seguros en un miembro para un grupo concreto. La SA de la fase 1 protege el intercambio GROUPKEY-PULL; puede haber múltiples intercambios GROUPKEY-PULL en una SA de fase 1, aunque sólo la primera es imprescindible. Este intercambio descarga las claves de datos seguros (TEKs) y las claves de cifrado de claves de grupo (KEK) o vector KEK bajo la protección de la SA de la fase 1.

Autorización

Existen dos alternativas para autorizar los mensajes GROUPKEY-PULL. En primer lugar, se puede usar la identidad de la fase 1 para autorizar la petición de la fase 2 (GROUPKEY-

PULL) por unas claves de grupo. Como segunda opción, se puede pasar una nueva identidad en las peticiones GROUPKEY-PULL; la nueva identidad podria ser específica del grup y usar un certificado que se firmaría por el propio grupo para identificar al propietario como miembro autorizado del grup. El payload de Prueba de Posesión valida que el titular posee la clave secreta asociada con la identidad de la fase 2.

Mensajes

GROUPKEY-PULL es un intercambio de la fase 2. La fase 1 calcula SKEYID_a [RFC2409] que es la clave en los payloads GROUPKEY-PULL HASH. Como el payload IKE HASH [RFC2409], cada mensaje GROUPKEY-PULL genera un único grupo de valores. Una marca del tiempo (Nonce) permuta con el HASH y provee protección contra ataques de repetición. Esta protección es importante para proteger el GCKS de ataques a los que es susceptible el servidor de claves.

Este intercambio usa la marca de tiempo actual (Nonce) para garantizar que el miembro esté activo, o contra repeticiones de mensajes recientes GROUPKEY-PULL. El ataque por repetición sólo es posible en el contexto de la fase 1. Si un mensaje GROUPKEY-PULL se repite basandose en una fase 1 anterior, el cálculo de HASH fallará por un SKEYID_a equivocado; el mensaje falla en el proceso antes que el payload Nonce sea evaluado. PPara que cualquier extremo consiga la protección contra repeticiones, se pospone el proceso tanto com sea posible hasta que se recibe el mensaje en el protocolo que prueba que el otro extremo está activo. Por ejemplo, el GCKS no instala nuevas SAs hasta que recibe un mensaje con Nr incluido de forma correcta en el HASH.

El Nonce requiere un mensaje adicional en el intercambio para asegurar que el GCKS no añade un miembre nuevo hasta que este pruebe su estado activo. El miembro GROUPKEY-PULL espera encontrar el payload Nonce, Ni, en el HASH del mensaje de respuesta. Y el GCKS GROUPKEY-PULL espera ver su Nonce, Nr, en el HASH del mensaje de respuesta antes de dar el material de claves de grup en el siguiente intercambio.

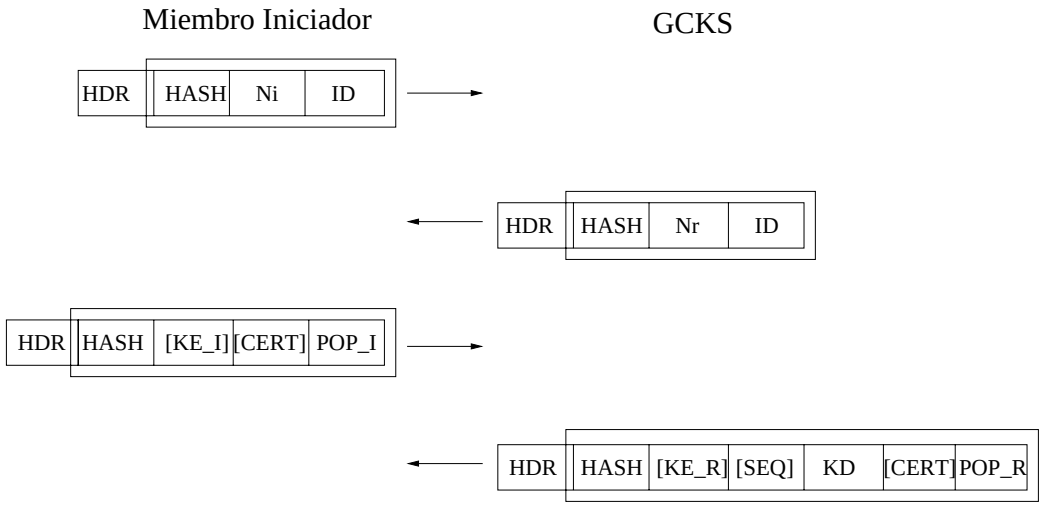


Figura B.1: Intercambio de mensajes fase 1 GDOI

Las funciones de hash se calculan de la siguiente manera (prf significa pseudo random

function):

HASH (1) = prf (SKEYID_a, M-ID | Ni | ID)
HASH (2) = prf (SKEYID_a, M-ID | Ni_b | Nr | SA)
HASH (3) = prf (SKEYID_a, M-ID | Ni_b | Nr_b [| KE_I] [| CERT] [| POP_I])
HASH (4) = prf (SKEYID_a, M-ID | Ni_b | Nr_b [| KE_R] [| SEQ |] KD [| CERT] [| POP_R])

En los mensajes de la FigB.1 HDR está protegido por la SA de la fase 1, se encripta después de HDR. HDR es una cabecera ISAKMP que usa los cookies de la fase1 y un identificador de mensaje (M-ID) como en IKE [RFC2409]. El payload Nonce está incluido en los primeros dos intercambios, el GCKS devuelve sólo el payload de la política SA antes que se pruebe que el miembro está activo. Los payloads HASH [RFC2409] prueban que la pareja tiene el secreto de la fase 1 (SKEYID_a) y el Nonce para el intercambio identificado con el identificador de mensaje (M-ID). Una vez se ha demostrado la actividad de los comunicantes, el último mensaje completa el proceso descargando el payload KD.

Además de los payloads Nonce y HASH, el miembro iniciador identifica el grupo al cual desea unirse con el payload ISAKMP ID. El GCKS responde informando al miembro del valor actual del número de secuencia en el payload SEQ; el número de secuencia ordena los datagramas GROUPKEY-PUSH; el miembro comprueba que el número de secuencia es mayor que en un campo SEQ previo antes de instalar qualquier SA nueva. El payload SEQ siempre está presente si el payload SA contiene el atributo SA KEK. El GCKS informa al miembro de las políticas criptográficas del grupo en el payload SA, donde se describen el DOI, el material KEK y TEK, y las transformaciones de autentificación.

Los SPIs también son determinantes para el GCKS y se descargan en la cadena de payloads SA. El atributo SA KEK contiene el par cookie ISAKMP para la SA de Rekey. El atributo SA TEK contiene un SPI. El segundo mensaje descarga este payload SA. Si el payload SA define un SA Rekey, entonces el payload KD contiene el KEK; y si SA define una o más SAs de Datos Seguros, entonces el payload KD contendrá los TEKs correspondientes. Este sistema es útil en el caso que haya un grupo inicial de TEKs para un grup particular y se pueda evitar la necesidad de mensajes GROUPKEY-PUSH futuros.

El participante puede establecer una identidad en el intercambio GROUPKEY-PULL en el payload opcional CERT que no será la misma que la identidad de la fase 1. Cuando el miembro pasa un CERT nuevo, el payload POP lo acompaña. Este demuestra que el participante o GCKS ha usado el mismo secreto que lo autentica.

POP_I es un payload ISAKMP SIG que contiene un hash que incluye los Ni y Nr firmados por el miembro, cuando este pasa un CERT, firmado por el grupo para probar su autorización. POP_R contiene un hash que incluye los Ni y Nr firmados por el GCKS, cuando el GCKS pasa un CERT, firmado por el grupo, para probar su autoridad para proveer claves a un grup particular. El uso del par de nonces para el payload POP, transformados a través de una función pseudo-aleataria (prf) y cifrada, se designa para mantener el compromiso con la fase 1. La implementación de los payloads CERT y POP es opcional.

Perfect Forward Secrecy

Si es desea PFS y se usa el payload opcional KE en el intercambio, entonces los dos extremos calculan el número secreto DH (Diffie-Helman) y lo usan para proteger el nuevo material de claves contenido en el KD. El GCKS harà una XOR con el número DH y el KD, y envía el resultado al miembro iniciador. Este recupera el payload KD repitiendo la operación [RFC2412]. La implementación del payload KE es opcional.

Inicialización de la Cabecera ISAKMP.

Los cookies se usan en la cabecera ISAKMP como una forma débil de negación de protección. El intercambio GROUPKEY-PULL usa los cookies de acuerdo con ISAKMP [RFC2408].El campo Sigüiente Payload identifica un payload ISAKMP o GDOI. La versión va desde la 0 (Minor Version) a la 1 (Major Version) de acuerdo con ISAKMP [RFC2408]. El tipos de intercambio tiebe el valor 32 para el intercambio GROUPKEY-PULL GDOI. Los Flags, ID del mensaje, y Longitud están según ISAKMP [RFC2408].

B.2. Mensaje *GroupKey-Push*

El protocolo GDOI envía información de control de forma segura usando comunicaciones de grupo. Típicamente utiliza IP multicast para distribuir los mensajes GROUPKEY-PUSH pero también se puede usar unicast si no es posible. Los mensajes GROUPKEY-PUSH sustituyen los vectores de claves KEK, y crean nuevas SAs de Datos Seguros.

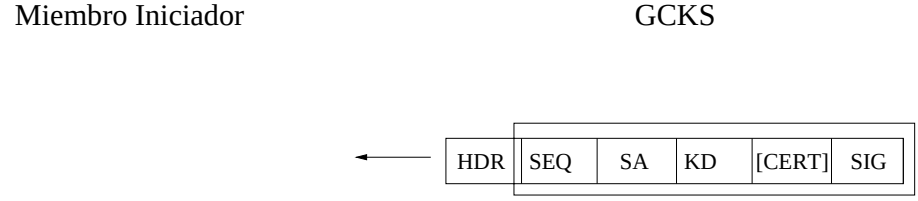


Figura B.2: Intercambio de mensajes fase 2 GDOI

HDR está protegido por la SA KEK, se encripta después de HDR.

El payload SEQ prove el número secuencial de mensajes GROUPKEY-PUSH. El payload SA define la política y los atributos para las SAs Rekey y de Datos Seguros. El GCKS o opcionalment un delegado puede proveer un payload CERT para verificación de SIG. El payload KD (Key Download) contiene todas las claves necesarias.

El payload SIG es una firma de un hash del mensaje antes de cifrarse (incluye la cabecera y excluye el SIG), prefijado con el string rekey. El string prefijado asegura que la firma del datagrama Rekey no pueda ser usado para otro propósito en el protocolo GDOI.

Si la SA define un vector LKH KEK o un único KEK, KD contiene un KEK o un vector KEK para nuevas SA Rekey, que tendrán un nuevo par de cookies. En cuanto el payload KD con nuevos atributos SA KEK, una SA Rekey se reemplaza con una nueva SA que tiene el mismo identificador de grupo (ID especificada en el primer mensaje del intercambio GROUPKEY-PULL) e incrementa el mismo contador secuencial, que se inicializa en el

cuarto mensaje del GROUPKEY-PULL. Si la SA define un payload SA TEK, este informa al participante que una nueva SA de Datos Seguros ha sido creada, con el material de claves en el payload KD.

Perfect Forward Secrecy.

El mensaje GROUPKEY-PUSH está protegido por las KEK de grupo. El mensaje contiene las nuevas claves entre otra información. Las claves generadas protegen les claves para los mensajes GROUPKEY-PUSH para tener PFS.

Forward i Backward Access Control

Con los intercambios GROUPKEY-PUSH, el protocolo GDOI soporta algoritmos como LKH que tienen la propiedad de denegar el acceso a una nueva clave de grupo a un miembro revocado del grupo (*Forward Access Control*) y de una clave vieja del grupo a un miembro nuevo del grup (*Backward Access Control*). Estos mecanismos permiten prestar los servicios de *Perfect Forward Security* y *Perfect Backward Security* [RFC2627].

A parte de LKH, otro algoritmos de gestión del grupo con control de acceso también se pueden usae con el GDOI, OFT o *Subset Difference* son ejemplos de ellos. La elección del algoritmo de gestión de grupos se realiza mediante el atributo KEK_MANAGEMENT_ALGORITHM, que se transmite en el payload SA KEK.

Requisitos del Forward Access Control.

Cuando los integrantes de un grupo cambian, usualmente, el algoritmo de gestión de grupo necesita nuevas SA TEKs (y sus claves asociadas). Las nuevas SAs aseguran que los miembros a quien se ha denegado el acceso no puedan participar en el grupo.

Si se desea *Forward Access Control*, las nuevas SA TEKs y sus paquetes de claves en el payload KD no van incluidos en el mensaje GROUPKEY-PUSH que contiene los cambios para el nuevo grupo. Esto se explica porque la política SA TEK y los paquetes de claves asociados en el KD no están protegidos con la nueva KEK. Un segundo mensaje GROUPKEY-PUSH puede entregar los nuevos SA TEK y sus claves porque ya están protegidos con la nueva KEK y no son visibles por los miembros a quien se ha denegado el acceso.

Delegación de la Gestión de Claves.

El protocolo GDOI soporta la delegación de los datagramas GROUPKEY-PUSH con las capacidades de delegación de la PKI. GDOI no especifica como el GCKS identifica a los delegados.

Uso de Claves Firmadas.

El GCKS normalmente no usa la misma clave para firmar el payload SIG en el mensaje GROUPKEY-PUSH y para la autorización en el POP de GROUPKEY-PULL. Si se usa la misma clave, se deben utilizar funciones resumen diferentes como base para el payload POP y para el payload SIG.

Inicialización de la Cabecera ISAKMP.

A diferencia de ISAKMP o IKE, el par de cookies se determina por el GCKS. El par de cookies en la cabecera ISAKMP GDOI identifica la SA Rekey para diferenciar los diferentes grupos seguros gestionados por GCKS. Así el GDOI usa los campos cookies como SPI.

El Siguiente Payload identifica un payload ISAKMP o GDOI. La versión va des de la 0 (Minor Version) a la 1 (Major Version) según ISAKMP [RFC2408]. El Tipo de Intercambio tiene el valor 33 para el mensaje GROUPKEY-PUSH GDOI. Los Flags tienen activado el bit Encryption según RFC2008. Todos los demás bits están a 0. La ID del mensaje está a 0. La Longitud está según ISAKMP [RFC2408].

Supresión de SAs.

Algunas veces el GCKS puede querer indicar a los receptores que supriman SAs, por ejemplo al final de un broadcast. La supresión de claves se consigue enviando un payload ISAKMP Delete [RFC2408] como parte del mensaje GROUPKEY-PUSH.

Los payloads Delete (uno o más) se situan a continuación del payload SEQ en el mensaje GROUPKEY-PUSH. Si el GCKS no ha de enviar otras SAs a los miembros del grupo, los campos SA y KD no forman parte del mensaje.

Los campos en el payload Delete se definen a continuación:

- El campo Domain Of Interpretation contiene el DOI GDOI.
- El campo ID del protocolo contiene la ID del protocol TEK. Para suprimir una SA KEK, se debe poner a 0 el campo ID del protocolo. Como sólo se puede poner una ID de protocolo en un payload Delete, si se quiere suprimir una SA TEK y una SA KEK, se deben enviar en diferentes payloads Delete.

B.3. Campos y definición de valores

Algunos de los payloads ISAKMP definidos en concordancia con RFC2408, que seran detallats més endavant.

Tabla B.1: Valores del Tipo de Siguiente Payload

Tipo de Siguiente Payload	Valor
Asociación de Seguridad SA	1
Identificación ID	5
Nonce N	10

Se requieren varios formatos de payloads nuevos en los intercambios seguros del grupo. Los tipos de payloads para las nuevas cabeceras están definidos en el rango Private Use de ISAKMP.

Tabla B.2: Nuevos valores del Tipo de siguiente Payload en el rango de uso privado de ISAKMP

Tipo de Siguiente Payload	Valor
RESERVADO	128 - 129
SA KEK (SAK)	130
SA TEK (SAT)	131
Key Download (KD)	132
Número de secuencia (SEQ)	133
Prueba de Posesión (POP)	134
RESERVADO	135-200
Uso privado GDOI	201-255

Payload Identificación (ID).

Se usa paar identificar un grupo que tendra asociadas varias Asociaciones de Seguridad. Una identidad de grupo mapea una IP multicast específica, o especifica un identificador más general (como una representación de un conjunto de streams multicast).

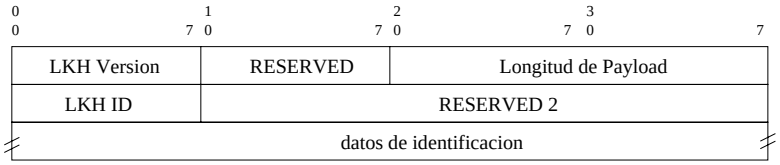


Figura B.3: Formato del Payload de Identificación

Los campos del payload de Identificación se definen a continuación:

- Siguiente Payload (1 byte) : Identificador del siguiente tipo de payload en el mensaje. Si el payload actual es el último del mensaje, este campo será 0.
- RESERVADO (1 byte) : No usado, ha de ser 0.
- Longitud del Payload (2 bytes) : Longitud, en bytes, de los datos de identificación, incluyendo la cabecera genérica.
- Tipo de Identificación (1 byte) : Valor que describe la información de identidad en el campo de Datos de identificación.
- RESERVADO2 (3 bytes) : No usado, ha de ser 0.
- Datos de Identificación (Longitud Variable) : Valor identificado por el campo Tipo de Identificación.

Valors del Tipo de Identificación.

La sieguiente tabla lista los valores asignados por el Tipo de Identificación en el Payload de Identificación.

En el contexto del campo GDOI ID, ID_KEY_ID especifica un identificador de grupo de 4 bytes.

Tabla B.3: Valores del tipo de identificación

Tipo de ID	Valor
RESERVADO	0-10
ID_KEY_ID	11
RESERVADO	12-127
Uso Privado	128-255

Payload Asociación de Seguridad (SA).

Este payload está descrito en el RFC2408. En el GDOI, el GCKS lo usa para definir los atributos de seguridad de las SAs de Rekey y de Datos Seguros.

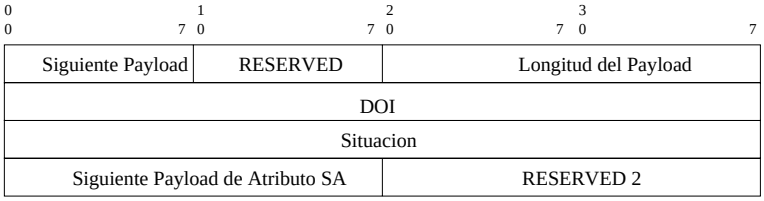


Figura B.4: Formato del Payload de SA

Los campos del payload de la Asociación de Seguridad se definen a continuación:

- Siguiente Payload (1 byte) : Identifica el seguiente payload en el mensaje GROUPKEY-PULL o GROUPKEY-PUSH. El siguiente payload no debe ser de tipo SAK o SAT, sino de un tipo que no sea SA.
- RESERVADO (1 byte) : Ha de ser 0.
- Longitud del Payload (2 bytes) : Es la longitud en bytes del campo actual incluyendo la cabecera genérica y todos los campos TEK y KEK.
- DOI (4 bytes) : Es el GDOI, que es 196 pendiente de asignación por la IANA.
- Situación (4 bytes) : Ha de ser 0.
- Siguiente Payload de Atributo SA (1 byte) : Ha de ser o SAK o SAT.
- RESERVADO2 (2 bytes) : Ha de ser 0.

Payloads que siguen al payload SA.

El payload SA debe ir seguido por payloads que definen los atributos específicos de la asociación de seguridad para la KEK y/o TEKs del grupo. La cantidad dependerá de la política de grupo. Puede haber 0 o un payload SAK y 0 o más payloads SAT, pero en total debe haber como mínimo un payload SAK o SAT.

Este método permite soportar varias políticas de grupo. Por ejemplo, si la política del grupo no requiere SA Rekey, el GCKS no necesita enviar los atributs SA KEK a los miembros,

ya que todas las actualizaciones de SA se realizarían usando la SA de registro (unicast). Alternativamente, se podria usar la SA Rekey pero escogiendo guardar una KEK a un miembro sólo como parte del registro. Por tanto, la política KEK (en el atributo SA KEK) no es necesaria como parte integrante del payload SA del mensaje Rekey.

Especificar múltiples SATs permite múltiples sesiones en un mismo grupo multicast y múltiples streams asociados a una sesión (vídeo, audio, texto), cada uno con política de asociación de seguridad individual.

Payload SA KEK.

El payload SA KEK (SAK) contiene los atributos de seguridad según el metodo KEK para un grupo y los parámetros específicos para el intercambio GROUPKEY-PULL. Las identidades de la fuente y el destino descriven las identidades que usa el datagrama GROUPKEY-PULL.

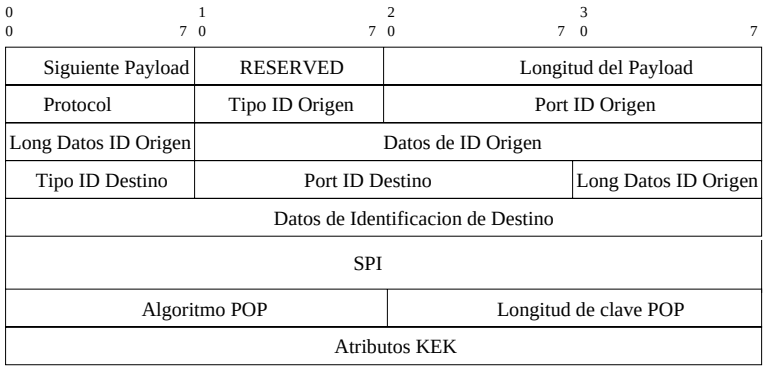


Figura B.5: Formato del Payload de SA KEK

Los campos del payload SAK se definen a continuación:

- Siguiente Payload (1 byte) : Identifica el seguiente payload en el mensaje GROUPKEY-PULL o GROUPKEY-PUSH. Los únicos tipos de siguientes payloads válidos son SAT o 0 para indicar que no es un campo SA TEK.
- RESERVADO (1 byte) : Ha de ser 0.
- Longitud de la Payload (2 bytes) : Longitud de la Payload, incluyendo los atributos KEK.
- Protocolo (1 byte) : Valor que describe la ID de un protocol IP (TCP/UDP) para datagramas rekey.
- Tipus de ID de Origen (1 byte) : Describe la información que se encuentra en el campo de datos de identidad de la fuente. Los valores definidos están especificados en la sección IPSec Identification Type de IANA isakmpd-registry.
- Puerto de ID de Origen (2 bytes) : Especifica el puerto asociado con la identidad de la fuente. El valor 0 significa que este campo se ha de ignorar.
- Longitud de los Datos de ID del Origen (1 byte)

- Datos de ID del Origen (variable) : Valor indicado por el Tipo de ID de Origen.
- Tipo de ID de Destino (1 byte) : Describe la información del campo de datos de identidad del destino. Los valores definidos están especificados en la sección IPSec Identification Type de IANA isakmpd-registry.
- Puerto de ID de Destino (2 bytes) : Especifica el puerto asociado con la identidad de la fuente. El valor 0 significa que este campo se ha de ignorar.
- Longitud de les Datos de ID de Destino (1 byte)
- Datos de ID de Destino (variable) : Valor indicado por el Tipo ID del Destino.
- SPI (16 bytes) : Índice de los Parámetros de Seguridad para la KEK. El SPI ha de ser el par de cookies de la cabecera ISAKMP, donde los primeros 8 bytes serán el campo Initiator Cookie del HDR ISAKMP del mensaje GROUPKEY-PUSH, y los segundos 8 bytes el Responder Cookie en la misma cabecera. Estos cookies son asignados por el GCKS.
- Algoritmo POP (2 bytes) : Los valores definidos están especificados en la tabla. Si no se quiere especificar ningún algoritmo POP, el campo ha de ser 0.
- Longitud de la Clave POP (2 bytes) : Longitud del Payload de la Clave POP. Si no se especifica ningún algoritmo en el campo Algoritmo POP en la política KEK, este campo ha de ser 0.
- Atributos KEK : Contiene los atributos de la política KEK asociados con el grupo. Algunos o todos los atributos son opcionales dependiendo de la política.

Tabla B.4: Valores del tipo de algoritmo del payload POP

Tipo de Algoritmo	Valor
RESERVADO	0
POP_ALG_RSA	1
POP_ALG_DSS	2
POP_ALG_ECDSS	3
RESERVADO	4-127
Uso privado	128-255

Atributos KEK.

Los siguientes atributos deben estar presentes en el payload SAK y han de seguir el formato establecido en ISAKMP [RFC2408]. En la tabla, los atributos TV están marcados como Básics (B) con longitud constante y los atributos TLV como Variable (V).

Los atributos KEK_MANAGEMENT_ALGORITHM y KE_OAKLEY_GROUP sólo han de formar parte de los mensajes GROUPKEY-PULL.

Tabla B.5: Valores de Identificación y Tipo de atributos KEK

Id de Calse	Valor	Tipo
RESERVADO	0	
KEK_MANAGEMENT_ALGORITHM	1	B
KEK_ALGORITHM	2	B
KEK_KEY_LENGTH	3	B
KEK_KEY_LIFETIME	4	V
SIG_HASH_ALGORITHM	5	B
SIG_ALGORITHM	6	B
SIG_KEY_LENGTH	7	B
KE_OAKLEY_GROUP	8	B

Tabla B.6: Valores del tipo de gestión KEK

Tipo de gestión KEK	Valor
RESERVADO	0
LKH	1
RESERVADO	2-127
Uso privado	128-255

La clase KEK_MANAGEMENT_ALGORITHM especifica el algoritmo de gestión del grup KEK para proporcionar Control de Acceso Backward y Forward.

La clase KEK_ALGORITHM especifica el algoritmo de cifrado que se usa en la KEK.

Tabla B.7: Valores del tipo de algortimo de cifrado para las KEK

Tipo de algoritmo	Valor
RESERVADO	0
KEK_ALG_DES	1
KEK_ALG_3DES	2
KEK_ALG_AES	3
RESERVADO	4-127
Uso privado	128-255

Si se utiliza un KEK_MANAGEMENT_ALGORITHM que define múltiples claves, y el algoritmo de gestión no especifica el algoritme de cifrado para estas claves, entonces el algoritmo definido por el KEK_ALGORITHM se usa por todas las claves de gestión.

El algoritmo KEK_ALG_DES especifica el DES usando el mode de cifrado por bloques o Cipher Block Chaining (CBC). El algoritmo KEK_ALG_3DES especifica el 3DES usando tres claves independientes. El algoritme KEK_ALG_AES especifica el AES. El modo de operación de AES es CBC. Una implementación práctica del protocol GDOI debe soportar el atributo de algoritmo KEK_ALG_3DES.

La clase KEK_KEY_LENGTH es la longitud de la clave del algoritme KEK (en bits).

La clase KEK_KEY_LIFETIME especifica el máximo tiempo que una KEK es válida.

El GCKS refresca la KEK antes del final del periodo. El valor es un número de cuatro bytes que define un periode de tiempo en segundos.

El SIG_HASH_ALGORITHM especifica el algoritmo hash del campo SIG.

Tabla B.8: Valores del tipo de algortimo hash de la firma

Tipo de algoritmo	Valor
RESERVADO	0
SIG_HASH_MD5	1
SIG_HASH_SHA1	2
RESERVADO	3-127
Uso privado	128-255

El SIG_HASH_ALGORITHM no es necesario si el SIG_ALGORITHM es SIG_ALG_DSS o SIG_ALG_ECDSS porque implican que es SIG_HASH_SHA1.

La clase SIG_ALGORITHM especifica el algoritmo de firma en el payload SIG.

Tabla B.9: Valores del tipo de algortimo de la firma

Tipo de algoritmo	Valor
RESERVADO	0
SIG_ALG_RSA	1
SIG_ALG_DSS	2
SIG_ALG_ECDSS	3
RESERVADO	4-127
Uso privado	128-255

El SIG_ALG_RSA especifica el algoritmo de la firma digital RSA. El SIG_ALG_DSS especifica el DSS. El SIG_ALG_ECDSS especifica el algoritmo de la firma digital de Curva Elíptica. Una implementación práctica del protocol GDOI debe soportar el atributo de algoritmo SIG_ALG_RSA.

La clase SIG_KEY_LENGTH es la longitud de la clave del campo SIG.

La clase KE_OAKLEY_GROUP define el grupo OAKLEY que se usa per computar el PFS en el campo opcional KE del intercambio GDOI GROUPKEY-PULL.

Payload SA TEK.

El payload SA TEK (SAT) contiene los atributos de seguridad de una única TEK asociada a un grupo.

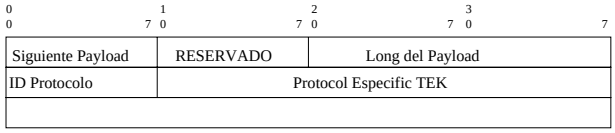


Figura B.6: Formato del Payload de SA TEK

Los campos del payload SAT se definen a continuación:

- Siguiente Payload (1 byte) : Identifica el siguiente payload en los mensajes GROUPKEY-PULL y GROUPKEY-PUSH. Los únicos tipos de payloads válidos son SAT o 0 para indicar que no hay más atributos de asociaciones de seguridad.
- RESERVADO (1 byte) : Ha de ser 0.
- Longitud de Payload (2 bytes) : Longitud del SAT incluyendo el campo Protocol-Especific TEK.
- ID Protocol (1 byte) : Valor que especifica el Protocolo de Seguridad.
- Protocol-Especific TEK (Variable) : Campo que describe los atributos específicos del protocolo de seguridad definido en ID Protocol.

Tabla B.10: Valores de las identidades de los protocolos de seguridad

Tipo de algoritmo	Valor
RESERVADO	0
GDOI_PROTO_IPSEC_ESP	1
RESERVADO	2-127
Uso privado	128-255

Protocol PROTO_IPSEC_ESP.

El payload Protocol-Especific TEK para ESP es:

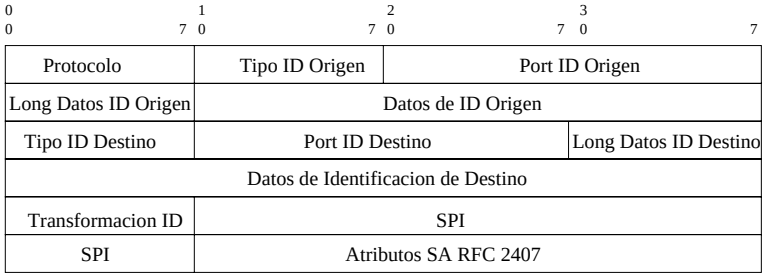


Figura B.7: Formato del Payload protocolo específico TEK para ESP

Los campos del payload SAT se definen a continuación:

- Protocol (1 byte) : Valor que describe una ID de un protocol IP (TCP/UDP). El valor 0 indica que este campo se ignora.
- Tipo de ID del Origen (1 byte) : Describe la información que se encuentra en el campo de dato de identidad de la fuente. Los valores definidos están especificados en la sección IPSec Identification Type de IANA isakmpd-registry.

- Port ID del Origen (2 bytes) : Especifica el puerto asociado con la identidad de la fuente. El valor 0 significa que este campo se ha de ignorar.
- Longitud de los Datos de ID de Origen (1 byte)
- Datos ID de Origen (variable) : Valor indicado por el Tipo de ID de Origen. Para grupos multicast con múltiples fuentes que usan una TEK comuna se fijan tres bytes a cero.
- Tipo ID de Destí (1 byte) : Describe la información que hay en el campo de datos de identidad de destino. Los valores definidos están especificados en la sección IPSec Identification Data de IANA isakmpd-registry.
- Port ID de Destino (2 bytes) : Especifica el puerto asociado con la identidad de la fuente. El valor 0 significa que este campo se ha de ignorar.
- Longitud de los Datos de ID de Destino (1 byte)
- Datos de ID de Destino (variable) : Valor indicado por Tipo ID de Destino.
- Transformación ID (1 byte) : Valor que especifica qué transformación ESP se utiliza. La lista de valores válidos está definida en la sección IPSec ESP Transform Identifiers de IANA isakmpd-registry.
- SPI (4 bytes) : Índice de Parámetros de Seguridad para ESP.
- Atributos RFC2407 : Atributos ESP de RFC2407. El GDOI soporta todos los Atributs SA de IPSec DOI para PROTO_IPSEC_ESP, excluyendo la Descripción de Grupo [RFC2407], que en las implementaciones de GDOI no se envía y se ignora cuando se recibe. Todos los atributos obligatorios en IPSec DOI son obligatorios en GDOI PROTO_IPSEC_ESP. En el GDOI el atributo Algoritmo de Autentificación del IPSec DOI es la autentificación de grup.

Otros Protocolos de Seguridad.

Además del protocolo ESP, el GDOI también sirve para establecer SAs para grupos seguros necesarias para otros Protocolos de Seguridad que operan en las capas de transporte, aplicación y Internet. Estos otros Protocolos de Seguridad están en proceso de desarrollo o no existen.

Todos los Protocolos de Seguridad han de proporcionar información para que se pueda dirigir la especificación del GDOI para este protocolo. Para un Protocolo de Seguridad se ha de proporcionar al GDOI:

- La ID del Protocolo para un Protocolo de Seguridad particular.
- El Tamaño del SPI.
- El método de generación del SPI.
- Las transformaciones, atributos y claves necesarios para el Protocolo de Seguridad.

Un Protocolo de Seguridad puede proteger el tráfico de la red a cualquier nivel, pero en qualquier caso debe proteger el tráfico que se comparta entre más de dos entidades.

Payload Descarga de Claves (KD).

Este payload contiene las claves de grupo para el grupo especificado en el payload SA. Puede tener varios atributos de seguridad que están basados en la política de seguridad del grupo definido en el payload SA.

Cuando se incluyen como parte de la SA rekey con el payload opcional KE, el payload KD realiza una xor con el nuevo número DH. La operación xor empieza en el campo Número de Paquetes de Claves.

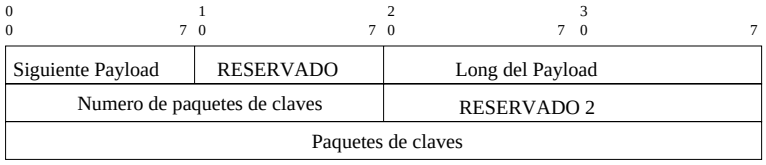


Figura B.8: Formato del Payload KD

Los campos del payload KD se definen a continuación:

- Siguiente Payload (1 byte): Identificador del tipo de payload del siguiente payload en el mensaje. Si el actual es el último, este campo es 0.
- RESERVADO (1 byte) : Ha de ser 0.
- Longitud de Payload (2 bytes) : Longitud en bytes del payload actual incluyendo la cabecera genérica.
- Número de Paquetes de Claves (2 bytes) : Contiene el número total de TEKs y vectores rekey en este bloque de datos.

Paquetes de Claves

Se ha definido algunos tipos de paquetes de claves. Cada uno tiene el siguiente formato.

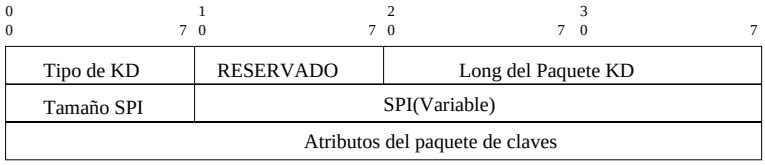


Figura B.9: Formato del paquete de claves del Payload KD

- Tipo de KD (1 byte) : Identificador para los datos de las claves en este paquete de claves.
- RESERVADO (1 byte) : Ha de ser 0.
- Longitud del KD (2 bytes) : Longitud en bytes de los datos KD incluyendo la cabecera del Paquete de Claves.
- Tamaño SPI (1 byte) : Especifica la longitud en bytes del SPI.

- SPI (Variable) : Índice de Parámetros de Seguridad en concordancia con el SPI enviado previamente en el payload SAK o SAT.
- Atributo del Paquete de Claves (Variable) : Contiene información de Claves. El formato de este campo es específico del valor del campo Tipo de KD.

Tabla B.11: Valores del tipo de paquetes KD

Tipo de KD	Valor
RESERVADO	0
TEK	1
KEK	2
LKH[1]	3
RESERVADO	4-127
Uso privado	128-255

Paquete TEK.

Los atributos de la tabla han de formar parte del paquete TEK. Exactamente uno por cada tipo enviado en el payload SAT. Estos atributos siguen el formato especificado en ISAKMP [RFC2408].

Tabla B.12: Valores de identificación y tipo de atributos de los paquetes TEK

Id de Clase	Valor	Tipo
RESERVADO	0	
TEK_ALGORITHM_KEY	1	V
TEK_INTEGRITY_KEY	2	V
TEK_SOURCE_AUTH_KEY	3	V

Si no hay paquetes TEK en KD, los miembros del grupo pueden esperar a recibir las TEKs como parte de la SA rekey. Como mínimo se debe incluir una TEK en cada payload KD de rekey. Se pueden incluir múltiples TEKs si se han de actualizar diversos streams asociados a la SA.

La clase TEK_ALGORITHM_KEY declara que la clave de cifrado para esta SPI está contenida en este atributo. El algoritmo de cifrado que se ha de usar para esta clave está especificado en el payload SAT. En el caso que el algoritmo requiera múltiples claves (3DES), todas las claves se incluyen en un atributo.

Las claves DES son de 64 bits (56 bits más paridad). Les claves del triple DES se especifican en un atributo de 192 bits (incluyendo paridad) en el orden que se usan para cifrar (DES_KEY1, DES_KEY2, DES_KEY3).

La clase TEK_INTEGRITY_KEY declara que la clave de integridad para este SPI está contenida en este atributo. El algoritmo de integridad es el especificado en el payload SAT. Así el GDOI asume que la clave simétrica de cifrado y la clave de integridad son dadas al miembro. Las claves SHA consisten en 160 bits y las MD5 son de 128 bits.

La clase TEK_SOURCE_AUTH_KEY declara que la clave de autentificación de la fuente para este SPI está contenida en este atributo. El algoritmo que se usa con esta clave es el especificado en el payload SAT.

Paquete KEK.

Los atributos de la tabla han de formar parte del paquete KEK. Exactamente uno por cada tipo enviado en el payload SAK. Estos atributos siguen el formato especificado en ISAKMP.

Tabla B.13: Valores de identificación y tipo de atributos de los paquetes KEK

Id de Clase	Valor	Tipo
RESERVADO	0	
KEK_ALGORITHM_KEY	1	V
SIG_ALGORITHM_KEY	2	V

Si se incluye el paquete de claves KEK en el payload KD, almenos habráde existir uno de los atributos.

La clase KEK_ALGORITHM_KEY declara la clave de cifrado para este SPI contenida en este atributo. El algoritmo de cifrado que usa esta clave está especificado en el payload SAK.

Si el modo de operación para el algoritmo requiere un Vector de Inicialización (IV), este se ha de incluir en el KEK_ALGORITHM_KEY antes de la clave actual.

La clase SIG_ALGORITHM_KEY declara que la clave pública para este SPI está contenida en este atributo. Podria ser útil cuando no hay una infraestructura de clave pública disponible. El algoritmo para esta clave está especificado en la SAK.

Paquete LKH.

El paquete de claves LKH es un conjunto de atributos que representan los diferentes eventos en el árbol LKH. Los atributos de la tabla se utilizan para pasar un vector LKH KEK en el payload KD. Estos atributos siguen el formato especificado en ISAKMP.

Tabla B.14: Valores de identificación y tipo de atributos de los paquetes LKH

Id de Clase	Valor	Tipo
RESERVADO	0	
LKH_DOWNLOAD_ARRAY	1	V
LKH_UPDATE_ARRAY	2	V
SIF_ALGORITHM_KEY	3	V
RESERVADO	4-127	
Uso privado	128-255	

Si se incluye el paquete de claves LKH en el payload KD, almenos tendrá que haber uno de los atributos.

LKH_DOWNLOAD_ARRAY

Este atributo se usa para descargar un conjunto de claves a un miembro del grupo. No se ha de incluir en el payload KD del mensaje GROUPKEY-PUSH si se envía a más de un miembro.

Este atributo consiste en una cabecera seguida de una o más claves LKH.

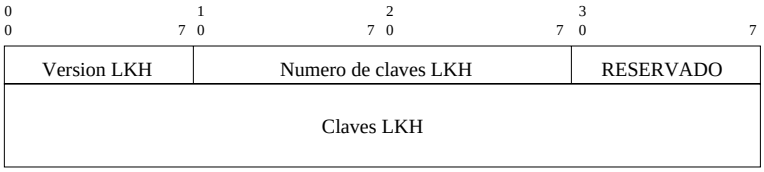


Figura B.10: Formato del atributo LKH_DOWNLOAD_ARRAY de un paquete LKH KD

Los campos del atributo LKH_DOWNLOAD_ARRAY se definen a continuación:

- Versión LKH (1 byte) : Contiene la versión del protocolo LKH. Ha de ser 1.
- Número de Claves LKH (2 bytes) : Es el número de claves en esta secuencia.
- RESERVADO (1 byte) : Ha de ser 0.

Claves LKH

Cada clave se define como sigue:

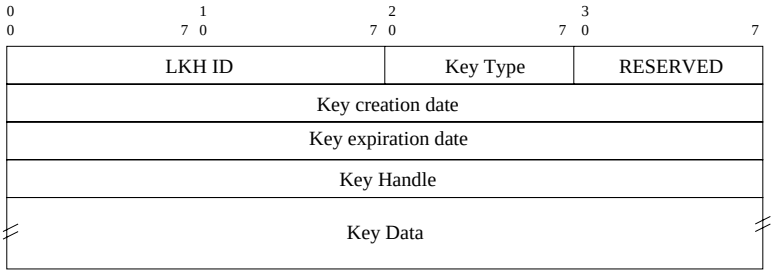


Figura B.11: Formato de una clave LKH

- ID LKH (2 bytes) : Es la posición de esta clave en la estructura de árbol binario LKH.
- Tipo de Clave (1 byte) : Es el algoritmo de cifrado que ha de usarse con esta clave.
- RESERVADO (1 byte) : Ha de ser 0.
- Fecha de Creación de la Clave (4 bytes) : Es el valor del momento de la generación original de los datos de la clave. Un tiempo cero indica que no hay tiempo anterior y la clave no es válida.
- Fecha de Expiración de la Clave (4 bytes) : Es el valor del momento a partir del cual la clave ja no será válida. Un valor cero indica que la clave no tiene tiempo de caducidad.

- ID de Clave o Key Handle (4 bytes) : Es un valor generado aleatoriament que identifica completamente una clave junto con el ID LKH.
- Datos de Clave (Variable) : Son los datos de la clave de cifrado actual. El formato depende del algoritmo del Tipo de Clave. Si el algoritmo especifica un IV, este se debe incluir en el campo de los Datos de Clave antes de la clave.

La Fecha de Creación de la Clave y la Fecha de Expiración de la Clave puede ser 0. Esto es necesario en caso que no sea posible la sincronización de tiempos en el grupo.

La primera estructura de Clave LKH en el atributo LKH_DOWNLOAD_ARRAY contiene el identificador de hoja y la clave del miembro del grupo. El resto de estructuras de Clave LKH contienen las claves a lo largo del path del árbol en orden desde la hoja, acabando en la KEK del grupo.

LKH_UPDATE_ARRAY

Este atributo se usa para actualizar las claves en un grupo. Normalmente formará parte del payload KD de los mensajes GROUPKEY-PUSH para actualizar todo el grupo. Este atributo consiste en una cabecera seguida por una o más claves LKH.

El número de atributos LKH_UPDATE_ARRAY en un payload KD no es limitado.

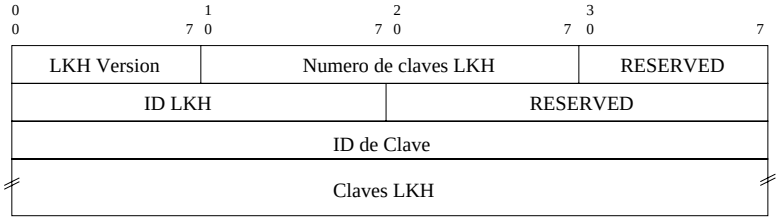


Figura B.12: Formato del atributo LKH_UPDATE_ARRAY de un paquete LKH KD

Los campos del atributo LKH_UPDATE_ARRAY se definen a continuación:

- Versión LKH (1 byte) : Contiene la versión del protocolo LKH. Ha de ser 1.
- Número de Claves LKH (2 bytes) : Es el número de las distintas claves en esta secuencia.
- RESERVADO (1 byte) : Ha de ser 0.
- ID LKH (2 bytes) : es el identificador del nodo asociado con la clave usada para cifrar la primera clave LKH.
- RESERVADO2 (2 bytes) : Ha de ser 0.
- ID de Clave (2 bytes) : Es el valor que identifica la clave, junto con ID LKH, usada para cifrar la primera clave LKH.

Las estructuras de claves LKH contienen las claves, a lo largo del path del árbol LKH, en orden desde ID LKH de la cabecera del LKH_UPDATE_ARRAY, hasta la KEK del grupo. El campo de Datos de la Clave de cada clave LKH se encripta con la clave LKH que la

precede en el atributo LKH_UPDATE_ARRAY. La primera clave LKH se encripta con la clave definida por los campos ID LKH y ID de la Clave que se encuentran en la cabecera de LKH_UPDATE_ARRAY.

SIG_ALGORITHM_KEY.

La clase SIG_ALGORITHM_KEY declara que la clave pública para este SPI está contenida en este atributo. Podría ser útil cuando no hay una infraestructura de clave pública disponible. El algoritmo de firma para esta clave está especificado en el payload SAK.

Payload Número de Secuencia.

El payload Número de Secuencia (SEQ) provee de una protección contra repeticiones para los mensajes GROUPKEY-PUSH. Se utiliza de manera muy similar al campo Número de Secuencia definido en el protocolo IPSec ESP.

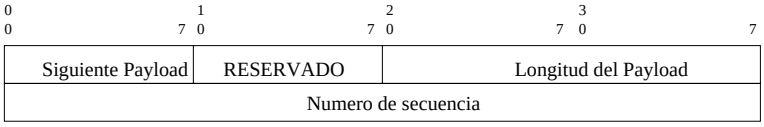


Figura B.13: Formato del payload Número de secuencia

Los campos del payload SEQ se definen a continuación:

- Siguiente Payload (1 byte) : Identificador del tipo de payload del siguiente payload en el mensaje. Si el actual es el último en el mensaje, este campo será 0.
- RESERVADO (1 byte) : Ha de ser 0.
- Longitud de Payload (2 bytes) : Longitud en bytes del payload actual, incluyendo la cabecera genérica de payload.
- Número de Secuencia (4 bytes) : Este campo contiene el valor de un contador incremental para el grupo. Se inicializa a cero por el GCKS y se incrementa en cada mensaje que se transmite. Así el primer paquete enviado por una SA rekey concreta tiene un Número de Secuencia de 1. La implementación del GDOI guarda el contador de secuencia como a atributo de la SA rekey y incrementa el contador para cada mensaje GROUPKEY-PUSH. El valor actual del Número de Secuencia se ha de transmitir a los miembros como parte del registro de SA. Los miembros del grupo han de guardar la ventana de desplazamiento. La ventana se trata como en el protocolo ESP [RFC2406].

Payload Prueba de Posesión.

El payload Prueba de Posesión se usa como parte de la autorización de grupo durante los intercambios GDOI. El payload POP es idéntico al payload SIG ISAKMP. Pero el uso es diferente.

El GCKS, el GCKS delegado o el miembro firma un hash de los sieguientes valores:

POP_HASH = hash ("pop" | Ni | Nr)

Donde hash( ) es la función hash que se usa en la firma.

El prefijo "pop" asegura que la firma del payload POP no se pueda usar para ningún otro propósito en el protocolo GDOI.

Payload Nonce.

La parte de datos del payload Nonce (Ni_b, Nr_b incluidos en los HASHs) han de ser valores entre 8 y 128 bytes.

B.4. Consideraciones de seguridad

El GDOI es un protocolo de gestión de asociaciones de seguridad para grupos de remitentes y receptores. La gestión de SA incluye un protocolo de establecimiento de claves para establecer con seguridad claves en los extremos de la comunicación. Este protocolo realiza la autentificación de las entidades GCKS y miembros, provee de confidencialidad a los mensajes de gestión de claves, y provee de autentificación de fuente para estos mensajes.

Este protocolo también usa las práctiques más conocidas para la defensa, en redes inseguras, de los ataque man-in-the-middle, apropiación de la conexion, repeticiones, reflexión, y negación de servicio (denial-of-service) [24,25,RFC2522,]. El GDOI asume que la red no es segura y puede estar bajo el control de un atacante.

El GDOI asume que el host es seguro y que la red es insegura. El GDOI establece claves entre los miembros, que han de ser fiables para usarlas de manera autorizada de acuerdo con la política del grupo. La seguridad del GDOI, por tanto, es mejor cuanto más fiables sean los miembros del grupo para proteger las autenticidades, las claves de cifrado y descifrado, y las claves de autentificación de mensajes.

Fase 1 ISAKMP.

Como se ha descrito, el GDOI utiliza los intercambios de la fase 1 en RFC2409 para proteger los intercambios GROUPKEY-PULL. Por tanto, todas las propiedades y consideraciones de seguridad de estos intercambios son relevantes en el GDOI.

Autentificación.

Los mecanismos definidos en RFC2409, denominados de Clave Pública (Public Key Encryption), proveen autentificación.

Confidencialitat.

La confidencialidad se consigue en la fase 1 con el intercambio Diffie-Hellman, que provee el material de las claves, a través de la negociación de las transformaciones de cifrado.

El protocolo de la fase 1 protege las claves enviadas en el protocolo GROUPKEY-PULL. La robustez del cifrado de la fase 1 supera el de las claves que se envian en el protocol GROUPKEY-PULL.

Protección contra Ataques Man-in-the-middle.

Un ataque man-in-the-middle o de secuestro de la conexión falsea la autentificación de una o más entidades de la comunicación durante el establecimiento de las claves. El GDOI confia en la autentificación de la fase 1 para superar los ataques man-in-the-middle.

Protección contra Ataques de Repetición/Reflexión.

En estos tipos de ataque, el atacante captura los mensajes entre las entidades del GDOI y, a continuació las remite a una entidad GDOI. Estos ataques buscan gobtener información a partir de les respuestas de los mensajes GDOI, o buscan interrumpir las operaciones de un miembro o del GCKS. El GDOI confia en los mecanismos de marcas temporals (noncea) de la fase 1 en combinación con la autentificación de mensajes basada en los hash, para proteger la reflexión o repetición de mensajes de gestión de claves previos.

Protección contra Negación de Servicio.

Un atacante de negación de servicio envía mensajes a una entidad GDOI para provocar que esta entidad realice operaciones de autentificación de mensajes innecesarias. El GDOI utiliza los mecanismos de cookies de la fase 1 para identificar falsos mensajes antes de processar los hash. Esta es una forma débil de protección contra negación de servicio donde la entidad GDOI comprueba los cookies, esta se puede imitar para un atacante sofisticado. El mecanismo de cookies de la fase 1 trabaja con estados, y utiliza los recursos de memoria para los cookies, pero los cookies sin estado son una mejor defensa contra los ataques de negación de servicio.

Intercambio GROUPKEY-PULL.

El intercambio GROUPKEY-PULL permite que un miembro del grup solicite SAs y claves a un GCKS. Trabaja, como un protocolo de la fase 2, bajo la protección de la asociación de seguridad de la fase 1.

Autentificación.

En el protocolo GROUPKEY-PULL no se requiere la autentificación punto-a-punto. Esto es necesario en el contexto del protocolo de la fase 1, que previamente ha autenticado las identidades.

La autentificación de los mensajes se provee en cada mensaje en los payloads HASH, donde el HASH se define sobre SKEYID_a (en el intercambio de la fase 1), la ID del missatge ISAKMP, y todos los payloads en el missatge. Sólo los dos extremos del intercambio conocen el valor SKEYID_a, este prove confianza sobre el emisor del mensaje.

Confidencialidad.

La confidencialitat se provee en la asociación de seguridad de la fase 1, descrita en RFC2409.

Protección contra Ataques Man-in-the-middle.

La autenticidad de mensajes incluye un secreto compartido per sólo el miembro del grup y el GCKS, cuando se construye el payload HASH. Este previene los ataques de man-in-the-middle y secuestro de la conexión porque un atacante no puede cambiar un mensaje que no se ha detectado.

Protección contra Ataques de Repetición/Reflexión.

Las marcas temporales proporcionan la vivacidad de los intercambio GROUPKEY-PULL. El miembro del grupo y el GCKS intercambian los valores nonce en los dos primeros mensajes. Estos valores se incluyen en los cálculos de los següientes payloads HASH. El miembro del grup y el GCKS no realizan ninguna tarea computacionalmente cara antes de recibir el HASH con su propio nonce incluido. El GCKS no actualiza el estado de gestión del grup (por ejemplo, el arbol de claves LKH) hasta que recibe el tercer mensaje en el intercambio con el payload HASH valido con su propio nonce.

Las implementaciones guardan un fichero de los mensajes GROUPKEY-PULL recibidos más recientemente y rechazan los mensajes que ya han sido procesados. Esto permite que los mensajes repetidos sean rápidamente descartados.

Protección contra Negación de Servicio.

Un mensaje GROUPKEY-PULL se identifica usando los cookies del intercambio de la fase 1 que lo preceden. Los cookies proveen una forma débil de protección contra la negación de servicio, en el sentido que un mensaje GROUPKEY-PULL que no tenga los cookies válidos se descarta.

Los mecanismos de protección contra repeticiones descritos proveen la base para la protección contra la negación de servicio.

Autorización.

El payload CERT en el intercambio GROUPKEY-PULL permite a un miembro del grupo o al GCKS presentar un certificado que contiene los atributos de autorización, e identificar un par de claves pública/privada. El payload POP de GROUPKEY-PULL permite que se consiga la autorización, donde la infraestructura de la autorización es diferente de la infrastructura de la autorización del GROUPKEY-PULL probando que posee la clave privada.

Intercambio GROUPKEY-PUSH.

El intercambio GROUPKEY-PUSH es un único mensaje que permite al GCKS enviar SAs y claves hacia los miembros del grupo. Este usualmente se envía a todos los miembros usando un grupo IP multicast. Esto proporciona un rekeying eficiente y capacidad de ajuste en el grupo de miembros.

Autentificación.

El intercambio GROUPKEY-PULL identifica una clave pública que se usa para la autentificación de los mensajes. Los mensajes GROUPKEY-PUSH se firman digitalmente con la correspondiente clave privada por el GCKS o su delegado. Esta firma digital proporciona autenticidad de fuente en el mensaje. Así, el GDOI protege el GCKS de las imitaciones en el grupo.

Confidencialidad.

El GCKS encripta los mensajes GROUPKEY-PUSH con una clave que se establece en el intercambio GROUPKEY-PULL.

Protección contra Ataques Man-in-the-middle.

Esta combinación de los servicios de confidencialidad y autentificación de los mensajes protegen los mensajes GROUPKEY-PUSH de los ataques man-in-the-middle y de secuestro de la conexión.

Protección contra Ataques de Repetición/Reflexión.

Los mensajes GROUPKEY-PUSH incluyen un número de secuencia incremental que protege de ataques de repetición y reflexión. Un miembro del grup reconoce un mensaje repetido comparando el número de secuencia con la ventana de desplazamiento, de la misma manera que el protocolo ESP usa el número de secuencia.

Las implementaciones guardan un fichero de mensajes GROUPKEY-PUSH recibidos recientemente y rechazan los mensajes que ya han sido procesados. Esto permite que los mensajes repetidos sean rápidamente descartados.

Protección contra Negación de Servicio.

El par de cookies identifica la asociación de seguridad de los mensajes GROUPKEY-PUSH. Los cookies sirven como una forma débil de protección contra negación de servicio en los mensajes GROUPKEY-PUSH.

La firma digital, usada para la autentificación de mensajes, tiene un coste computacional mayor que un codigo de autentificación de mensajes y podria amplificar los efectos de los ataques de negación de servicio contra los miembros del GDOI que procesan los mensajes GROUPKEY-PUSH. El coste añadido de las firmas digitals se justifica por la necesidad de preveer las imitaciones del GCKS. Si se utilizase una clave simétrica compartida para autentificación de mensajes GROUPKEY-PUSH, entonces la autentificación de la fuente GCKS seria imposible y cualquier miembro seria capaz de imitar al GCKS.

El potencial de la firma digital, que amplifica un ataque de negación de servicio, se atenua con el orden en que el miembro del grupo ejecuta las operaciones, donde la operación criptográfica menos costosa es la que se realiza primero. Primero el miembro del grupo desencripta el mensaje usando una clave simétrica. Si es un mensaje formado de manera válida, entonces se comprueba el número de secuencia con la ventana de repetición. Sólo si el número de secuencia es válido se verifica la firma digital. Así para que se realice un ataque de negación de

servicio, el atacante ha de conocer la clave de cifrado simétrica usada para la confidencialidad, y un número de secuencia válido. Generalmente esto significa que sólo los actuales miembros del grupo pueden desplegar un ataque de negación de servicio.

Forward Access Control.

Si se utiliza un algoritmo de gestión de grupo (como LKH), el control de acceso hacia adelante (forward access control) no se puede asegurar en algunos casos. Esto puede pasar si se niega el acceso a algunos miembros del grupo en el mismo mensaje GROUPKEY-PUSH que entrega una nueva política y TEKs al grupo. El control se puede mantener enviando múltiples mensajes GROUPKEY-PUSH, donde los cambios en el grupo se envían desde el GCKS de forma separada de la nueva política y TEKs

Bibliografía

[Adams and Lloyd, 2002] Adams, C. and Lloyd, S. (2002). *Understanding PKI: Concepts, Standards and Deployment Considerations*. Sams.

[Almeroth and Ammar, 1996] Almeroth, M. and Ammar, K. (1996). Collecting and modeling the join/leave behavior of multicast group members in the mbone. In *Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing*.

[Almeroth and Ammar, 1997] Almeroth, M. and Ammar, K. (1997). Multicast group behavior in the internet multicast backbone (mbone). *IEEE Communications*.

[Ballenson et al., 2000] Ballenson, D., McGrew, D., and Sherman, A. (2000). Key Management for Large Dynamic Groups: One Way Function Trees and Amortized Initialization. Internet Draft. Work in Progress.

[Baugher et al., 2002a] Baugher, M., Canetti, R., Dondeti, L., and Lindholm, F. (2002a). Group key management architecture. Internet Draft. Work in Progress.

[Baugher et al., 2002b] Baugher, M., Canetti, R., Rohatgi, P., and Cheng, P. (2002b). Mesp: Multicast encapsulating security payload. Internet Draft. Work in Progress.

[Baugher et al., 2003] Baugher, M., Weis, B., Hardjono, T., and Harney, H. (2003). The group domain of interpretation. RFC 3547.

[Bellare and Rogaway, 1993] Bellare, M. and Rogaway, P. (1993). Entity authentication and key distribution. In *Proceedings of Crypto 1993. Springer-Verlag, 1993. LNCS 773.*

[Bin and Hua, 2002] Bin, W. and Hua, J. (2002). Optimal key storage for secure multicast. Internal Report. Shangai Jiatong University.

[Blum and Micali, 1982] Blum, M. and Micali, S. (1982). How to generate cryptographically strong sequences of pseudo random bits. In *23rd IEEE Annual Symposium on Foundations of Computer Science, pages 112-117.*

[Boneh et al., 2001] Boneh, D., Durfee, G., and Franklin, M. (2001). Lower bounds for multicast message authentication. In *Proceedings of EUROCRYPT, pages 437-452, Innsbruck(Tyrol), Austria. Springer-Verlag. LNCS 2045.*

[Borella, 2000] Borella, M. (2000). Source models of network game traffic. *Computer Communications*, 23(4):403–410.

[Burmester and Desmedt, 1994] Burmester, M. and Desmedt, Y. (1994). A secure and efficient conference key distribution system. In *Proceedings of EUROCRYPT, pages 275-286, Perugia, Italy. Springer-Verlag. LNCS 950.*

[Canetti et al., 1999a] Canetti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M., and Pinkas, B. (1999a). Multicast security: A taxonomy and some efficient constructions. In *INFOCOMM'99.*

[Canetti et al., 1999b] Canetti, R., Malkin, T., and Nissim, K. (1999b). Efficient communication-storage tradeoffs for multicast encryption. In *Proceedings of Eurocrypt'99 pp.456-470.*

[Canetti and Pinkas, 2000] Canetti, R. and Pinkas, B. (2000). A taxonomy of multicast security issues. Internet Draft. Work in Progress.

[Caronni et al., 1998] Caronni, G., Waldvogel, M., Sunand, D., and Platter, B. (1998). Efficient Security for Large and Dynamic Groups. In *IEEE Workshop on Enabling Technologies WETICE 98.* IEEE Computer Society Press.

[Diffie and Hellman, 1976] Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654.

[Dinsmore et al., 2000] Dinsmore, P., Balenson, D., Heyman, M., Kruus, P., Scace, C., and Sherman, A. (2000). Policy based security management for large dynamic groups: an overview of the dccm project. In *DARPA Information Survivability Conference and Exposition. DISCEX'00.*

[Dondeti et al., 1999] Dondeti, L., Mukherjee, S., and Samal, A. (1999). Comparison of scalable key distribution schemes for secure one-to-many group communication. In *Proceedings of IEEE GLOBECOM Global Internet Symposium.* IEEE Press.

[Dondeti et al., 2000] Dondeti, L. R., Mukherjee, S., and Samal., A. (2000). Disec: A distributed framework for scalable secure many-many communication. In *Proceedings of IEEE International Symposium on Computer Communications (ISCC), Antibes, France.*

[Feldmann et al., 1998] Feldmann, A., Gilbert, A., Willinger, W., and Kurtz, T. (1998). The changing nature of network traffic: scaling phenomena. *Computer Communication Review*, 28(2):5–29.

[Fiat and Naor, 1993] Fiat, A. and Naor, M. (1993). Broadcast encryption. *Advances in Cryptology. CRYPTO93.*

[Gong and Shacham, 1994] Gong, L. and Shacham, N. (1994). Elements of trusted multicasting. In *Proc. IEEE Int. Conf. on Network Protocols, pages 23-30, Boston, MA, USA.*

[Hardjono et al., 2001] Hardjono, T., Baugher, M., and Harney, H. (2001). Group security association (gsa) management in ip multicast. In *Proceedings of the 16th International Conference on Information Security (IFIP/SEC), Paris, France.*

[Hardjono et al., 2000a] Hardjono, T., Cain, B., and Doroswamy, N. (2000a). A Framework for Group Key Management for Multicast Security. Internet Draft. Work in Progress.

[Hardjono et al., 2000b] Hardjono, T., Canetti, R., Baugher, M., and Dinsmore, P. (2000b). Secure IP Multicast: Problem areas, Framework and Building Blocks. Internet Draft. Work in Progress.

[Harkins and Carrel, 1998] Harkins, D. and Carrel, D. (1998). The internet key exchange (ike). IETF RFC 2409.

[Harney and Harder, 1999] Harney, H. and Harder, E. (1999). Logical Key Hierarchy Protocol. Internet Draft. Work in Progress.

[Harney and Muckenhirn, 1997] Harney, H. and Muckenhirn, C. (1997). Group Key Management Protocol (GKMP) Specification. RFC 2023.

[Henderson and Bhatti, 2001] Henderson, T. and Bhatti, S. (2001). Modelling user behavior in networked games. In *Proceedings of ACM Multimedia 2001, Otawa, Canada.*

[Hernández-Serrano and Pegueroles, 2003] Hernández-Serrano, J. and Pegueroles, J. (2003). Comercio electrónico a través de web de contenido multimedia sobre transporte multicast. In *Proceedings del II Simposio Español de Comercio Electrónico. SCE'03, Barcelona, España.*

[Hernández-Serrano et al., 2003a] Hernández-Serrano, J., Pegueroles, J., and Soriano, M. (2003a). Heuristic method for synthetic traffic generation for multicast networked games. In *Proceedings of the 2nd IASTED International Conference on Communication Systems and Networks. CSN 2003. Benalmádena, Spain.*

[Hernández-Serrano et al., 2003b] Hernández-Serrano, J., Pegueroles, J., and Soriano, M. (2003b). Método heurístico de generación de tráfico sintético de juegos en red multicast. In *Proceedings de las IV Jornadas de Ingeniería Telemática. Jitel'03, Las Palsmas, España.*

[Judge and Ammar, 2002] Judge, P. and Ammar, M. (2002). Gothic: A group access control architecture for secure multicast and anycast. In *In Proceedings of IEEE INFOCOM, New York, NY.*

[Lotspiech et al., 2001] Lotspiech, J., Naor, M., and Naor, D. (2001). Subset-Difference based Key Management for Secure Multicast. Internet Draft. Work in Progress.

[Menezes et al., 2001] Menezes, A. J., van Oorrschot, P. C., and Vanstone, S. A. (2001). *Handbook of applied cryptography.* CRC Press, fifth edition.

[Mittra, 1997] Mittra, S. (1997). Iolus: A framework for scalable secure multicasting. In *Proceedings of ACM SIGCOMM.* ACM Press.

[Moyer et al., 2001] Moyer, M. J., Rao, J. R., and Rohatgi, P. (2001). Maintaining balanced key trees for secure multicast. Internet Draft. Work in Progress.

[Nash et al., 2001] Nash, A., Brink, D., and Duane, B. (2001). *PKI: Implementing and Managing E-Security.* McGraw-Hill.

[Okamoto, 1988] Okamoto, T. (1988). A digital multisignature scheme using bijective public-key cryptosystems. *ACM Transactions on Computer Systems*, 6:432–441.

[Paxson and Floyd, 1995] Paxson, V. and Floyd, S. (1995). Wide area traffic: the failure of poisson modeling. *IEEE/ACM Trabsactions on Networking*, 3(3):226–244.

[Pegueroles et al., 2003a] Pegueroles, J., Hernández-Serrano, J., Rico-Novella, F., and Soriano, M. (2003a). Adapting gdoi for balanced batch-lkh, draft-irtf-gsec-gdoi-batch-lkh-00.txt. proceedings of the 57th ietf meeting. july 2003, vienna, austria. Internet Draft. Work in Progress.

[Pegueroles and Rico-Novella, 2002a] Pegueroles, J. and Rico-Novella, F. (2002a). Evaluación de herramientas java para ofrecer confidencialidad en servidores de vídeo. In *Proceedigs de la VII Reunión Española sobre Criptología y Seguridad de la Información. VII RECSI Oviedo, España.*

[Pegueroles and Rico-Novella, 2002b] Pegueroles, J. and Rico-Novella, F. (2002b). Performance evaluation of cryptographic algorithms for multicast secrecy protection. In *Proceedings of the 7th Nordic Workshop on Secure IT Systems. Nordsec2002. Karlstad University, Sweden.*

[Pegueroles and Rico-Novella, 2002c] Pegueroles, J. and Rico-Novella, F. (2002c). Rekeying de grupo en multicast seguro usando el teorema de fermat. In *Proceedigs del XVII Simposium Nacional de la Unión Científica Internacional de Radio XVII URSI, Alcalá de Henares, Madrid, España.*

[Pegueroles and Rico-Novella, 2003a] Pegueroles, J. and Rico-Novella, F. (2003a). Balanced batch lkh: New proposal, implementation and performance evaluation. In *Proceedings of the IEEE Symposium on Computers and Communications . ISCC'2003. Antalya, Turkey.*

[Pegueroles and Rico-Novella, 2003b] Pegueroles, J. and Rico-Novella, F. (2003b). Balanced lkh for secure multicast. In *Proceedings of the 2nd NATO PfP/PWP International Scientific Conference on Security and Protection of Information. SPI03. Brno, Czech Republic.*

[Pegueroles and Rico-Novella, 2003c] Pegueroles, J. and Rico-Novella, F. (2003c). Enabling secure multicast using a new java lkh rekeying tool. In *Proceedings of the 3rd International Conference on Web Engineering ICWE2003. Oviedo, Spain. Published by Springer Verlag, LNCS 2722.*

[Pegueroles and Rico-Novella, 2003d] Pegueroles, J. and Rico-Novella, F. (2003d). Lkh mejorado para gestión de claves en grupos multicast. In *Proceedings de las IV Jornadas de Ingeniería Telemática. Jitel'03, Las Palsmas, España.*

[Pegueroles et al., 2003b] Pegueroles, J., Rico-Novella, F., Hernández-Serrano, J., and Soriano, M. (2003b). Improved lkh for batch rekeying in multicast groups. In *IEEE International Conference on Information Technology Research and Education. ITRE2003. New Jersey, USA.*

[Quinn, 1999] Quinn, B. (1999). IP Multicast Applications: Challenges and Solutions. In *Proceedings of 44th Internet Engineering Task Force*. IETF.

[Rafaeli, 2000] Rafaeli, S. (2000). A descentralized architecture for group key management. Technical report, Computing Department. Lancaster University.

[Sampigethaya et al., 2002] Sampigethaya, R., Li, M., and Poovendran, R. (2002). Centralized Key Management and Distribution for Dynamic Multicast Groups: Scalability Issues. Internet Draft. Work in Progress.

[Schneier, 1996] Schneier, B. (1996). *Applied Cryptography. Protocols, algorithms and source code in C*. John Wiley & Sons, second edition.

[Schneier, 2000] Schneier, B. (2000). *Secrets & Lies. Digital Security in a Networked World*. John Wiley & Sons.

[Shamir, 1979] Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22:612–613.

[Simmons, 1992] Simmons, G. (1992). An introduction to shared secret and/or shared control schemes and their application. *Contemporary Cryptology: The Science of Information Integrity*, pages 441–497.

[Snoeyink et al., 2001] Snoeyink, Suri, and Varghese (2001). A lower bound for multicast key distribution. In *In Proceedings of IEEE INFOCOM, New York, NY*.

[Szuprowicz, 1995] Szuprowicz, B. (1995). *Multimedia Networking*. McGraw-Hill, first edition.

[varios autores, 1999] varios autores (1999). NDS. Internet Multicast Security - Overview of Issues and Technologies.

[Wallner et al., 1999] Wallner, D., Harder, E., and Agee, R. (1999). Key Management for Multicast: Issues and Architectures. RFC 2627.

[Wong et al., 1998] Wong, C., M.Gouda, and Lam, S. (1998). Secure group communications using key graphs. In *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 68–79.

[Yang et al., 2001] Yang, Y. R., Li, X. S., Zhang, X. B., and Lam, S. S. (2001). Reliable group rekeying: a performance analysis. In *Proceedings of the ACM SIGCOM 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM Press.

[Zhang et al., 2001] Zhang, X. B., Yang, Y. R., Lam, S. S., and Lee, D. Y. (2001). Protocol design for scalable and reliable group rekeying. In *Proceedings of SPIE conference on scalability and traffic control in IP networks, Denver, CO*.