UNIVERSIDAD DE ALMERÍA

CONTRIBUTIONS TO SECURE MULTICAST AND PRIVACY-PRESERVING COMPUTATIONS ON PEER-TO-PEER NETWORKS

Tesis doctoral presentada por
JUAN ÁLVARO MUÑOZ NARANJO

Dirigida por
Dr. LEOCADIO GONZÁLEZ CASADO

UNIVERSIDAD DE ALMERÍA

CONTRIBUTIONS TO SECURE MULTICAST AND PRIVACY-PRESERVING COMPUTATIONS ON PEER-TO-PEER NETWORKS

Tesis doctoral presentada por
JUAN ÁLVARO MUÑOZ NARANJO

Dirigida por
Dr. LEOCADIO GONZÁLEZ CASADO

El doctorando El director

Almería, marzo de 2013

Contributions to Secure Multicast and Privacy-Preserving Computations on Peer-to-peer Networks

Doctorando: Juan Álvaro Muñoz Naranjo
Director: Leocadio González Casado

La siguiente página web contiene información actualizada sobre los temas relacionados con esta tesis doctoral:
http://www.hpca.ual.es/

Texto impreso en Almería, marzo de 2013

A mis padres, por darme algo que el dinero
no puede comprar: una buena educación.

Preface

This thesis dissertation addresses two different areas that fall below the term *privacy*. The first of them is private communications within closed groups, a matter that commonly receives the name of *secure multicast*. A set of participants would like to establish communications in such a way that no outsider is able to listen to the information transmitted. This problem has many applications like pay-per-view services in multimedia streaming platforms, private multiconferences in business or even military operations.

The typical solution to the problem consists of agreeing on a symmetric encryption key, namely the *session key*, which is known by the whole group and used to encrypt and decrypt every message sent. However, implementing this solution in practice is not trivial mainly due to two reasons: using an encryption key for long periods of time makes it vulnerable to attacks, and variations in the group of members (commonly known as *churn*) forces to use a different key after every new arrival or leaving of a member. Because of that, the session key must be refreshed periodically and upon changes in the group composition. Chapter 1 introduces the aforementioned concepts in a more formal way and gives a wider vision on the topic.

According to how the session key is refreshed we can classify secure multicast schemes into three categories: centralized, decentralized and distributed. In the first one, a central entity, the *Key Server*, is in charge of establishing new session keys and dealing them to the whole network. This is the simplest way of solving the problem, at the cost of limited audience sizes, and most of the proposals to the date belong to this category. The second type, decentralized schemes, is a natural

extension of centralized solutions: larger audiences are managed by arranging participants into disjoint clusters, each managed by a secondary Key Server by means of a centralized scheme. Finally, in distributed schemes there is no single entity managing the re-key process, so all participants must agree on the new cryptographic material each time it is refreshed. Distributed schemes avoid single points of failure and prevent a single entity from controlling the group, at the cost of poor scalability.

This thesis dissertation deals with secure multicast schemes due to their importance as standalone solutions or as building blocks for decentralized solutions. A wide review of the state of the art is provided in Chapter 2. Chapter 3 presents *(i)* a scheme of this nature, discusses its properties and puts its performance to the test. The mathematics behind the scheme are mainly based on the Extended Euclidean Algorithm, which we have also used to develop *(ii)* an authentication solution for messages coming from the Key server and *(iii)* a zero-knowledge proof that a participant can use to check the membership of others.

The whole solution has already been the object of a cryptanalysis by other researchers, with some vulnerabilities being found in the message authentication and zero-knowledge proof schemes, but not in the main secure multicast algorithm. We mention these flaws and show some solutions to them developed by other researchers. We would like to make clear that those last solutions are no merit of the author of this thesis, and are referred here for the sake of completion.

Now, let us turn to the second topic this thesis dissertation deals with: privacy-preserving distributed computations. In this scenario, nodes in a network would like to follow the evolution of a global variable to which every node contributes. However, individual contributions from each node should not be disclosed to others. A typical example is the *Millionaire's problem*: two millionaires want to know who is the richest without revealing their fortunes. Solving this problem for

many participants and other functions has many applications like data aggregation, trust computing and data mining, all of which are currently hot topics.

This research topic is not new and many algorithms have been proposed so far, most of them being applied to a reduced, static set of players and with fault tolerance issues not usually addressed. However, we foresee that addressing practical issues will become more and more important with the popularization of different types of networks. For example, in peer-to-peer networks nodes can enter and leave at any time, just like in the secure multicast problem, and computations should not be halted or restarted because of that. In wireless sensor networks, the transmission medium is failure prone, and participants may come and go. Chapter 4 reviews the related literature.

We have thus developed an iterative privacy-preserving distributed algorithm for peer-to-peer networks that focuses on those practical problems. Our algorithm follows an *asynchronous* message passing model given that asynchrony tolerates churn and message faults in a natural way, in opposition to *synchrony*. To prove this, we compare it to a closely related synchronous proposal from the literature. We believe our algorithm is the first of its kind. Chapter 5 presents it.

Regarding the types of adversary models considered in the literature, they can be either *semi-honest* or *malicious*. The former will analyze the received information in order to guess other players' private information, but will follow the protocol faithfully. The latter is assumed to take arbitrary actions (forging messages, for example) to obtain the desired knowledge. Our algorithm deals with the semi-honest adversary type.

To summarize, this thesis dissertation *(i)* surveys the state of the art in centralized secure multicast algorithms, *(ii)* introduces a centralized secure multicast solution with low computational requirements as well as complementary algorithms to authenticate re-key messages and members, *(iii)* reviews the state of the art in privacy-preserving distributed

computations and *(iv)* introduces an iterative asynchronous privacy-preserving algorithm for distributed computations that tolerates node churn and message faults.

Prefacio

El término *privacidad* engloba un amplio campo de investigación compuesto por diferentes áreas. Esta tesis doctoral se centra en dos de ellas. La primera son las comunicaciones privadas en grupos restringidos, o *secure multicast*. Un grupo de participantes desea mantener comunicaciones de forma que ningún oyente externo al grupo sea capaz de interpretar la información transmitida. Este problema tiene diferentes aplicaciones como servicios pay-per-view en plataformas de streaming multimedia o multiconferencias privadas en el ámbito empresarial e incluso militar.

La solución estándar al problema consiste en establecer una clave de cifrado simétrico común al grupo, llamada *session key* o *clave de sesión*, con la cual todos los participantes cifran o descifran la información transmitida. Sin embargo, las implicaciones prácticas de esta solución no son triviales por dos motivos: el uso de una misma clave de cifrado durante largo tiempo la hace vulnerable a ataques, y los cambios en la composición del grupo (un fenómeno conocido como *churn*), de ocurrir, fuerzan a cambiar la clave después de cada entrada o salida de participantes. Por ello es necesario refrescar la clave de sesión periódicamente, y en especial después de cada cambio en la composición del grupo. El capítulo 1 introduce más formalmente estos conceptos y da al lector una visión más amplia del problema.

Las propuestas de secure multicast existentes pueden clasificarse en tres categorías dependiendo de cómo realicen la renovación de la clave de sesión: centralizadas, descentralizadas y distribuidas. En las primeras, una entidad central, el *Key Server* o *Servidor de Claves*, está a cargo del establecimiento de nuevas claves de sesión y la distribución

a toda la red. Ésta es la forma más sencilla y popular de resolver el problema a cambio de soportar tamaños de grupo limitados. La segunda categoría, las propuestas descentralizadas, es una extensión natural de la primera: para conseguir mayores tamaños de grupo los participantes se distribuyen en subgrupos disjuntos, cada uno gestionado por un Servidor de Claves secundario mediante secure multicast centralizado. Finalmente, en las propuestas distribuidas no existe una única entidad a cargo del proceso de renovación de claves. En su lugar, todos los participantes colaboran en la generación del material criptográfico cada vez que sea necesario. Los esquemas distribuidos ofrecen por tanto más fiabilidad al evitar un único punto de fallo a cambio de una escalabilidad muy limitada.

Esta tesis se enfoca hacia la categoría centralizada debido a su importancia como solución en sí misma y como pilar de construcciones descentralizadas. El capítulo 2 realiza una amplia revisión del estado del arte, y el capítulo 3 presenta *(i)* un nuevo esquema centralizado, discute sus propiedades y pone a prueba su rendimiento. Los fundamentos matemáticos del esquema se basan principalmente en el Algoritmo Extendido de Euclides, gracias al cual también hemos desarrollado *(ii)* una solución de autenticación para los mensajes de renovación provenientes del Servidor de Claves, y *(iii)* una prueba de conocimiento cero (zero-knowledge proof) que permite a miembros legales del grupo verificar la legalidad de otros.

Los tres esquemas han sido ya objeto de criptoanálisis por parte de otros investigadores, los cuales han hallado algunas vulnerabilidades en el segundo y tercero (autenticación de mensajes y prueba de conocimiento cero). Sin embargo, el esquema principal de secure multicast sigue siendo seguro a día de hoy. Mencionaremos las vulnerabilidades encontradas por el criptoanálisis, así como soluciones propuestas por otro conjunto de investigadores. Dichas soluciones no son mérito del autor de esta tesis y se muestran aquí con el único propósito de dar al lector una perspectiva completa.

La segunda parte de esta tesis se centra en computaciones distribuidas con preservación de privacidad. En este caso, los nodos que conforman una red desean seguir la evolución de una variable global a la que todo el mundo contribuye. Sin embargo, las contribuciones individuales hechas por cada nodo no deberían ser conocidas por los demás. Un ejemplo típico, muy ilustrativo, es el *Problema de los millonarios*: dos millonarios quieren saber quién es más rico sin revelar la cuantía de sus fortunas. Situaciones similares, especialmente aquellas que involucran un número indeterminado de participantes y el cómputo de otras funciones, son útiles a la hora realizar agregación de datos, cálculo de confianza y minería de datos, operaciones todas de creciente interés.

El problema de la preservación de privacidad en computación distribuida no es nuevo: existen ya un amplio número de soluciones. Sin embargo, la mayoría de ellas es aplicable sólo a un grupo reducido de participantes y no suelen ser tolerantes a fallos. El autor de esta tesis cree que esas dos características van a cobrar mucha importancia con el tiempo debido a la popularización de diferentes tipos de redes. Por ejemplo, en redes peer-to-peer los nodos pueden entrar y salir de la red de forma inesperada (tal y como hemos comentado previamente para los escenarios de secure multicast), y los cálculos no deberían detenerse o reiniciarse por ello. En redes de sensores (wireless sensor networks) el medio de transmisión es propenso a fallos, y también los participantes pueden aparecer y desaparecer sin previo aviso. El capítulo 4 repasa de la literatura disponible en este campo.

En vista de la importancia de la tolerancia a fallos y al dinamismo de las redes, en el capítulo 5 hemos desarrollado un algoritmo de cómputo distribuido iterativo con preservación de privacidad enfocado a redes peer-to-peer que se adapta a ese tipo de situaciones prácticas. Nuestro algoritmo sigue un modelo de paso de mensajes *asíncrono* que tolera churn y pérdida o retraso de mensajes de forma natural, en oposición a los algoritmos *síncronos*. Para demostrarlo, lo comparamos con una

propuesta síncrona parecida tomada de la literatura. Creemos que nuestro algoritmo es el primero de este tipo.

La literatura en este campo considera dos modelos de adversario a tener en cuenta: *semi-honestos* y *maliciosos*. Los primeros analizan cualquier dato a su disposición para obtener información privada de otros participantes, pero se asume que ejecutan el protocolo de forma correcta. De los segundos se espera que lleven a cabo acciones arbitrarias para obtener la información privada deseada, tales como enviar mensajes falsos. Nuestro algoritmo tolera adversarios semi-honestos.

A modo de resumen de este prefacio, la presente tesis *(i)* analiza el estado del arte en algoritmos de secure multicast centralizado, *(ii)* presenta una solución de secure multicast centralizada con bajos requerimientos computacionales así como algoritmos complementarios para la autenticación de los mensajes de refresco y los miembros, *(iii)* repasa el estado del arte en computación distribuida con preservación de privacidad y *(iv)* presenta un algoritmo iterativo asíncrono para computación distribuida con preservación de privacidad que tolera churn y fallos en el envío de mensajes.

Agradecimientos

Completar una tesis doctoral es una dura carrera de fondo durante la cual numerosas personas contribuyen de una forma u otra. Para empezar, quiero agradecer su labor de guía al Dr. Leocadio González Casado. Son muchas las horas, algunas intempestivas, las que ha empleado en ayudarme. Parece que por fin tanto trabajo tiene su recompensa.

Desde el comienzo de esta tesis mi *hogar* ha sido, en sentido figurado y a veces no tanto, el grupo de investigación TIC-146 Supercomputación: Algoritmos. Quiero dar las gracias a su líder, la Dra. Inmaculada García Fernández, y a su responsable actual, la Dra. Gracia Ester Martín Garzón, por sus esfuerzos para darme siempre las mayores facilidades y recursos posibles a la hora de investigar. Mencionaré ahora al Ministerio de Economía y Competitividad, la Junta de Andalucía y el Fondo Europeo de Desarrollo Regional (FEDER) por la financiación de los proyectos TIN2012-37483-C03-03, TIN2008-01117 y P11-TIC7176 en los que estoy involucrado.

No puede faltar tampoco un gran reconocimiento al Dr. Juan Antonio López Ramos por sus inestimables e indispensables contribuciones a la parte matemática de esta tesis. A él debo mi entrada en el mundo de la investigación desde que dirigiese mi Proyecto de Fin de Carrera allá por el año 2007. Guardo un muy buen recuerdo de aquella época.

Asimismo, la segunda parte de esta tesis fue concebida y desarrollada durante una estancia en el Grupo de Investigación en Inteligencia Artificial de la Universidad de Szeged, Hungría, bajo la tutela del Dr. Márk Jelasity. Veo en él aquello a lo que todo investigador debería aspirar: espero haber aprendido de su profesionalidad algo más que privacidad en computaciones distribuidas. Además, me siento en deuda con los

investigadores del grupo que me recibieron con los brazos abiertos, en especial (aunque no únicamente) István Hegedűs y Róbert Ormándi.

Durante los últimos años he desarrollado labores docentes en el Departamento de Informática de la Universidad de Almería (antiguo Departamento de Arquitectura de Computadores y Electrónica). Mis agradecimientos también a sus integrantes, así como a los del Departamento de Matemáticas de la misma universidad (antiguo Departamento de Álgebra y Análisis Matemático) al que pertenecí previamente, en especial a los Drs. Justo Peralta López y José Escoriza López.

Además de comparar el desarrollo de una tesis con una carrera de fondo, siempre me viene a la mente la expresión *montaña rusa emocional* cuando pienso en los años invertidos en ella. En el mismo vagón han viajado también bastantes compañeros del grupo de investigación a los cuales estoy muy agradecido. Quiero destacar a José Ignacio Agulleiro Baldó, José Manuel Molero Pérez, Juan Francisco Rodríguez Herrera y José Román Bilbao Castro. También quiero agradecer a mi círculo de amigos, los de toda la vida, los momentos de descanso, risas y comprensión que me han regalado.

Finalmente, mis padres lo han dado siempre todo para proporcionarnos a mi hermana y a mí la mejor educación posible y las mejores oportunidades. A ellos tres, mi familia, gracias.

Y, para terminar, el mayor de los agradecimientos a Carmen. Convivir con un estudiante de doctorado ha debido de ser la tarea más dura de todas las llevadas a cabo durante la elaboración de esta tesis.

Gracias.

Marzo de 2013

Contents

List of Figures

List of Tables

List of Algorithms

Chapter

1

Introduction

Security and privacy form a wide body of research that covers a vast set of topics. It is not necessary to explain its great importance in today's society. Virtually every activity that we can benefit from or that we can develop is related to, or implies, the use of computers, digital devices and the networks connecting them. These systems have a great exposure to threats since attackers can benefit in many ways: economical profit, information stealing, denial of service of commercial competitors or enemies and more [96].

This thesis dissertation deals with two different topics within the privacy world: centralized secure multicast and privacy-preserving distributed computations on peer-to-peer networks.

1.1 Centralized secure multicast

Multicast communications allow hosts to send information to other hosts within a group avoiding the establishment of point-to-point connections with all of them. IP multicast technologies [65] (which use routing techniques at a low level over a network, such as the IGMP protocol) have not achieved the expected success due to several reasons (need for compatible routers, implantation costs, lack of support from Internet providers, etc.). As a recent alternative, application level multicast

has taken over, since it offers the same functionality at a lower cost and easier deployment: instead of requiring the deployment of specific protocol-compatible architectures, a logical network is built and hosts resend messages themselves.

Multicast communications can be either *one-to-many*, if the source of the transmitted data is one entity only over time (such as IPTV or P2PTV services) or *many-to-many*, if several clients or all act as a source of data. Multiconferences are an example of this (strictly, each data source establishes a one-to-many multicast communication).

There are services that take advantage of multicast but need to keep communications private. Those technologies that make it possible are known as *secure multicast*. Applications of secure multicast are, among others, pay-per-view IPTV or P2PTV, private multiconferences (oriented to business, politics or even military affairs), or any private service that involves several participants or clients.

The typical approach to establish secure multicast communications is to agree on one or several symmetric encryption keys (depending on the topology and size of the network) to encipher messages. However the key, or keys, must be renewed periodically to prevent attacks from outsiders or even insiders.

Depending on how key distribution and management are carried out, secure multicast schemes are divided into centralized, decentralized and distributed. Centralized schemes depend directly on a single entity to generate and distribute every cryptographic key. Decentralized schemes replicate the centralized infrastructure in order to reach larger audiences, usually involving entities that act as local subservers and manage subgroups of users, and requiring full or partial re-encryption of the multicast information in some cases. Finally, in a distributed approach there is no group controller, and therefore all members participate actively in the key generation process. Generally, the final encryption key is therefore the result of the contributions of all members, which makes the process time and communication consuming. The popular survey in [90] clearly describes this classification.

Centralized secure multicast schemes are of great use thanks to their simplicity and the popularization of services like IPTV [22] and ad-hoc networks. Even in decentralized architectures for huge audiences they appear at the core of every separate group. Therefore, they play an important role in the secure multicast global field.

Regardless of their nature, any secure multicast scheme must provide: information privacy while in transit, an efficient and fault-tolerant re-keying process so an acceptable quality of service (QoS) is guaranteed, and forward and backward secrecy.

Definition 1 **Forward secrecy** *implies that a member that leaves the network (i.e., her membership expires) should not be able to decrypt any ciphered information transmitted thereafter.*

Definition 2 **Backward secrecy** *implies that an arriving member should not be able to decrypt any ciphered information transmitted before her arrival.*

Both impose a refreshment of the encryption key used to cipher the transmitted information. These two constraints may become an efficiency problem at high churn rates (see Def. 3), though some less restrictive scenarios may not require backward secrecy.

Definition 3 *The* **churn** *phenomenon refers to nodes unexpectedly entering or leaving the network at any time.*

Additionally, schemes must be resistant to collusion. A collusion attack can be defined differently depending on the research field we are addressing. Next, we provide a definition for the secure multicast field.

Definition 4 *In a* **collusion attack** *to a secure multicast algorithm, two or more revoked users ally together and use their expired keying material in order to illegally decrypt new multicast information.*

There are other features of schemes that are directly related to reliability. We define them next.

Definition 5 *A secure multicast scheme is* **self-healing** *if members are able to obtain lost keying material from new re-key messages received from the Key Server. This is, members do not need to request lost messages in order to recover previous keying material.*

A closely related feature is recoverability.

Definition 6 m-recoverability *implies that a member may still recover the current keying material after missing a maximum of m key updates.*

Finally, secure multicast protocols can be divided into *stateful* and *stateless*.

Definition 7 *In a* **stateful** *secure multicast scheme, the state of the keying material at a given time is defined by the history of that material and the modifications made on it. Thus, re-key messages contain modifications done on the previous state. This implies that members must be aware of all re-key operations performed since their arrival.*

Definition 8 *In a* **stateless** *secure multicast scheme, the state of the keying material at a given time is independent of the history of the material. As a consequence, members may obtain all the keying material from scratch at every re-key operation.*

The stateless property clearly makes schemes more resilient against faulty networks. Addressing reliability has become the trend in the last years due to the popularization of ad-hoc networks.

Chapters 2 and 3 deal with secure multicast. The former surveys the state of the art, while the latter introduces a centralized secure multicast scheme.

1.2 Privacy-preserving distributed computations on peer-to-peer networks

The second part of this dissertation deals with the challenge of adding privacy-related routines to fully distributed computations. The motivation for it relies on the fact that a large part of the networked systems is evolving towards a fully distributed paradigm. Peer-to-peer networks are examples of this, but we can also find ad-hoc networks composed by powerful mobile phones, wireless sensor networks (WSNETs), vehicular networks (VANETs), smart power grids and more. These systems are composed by autonomous nodes that cooperate without the intervention of centralized third parties, therefore executing fully distributed algorithms. An interesting subset of those algorithms corresponds to functions that obtain global results based on local attributes of the nodes. With them, nodes can learn global

properties of the whole network by updating a local value with the help of their neighbour nodes. Typical applications are data aggregation [47, 52, 101], spectral analysis [51], trust management [49], and distributed ranking and data mining [8, 25, 85]. These algorithms can simply monitor a system, or they can be used for control and optimization as well. A crucial point here is that distributed algorithms should be *asynchronous* rather than *synchronous*. We define this concepts next.

Definition 9 *A* **synchronous distributed algorithm** *requires that messages are sent and received within a strict period of time. This type of algorithms imposes barriers on the progress of processes, thus forcing all of them to carry out executions at the same pace.*

Definition 10 *An* **asynchronous distributed algorithm** *does not impose deadlines on the exchanged messages. As a consequence, processes in these algorithms can progress at different speeds without affecting the correctness of computations.*

The reason why asynchronous algorithms are preferred for the aforementioned applications is that in large networks one can not assume that a given node will be online at a given specific time, or that messages will arrive properly.

As usual, some of those algorithms may require privacy mechanisms that prevent nodes from learning others' attributes. To give the reader an idea of this, let us refer to the illustrative *Millionaires' problem*, in which two millionaires want to discover who is the richer without revealing their actual fortunes [108]. Privacy preserving computations have been studied for a long time, however most of the existing solutions to the date address scenarios with a few participants in a very controlled environment, such as e-commerce [91]. Let us now define the term "collusion attack" for a privacy-preserving distributed computations environment.

Definition 11 *In a* **collusion attack** *to a privacy-preserving distributed computations algorithm, two or more users ally together and use they knowledge in order to illegally obtain secret information from other user.*

Note that the main difference of this collusion attack definition with Def. 4 relies on the fact that the colluding users are still active.

Instead of focusing on a few users, the scenarios we tackle here are composed by thousands of nodes (or even more). For example, nodes in a trust computing algorithm will need to calculate the reputation of a peer in the network with the restriction of not being able to extract the individual opinions contributed by other participants. As another example, collective measurements might be taken in a smart power grid without learning the consumption of every single house. The list of possible applications is extensive. More specifically, the IT industry has recently started to collect and analyze vast amounts of personal data centrally (specially social networks companies). These practices raise legal and ethical concerns and foster research on the privacy preservation field.

Peer-to-peer networks offer an alternative to centrally managed databases of user profiles and behaviour, hence we are focusing on privacy preserving routines on this kind of networks, or networks with similar properties. Peer-to-peer networks have two characteristics that make the execution of fully distributed algorithms and specially privacy preserving routines a challenge. First, they are inherently unreliable: messages may be lost or delayed, affecting algorithms that do not take this fact into account. Second, it is impossible to assure that every node in such a large network will be online at a given moment: algorithms should be ready to cope with churn (see Def. 3) without stopping or restarting computations. For these reasons, developing privacy preserving routines in peer-to-peer networks is an interesting and challenging problem, specially if we want communications to be asynchronous.

Chapter 4 provides the background for this topic and reviews the literature up to date. In Chapter 5 we introduce our own proposal which, to the best of our knowledge, is the first asynchronous privacy-preserving algorithm for fully distributed computations. Finally, Chapter 6 concludes this dissertation.

Part I

Contributions to secure multicast

Chapter

2

Related work on centralized secure multicast

2.1 Introduction

This chapter extends the work presented in [69, 70] in order to provide a selection of the most important centralized secure multicast protocols so far. First, a classification of the field is presented according to the scenario of application. Three categories arise:

(1) **General-purpose schemes**, suitable for a wide range of applications.

(2) **Multi-group schemes**, which address scenarios that involve access control to more than one information channel and different, hierarchical degrees of privilege.

(3) **Self-healing schemes**, for ad-hoc networks, which focus mainly on reliability since the environment they address is prone to message loss.

Within each category, schemes are discussed and compared attending to the properties mentioned in the introduction. It is important to remark that many recent schemes that were not considered in popular but older surveys [16, 90, 115] are

shown here. We would also like to mention that [69, 70] was the first survey to include multi-group schemes to the best of our knowledge. To conclude the chapter a general comparison of all the schemes is shown so the reader can have a global and clear view of the state of the art.

The following notation will be used for the rest of the chapter. The single entity that manages the re-keying process receives the name *Key Server*. Hosts that conform the main body of the network are named *members*. h and d denote a tree's height and degree, respectively. The total number of members is given by n. b is the bit-length of a symmetric key and H is the output length in bits of a hash function. q is the length in bits of a typical RSA problem large prime number, while r is the number of revoked members. Additional specific notation will be indicated where needed.

2.2 General-purpose schemes

General purpose schemes were the first ones to appear and have been around for more than ten years now. One of the earliest is the Group Key Management Protocol (GKMP) [39], in which the Key Server shares a key with every member in the audience and a common group key. Some re-key operations require a unicast connection with each member, hence the scheme scales poorly. More efficient solutions were soon proposed. We can divide them in two main approaches: the *key hierarchy tree approach*, which arranges users into a logical structure in order to optimize the key refreshment operation and the *computational approach*, which rather relies on mathematical strategies.

2.2.1 The key hierarchy tree approach

The hierarchical tree of keys is clearly the most popular idea due to its small computational requirements, and to the fact that the user arrangement is *logical*, i.e., no costly underlying topology is created.

The first scheme of that kind was the Logical Key Hierarchy (LKH) [103, 105]. In order to reduce the number of re-key messages per join/leave operation of the trivial approach, a logical tree is built with randomly chosen *user keys* at the leaves.

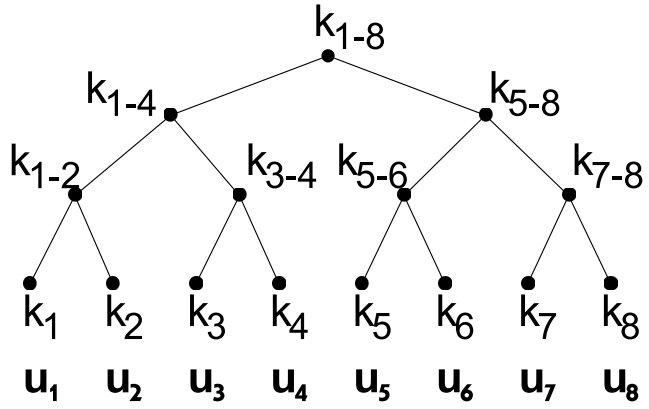


Figure 2.1: **A key hierarchy tree.** u_i and k_i denote user i and her user key, respectively. k_{1-8} is the group key. The rest are KEKs.

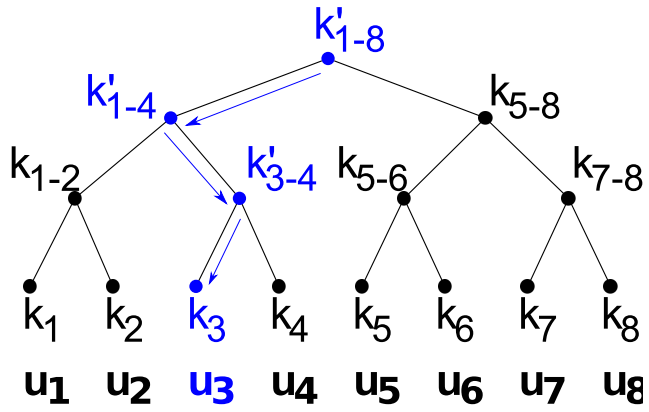


Figure 2.2: **Re-keying associated to a join.** User u_3 enters the system.

Figure 2.1 depicts such tree. Every user key is only known by its owner and the Key Server. A Data Encryption Key (DEK), namely the *group key*, is placed at the root node: it encrypts the actual broadcast information, while intermediate tree nodes keep Key Encryption Keys (KEKs), used in the re-key process. A user knows only the keys in her *key path*, i.e., the keys in the path from her leaf to the root. The arrival or departure of a member implies that the group key, the member's key-path and those nodes that are siblings to the key path must be refreshed. Figure 2.2 shows the join process. User u_3 is joining the tree: as a consequence, the Key Server sends the user a key under a different secure channel, say k_3, and renews

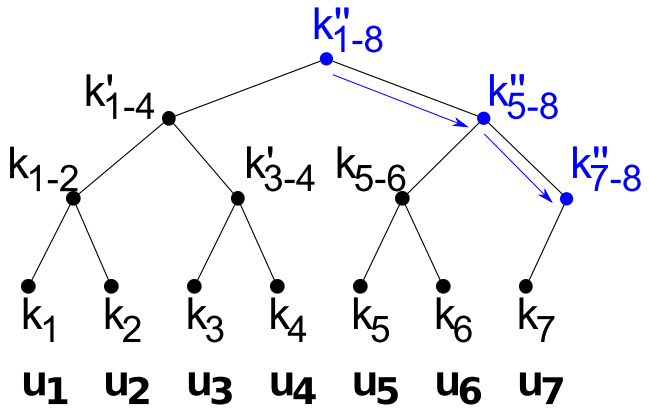


Figure 2.3: **Re-keying associated to a leave.** User U_8 leaves the system.

the keys in her way to the root: k', k'_{1-4} and k'_{3-4}. Next, these keys are sent to all members as follows (notation ab means b encrypted with key a):
Users 1 and 2 receive: $k_{1-2}\{k'_{1-4}\}$, $k'_{1-4}\{k'_{1-8}\}$
User 3 receives: $k_3\{k'_{3-4}\}$, $k'_{3-4}\{k'_{1-4}\}$, $k'_{1-4}\{k'_{1-8}\}$
User 4 receives: $k_4\{k'_{3-4}\}$, $k'_{3-4}\{k'_{1-4}\}$, $k'_{1-4}\{k'_{1-8}\}$
Users 5 to 8 receive: $k_{5-8}\{k'_{1-8}\}$.

User leaves are treated in a similar way. In Figure 2.3, user u_8 is leaving the system, hence the whole key-path from her leaf to the root is refreshed: k'', k''_{5-8} and k''_{7-8}. The messages sent are the following:
Users 1 to 4 receive: $k'_{1-4}\{k''_{1-8}\}$
Users 5 and 6 receive: $k_{5-6}\{k''_{5-8}\}$, $k''_{5-8}\{k''_{1-8}\}$
User 7 receives: $k_7\{k''_{7-8}\}$, $k''_{7-8}\{k''_{5-8}\}$, $k''_{5-8}\{k''_{1-8}\}$.

For balanced trees the number of required multicast messages for a single join operation is $2h-1$ while the new member must receive $h+1$ keys. A leave requires $2h$ keys updated. Appropriate tree configurations for optimal bandwidth usage are discussed in [18] and [17].

Some improvement in storage overhead at the Key Server can be achieved if members are grouped within clusters of fixed size (say c). The Clustering approach [14] uses this idea: every cluster is placed at a tree leaf and assigned a KEK. Member departures, however, pose an important drawback since they require one by one key encryptions withing the affected cluster.

Subsequent efforts seek to reduce the number of messages sent by LKH. A widely adopted approach is to use one-way functions or combinations of one-way functions with pseudo-random number generators (PRNGs). On one hand, LKH+ [102], LKH++ [26] and the recent SKD [58] use this approach *in a temporary manner*: new keys are derived from old ones by means of a one-way function, independently of their position within the tree. LKH++ and SKD approximately halve LKH's bandwidth usage in join operations. Additionally, SKD claims a large reduction of the computational effort required at the Key Server.

On the other hand, One-way Function Trees (OFT) [94], One-way Function Chain Trees (OFCT) [13] and ELK [88] use the one-way function approach in a spatial manner: every key in the tree is derived from its children, therefore the tree is built in a bottom-up fashion starting at user keys (which are randomly picked by the Key Server). These schemes assume one-way functions to be perfect, but Horng proposed a collusion attack against OFT in [43] which was taken into account by Ku and Chen in order to develop an improved collusion resistant version of that protocol [56]. ELK addresses bandwidth reduction and reliability simultaneously: a communication-computation tradeoff is introduced and some degree of tolerance against message losses is achieved. Besides, members need no aid from the Key Server to derive their key path and no multicast messages are required in joins if batch re-keying is used (see the end of Section 2.2.3 for more information on this).

The Flat Table (FT) approach [19] (contemporary to LKH) also uses a hierarchical key tree arrangement though there are subtle differences which allow FT to refresh exactly the lowest possible number of keys (since trees are not always balanced) [89]. Whenever one or more members leave the system, the Key Server uses a well-known boolean function minimization technique that indicates the optimal number of encryptions (and which KEKs to use) to deliver the new session key to the remaining members. However, vulnerabilities against collusion attacks were discovered as stated in [114]. In response to them, the recent EGK scheme [114] retakes the Flat Table idea and smartly solves the problem. Its authors claim to obtain storage-communication-optimality and constant small message sizes.

Another tree-based approach is the subset-difference (SDR) scheme [68]: in this case the tree is covered by the minimal number of subsets needed to leave all revoked members out at a given time, each subset receiving a different key. In every

refreshment, the data encryption key is ciphered with the keys of every subset at that time. Remaining members calculate which subset they belong to and use the proper key to decrypt the refreshment message. This scheme performs nicely when the set of renovated members is relatively small and stable; however its performance falls down with time, given that all revocations since the beginning must be taken into account in every re-key operation. An extension to SDR, the LSD scheme [38], makes possible a tradeoff between communication overhead and member storage. Both SDR and LSD are stateless (see Def. 8).

As a last remark we note that statistical improvements can be applied to member arrangement in tree-based schemes in order to gain efficiency and scalability [84].

2.2.2 The computational approach

A different group of general purpose schemes uses algebraic calculations to securely provide an encryption key to a given set of members. These solutions are usually stateless since the current encryption key can be obtained from scratch upon the reception of a re-key message. This kind of schemes usually combine member secrets (like a private key or a secret prime owned by each member) with the re-key information. As a result, only the authorized owners of those secrets can recover the re-key information.

Probably the most popular scheme of this kind is SecureLock [21]. In it, the Key Server builds a system of linear congruences (each of them including the keying material and a member's public key) and solves using the Chinese Remainder Theorem. The solution is then broadcast to the members, which are able to obtain the keying material with the aid of their respective private keys. SecureLock is a simple and smart solution, however the computational effort required to solve the Chinese Remainder Theorem soon becomes excessive when increasing the number of members.

The work in [110] by Yoon et al. combines the RSA problem [1] with the use of one-way functions in order to obtain an efficient solution and short re-key messages. In their paper, the authors propose the scheme as a building block for a grid computing platform, but clearly it can be used for other multicast purposes.

This work is an improvement on the proposal in [63] given that the latter had several security problems.

A different idea is used in [106]. Its authors use Maximum Distance Separable (MDS) Codes, a class of error control codes, to prevent unauthorized access to the re-key information multicast by the Key Server. The main idea is to assign a secret key to every user in the system, and to construct a code word from the all the authorized keys at a given moment plus the chosen session key. Upon reception of the code word, every member with an authorized secret key can obtain the session key. One can see this as a similar solution to [110], however MDS encoding is faster than modular exponentiation as shown in [106].

Regardless of the specific computational trick used, schemes within this category can not handle audiences as large as the hierarchical tree does due to the increment in the computational effort required as group size increases. Therefore it is typical to assume the use of the computational approach to provide service to small to medium sized member clusters managed by a super-member. These clusters are assumed to be connected in a hierarchical way by one of the tree constructions explained before. In this way, greater audiences can be handled at the cost of obtaining a stateful solution. For example, the authors of [92] combine SecureLock with LKH in order to reduce the number of congruences involved in computations (there is a congruence per cluster). The authors of MDS [106] propose its combination with LKH in their paper, too. Enhancing the efficiency of SecureLock from a fully computational perspective is an interesting future work line.

To finish, let us remark that the work presented in Chapter 3 falls within this category.

2.2.3 Considerations on general-purpose schemes

Many of the protocols reviewed above can handle large, dynamic audiences in many multicast services that demand privacy. Regarding security, they are secure so far except for FT [19] and OFT [94] (which are vulnerable to collusion) and [63] (vulnerable to several attacks). Concerning reliability we must remark that LKH was introduced in the late 90's, a time when the main part of communications were

	Tree/ Comp	Storage overhead		Communications overhead		
		Key Server	Member	Join		Leave
				Multicast	Unicast	Multicast
GKMP [39], 1997	Tree	$2b$	$2b$	$2b$	$2b$	$(n-1)b$
Clusters [14], 1999	Tree	$\frac{n}{c}\frac{d}{d-1}+\frac{n}{c}$	$log_d(\frac{n}{c})+2$	$c-1+d\,log_d(\frac{n}{c})$	$log_d(\frac{n}{c})+2$	$c-1+d\,log_d(\frac{n}{c})$
LKH [103, 105], 1999	Tree	$(2n-1)b$	$(h+1)b$	$(2h-1)b$	$(h+1)b$	$2hb$
ELK [88], 2001	Tree	$(2n-1)b$	$(h+1)b$	0	$(h+1)b$	$h(b_1+b_2)$
SDR [68], 2001	Tree	-	$0.5log^2n$	$(2r-1)b$	$0.5log^2n$	$(2r-1)b$
LKH++ [26], 2002	Tree	$(2n-1)b$	$(h+1)b$	$b+log_2n$	$(h+1)b$	$log_2n+(h-1)b$
OFT [94], 2003	Tree	$(2n-1)b$	$(h+1)b$	$(h+1)b$	$(h+1)b$	$(h+1)b$
SKD [58], 2009	Tree	$\frac{(dn-1)b}{d-1}$	hb	h	hb	$(d-1)hb$
EGK [114], 2010	Tree	log_2n	log_2n	b	$2b$	log_2n
SecureLock [21], 1989	Comp	nq	q	nq	0	nq
MDS [106], 2008	Comp	nb	b	nb	0	nb
Yoon [110], 2011	Comp	$2qn$	$2q$	$qn+q+H$	0	$qn+q+H$

Table 2.1: General-purpose secure multicast schemes comparison. See page 10 for notation.

held on dedicated links. That is the main reason why its extending proposals normally do not address the problem of recoverability and therefore they are stateful and not self-healing. Probably ELK is the most reliable protocol in this family.

Table 2.1 shows a comparison in terms of storage and communication costs of the most important schemes along with recent proposals. Data are expressed in bits; worst cases are shown. User keys are assumed to be sent in a separate channel and are not considered since they are delivered only once per user.

There are small but subtle variations in the results shown. It can be seen that

older tree-like schemes focus on reducing communications in re-key operations, while acquiescing in linear storage needs. More recent tree-like proposals focus on the latter, given that acceptable bandwidth usage results were already obtained. Other reasons for storage reduction are the popularization of smart devices (with low storage capabilities) and ever increasing audiences as multimedia multicast services become more and more popular.

In the computational approach it can be observed that the length of the multicast re-key information is always linear with the number of users. That is clearly a drawback which restricts the use of these schemes to reduced groups. As mentioned above, a typical approach is to split the audience into small enough groups, which are in turn joined together by a tree.

It is interesting to remark that some of the mentioned schemes may perform batch re-key operations [41], i.e. on a timely manner, rather than triggered by joins or leaves.

Definition 12 Batch re-key *operations are performed by the Key Server at a given fixed frequency (say, after a few minutes) rather than being triggered by a single join or leave. Batch re-key operations take into account both the new and evicted members since the last re-key.*

This strategy helps to reduce the burden of re-keying in high churn scenarios and can be used in every scheme, even if the authors do not make it explicit.

The authors of this survey do not expect great new improvements in the LKH category. However, new computational developments might appear given the increasing power of multi-core processors.

2.3 Multi-group schemes

There exist scenarios in which several, different information channels are encrypted separately and reach different, not disjoint groups of members. Typical examples are multimedia platforms with several pay-per-view channels and communications in hierarchically managed networks. Schemes shown next can be seen as an extension of the tree approach: multiple trees are built from a single, global set of leaves, thus obtaining several roots. Figure 2.4 shows an arrangement example. Given that

	Storage overhead		Communications overhead		
	Key Server	Member	Join		Leave
			Multicast	Unicast	Multicast
MG [98, 99], 2007	$O(\log n)$	$O(Mn)$	$O(d \log n)$	$O(Md \log n)$	$O(Md \log n)$
HAC [112], 2008	$O(\frac{d}{d-1}Mn_0)$	$O(\log n_0)$	0	$O(1)$	$O(d \log n_0)$
Zhang et al. [111], 2006	$O(n)$	$O(1)$	0	$O(M)$	$O(M)$

Table 2.2: Multi-group secure multicast schemes comparison. See page 10 for notation.

a single member may now belong to more than one group, her key path includes all keys from her leaf to the different roots it is connected to. Therefore re-key operations will normally affect more than one tree. However, note that not all users are always connected to all roots. The pioneer proposal, the MG scheme, appears in [98] and [99]. Due to the intricate resultant network, single re-key operations become significantly complex in terms of overhead and therefore batch re-keyings are used. The HAC scheme [112] reduces both bandwidth and communications overhead by improving the multi-tree arrangement and using one-way functions. Both schemes are stateful. Zhang et al. present a stateless protocol based on the bilinear Diffie-Hellman Problem [111].

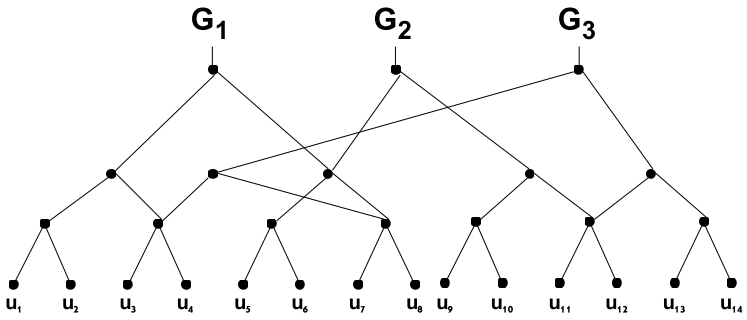


Figure 2.4: **A multi-group tree.** Users u_1 to u_{14} are arranged into three overlapping groups: G_1, G_2 and G_3.

Table 2.2 shows a comparison of [99, 111, 112] and [113] in $\mathrm{big}{-}O$ notation terms. M is the number of groups/trees and n_0 the average number of members in a subgroup. Other multi-group schemes are [107] and [53]. The authors of the

latter claim a reduction of up to 15.2% of communication overhead against MG by using trees with different degrees.

2.4 Self-healing schemes for ad-hoc networks

Last decade advances on smart devices have led to the development and wide use of ad-hoc networks, characterized by unreliable links, limited bandwidth, highly dynamic topologies and unpredictable member behavior (there is no guarantee that members will be online at a given time or for a given period of time). What's more, most of the devices used in these networks have low computational capabilities given their need to manage energy efficiently. Due to the specific restrictions that ad-hoc networks impose, re-keying is usually performed on a batch manner. An interval between two batch re-keys is called a *session*. Zhu et al. present one of the earliest schemes in [116]. They add reliability and self-healing to the aforementioned SDR general-purpose scheme ([68]) in order to obtain *m-recoverability* at a low additional bandwidth overhead. Self-healing schemes may be split into three main families according to the cryptographic technique they use [104].

Revocation polynomials

The first family uses *revocation polynomials* and secret sharing techniques. The pioneering work in self-healing schemes, presented by Staddon et al. [95], falls within this category. It combines the transmission to members of an initial set of shares with the transmission of additional, redundant shares in every update round. With that information, members are able to recover a given key even if they miss its corresponding update. The price is an increase in bandwidth overhead, up to $O(mt^2)$: m is the maximum number of updates a member can miss, while t is both the polynomial degree and the minimum number of ex-members that must collude in order to break the system. More et al. [67] found some reliability problems in [95]. Also Blundo et al. [11] made some criticism on [95] concerning security and proposed an alternative, more secure and efficient solution that reduces bandwidth overhead to $O(tj)$. The value j is the current session within the interval m: note that the re-key information transmitted is a multiple of j and therefore increases with time, to a maximum of m times. Other proposals are Liu et al. [60] and

Hong et al. [42]: the latter slightly reduces Blundo's requirements. Generally speaking, this family of schemes is secure enough against collusion, but requires large messages for re-keying.

Access polynomials

A different family uses *access polynomials*, which are formed by a product of terms $(x - ID_\alpha)$ for every legal member α plus the re-key material (ID_α is a unique user identifier in numeric format for member α). When a legal member α evaluates the polynomial on ID_α the re-key material is recovered. Schemes of this kind have been proposed by Zou et al. [117] and Tian et al. [100]. They claim to obtain optimal storage requirements: only 2 keys must be stored by every member. However, their main drawback lays on message lengths, which are linear on the number of members. As an anecdote, the authors of [100] point out an error in the communication overhead datum provided by Hong et al. [42], which they fix (the correct value is shown later in Table 2.3). Additionally, [117] allows to revoke users on a temporary manner: other previous schemes like [11] and [60] do not support that feature (revoked members can not rejoin).

One way functions

The last family of self-healing schemes relies on the use of one way functions. The scheme in [32] by Dutta et al. achieves a better bandwidth usage, constant member storage requirements, presumably unconditional security and is not restricted to only m sessions recoverability. However, Du et al. later revealed security weaknesses in [32] and proposed an improved, collusion-free protocol [29]. Finally, the same authors recently proposed another constant storage scheme [30] but they did not guarantee its resistance to collusion. Generally speaking, schemes in this category perform nicely in terms of member storage and communication overhead but are vulnerable to collusion attacks so far.

2.4.1 Considerations on self-healing schemes

Table 2.3 compares the schemes found in [11, 30, 32, 42, 60, 95, 100, 117] focusing on their storage requirements at the member and the communication overhead per

	Storage at member	Communication overhead	Collusion resistant	Key long life-span
Staddon et al. [95], 2002	$(m-j+1)^2\ log\ q$	$(mt^2+2mt+m+t)\ log\ q$	Yes	No
Blundo et al. [11], 2004	$(m-j+1)\ log\ q$	$(2tj+j)\ log\ q$	Yes	No
Liu et al. [60], 2003	$2(m-j+1)\ log\ q$	$(mt+jt+t+m+1)\ log\ q$	Yes	No
Hong et al. [42], 2005	$(m-j+1)\ log\ q$	$(tj+j+t)\ log\ q$	Yes	No
Zou et al. [117], 2006	$2\ log\ q$	$(t+n+3)\ log\ q$	Yes	No
Tian et al. [100], 2008	$2\ log\ q$	$n+2\ log\ q$	Yes	No
Dutta et al. [32], 2007	$3\ log\ q$	$(t+1+j)\ log\ q$	No	Yes
Du et al. [30], 2009	$3\ log\ q$	$(3t+2+j)\ log\ q$	No	Yes

Table 2.3: Self-healing secure multicast schemes comparison. See page 10 for notation.

key update (q is a large prime involved in calculations, greater than the audience size n). Data are expressed in terms of bits. Given the limited memory space of smart devices constant storage requirements are desirable: those schemes using access polynomials or one-way functions ([30, 32, 100, 117]) offer the best results in that sense.

Regarding communication overheads, one way function schemes ([32] and [30]) show the best results. However, their vulnerability to collusion attacks makes them a weak option to choose. Conversely, access polynomial schemes ([117] and [100]) exhibit a very high overhead since they depend on the number of members. That is clearly a drawback. Revocation polynomials show an intermediate communication overhead that, combined with their resistance to collusion, probably makes them the best option. Note that in most cases overhead depends among other values on the security parameter t, which results in a tradeoff between security and message length.

The "Key long life-span" column refers to member keys lifetime: most of the schemes require the member key to be renewed after m sessions. That is another drawback since it increases the workload at the Key Server side and bandwidth

usage, specially in not reliable networks which is the case.

To conclude, we believe that self-healing schemes still have several challenges to overcome, mainly: costly setup phases that must be repeated after m sessions, generalizing a constant use of storage and going further on bandwidth usage reduction while maintaining security and, finally, overcoming the usual security-communication overhead tradeoff. The author of [104] states that almost none of the schemes proposed so far is able to cope with real use conditions in large mobile or wireless sensor networks, and suggests the proposal in [42] as the most balanced. Finally, let us refer the reader to [12, 115] for a wider analysis on this specific group of schemes.

2.5 Conclusions

This chapter provides a survey on the Centralized Secure Multicast field that includes some of the latest proposals. Table 2.4 compares the different schemes reviewed according to their features:

- category (either general-purpose, multi-group or self-healing),
- whether stateful or stateless,
- whether secure so far or not, and
- whether a hierarchical tree of keys is used or not.

The main challenges that arise for the future come from the fact that Internet and ad-hoc networks have conquered the *small world*: smart devices with low computational and energy resources that operate in mobile, highly dynamic, sometimes infrastructure-less environments. Researchers therefore face the challenge of designing new protocols that can provide an acceptable degree of security, efficiency and flexibility for such scenarios. This translates into new requirements for a centralized secure multicast scheme: (1) reductions of the number and frequency of communications with the Key Server, (2) recoverability from information loss and (3) lightweight computations are highly desirable features for future proposals.

	Category	Stateful/ Stateless	Secure	Reliable	Keys tree
SecureLock [21], 1989	General	Stateless	✓		
GKMP [39], 1997	General	Stateful	✓		
Cluster [14], 1997	General	Stateful	✓		✓
LKH [103, 105], 1999	General	Stateful	✓		✓
LKH+ [102], 1999	General	Stateful	✓		✓
OFCT [13], 1999	General	Stateful	✓		✓
FT [19], 1999	General	Stateful			✓
ELK [88], 2001	General	Stateful	✓	✓	✓
SDR [68], 2001	General	Stateless	✓		✓
LSD [38], 2002	General	Stateless	✓		✓
LKH++ [26], 2002	General	Stateful	✓		✓
SecureLock+LKH[92],2002	General	Stateful	✓		✓
OFT [94], 2003	General	Stateful			✓
Ku [56], 2003	General	Stateful	✓		✓
MDS [106], 2008	General	Stateless	✓	✓	
MDS+LKH [106], 2008	General	Stateful	✓		✓
SKD [58], 2009	General	Stateful	✓		✓
EGK [114], 2010	General	Stateful	✓		✓
Yoon [110], 2011	General	Stateless	✓	✓	
Zhang [111], 2006	Multi-group	Stateless	✓	✓	✓
MG [98, 99], 2007	Multi-group	Stateful	✓		✓
HAC [112], 2008	Multi-group	Stateful	✓		✓
Xia [46], 2009	Multi-group	Stateless	✓		✓
Yan [107], 2009	Multi-group	Stateful	✓		✓
Koo [53], 2009	Multi-group	Stateful	✓		✓
Hur [44], 2010	Multi-group	Stateless	✓	✓	✓
Staddon [95], 2002	Self-healing	Stateless		✓	
Zhu [116], 2003	Self-healing	Stateless		✓	✓
Liu [60], 2003	Self-healing	Stateless	✓	✓	
Blundo [11], 2004	Self-healing	Stateless	✓	✓	
Hong [42], 2005	Self-healing	Stateless	✓	✓	
Zou [117], 2006	Self-healing	Stateless	✓	✓	
Tian [100], 2008	Self-healing	Stateless	✓	✓	
Dutta [32], 2007	Self-healing	Stateless		✓	
Du [29], 2008	Self-healing	Stateless	✓	✓	
Du [30], 2009	Self-healing	Stateless		✓	

Table 2.4: Feature comparison for the different schemes reviewed.

Chapter

3

A centralized secure multicast solution

3.1 Introduction

This chapter describes a centralized secure multicast solution which was first introduced in [76] and later extended in [77]. According to the classification outlined in Chapter 2, it is a general-purpose computational solution. The following features are provided:

(a) private communications and efficient key refreshment,

(b) key server messages authentication, and

(c) validation among members.

Three different and complementary schemes are proposed in order to achieve those goals. They can be implemented as a whole solution or on their own depending on the addressed scenario and its requirements. The core operation in all three schemes is the Extended Euclidean Algorithm, which is briefly described next. The Chinese Remainder Theorem and the practical infeasibility of factorizing very large numbers are also used.

After the publication of [76, 77], Peinado et al. made a cryptanalysis of the whole solution in [86, 87] and exposed some security flaws. In response to it, Antequera et al. made some remarks on the seriousness of some of the flaws and proposed solutions to others [3]. Their respective works are cited and discussed later only for the sake of completeness: the author of this thesis has not participated in them.

The chapter is organized as follows. Section 3.2 describes the Extended Euclidean Algorithm and the Chinese Remainder Theorem. Section 3.3 describes the scenario conditions we assume for our solution. Section 3.4 presents the key refreshment scheme, as well as a security and efficiency discussion, a comparison with the state of the art and simulation results. Sections 3.5 and 3.6 introduce and discuss the schemes for key server messages authentication and verification among hosts, respectively. Section 3.7 briefly mentions some extensions to our scheme proposed by Antequera et al in [2]. Finally, the conclusions of the chapter are presented in Section 3.8.

We would like to emphasize that the work in [2, 3, 86, 87] is not part of this thesis and that it is referenced here for the sake of completion.

3.2 Algebraic background

This section describes two algebraic tools that will be useful later. The information shown in this section is mainly taken from the Handbook of Applied Cryptography [64].

3.2.1 The Extended Euclidean Algorithm

The computational solution proposed in this chapter requires the calculation of the *greatest common divisor* (gcd), which can be easily computed for small numbers with Alg. 1.

However, the factorization of large numbers is still an open problem and eventually becomes computationally unfeasible as the number to be factorized grows. Fortunately, the Extended Euclidean Algorithm (EEA, Alg. 2) finds the greatest common divisor in $O((log\ n)^2)$ time. What's more, it allows to compute the values

Algorithm 1 Greatest common divisor of two positive integers using their prime powers.

INPUT:

$a,\ b$: positive integers, with $a \geq b$.

OUTPUT:

$gcd(a, b)$: the greatest common divisor of a and b.

1. Let $p_1 \cdots p_k$ the first k distinct prime numbers.
2. Write a as $p_1^{e_1} \cdots p_k^{e_k}$, where $e_i \geq 0$.
3. Write b as $p_1^{f_1} \cdots p_k^{f_k}$, where $f_i \geq 0$.
4. $result \leftarrow p_1^{min(e_1,f_1)} \cdots p_k^{min(e_k,f_k)}$
5. Return $result$ as $gcd(a, b)$.

Algorithm 2 The Extended Euclidean Algorithm.

INPUT:

$a,\ b$: positive integers with $a \geq b$.

OUTPUT:

$gcd(a, b)$: the greatest common divisor of a and b.

$u,\ v$: integers such that $au + bv = gcd(a, b)$.

1. **if** $b = 0$ **then**
2. $gcd(a, b) \leftarrow a,\ u \leftarrow 1,\ v \leftarrow 0$
3. **return** $gcd(a, b),\ u,\ v$
4. **else**
5. $u_2 \leftarrow 1,\ u_1 \leftarrow 0,\ v_2 \leftarrow 0,\ v_1 \leftarrow 1$
6. **while** $b > 0$
7. $q \leftarrow \lfloor a/b \rfloor,\ r \leftarrow a - qb,\ u \leftarrow u_2 - qu_1,\ v \leftarrow v_2 - qv_1$
8. $a \leftarrow b,\ b \leftarrow r,\ u_2 \leftarrow u_1,\ u_1 \leftarrow u,\ v_2 \leftarrow v_1,\ v_1 \leftarrow v$
9. **endwhile**
10. $result \leftarrow a,\ u \leftarrow u_2,\ v \leftarrow v_2$
11. **return** $result$ as $gcd(a, b)$ and $u,\ v$
12. **endif**

$u,\ v$ that satisfy Bezout's identity shown in Theorem 1. We will take advantage of this fact later.

Theorem 1 Bezout's identity. *Let a and b be non-zero integers. There exist integers $u,\ v$ for which $au + bv = gcd(a, b)$.*

3.2.2 The Chinese Remainder Theorem

The Chinese Remainder Theorem (CRT) is described in Theorem 2.

Theorem 2 *Chinese Remainder Theorem. Given the pairwise relatively prime integers $n_1, \ldots, n_k$, the system of simultaneous congruences*

$$x = a_1 \ mod \ n_1$$
$$\vdots$$
$$x = a_k \ mod \ n_k$$

(with $a_1, \ldots, a_k$ also integers) has a unique solution modulo $n = n_1 \cdot \ldots \cdot n_k$.

3.3 Scenario

The scenario we address is the following: private communications are to be established within a restricted group. There is a *Key Server* in charge of key management issues. There is also a set of *members* which may communicate among them and/or with the Key Server, depending on the nature of the service. Therefore communications can be either one-to-many or many-to-many. Both forward and backward secrecy (see Defs. 1 and 2) are required.

Private communications are the main goal of a secure multicast solution. However, we have tried to exploit our solution in order to achieve authentication of re-key messages and members.

3.4 Distribution of secrets within closed groups

The first scheme allows the Key Server to generate and privately distribute a secret piece of information among a restricted audience. In our target scenario this sensible information can be used directly as an encryption key or fed to a Key Derivation

Function in order to derive a proper key (we will discuss this issue shortly). The most relevant features of the scheme are:

- Only one message is generated per re-key operation.
- Suitable for all topologies.
- No need for message re-encryption.
- Only one secret piece of info is held by each client. We call this pieces *member tickets*.
- Cost-effective and easy to deploy.

Let us assume r is the secret to be multicast, and that there are n members at a given time. The following paragraphs explain how the scheme works.

When a member i joins, the Key Server assigns her a member ticket, x_i. Every ticket is a different large prime[1] and is communicated to the corresponding member under a secure channel (e.g. SSL/TLS [27]). This communication is made once per member only, so it does not affect global efficiency. x_i is known only by its owner and the Key Server, and r is shared by all members and the Key Server as a result of executing the protocol. Algorithm 3 shows the generation and distribution of r.

New values for $m,\ g,\ p$ and/or k must be chosen for each refreshment of r. Note that $\delta,\ u$ and v depend on them and will change as they do. Some remarks can be made to the algorithm which are stated next.

Finding a proper g

First, a proper value g at step 1iii of Alg. 3 is easy to calculate: once the Key Server has chosen $m = p \cdot q + 1$, a value a is chosen satisfying that $m - 1$ is the least integer such that $a^{m-1} \bmod m = 1$ (that is, a is a primitive value from $\mathbb{Z}_m$). Then $g = a^q \bmod m$.

Using r as a symmetric encryption key

Second, the secret r can be treated as a symmetric key to be used along with any symmetric encryption algorithm such as AES [82]. However, the bit-length of

[1] Strictly, it is sufficient that all x_i are coprime and greater than δ. In that case, however, it would be necessary that every x_i has a large prime factor in order to make the factorization of L harder (δ and L are introduced in Algorithm 3).

Algorithm 3 The re-keying algorithm.

1. The Key Server selects:

   i. $m,\ p$, large prime numbers, such that $m - 1 = p \cdot q$.

   ii. k and δ, such that $\delta = k + p$ and $\delta < x_i$, for every $i = 1 \dots n$.

   iii. g that verifies $1 = g^p \bmod m$.

   The secret to be distributed is $r = g^k \bmod m$.

2. The Key Server calculates $L = \prod_{i=1}^{n} x_i$. L is kept private in the Key Server.

3. The Key Server finds $u,\ v$, by means of the Extended Euclidean Algorithm, such that

$$u \cdot \delta + v \cdot L = 1 \tag{3.1}$$

4. The Key Server multicasts (makes public) $g,\ m$ and u on plain text.

5. Each member i calculates $u^{-1} \bmod x_i = \delta$ and $g^\delta \bmod m = g^k \bmod m = r$.

r can be as much as that of m, therefore some preprocessing should be applied to r before feeding it to a fixed key-length symmetric encryption algorithm (m will be presumably longer than a typical symmetric key). A trivial solution would be to truncate r at the desired length, while the most prudent is to apply a Key Derivation Function (KDF) to r. KDFs return cryptographically strong, arbitrarily long symmetric key material that depends on the input provided. Two well known standards of KDF are the NIST SP 800-108 [20] and HKDF [54]. The multicast information should then be encrypted and decrypted with KDF(r).

Recomputing L

Third, fortunately it is not necessary to recompute L from scratch between two consecutive re-key operations: L is simply multiplied by the incoming members tickets and divided by those of the leaving members. This dramatically speeds the process up.

Long intervals without joins nor leaves

Fourth, the Key Server might decide to refresh r after a long period of time with no members joining or leaving for security reasons.

Disclosure of public keys

The protocol can be used for an additional purpose: the refreshment of asymmetric key pairs and the disclosure of the public part. Recall that the encryption key delivered to the audience has the form $r = g^k \bmod m$, which is similar to an Elgamal public key [33]. An Elgamal key pair has the form:

$$K_{pub} : g, m, g^k \bmod m$$
$$K_{priv} : k$$

Therefore, the protocol can be seen as a way of *controlling the disclosure of Elgamal public keys* if the scenario requires it: the public key is only communicated to a closed group of recipients.

3.4.1 Proof of correctness

Given that $\delta < x_i,\ i = 1 \dots n$ and with every x_i prime (or coprime at least), it is clear that:

$$gcd(\delta, x_i) = 1, \text{ for every } i = 1, \dots, n$$

and hence,

$$gcd(\delta, L) = 1 \tag{3.2}$$

Equation (3.2) ensures, by the Extended Euclidean Algorithm, the existence of $u, v \in \mathbb{Z}$ such that $\delta \cdot u + v \cdot L = 1$, from where it is deduced that $\delta \cdot u = 1 \bmod x_i$ and so $u^{-1} = \delta \bmod x_i$, for every $i = 1, \dots, n$. The Chinese Remainder Theorem guarantees that the solution for $u^{-1} \bmod x_i = \delta$ and $\delta < x_i$, for every $i = 1, \dots, n$ is unique.

The value $r = g^k \bmod m$ is obtained as shown next (recall step 1iii of Alg. 3):

$$\begin{aligned} g^{\delta} &= g^{k+p} \bmod m \\ &= g^k \cdot 1 \bmod m \\ &= g^k \bmod m \end{aligned}$$

g is public, but the use of δ assures that an outsider will not be able to guess k and, therefore, r.

3.4.2 Efficiency considerations on the re-key scheme

Kruus [55] suggests five issues that a multicast key management protocol must address. They are:

1. efficiency in initial keying,
2. efficiency in re-keying,
3. computational requirements,
4. storage requirements,
5. scalability.

There is no difference in our scheme between first time keying (requirement 1) and further re-keying operations. Re-keying operations are simple (requirement 2): the Key Server generates a single message which is injected into the multicast network on plain text, since only authorized members will be able to process it correctly. Requirements 3, 4 and 5 are discussed next.

We can observe that L will be large, given that $L = \prod_{i=1}^{n} x_i$. So will be u (recall Eq. (3.1)). In order to estimate it, assume for the rest of the chapter that every x_i value is stored in an unsigned binary data type of b bits. The greatest value that can be represented is $2^b - 1$. Assume also there are n members. The maximum length of L is then $n \cdot b$ bits. That is also the maximum length of u.

From the previous consideration we assume that Kruus' requirements 3 and 5 are the weakest points of our scheme. The solution that allows to overcome these

problems consists of logically dividing the audience into s disjoint subgroups while delivering the same encryption key to all of them. This can be done by running a different instance of the algorithm for every subgroup, with some variations: values m, g, p and k are the same for every subgroup j (and so is δ), but each partial product, L_{part_j} with $j \in [1, s]$ from now on, contains the product of the tickets of the members within the given subgroup only. With this modification the same key is delivered to every subgroup by means of several shorter messages that are disseminated throughout the network, which is more convenient in terms of efficiency. Now every peer needs only to process the short message addressed to its subgroup.

However, the join and leave operations still require the whole set of members to obtain a new key, therefore s refreshment messages (g, m and the corresponding u) must be computed and multicast now; each one for a different subgroup. Figure 3.1 shows an example: note that all subgroups receive the same g and m (which guarantees that the same key is obtained) but a different u, due to the use of different L_{part} values, $L_{part1} = x_A x_B x_C x_D$, $L_{part2} = x_E x_F x_G x_H$ and $L_{part3} = x_I x_J x_K x_L$. It is important to remark that such division is logical and independent from the topology of the network, that is, the underlying topology is not affected, neither dissemination of the encrypted information nor re-key messages. Only the Key Server is aware of the global arrangement, while a given peer knows only which subgroup it belongs to. However, in the case that the network topology is naturally divided through time (such as a regional division, for example) a topology-aware logical arrangement may be used.

Adopting the subgroups approach brings many benefits, even though the final bandwidth requirement does not change. First, it is obvious that, for a fixed number of members, the length of u values decreases linearly as the number of subgroups increases. Second, the message generation process that takes place at the Key Server can be sped up. Every different u can now be computed by a separate process, which may run concurrently with the others. This is specially appropriate for current multi-core processors. The whole process can be sped up by almost s times if the software is properly tuned. As a consequence, a better scalability is achieved, allowing to increase the maximum number of members that can be handled. Let us remark that users should be assigned to subgroups in a balanced way in order to keep refreshment messages as short as possible. This raises other

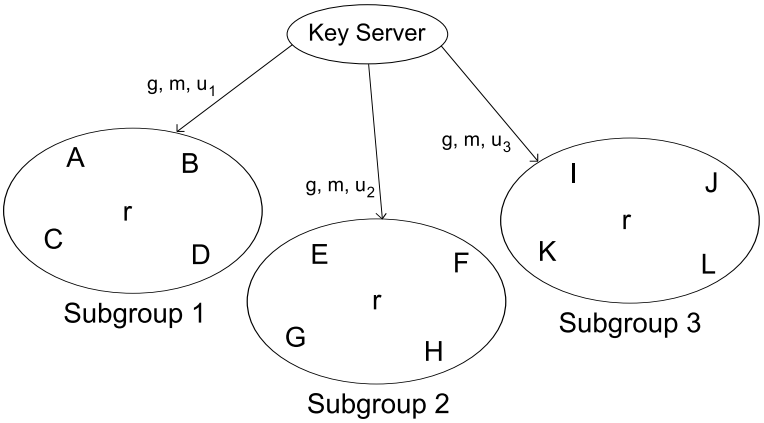


Figure 3.1: **The subgroups extension to the re-key scheme.** Capital letters denote members. r is the multicast secret.

issues, such as the problem of re-balancing subgroups after a leave avalanche, for example.

3.4.3 Security considerations on the disclosure scheme

Security in the distribution of r relies on the infeasibility of calculating the right δ in a reasonable time if a valid x_i is not known by the attacker (recall that values for Eq.(3.1) in Algorithm 3 are unique). The privacy of k and p is guaranteed if:

- a sufficiently large value is chosen for m,
- p and q have a similar bit-length (recall that $m - 1 = p \cdot q$).

In that case factorizing $m - 1$ will be more difficult. Additionally, a strong prime[1] can be chosen for m. Next, security is discussed considering three different types of attacker.

[1]From [64]: a prime number p is said to be *strong* if three conditions are satisfied,

i. $p - 1$ has a large prime factor, say f,
ii. $f - 1$ has a large prime factor, and
iii. $p + 1$ has a large prime factor.

Security against an outsider

First, we assume the presence of an adversary who neither has nor has had a valid ticket. Her intention is to learn about the secret value that is being disclosed to legal members. Those members, as was explained in step 5 of Algorithm 3, compute a modular inverse by using their corresponding ticket as the modulo. Let us assume that a non-member obtains a re-key message, i.e. $g,\ m,\ u$ on plain text. In order to guess r, the non-member must discover the value δ used by the Key Server. Now, if we take into account that δ is the unique solution of the congruence system $x = u^{-1} \bmod x_i$ in the interval $[0, L-1]$, then the calculus of δ implies knowing L or one valid ticket x_i. L is kept private in the Key Server but, even if it were public, its factorization is still a challenge as long as long enough tickets were used. On the other hand, in this attack we assume that the attacker does not know any valid ticket. Therefore, the only option to the best of our knowledge is a brute-force attack on r.

Security against an insider

Legal members might be interested in compromising other authorized member tickets. Consider such an attacker: she knows that $u \cdot \delta = 1 \bmod L$, i.e., L divides $u \cdot \delta - 1$. Then by factoring $u \cdot \delta - 1$ she would get all factors in L (the tickets). As in the previous attack, this factorization problem is computationally difficult to solve if tickets are long enough.

Security against an ex-insider

Finally, a former user who still holds her old ticket might intercept a refreshment message hoping that her ticket was reassigned to a new member. If that was the case it is clear that she would be successful in recovering the secret r. The way to prevent this attack is to never reuse tickets (or for a large period of time at least).

3.4.4 Security considerations on the disclosure scheme by Peinado et al. and Antequera et al.

As we mentioned earlier, Peinado et at. analyzed the whole solution in [86, 87]. Regarding the disclosure scheme, they claim to have found a security flaw by obtaining a multiple of the secret value L, or even the value itself with some probability. This can then be used for two different purposes: forging fake refreshment messages and (presumably) recovering user tickets.

For every attack we also show comments from Antequera et al. in their response work [3]. The notation used here is similar to that used in [87].

3.4.4.1 Attack 1

Let us recall Eq. 3.1

$$u \cdot \delta + v \cdot L = 1$$

After one execution of the key refreshment scheme, user h gets $\delta = u^{-1} \, mod \, x_h$, and consequently can obtain vL by simple manipulation of Eq. 3.1.

$$v \cdot L = 1 - u \cdot \delta. \tag{3.3}$$

The value $v \cdot L$ can be used to forge new fake key refreshment messages as explained in [87]:

1. Member h generates a new value $\delta' < \delta$ and computes u' and v' by means of the Extended Euclidean Algorithm such that

$$u' \cdot \delta' + v' \cdot (v \cdot L) = 1 \tag{3.4}$$

2. Member h sends $g,\ m$ and u' to every other member i. Such member i will obtain the new value $\delta' = u'^{-1} \, mod \, x_i$, and compute the refreshed key from it.

The result is member h controlling the key refreshment process.

3.4.4.2 Remark to attack 1

The attack above, while correct and smart, deals with impersonating the Key Server which is a complementary issue as stated in [3]. Every secure multicast scheme has a similar problem if no authentication measure is taken. As an example, take the Logical Key Hierarchy shown in Section 2.2.1: every member knows the whole key path to the root and can thus mislead all other members within the same branch.

A simple and standard countermeasure to this attack is therefore to add a digital signature to every refreshment message from the Key Server.

3.4.4.3 Attack 2

This attack deals with recovering the value L with the aid of two consecutive refreshment messages. First, let us assume that the members set has not changed in the meantime: this implies that the same L was used and thus the malicious member h can use two equations, each one corresponding to one refreshment.

$$\begin{array}{rcl} v \cdot L & = & 1 - u \cdot \delta \\ v' \cdot L & = & 1 - u' \cdot \delta' \end{array} \tag{3.5}$$

Therefore, h knows two different multiples of L, i.e., $v \cdot L$ and $v' \cdot L$. The greatest common divisor of them is L with high probability.

Second, if there was any change in the members set between the two refreshments then h obtains the following equations,

$$\begin{array}{rcl} v \cdot L & = & 1 - u \cdot \delta \\ v' \cdot L' & = & 1 - u' \cdot \delta' \end{array} \tag{3.6}$$

with L and L' different, but sharing many common factors (the tickets of the remaining members). The greatest common divisor of L and L', say L_{sub}, corresponds to the product of those remaining tickets. This process can be repeated with the hope that eventually L_{sub} will be formed by a single ticket in the case that only one member from the original set remains.

3.4.4.4 Remark to attack 2

In the first case, when L does not change, factoring L is still a problem for the attacker. The use of 1024 bits long primes already makes the factorization of L

computationally not affordable nowadays. Peinado et al. use a genetic algorithm for the factorization of L when tickets are 64 bits long in [87]. This short size is far from secure nowadays.

In the case of L_{sub}, Antequera et al. suggest to include an unassigned extra ticket x_* in L, so in the worst case we have $L_{sub} = x_i \cdot x_*$, which is still an intractable problem for long enough primes. Therefore no assigned ticket will be disclosed regardless of the size of L_{sub}.

3.4.5 Experiments

We have developed a Java 1.7 implementation of the scheme in order to obtain execution times. The BigInteger Java class was used for handling large numbers, and the Miller-Rabin test [64] was employed to check primality. The hardware used was an Intel Core i5 with 4GB RAM and four cores, each one at 3.33GHz. However, parallelization was not exploited: a single core was used. Each experiment was run ten times: in the following tables we show the average and the standard deviation per experiment.

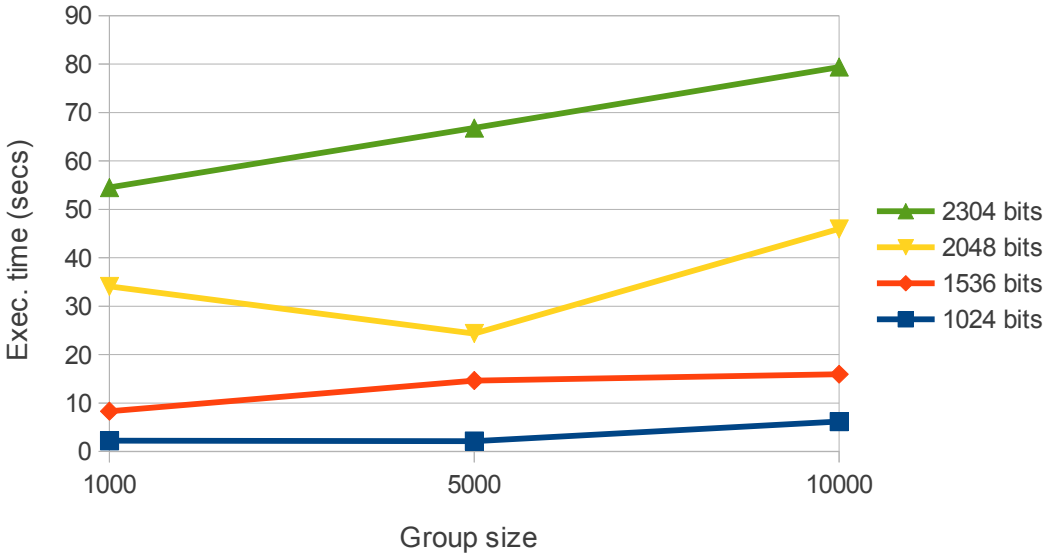


Figure 3.2: **Key server total execution time.**

		Audience size		
		1000	**5000**	**10000**
1024 bits tickets	**Total**	2.2218	2.1433	6.1864
	Total std	2.1865	1.2030	4.4841
	MPQ	2.1906	2.0602	6.0470
	MPQ std	2.1866	1.2032	4.4834
	Total - MPQ	**0.0312**	**0.0830**	**0.1395**
	Total - MPQ std	0.0012	0,0017	0.0033
1536 bits tickets	**Total**	8.3064	14.6533	15.9610
	Total std	6.736	16.508	17.157
	MPQ	8.2253	14.4729	15.6535
	MPQ std	6.7362	16.508	17.159
	Total - MPQ	**0.0811**	**0.1804**	**0.3075**
	Total - MPQ std	0.0003	0.0018	0.0073
2048 bits tickets	**Total**	34.0932	24.3556	45.9582
	Total std	27.49	15.846	23.436
	MPQ	33.9226	24.0146	45.4015
	MPQ std	27.49	15.847	23.44
	Total - MPQ	**0.1707**	**0.3409**	**0.5567**
	Total - MPQ std	0.0007	0.0014	0.0079
2304 bits tickets	**Total**	54.5011	66.8079	79.3990
	Total std	31.828	38.907	51.583
	MPQ	54.2676	66.3647	78.6849
	MPQ std	31.827	38.906	51.584
	Total - MPQ	**0.2334**	**0.4432**	**0.7142**
	Total - MPQ std	0.0005	0.0018	0.0035

Table 3.1: Key Server execution times in seconds for different ticket lengths and audience sizes.

Table 3.1 shows execution times for the Key Server part of the Algorithm 3 with different ticket bit-lengths and audience sizes. The total figures (rows **Total**) are also depicted in Figure 3.2. In order to obtain a better understanding of the composition of total time results we have broken them down into MPQ time and (Total - MQP) time: MPQ refers to step 1i in Algorithm 3, i.e., finding two primes m, p such that $m - 1 = p \cdot q$. For this, we used the method to find strong primes suggested in [37]. This method has proven to be relative slow for large bit-lengths

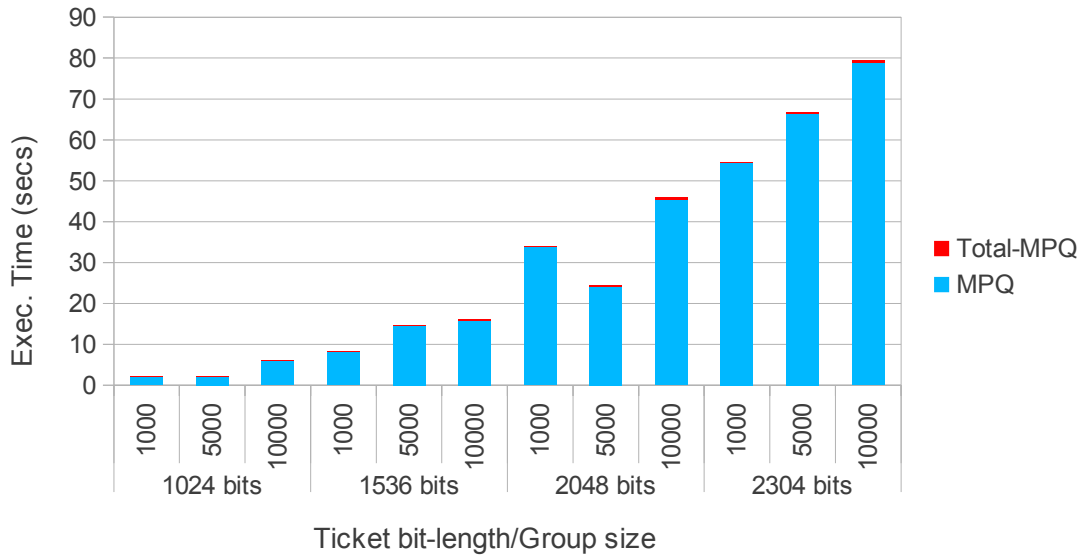


Figure 3.3: **Key server total execution time breakdown.**

(see rows **MPQ**) and its execution times are not deterministic as the standard deviation indicates (see rows MPQ std). Figure 3.3 shows this breakdown: we can see the main part of the execution is spent in searching for strong primes for m and p, while the rest of the algorithm consumes a negligible part of the total time.

Fortunately, the MPQ part of the algorithm is completely independent and can be run apart in a different thread. Thus, the Key Server may concurrently generate m, p, q threesomes in advance so they are immediately available for the re-keying algorithm when needed. Rows **Total - MPQ** and Figure 3.4 show the execution time of the re-key algorithm without the MPQ part: it is small for all bit-lengths and audience sizes and suggests that the algorithm can cope with high re-key rates.

Regarding the key recovery part of the algorithm run at the member, Table 3.2 and Figure 3.5 show the total execution times. These are also small, which is a positive feature since member hardware can vary from computers to low performance set-top boxes (dedicated hardware).

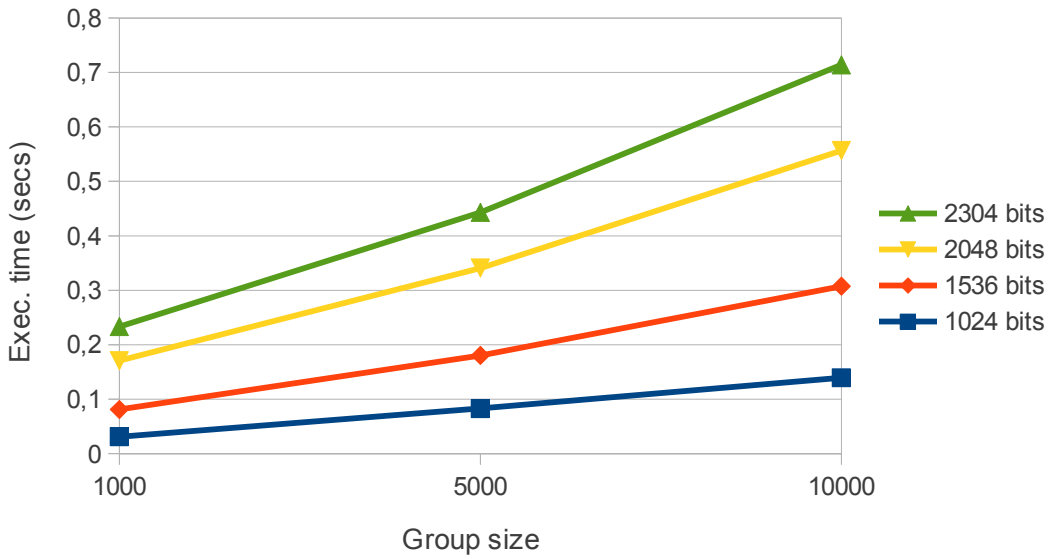


Figure 3.4: **Key server Total - MPQ execution time breakdown.**

		Audience size		
		1000	**5000**	**10000**
1024 bits	**Total**	**0.0267**	**0.0631**	**0.1127**
tickets	Total std	0.0276	0.0300	0.0322
1536 bits	**Total**	**0.0464**	**0.1290**	**0.2327**
tickets	Total std	0.0275	0.0319	0.0354
2048 bits	**Total**	**0.0841**	**0.2219**	**0.3954**
tickets	Total std	0.0371	0.0294	0.0305
2304 bits	**Total**	**0.1064**	**0.2791**	**0.4925**
tickets	Total std	0.0291	0.0282	0.0283

Table 3.2: Member execution times in seconds for different ticket lengths and audience sizes.

3.5 Key refreshment message authentication

At this point we have achieved privacy in multicast communications. This section presents a mechanism that authenticates the refreshment messages from the

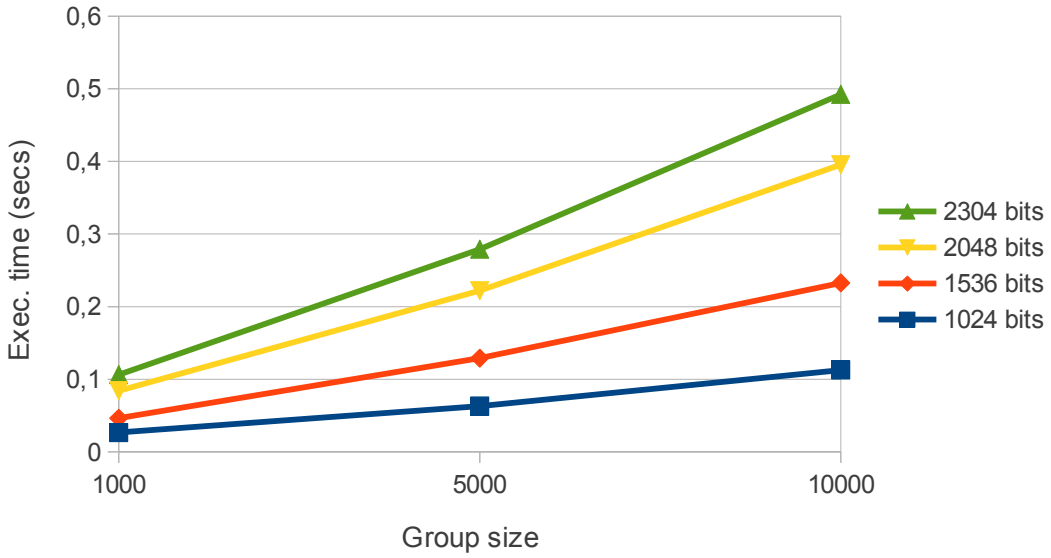


Figure 3.5: **Member total execution time.**

Key Server: such mechanisms are required to protect the system against forged re-keying messages. The usual technology for message authentication is digital signature: a hash of the message is encrypted with the sender's private key. The receiver can then decrypt the hash and compare it with its own result of a hash operation on the received information.

Instead, we propose an alternative approach that is not based in the use of public key cryptography, as a fast, straight and simple alternative for scenarios in which a public key infrastructure is not available. Our solution intends to prove that the sender either knows or ignores the recipient's ticket. The two only entities in the system that know any given ticket are its owner member and the Key Sever. Assuming the ticket has not been stolen, any message received by a member that successfully runs the verification scheme should only come from the Key Server. The authentication method naturally arises from the re-keying material and no additional infrastructure is needed.

Unfortunately, an attack against our authentication scheme was developed by Peinado et al. in [86, 87], which we discuss later.

For the moment, let us introduce the scheme. Algorithm 4 shows the re-key message authentication process. We assume the Key Server is performing a refreshment of r so the authentication process is complementary to that described in Section 3.4.

Algorithm 4 The key refreshment message authentication algorithm.

1. The Key server:
   i. computes $s = (g^k)^{-1} \bmod L$ by means of the Extended Euclidean Algorithm,
   ii. chooses a random number a, such that $a < x_i$, for every x_i, and
   iii. multicasts $\{a \cdot s, h(a)\}$, where $h(a)$ is the output of a one-way operation on a. Such operation is not specified here.
2. Every member i receives the authentication message and computes $h(a \cdot s \cdot r \bmod x_i)$, which should be equal to the value $h(a)$ received if x_i is a factor of L.

It is convenient that the authentication message is attached to the refreshment message so authenticity can be verified upon reception. If the subgroups approach is used for re-keying (see Section 3.4.2) then each partial re-keying message for group j must be authenticated separately, using the corresponding L_{part_j} value.

3.5.1 Efficiency considerations on the message authentication scheme

Regarding efficiency, the arbitrary-precision arithmetic additional operations required at the Key Server side are a modular inverse and a multiplication. On the other hand, every client must compute a modular multiplication. Those operations have very little impact on the final runtime since they can be run very efficiently by most of the hardware with arbitrary-precision arithmetic capabilities.

The scheme poses a disadvantage, however: the authentication message can be as long as the key refreshment message.

3.5.2 Security considerations on the message authentication scheme by Peinado et al. and Antequera et al.

Peinado et al [86, 87] show that the message authentication protocol can be broken by using a trick similar to that of attack 1 (Section 3.4.4.1). We show the method next.

3.5.2.1 Attack 3

Let us assume that a forged key refreshment message is sent by member h as stated in attack 1. Therefore forged parameters $m',\ p',\ g',\ \delta',\ k'$ are used by h along with $v \cdot L$. By means of the EEA $u',\ v'$ are also obtained. Member h now follows the message authentication protocol as stated in Algorithm 4 with those forged values:

$$s' = (g'^{k})^{-1}\ mod\ v \cdot L \tag{3.7}$$

Finally, a random value a' is used and the authentication message is $(a' \cdot s',\ h(a'))$. It is clear that this data correctly authenticates the forged refreshment message since they were generated from the same initial parameters. Members receiving this information will therefore be misled to believe it comes from the Key Server.

3.5.2.2 Remarks to attack 3

Antequera et al. [3] propose a solution to the message authentication flaw that somehow resembles the Secure Lock solution [21]. In it, the Key Server assigns a second ticket $b_i \in [0, x_i - 1]$ to every member i and solves a system of congruences. Algorithm 5 shows the steps taken by the Key Server. We assume the algorithm is run along with Alg. 3.

The new algorithm is correct, however it has the same drawbacks than Secure Lock: solving a huge system of congruences, a problem which does not scale well.

3.6 Peer validation: a zero-knowledge proof

Our last proposal deals with authentication among peers. Its aim is to verify whether a given peer j holds a valid ticket x_j without gaining knowledge of the

Algorithm 5 The new message authentication algorithm by Antequera et al.

1. The Key Server:
   i. selects a such that $a < x_i$ for every member i.
   ii. calculates $s = (g^k)^{-1}\ mod\ L$ and solves the system of congruences $x = a \cdot s + b_i\ mod\ x_i,\ i = 1, \ldots, n$, obtaining the solution S.
   iii. attaches $(S, h(a))$ to the key refreshment message.
2. Member i, upon reception of the message:
   i calculates $S_i = S - b_i\ mod\ x_i$.
   ii checks whether $h(S_i \cdot g^k\ mod\ x_i)$ equals $h(a)$. In that case the message is authentic.

latter: this means that j is a legal peer, assuming no information leakage. Verification is carried out by means of a challenge, with no disclosure of any private nor sensible information. The scheme is presented next.

Assume that peer i wants to verify whether peer b is a legal peer, prior to establishing communications with it. Algorithm 6 shows the process.

Algorithm 6 The peer validation algorithm.

1. Peer i chooses a random integer w_i such that $1 < w_i < m$ and sends it to the Key Server.
2. The Key Server computes the challenge $inv_i = w_i^{-1} \bmod L$ and sends it to i.
3. Peer i sends $\{inv_i,\ g^{x_i} \bmod m\}$ to b.
4. Peer b calculates $w_b = inv_i^{-1} \bmod x_b$, $\beta_j = w_b \cdot (g^{x_i})^{x_b}$ and sends $\{\beta_b,\ g^{x_b}\}$ to i.
5. Peer i computes $\beta_i = w_i \cdot (g^{x_b})^{x_i}$, which should be equal to β_b.

From the algorithm, it is clear that b owns a valid ticket x_b if $\beta_i = \beta_b$. Otherwise peer i should warn the Key Server so preventive measures can be taken against b.

In case this protocol is implemented in a standalone manner and no public key disclosure algorithm is being run then the Key Server must choose the values g and m as shown in Section 3.4 and communicate them to peers before any authentication is done.

3.6.1 Efficiency considerations on the peer validation scheme

Regarding efficiency, the protocol involves one communication with the Key Server, which clearly limits its application range. Next, two extensions that partially alleviate the problem are proposed. Both can be combined together.

3.6.1.1 Challenge precomputation

Peer i sends a list of several w_i values, $(w_{i1}, \ldots, w_{in})$, to the Key Server on each request. The Key Server replies with the corresponding list of challenges, $(inv_{i1}, \ldots, inv_{in})$, that can then be used by i when needed. A new request to the Key Server is issued when all, or near to all, challenges have been used. Note, however, that challenges are only valid until the next re-keying operation due to the change of L.

3.6.1.2 Subgroups approach with trusted super-peers

In the subgroups approach (see Section 3.4.2) the global L value is split into different, smaller L_{part_j} values. If fully trusted super-peers are introduced, each one receiving an updated version of one or more distinct L_{part_j} values from the Key Server, then they can act as *signature servers*, thus alleviating workload at the Key Server side and increasing overall scalability. Peers can now send their challenge requests to the corresponding super-peer. Given that super-peers are fully trusted our security considerations still hold, and tickets within the product L_{part_j} still remain private. Even if a super-peer went malicious and tried to gain access to the tickets, it still should have to factorize L_{part_j}, which is a computationally impractical task if a proper ticket bit-length is chosen.

3.6.2 Security considerations by Peinado et al. and Antequera et al. on the peer validation scheme

Peinado et al. [86, 87] developed an attack that allows a malicious member i to obtain the ticket of the victim j.

3.6.2.1 Attack 4

Alg. 7 shows the process. Here the notation used by Peinado et al. is not used since it conflicts with ours.

Algorithm 7 Attack 4, proposed by Peinado et al. on the peer validation scheme.

1. Member i computes $g^{x_i} \ mod \ m$, chooses a random inv and sends $(inv, g^{x_i} \ mod \ m)$ to j.

2. Member j computes $w_j = inv_i^{-1} \ mod \ x_j$, $g^{x_j} \ mod \ m$ and $\beta_j = w_j \cdot (g^{x_i})^{x_j} \ mod \ m$. Then she sends $(\beta_j, g^{x_j} \ mod \ m)$ to i.

3. Member i computes $\beta_i = (g^{x_j})^{x_i} \ mod \ m$. Then, she recovers $w_j = \beta_j \cdot (g^{x_i x_j})^{-1} \ mod \ m$. If $m \geq x_j$, then $w_j = w_j \ mod \ m$ and $inv \cdot \beta_j - 1$ is a multiple of x_j. Therefore, x_j could be computed as $x_j = gcd(inv \cdot \beta_j - 1, v \cdot L)$ with high probability.

Note that member i does not communicate with the Key Server. Instead, she chooses a random inv.

3.6.2.2 Remark to attack 4

Antequera et al. [3] show that this attack works only either if $m \geq x_j$ or $w_j \in [0, m]$. Therefore the attack fails if $m < w_j < x_j$ since $w_j \neq w_j \ mod \ m$. Then they propose the following extension based on the extension in Section 3.6.1.1: member i must send a list of w_i candidates to the Key Server. From them, the Key Server chooses one such that $m < w_i^{-1} \ mod \ x_l$, where x_l is the largest ticket in the system (so it can be used with any member j). Then, the Key Server signs w_i and sends the value and the signature back to i, who forwards it j. This way, j knows that the value comes from the Key Server and was not maliciously chosen.

Additionally, we propose a simplification of this solution: the Key Server can directly choose w_i so it does not need to come from i.

3.7 Other extensions by Antequera et al.

In a different work, Antequera et al. refer to the scheme as *Euclid* and propose two different extensions [2]. The first one combines the key refreshment scheme with the Hierarchical Tree Approach in order to address larger audiences and shorter refreshment messages.

In the second extension, they propose a distributed multicast protocol. In this scenario the audience is divided into subgroups, each one controlled by a trusted *group manager*. We refer the reader to [2] for further reading.

3.8 Conclusions

This chapter introduces three different applications of the Extended Euclidean Algorithm, all of them focusing on privacy and security in multicast scenarios. The main one, a re-keying mechanism, allows a single entity to manage the distribution and renewal of encryption keys within restricted groups with the final goal of holding private communications. Despite an attack introduced by Peinado et al. the protocol is secure so far as stated by Antequera et al.: the difficulty of the attack increases with ticket bit-length. Tickets from 1024 bits length on can be considered secure so far. That bit-length is common in today's public key cryptography.

The second application is a message authentication mechanism which is not based on public key cryptography. It was intended for situations in which the latter is not available, and can be run along with the first scheme. Unfortunately, Peinado et al. exposed a flaw in which the Key Server can be impersonated: its implications are that this scheme is not secure for practical use. The Chinese Remainder Theorem based solution proposed by Antequera et al. might be an alternative, however the simplest and fastest solution is to use a public key based signature.

Finally, a zero-knowledge protocol was presented which can be used for validation between two members. By using this protocol clients can decide whether to

trust others or not before establishing communications with them. It works by challenging clients to demonstrate that they own a valid ticket. A flaw in the protocol exposed by Peinado et al. was later fixed by Antequera et al. The new solution proposed by them achieves an applicability similar to that of the initial sub-protocol.

Part II

Contributions to privacy-preserving distributed computations on peer-to-peer networks

Chapter

4

Related work on privacy-preserving distributed computations

After having introduced our centralized secure multicast solution we now move forward to the second part of this thesis dissertation: privacy-preservation communication models for fully distributed computations in peer-to-peer networks. The present chapter reviews the related work so far and provides some needed background, while Chapter 5 introduces and discusses our proposal.

The outline of this chapter is as follows. First, Section 4.1 introduces Shamir′s Secret Sharing scheme, a fundamental primitive of cryptography widely used in this research field. Section 4.2 introduces the two different types of adversaries considered in the literature. Section 4.3 reviews some pioneering solutions, which address small scenarios with a reduced number of participants. They are interesting because they serve as a basis for more recent works. Section 4.4 shows some recent proposals that deal with large networks. Finally, Section 4.5 goes into the details of one of those recent proposals, which we will compare to a solution proposed by us in the next chapter.

4.1 Shamir's Secret Sharing scheme

Secret sharing schemes are a core routine in cryptography. A (t, n) secret sharing scheme can be defined as follows.

Definition 13 A $(\mathbf{t}, \mathbf{n})$ Secret Sharing Scheme *distributes a secret x in n shares, needing $t \leq n$ shares to reconstruct it, and being this impossible with a number of shares below a threshold t.*

In other words, player i wants to share a secret x into n pieces, such that t of them are enough to recover it. Among all secret sharing schemes Shamir's is undoubtedly the most popular. Algorithm 8 shows its steps.

Algorithm 8 Shamir's Secret Sharing algorithm.

1. Player i generates a polynomial $P(x) = x + \sum_{s=1}^{t-1} a_s x^s$. Note that x is the free coefficient.
2. Player i creates n shares by evaluating $P(x)$ at different points. For example, share j corresponds to $P(j)$.
3. Player i gives each other player a different share.
4. To recover x, at least t players must gather and interpolate their shares, thus obtaining a new polynomial $P'(x)$ with free coefficient x.

4.2 Adversary models

During the execution of a distributed algorithm an unknown arbitrary number of players may be interested in obtaining private information from other nodes. A privacy-preserving algorithm tries to prevent them from succeeding. Typically, two assumptions are made when modelling enemies.

First, the enemy is seen as an entity controlling a set of nodes. This means assuming the worst case scenario: that all adversary nodes collaborate together in order to achieve the same goal. Therefore the collusion attack is commonly considered in the literature (see Definition 11 in Chapter 1).

The second assumption deals with the actions the adversary may carry out. Based on them, an adversary may either be semi-honest or malicious [59].

Definition 14 *A* **semi-honest adversary** *tries to obtain hidden knowledge from the information disclosed by the protocol without deviating from the correct execution of the latter.*

That means that a typical semi-honest adversary will never forge messages, for example, but will store incoming messages for later analysis.

Definition 15 *A* **malicious adversary** *deviates from the behaviour assumed by the protocol and tries to obtain private information by taking arbitrary actions.*

Malicious adversaries will probably forge messages, or in other words, fake their input of the algorithm in order to expose other players′ private information. Preventing attacks by this kind of enemies is far more difficult than in the semi-honest case, resulting in complex and usually less efficient algorithms.

4.3 Foundations of privacy-preserving computations

Yao published the first important works on the field in 1982 [108] and 1986 [109]. In them he posed the Millionaire′s problem and similar ones, and presented a framework that allows two players to privately compute the output of a distributed function as well as some applications of this framework. He also extended this solution to the multiplayer case.

A similar, more advanced framework was proposed by Malkhi et al. in 2004 [62], and its multiplayer extension in 2008 [5]. Other important previous multiplayer protocols are [6] and [35].

The proposals above allow to compute any generic function that can be previously transformed into a Boolean circuit. Apart from them, Clifton et al. [23] proposed also an early framework based on secret sharing [64] for executing different privacy preserving operations in distributed algorithms, such like sum, set intersection and set union. These operations can be combined to create more complex ones. Also, Ishai et al. [45] show a very recent solution to the problem.

Even though most of the solutions above have multiplayer extensions and target both semi-honest and malicious adversaries they are complex and, most importantly, not very efficient since they impose a synchronous execution. Therefore, they are not suitable for peer-to-peer networks.

4.4 Proposals for the computation of specific functions

A different group of more recent proposals achieve better applicability in large and dynamic scenarios at the cost of focusing on the computation of a single operation rather than of any function. For example, Canny addresses collaborative filtering in peer-to-peer networks [15], while Kamvar et al. compute the Eigentrust algorithm for reputation management purposes [50]. Let us remark that, again, all solutions reviewed here impose synchronous constraints on the execution.

He et al. [40] propose a solution for wireless sensor networks that combines secret sharing with encryption. However the resulting algorithm does not address churn and accuracy may be affected by message loss (a typical phenomenon in such networks). Even vehicular networks (VANETs) have been considered as scenarios for the problem [57].

Das et al. [24] address sum computations in peer-to-peer networks with the additional advantage that nodes can set their desired level of privacy. However, their proposal is partially synchronous and problems like message loss and node churn have not been taken into account.

Other ideas that have been applied to peer-to-peer privacy preservation are random perturbations [31] and homomorphic encryption [83]. The first can be applied only to aggregation and consists of adding a random, controlled noise to every piece of data sent: when summing all the information received the noise will most probably cancel out. Though efficient and simple, this technique lacks accuracy and offers limited privacy as shown in [28].

Homomorphic encryption has also been proposed as a solution. However, it requires a trusted third party and imposes synchronism, thus slowing down computations. Bickson et al. show that neither random perturbations nor homomorphic encryption are the best choices for large dynamic networks [9]. In the same paper, Bickson et al. propose SSS, the most advanced solution so far to the best of our knowledge. We discuss it in the next section.

4.5 SSS: the proposal by Bickson et al.

Bickson et al. introduce their algorithm SSS in [9]: a method for the computation of privacy preserving sums in peer-to-peer networks in presence of semi-honest adversaries. We note that although the SSS method has been extended to cope with malicious adversaries [10] it has not been prepared to deal with the realistic failure models we wish to target here, so here we focus on the version presented in [9].

Let us discuss it in a more formal way. Assume a peer-to-peer overlay network with N nodes, where each node i has a private value x_i and each link (i, j) has an assigned weight w_{ij}. The ultimate goal of the method is having every node i compute the sum

$$\sum_{j \in IN^{[i]}} w_{ji} x_j, \tag{4.1}$$

without being able to extract every independent value x_j, being $IN^{[i]}$ the set of in-neighbors of i. This implies that every private value x_i in the network remains a secret except for its owner. Additionally, the sum in Eq. (4.1) is known only by i.

The strategy followed to achieve this goal is based in Shamir's Secret Sharing scheme [93] (hence the name SSS), and is stated in Algorithm 9 (taken from [9]; notation is adapted to ours). In a nutshell, each in-neighbour of i, i.e. each $j \in IN^{[i]}$ generates k shares from its secret value x_j and delivers each one to a different in-neighbour of i (including itself). Then, every node j sums all the shares received within this iteration and sends the result to the central node i. Finally, i interpolates the aggregated shares received and obtains a polynomial $P(x)$. The free coefficient of $P(x)$, i.e. $P(0)$, corresponds to Eq. (4.1).

Figures 4.1 and 4.2 shed light on the process. We consider the neighbourhood formed by node i and $j,\ l,\ m\ \in IN^{[i]}$. The secret values of the latter nodes are 5, 6 and 7, respectively, and all link weights are 1 for simplicity. Therefore, node i should learn the following linear combination

$$w_{ji} \cdot x_j + w_{li} \cdot x_l + w_{mi} \cdot x_m = 1 \cdot 5 + 1 \cdot 6 + 1 \cdot 7 = 18 \tag{4.2}$$

after one round of execution.

Each in-neighbour of i is assigned a different random x-coordinate, say 1, 3, and 2, respectively. The upper part of Figure 4.1 shows the polynomials chosen by

Algorithm 9 The SSS algorithm by Bickson et al.

INPUT:

x_j at every node $j \in IN^{[i]}$

k: a fixed security parameter, such that $k \leq |IN^{[i]}|$

OUTPUT:

The value computed in Eq. (4.1) at node i.

1. Each node j generates a random polynomial $P_{ji} = w_{ji}x_j + \sum_{s=1}^{k_i-1} a_s x^s$ of degree $k_i - 1$ (where $k_i \leq |IN|^{[i]}$).
2. For each neighbour l of node i, i.e. $l \in IN^{[i]}$, each node j creates a share C_{jil} of the polynomial $P_{ji}(x)$ by evaluating it on a single point x_l.
3. Each node j sends C_{jil} to node l, which is also i's neighbour.
4. Each neighbour l of node i aggregates the shares she received from all neighbours of node i and computes the value $S_{li} = \sum_{j \in N_i} P_{ji}(x_l)$[1].
5. Each neighbour l sends the sum S_{li} to node i.
6. Node i treats the value received from node l as a value of a polynomial of degree $k_i - 1$ evaluated at the point x_i.
7. Node interpolates $P_i(x)$ for extracting the free coefficient, which in this case is the weighted sum of all messages $\sum_j \in IN^{[i]} w_{ji}x_j$.

the in-neighbours

$$\begin{aligned} P_{ji} &= 2x^2 + x + 5 \\ P_{li} &= 4x^2 + 3x + 7 \\ P_{mi} &= 3x^2 + 2x + 6, \end{aligned}$$

and the shares generated by each in-neighbour: C_{jij}, C_{jil}, C_{jim}, C_{lij}, C_{lil}, C_{lim}, C_{mij}, C_{mil}, C_{mim}. We can see that the shares are actually points in their respective polynomials. Then, those shares are spread among the neighbourhood (Fig. 4.2(a) depicts j sending its own), and each in-neighbour sums every share received (Fig. 4.2(b) shows node l receiving all shares). Finally, every in-neighbour sends its

[1]Note that the result of this computation is equal to the value of a polynomial of degree $k_i - 1$, whose free coefficient is equal to the weighted sum of all messages sent to node i by its neighbours.

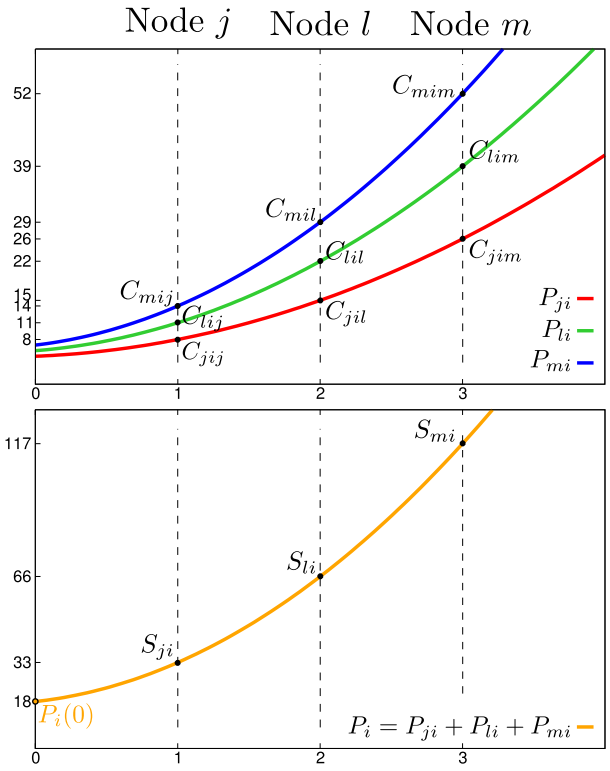
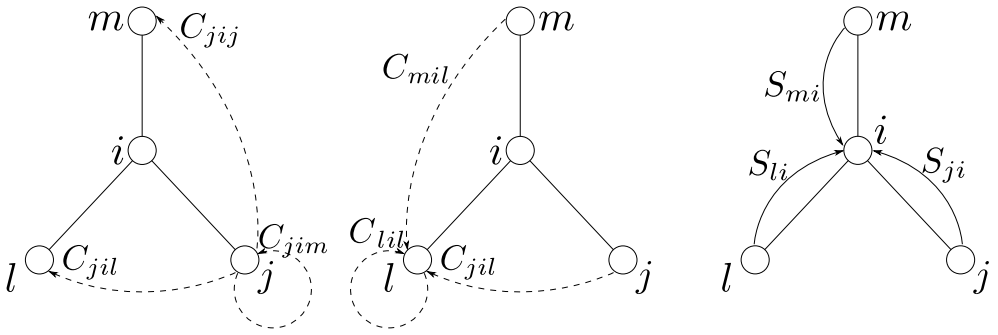


Figure 4.1: **Aggregation and interpolation of shares in the SSS algorithm.**

summation of shares to i: S_{ji}, S_{li}, S_{mi} (see Fig. 4.2(c)). These S messages are actually points of the polynomial $P_i = P_{ji} + P_{li} + P_{mi}$, depicted in the lower part of Figure 4.1. Node that i obtains P_i by simply interpolating the three S messages. The free coefficient is the value shown in Eq. (4.2).

The value k is actually a security parameter. From Shamir's Secret Sharing scheme it is straight that i must collude with $k-1$ in-neighbours if she wants to discover the secret value of j. This parameter is fixed for every neighbourhood and must be established prior to executing the algorithm. This means that nodes can not decide autonomously on their security degree. In general, any change in parameter values has to be agreed on in every neighbourhood. In practical implementations parameters will be probably imposed by i.

Regarding tolerance to message loss, the algorithm tolerates the loss of up to $|IN^{[i]}| - k$ type S messages (those addressed to i), but each in-neighbor $j \in IN^{[i]}$



(a) Sending shares (b) Collecting shares (c) Sending share aggregations

Figure 4.2: **Messages sent in the SSS algorithm.**

must receive every share of type C sent to it by all the other in-neighbors of i in order to be able to send a correct S message to i. Note that in case this did not happen then computations would be corrupted. Also note that even if message losses were detected by some mechanism (not mentioned in [9]) the result of the affected iteration should be discarded.

Message complexity is $O(|IN^{[i]}|^2 + |IN^{[i]}|)$ for each neighbourhood because all the possible pairs of node in $IN^{[i]}$ must communicate with each other, and later every node sends a message to i.

4.6 Conclusions

This chapter reviews the field of privacy preservation in peer-to-peer networks. Despite researchers have paid attention to the problem of computing a common result without revealing individual inputs for a long time, the scenarios considered always involved a few players only, specially in the older works. Furthermore, practical issues like fault tolerance and node churn were not seriously considered.

More recently, some authors have moved the same problem to a variety of networks, like peer-to-peer, wireless sensor networks (WSNETs) or even vehicular networks (VANETs). However, even in those works the aforementioned problems are barely addressed. Additionally, all the schemes reviewed here impose some level of synchrony on the execution, even if the underlying computation algorithm

does not.

We have also discussed one of the most relevant proposals to the date by Bickson et al. We will use this work as a base for the research presented in the next chapter.

Chapter

5

A privacy-preserving distributed computations algorithm

5.1 Introduction

In this chapter, based in the work introduced in [78], we present a privacy-preserving distributed algorithm against semi-honest adversaries that suits peer-to-peer networks. Our algorithm serves as a privacy-preserving layer to be run on top of distributed iterative algorithms.

The kind of computations supported by the algorithm are linear combinations of the values provided by a node's in-neighbours. It is privacy-preserving because (1) the node can not obtain the individual value provided by each in-neighbour, and (2) no other node is aware of the final sum computed at a given node.

As we mentioned in the previous chapter, peer-to-peer networks present distributed algorithms with harsh conditions that can make the execution difficult. The main problems are unreliability (messages can be lost or delayed) and node churn (nodes can enter and leave unexpectedly at any time). For this reason our algorithm is asynchronous, i.e., makes no assumptions about the arriving time of a message (if

it arrives at all), and tolerates node churn in a natural way. The result is a flexible algorithm that tolerates realistic message loss and delay rates as well as realistic churn patterns. At the end of the chapter we will show experimental results to support these claims.

The rest of the chapter is organized as follows. Section 5.2 presents the scenario we consider and our goal. Section 5.3 shows the main idea behind our proposal. Section 5.4 discusses how that idea can be applied in practice and describes the algorithm itself. Section 5.5 deals with practical experimentation and discusses the results obtained. Finally, Section 5.6 concludes the chapter.

5.2 Scenario

Let us assume a network with N nodes, with each node being connected with directed links by at least k other nodes. Nodes communicate by exchanging messages that may get lost or delayed. Besides, the network may suffer from churn: nodes can leave but they are assumed to rejoin eventually remembering their previous state and neighbours, i.e. they rejoin in the same neighbourhood. Each node i has a secret value x_i known by no other, and each directed link (i, j) is assigned a weight w_{ij}. The set formed by the in-neighbours of i is denoted $IN^{[i]}$, while the set formed by the out-neighbours of i is $OUT^{[i]}$. As in the SSS scheme by Bickson et al. [9] every node i computes the value

$$\sum_{j \in IN^{[i]}} w_{ji} x_j, \tag{5.1}$$

in a way such that no other node apart from i can learn this value, and i can not guess x_j for any $j \in IN^{[i]}$.

Adversaries we will be facing belong to the semi-honest type (see Definition 14): thus they are not assumed to forge messages, although nodes can collude in order to try to guess any x_i value. Eavesdropping on arbitrary links without corrupting nodes is not considered either. This is not a crucial restriction, since eavesdropping can be easily prevented with public key encryption.

5.3 A sum-splitting secret sharing scheme

The use of a $(k, |IN^{[i]}|)$ Shamir's Secret Sharing scheme with $k < |IN^{[i]}|$ introduces a $(|IN^{[i]}| - k)$ redundancy of shares in the secret recovering process, so the SSS scheme might be seen as a highly fault-tolerant protocol. This is not exactly the case since, as we already mentioned in Section 4.5, SSS tolerates the loss (or delay beyond deadline) of up to $(|IN^{[i]}| - k)$ type S messages only. Having one type C message lost or delayed corrupts computations for the current iteration. This is worsened by the fact that a change in a secret value forces its owner to choose a new polynomial and send new shares (recall Alg. 9), which imposes a synchronous flow on the execution. We can see this last fact as if all shares participating in one round were *entangled*: shares from different rounds can not be mixed, therefore synchrony is imposed.

Instead of Shamir's Secret Sharing scheme, we propose the use of a sum-splitting scheme as follows: for a secret $s \in \mathcal{R}$, let the first $k-1$ shares be random elements of $\mathcal{R}$ (i.e., $s_1, \ldots, s_{k-1} \in \mathcal{R}$), and let

$$s_k = s - \sum_{i=1}^{k-1} s_i. \tag{5.2}$$

This turns out to be a (k, k) secret sharing scheme with zero redundancy. Now, we can replace Shamir's scheme for this one in SSS: being i the central node in the neighbourhood, every $j \in IN^{[i]}$ sends random shares to other in-neighbours of i and keeps the last share (see Eq. (5.2)) for itself. At every node all received shares are summed up and the result is sent to i. Finally, i simply sums all the incoming messages, obtaining the desired value (see Eq. (5.1)).

As a consequence, we lose the aforementioned $(|IN^{[i]}| - k)$ fault-tolerance. However, the benefits largely compensate this minor loss as we explain next.

Share reuse

A nice property of the new scheme relies on the fact that any share can be computed from the secret and the other $k-1$ shares. This allows node j to recompute the last share s_k without sending a new set of shares whenever its secret value changes. Thanks to that, the imposed entanglement drawback is overcame: random shares

$s_1, \ldots, s_{k-1}$ can be reused if s_k is recalculated after every change of the secret value. Additionally, any node can change any of the random secret shares, if desired, by sending the new one. The other shares will remain unchanged.

Asynchronism

Reusing shares brings the most important feature of the new scheme: the elimination of synchronization among nodes. Now the central node can process every message asynchronously as it arrives, independently from the others. Nodes can therefore work at a different pace and there are no deadlines.

Different and flexible k values

As a second benefit, every node can now set its own value of k (number of shares) and even a different set of share recipients. This means that node $j \in IN^{[i]}$ needs not to send a share to every other $\alpha \in IN^{[i]}$ any more.

Lower message complexity

Finally, the new message complexity per neighbourhood of one round of execution is $O(|IN^{[i]}|k + |IN^{[i]}|)$ while for SSS it was $O(|IN^{[i]}|^2 + |IN^{[i]}|)$. Note that if $k << |IN^{[i]}|$ the complexity of the sum-splitting scheme is much lower so, even though we are using a (k, k) secret sharing scheme, sending fewer messages implies more overall robustness to failure. In the worst case, i.e. $k = |IN^{[i]}|$, message complexity is equivalent to that of SSS. For the sake of completeness, the final algorithm shown in Section 5.4 has an additional $O(|IN^{[i]}|)$ complexity since some control messages are sent in one round, being the final total complexity $O(|IN^{[i]}|k + 2|IN^{[i]}|)$. However, this fact does not affect the discussion above.

5.3.1 Collaborator nodes

Being i the central node of a neighbourhood (that who will receive the privacy-preserving sum of values) and nodes $j,\ \alpha\ \in IN^{[i]}$, we note a share sent from j to α by $s^{[j\alpha i]}$. Given that j is using node α as a collaborator in the privacy-preserving process, we say that α is an *out-collaborator* of j. We note the set of

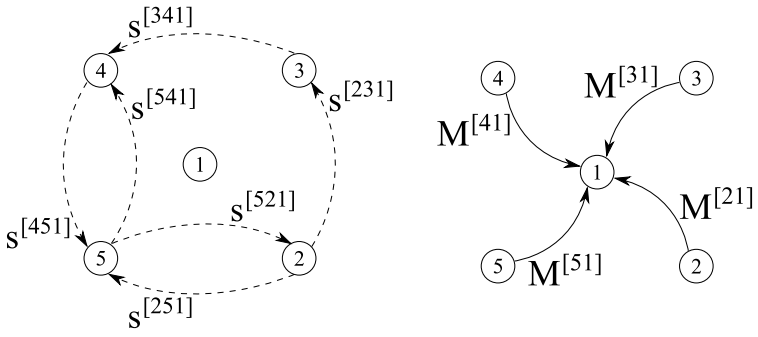


Figure 5.1: **Communication during the sum-splitting scheme.**

out-collaborators of j as $OC^{[ji]}$, and remark that $OC^{[ji]} \subseteq IN^{[i]}$. The size and composition of $OC^{[ji]}$ can be freely chosen by node j. So, if we rewrite Eq. (5.2) in terms of the new notation, we have

$$s^{[jij]} = w_{ji}x_j - \sum_{\alpha \in OC^{[ji]}} s^{[j\alpha i]}. \tag{5.3}$$

Node j keeps $s^{[jij]}$ to itself.

At the same time, node j can receive shares from other in-neighbours of i. For example, share $s^{[\beta ji]}$ is sent by β to j. In-neighbours of i that send shares to j are called *in-collaborators* of j, and this set is noted as $IC^{[ji]}$, with $IC^{[ji]} \subseteq IN^{[i]}$. The size of this set is not chosen by j, it is rather given by the number of in-neighbours of i that decide to send a share to j.

So, the privacy-preserving message that node j sends to central node i is the last of j's own shares ($s^{[jij]}$) plus the shares received from its in-collaborators:

$$M^{[ji]} = s^{[jij]} + \sum_{\beta \in IC^{[ji]}} s^{[\beta ji]} = w_{ji}x_j - \sum_{\alpha \in OC^{[ji]}} s^{[j\alpha i]} + \sum_{\beta \in IC^{[ji]}} s^{[\beta ji]}, \tag{5.4}$$

If we could guarantee that there is no delay or message loss, and that all nodes have completely up-to-date information about the secret shares, then node i could simply compute

$$x_i = \sum_{j \in IN^{[i]}} M^{[ji]} = \sum_{j \in IN^{[i]}} w_{ji}x_j. \tag{5.5}$$

Figures 5.1(a) and 5.1(b) illustrate the scheme we propose using the notation defined above. Let us assume nodes $\{2,3,4,5\} = IN^{[1]}$ want to share a linear combination of their private values with node 1. Figure 5.1(a) depicts the nodes sending random shares to other randomly chosen in-neighbors of 1. Note that the number of shares, i.e., the value of k, can be different for each node. Figure 5.1(b) shows the nodes sending the final messages to 1. Node 1 simply adds these final messages to obtain the linear combination. From this moment on, any node $j \in \{2,3,4,5\}$ can freely send $M^{[j1]}$ messages at the desired rate: node 1 will always compute Eq. 5.5 correctly since shares have not changed. This fact makes our scheme asynchronous.

5.4 Implementing asynchronous distributed algorithms

Once we are able to compute the sum in Eq. (5.1) we can compute global functions, with additional assumptions. For example, if these sums are computed iteratively in a synchronized fashion then we can implement many iterative methods including one for solving linear systems of equations or finding the dominant eigenvector of the matrix $W = [w_{ij}]_{i,j=1}^{N}$ [36]. For example, the well-known power method [4] for calculating the dominant eigenvector of W is based on the iteration defined by

$$x_i^{(t+1)} = \sum_{j \in IN^{[i]}} w_{ji} x_j^{(t)}, \tag{5.6}$$

where the vector $[x_i^{(t)}]_{i=1}^{N}$ converges to the dominant eigenvector of W under very mild conditions provided that the spectral radius of W is 1. Several key algorithms, including PageRank, rely on finding the dominant eigenvector of an appropriate matrix [7].

If we relax the assumption on synchronization, then many of the algorithms mentioned above can still be executed in a rather simple way, under very similar conditions [34, 48, 61]. For example, Algorithms 10, and 11 show the Power Iteration algorithm, a non-secure asynchronous distributed implementation of the aforementioned power method, taken from [48] and adapted to our own notation. The active part, Alg. 10, is in charge of sending the state of the owner node to its neighbours. The passive part, Alg. 11, receives and processes incoming messages.

Algorithm 10 Async. distrib. Power Iteration at node i, active thread

1. **while**(true)
2. wait(Δ)
3. **for-each** $j \in OUT^{[i]}$ **do**
4. send $w_{ji}x_i$ to j
5. $b_i \leftarrow \sum_{k \in IN^{[i]}} b_{ki}$
6. $x_i \leftarrow b_i$

Algorithm 11 Async. distrib. Power Iteration at node i, passive thread

1. **while**(true)
2. $msg \leftarrow receive\_message()$
3. $k \leftarrow msg.sender$
4. $b_{ki} \leftarrow x$

For a privacy-preserving method to be applied on top of those algorithms, it should be able to run asynchronously. We have not found such a method on the literature. Our proposal does, thanks to the lack of entanglement of shares. After having introduced the basic idea behind our algorithm, we show now how to implement it on top of asynchronous distributed algorithms and under the harsh conditions of peer-to-peer networks. We will do this in the context of the *Power Iteration* algorithm, given it is a popular and useful one. We will use Algorithms 10 and 11 later as a skeleton. For other numerical methods that support asynchronism, like those that solve systems of linear equations, the same method is applicable with trivial modifications.

5.4.1 Share versions

Between iterations, a node may want to change one or more shares to increase the security obtained from the algorithm. As we mentioned before, the node may change either one or more shares at the same time. Actually, it may even change the composition and size of its out-collaborators set. All nodes can do this independently. Since we assume there is no synchronization, we assign a version number t to the shares, and we use the notation $s_t^{[j\alpha i]}$. For example, $s_1^{[j\alpha i]}$ indicates that the

old value $s_0^{[j\alpha i]}$ has been updated to a newer one. From now on, when the version number t is omitted from the notation, we assume t is the latest existing version.

5.4.2 Fault tolerance mechanisms

We have just mentioned that nodes manage their out-collaborator sets independently as well as the version of their shares. Since we assume the existence of message loss and delay, and churn, it is not guaranteed any more that the information that reaches node i is consistent. For example, node j can update share $s_t^{[j\alpha i]}$ by sending $s_{t+1}^{[j\alpha i]}$ to $\alpha \in OC^{[ji]}$, but the message might be lost. As a consequence, j would start to use the newer version of the share for Eq. (5.4) while α would use the old one in its own computations: the result at i would be therefore corrupted.

To solve this problem, node j maintains two lists of share-related information, one per set of collaborators, $LOC^{[ji]}$ for $OC^{[ji]}$ and $LIC^{[ji]}$ for $IC^{[ji]}$:

$$LOC^{[ji]} = \{(j, \alpha, t) \mid \alpha \in OC^{[ji]}\} \tag{5.7}$$

$$LIC^{[ji]} = \{(\beta, j, t) \mid \beta \in IC^{[ji]}\}, \tag{5.8}$$

where j, α, β, t are taken from $s_t^{[j\alpha i]}$ and $s_t^{[\beta j i]}$. We would like to make clear that different shares within the same list may have different t values, since shares may be updated on a one-by-one basis. Note also that the actual value of the shares is not included in the list.

These lists therefore indicate the current state of the $OC^{[ji]}$ and $IC^{[ji]}$, respectively, and are sent by j to i along with every message $M^{[ji]}$ (see Eq. (5.4)). Based on the lists, node i will update its value according to Eq. (5.5) only if the following condition is satisfied:

$$\bigcup_{j \in IN^{[i]}} LOC^{[ji]} = \bigcup_{j \in IN^{[i]}} LIC^{[ji]}, \tag{5.9}$$

which expresses the fact that all collaborator groups are consistent, and use the same version of the secret shares.

The mechanism above helps node i to detect possible inconsistencies. Now, in order to solve them, node i sends information back to j regarding the share versions other nodes use from j. This is, i periodically sends back the versions of

shares used by nodes in $OC^{[ji]}$. Note that i knows this information because she receives LOC and LIC lists from every node in the neighbourhood. We call these information pieces *checklists*, and denote them by $CH^{[ji]}$. More formally,

$$CH_t^{[ji]} = \{(j, \alpha, t) \in \bigcup_{\alpha \in IN^{[i]}} LIC^{[\alpha i]},\ (\beta, j, t) \in \bigcup_{\beta \in IN^{[i]}} LOC^{[\beta i]}\}. \tag{5.10}$$

Upon reception of the latest checklist, j can check which version of the shares are effectively in use by its out-collaborators. If the version of any of the shares is not the latest, node j re-sends the latest share version again to the corresponding out-collaborator. A different checklist is elaborated by i for every in-neighbour, therefore no consensus among nodes is needed. Checklists have their own version index t too.

Finally, we have included a third mechanism to prevent the following situation: let us assume that node j sends $s_{t+1}^{[j\alpha i]}$ to α and, consequently, starts using version $t+1$ when computing Eq. (5.4). However, let us also assume that α is offline for a long period of time and the last message $M^{[\alpha i]}$ received by i had used $s_t^{[j\alpha i]}$. Node i informs j about the problem with a checklist, however even if j resends version $t+1$ the problem will not be solved because that α is offline. As a consequence node i can not compute the correct result.

In order to solve this, i sends a new list along with every checklist. This new list indicates which nodes i *believes* to be online, i.e., which nodes recently contacted i with a message. More formally,

$$online\_nodes_t^{[i]} = \{\gamma \in IN^{[i]},\ \text{that } i \text{ believes to be online}\}. \tag{5.11}$$

Now, if node j realizes that α is offline then it will switch back to $s_t^{[j\alpha i]}$ rather than insisting on $s_{t+1}^{[j\alpha i]}$. The information reaching i should be consistent now.

5.4.3 The algorithm

After having introduced mechanisms that provide fault tolerance to the sum-splitting scheme we present the full algorithm in a formal manner. For this, we specify three types of messages:

Type 1 (random share): node $j \in IN^{[i]}$ sends $s_t^{[j\alpha i]}$ to node $\alpha \in OC^{[ji]}$.

Type 2 (checklist): node i sends to $j \in IN^{[i]}$ two sets,

$$CH_t^{[ji]} = \{(j,\alpha,t) \in \bigcup_{\alpha \in IN^{[i]}} LIC^{[\alpha i]},\ (\beta,j,t) \in \bigcup_{\beta \in IN^{[i]}} LOC^{[\beta i]}\},$$

and

$$online\_nodes_t^{[i]} = \{\gamma \in IN^{[i]},\ \text{that } i \text{ believes to be online}\},$$

Type 3: node $j \in IN^{[i]}$ sends $V_t^{[ji]} = (M^{[ji]}, LOC^{[ji]}, LIC^{[ji]})$ to node i.

We remark again that version numbers t refer only to the entity they index and do not necessarily have the same value. They have also been included in Type 2 and 3 messages so that nodes can drop delayed out-of-order messages. In $s_t^{[j\alpha i]}$ the index t denotes the active version number. The set $online\_nodes^{[i]}$ is maintained by each node i based on recording the senders of recently received messages. In a Type 2 message, node i sends this set to node j.

The algorithm at a node j takes as input the local structure of the communication graph (the overlay network), that is, $OUT^{[j]}$, $IN^{[i]}$ and w_{ji} for all nodes $i \in OUT^{[j]}$, where w_{ji} is the link weight. In addition, node j has an initial secret value x_j, and a period Δ that determines the frequency of the execution of the loop in the active thread. All the nodes could in principle select a different Δ or they could execute their active thread even irregularly since the asynchronous iteration tolerates this. However we assume that all the nodes use the same period Δ for the sake of simplicity.

The algorithm is actually composed of two different threads running in parallel. Algorithm 12 shows the active thread, while Algorithm 13 shows the passive thread: they are the privacy-preserving versions of Algorithms 10 and 11, respectively. We describe them in the following paragraphs.

The active thread is in charge of sending information. An instance of it is run at node j for all $i \in OUT^{[j]}$, that is, for all the nodes i for which $j \in IN^{[i]}$. Its main tasks are (1) managing the set of out-collaborators, OC, and sending shares (Type 1 messages) to its members, (2) verifying the received checklists to check the correct arrival of the shares and resending them if needed, (3) computing and sending Type 3 messages and (4) sending Type 2 messages (given that the node is also the central node in its own neighbourhood).

The algorithm first initializes several variables: a copy of the last received Type 3 message from each in-neighbor (line 1), the set $OC^{[ji]}$ and a timeout for updating the shares of the members of the set (line 3). The first set of shares is sent in line 5.

The main loop runs with a period of Δ time units that defines the frequency of sending messages. First, we clear the set of nodes that j estimates to be online. This set is filled in the passive thread during the waiting period (see Algorithm 13). Then, if the share renewal timeout expires, we first identify those out-collaborators that are probably online, we try to add new collaborators if the old ones seem to be offline (line 11), and then we send the new shares (line 15). Note that sending new shares to collaborators that are offline is not a problem. From line 17 to 19, the freshest available checklist is used to discover which shares $s^{[j\alpha i]}$ have not been installed successfully, and these are re-sent. Node j then creates new Type 2 messages for its own in-neighbors (line 20).

Finally, a Type 3 message is created and sent to node i in line 38. For those shares sent by j (lines 25 and 27), node j uses the version in the checklist received from i. This way j makes sure it is using the last share that reached i. The case of shares received by j is more interesting. The sender β wants proof that node j has started using the new version before switching to it itself. However, if β is offline, it can get the proof only when it comes back online, which may in fact never happen. So node j is cautious and switches to the new version only if there is a good chance that β will detect it, and will switch too. This is only a heuristic, since with a small probability it might happen that β is incorrectly considered to be online. However, this is not a problem, because node j will switch back to the working version included in the checklist in the next round, should β stay offline.

The passive thread in Algorithm 13 handles message arrivals at node j, updates its private state when possible and fills the set $online\_nodes^{[j]}$. If the incoming message is of Type 1 (line 3) then the received share is stored (line 4). For a Type 2 message (line 6) the checklist is stored and the set $online\_nodes^{[i]}$ is added to $online\_nodes^{[j]}$ (line 8). Incoming Type 3 messages are stored and if the condition in (5.9) holds (line 12) then the internal state x_j is updated and the corresponding share timeout is decreased. Note that it is not necessary to update shares if the private state is not changing, so in that case we do not decrement the timer. The sender of any message is added to $online\_nodes^{[j]}$ (lines 5, 8 and 11).

Algorithm 12 Async. privacy-preserving Power Iteration at j, active thread

1. **for-each** node $\lambda \in IN^{[j]}$ *#initialize incoming type 3 msgs*
2. $V^{[\lambda j]} \leftarrow (0, \{\}, \{\}, \text{timestamp})$
3. choose random $s\_timeout^{[ji]}$ and a randomly sized set of nodes $OC^{[ji]} \in IN^{[i]}$
4. **for-each** node $\alpha \in OC^{[ji]}$ *#send shares*
5. send type 1 msg ($s_{t=0}^{[j\alpha i]}$) to α
6. **while**(true)
7. $online\_nodes^{[j]} \leftarrow \{\}$ *#it is updated in the passive thread*
8. wait(Δ) *#determines message sending frequency*
9. **if** $i \in online\_nodes^{[j]}$ **and** $s\_timeout^{[ji]} \leq 0$ *#renew one share*
10. **if** $OC^{[ji]} \cap online\_nodes^{[j]} = \{\}$ **and** $IN^{[i]} \cap online\_nodes^{[j]} \neq \{\}$
11. add a node α to $OC^{[ji]}$ from $IN^{[i]} \cap online\_nodes^{[j]}$
12. **if** $OC^{[ji]} \cap online\_nodes^{[j]} \neq \{\}$
13. choose one node $\alpha \in OC^{[ji]} \cap online\_nodes^{[j]}$
14. generate new share $s_{t \leftarrow t+1}^{[j\alpha i]}$
15. send new type 1 msg ($s_t^{[j\alpha i]}$) to α *# t was increased by 1*
16. choose new random $s\_timeout^{[ji]}$
17. **for-each** share $s_t^{[j\alpha i]}$ *#check shares and versions in checklist*
18. **if** $(j, \alpha, t) \notin CH^{[ji]}$
19. resend $s_t^{[j\alpha i]}$ to α *#resend the share if needed*
20. **for-each** node $l \in IN^{[j]}$ *#send type 2 messages*
21. compose $CH^{[lj]}$ according to its definition
22. send type 2 message: ($CH^{[lj]}$, $online\_nodes^{[j]}$, timestamp) to node l
23. $LOC^{[ji]} \leftarrow \{\}$ *#send type 3 messages*
24. $M^{[ji]} \leftarrow w_{ji} x_j$
25. **for-each** $(j, \alpha, t) \in CH^{[ji]}$ *#share version t from $checklist^{[ji]}$*
26. $M^{[ji]} \leftarrow M^{[ji]} - s_t^{[j\alpha i]}$
27. store (j, α, t) in $LOC^{[ji]}$
28. **for-each** $(\beta, j, t) \in received\_shares^{[ji]}$ *#shares received from in-collaborators*
29. **if** $\beta \in online\_nodes^{[j]}$ *#decide which version of the share to use*
30. $M^{[ji]} \leftarrow M^{[ji]} + s_t^{[\beta j i]}$ with t the newest version received
31. store (β, j, t) in $LIC^{[ji]}$
32. **else**
33. $M^{[ji]} \leftarrow M^{[ji]} + s_{t'}^{[\beta j i]}$ with t' obtained from $CH^{[ji]}$
34. store (β, j, t') in $LIC^{[ji]}$
35. **if** $LOC^{[ji]} = \{\}$ **and** $LIC^{[ji]} = \{\}$ *#happens only during bootstrap*
36. send type 3 message $V^{[ji]}$: (0, $LOC^{[ji]}$, $LIC^{[ji]}$, timestamp) to node i
37. **else**
38. send type 3 message $V^{[ji]}$: ($M^{[ji]}$, $LOC^{[ji]}$, $LIC^{[ji]}$, timestamp) to node i

Algorithm 13 Async. privacy-preserving Power Iteration at j, passive thread

1. **while**(true)
2. $msg \leftarrow receive\_message()$
3. **if** type 1: $s_t^{[\beta ji]}, \beta \in IN^{[i]}$ # $s_t^{[\beta ji]} received$
4. store $s_t^{[\beta ji]}$ in $received\_shares^{[ji]}$ (replace $s_{t-1}^{[\beta ji]}$ if exists)
5. $online\_nodes^{[j]} \leftarrow online\_nodes^{[j]} \cup \beta$
6. **else if** type 2: $CH^{[ji]}, online\_nodes^{[i]}$ *# checklist received from i*
7. store $CH^{[ji]}$ (replace older version if exists)
8. $online\_nodes^{[j]} \leftarrow online\_nodes^{[j]} \cup online\_nodes^{[i]} \cup i$
9. **else if** type 3: $V^{[\lambda j]}, \lambda \in IN^{[j]}$ *#V msg received*
10. store $V^{[\lambda j]}$ (replace older version if exists)
11. $online\_nodes^{[j]} \leftarrow online\_nodes^{[j]} \cup \lambda$
12. **if** $\cup_{\lambda \in IN^{[j]}} LOC^{[\lambda j]} = \cup_{\lambda \in IN^{[j]}} LIC^{[\lambda j]}$ then *#check Eq. (5.9)*
13. $x_j \leftarrow \sum_{\lambda \in IN^{[j]}} M^{[\lambda j]}$ *#update internal state*
14. $s\_timeout^{[ji]} \leftarrow s\_timeout^{[ji]} - 1$

5.4.4 Practical considerations on the algorithm

It was already mentioned that our scheme runs in an asynchronous mode: this is due to the fact that node i can update its secret value upon the arrival of a single Type 3 message from any in-neighbour (as long as the condition in Eq. (5.9) holds). As a consequence, the impact of message loss and delay is dramatically reduced.

Node churn is also naturally supported by our algorithm. For example, in SSS every neighbourhood needs to agree on a given k. If the number of online nodes at a given moment is below k then the remaining nodes must agree on a new k. This issue is not discussed in [9] and is not straightforward. Having a static value of k for the whole network is neither a solution since neighbourhoods with less than k nodes would not be able to compute a result.

On the other hand, our algorithm allows nodes to set independent k values, and there is no lower limit on the number of online nodes at a given time in a neighbourhood. This means that computations can carry on even if there is only one active neighbour apart from the central node i: the active node can still send valid Type 3 messages to i even if all its out-collaborators are offline. There is one

problematic case, however, in which node i will not be able to update its state. This case was already mentioned: if there is a share version incompatibility at node i and both the sender and the collaborator are offline. As an example, let us suppose that node j sends $s_{t+1}^{[j\alpha i]}$ to α, who is offline at that time. Then, j sends a Type 3 message to i in which $s_{t+1}^{[j\alpha i]}$ was used and goes offline. However, the last Type 3 message that i had received from α involved $s_t^{[j\alpha i]}$ rather than the $t+1$ version. This inconsistency prevents i from updating its private value and can only be solved when either j or α become online again.

Finally, our scheme allows nodes to increase their individual degree of privacy by augmenting the value of k and consequently the size of their out-collaborators set. In Algorithm 12 we do this when the node decides to refresh a share but no out-collaborator is found online. Then a new out-collaborator is chosen.

5.4.5 Considerations on privacy

Here we discuss the level of privacy achieved by our algorithm and compare it with that of SSS. Remember that the adversary model considered both here and in SSS is the semi-honest one, and that eavesdropping on links is not considered given that public key encryption easily eliminates that threat. Having said that, there are two questions to considerate here.

First, our algorithm requires a collusion of more than k nodes in order to guess the private value of another node j, assuming that j has generated $k = |OC^{[ji]}|$ shares. To see this, let us recall Eq. (5.4),

$$M^{[ji]} = s^{[jij]} + \sum_{\beta \in IC^{[ji]}} s^{[\beta ji]} = w_{ji}x_j - \sum_{\alpha \in OC^{[ji]}} s^{[j\alpha i]} + \sum_{\beta \in IC^{[ji]}} s^{[\beta ji]}.$$

From the equation, it is clear that in order to guess $w_{ji}x_j$ all nodes in $OC^{[ji]}$ plus all nodes in $IC^{[ji]}$ plus i itself (the only receiver of the Type 3 message) must collude. Given that $|OC^{[ji]}|$ already equals k, we have that $|OC^{[ji]}| + |IC^{[ji]}| + |\{i\}| \geq k$. On the other hand, in SSS it is enough that $k-1$ nodes collude in order to guess the private value of a given node. To make things harder, in our algorithm out-collaborators do not know how many shares other nodes have created: only the central node knows this number.

Second, the central node i might be able to detect trends in $w_{ji}x_j$ if the expression $\sum_{\alpha \in OC^{[ji]}} s^{[j\alpha i]} + \sum_{\beta \in IC^{[ji]}} s^{[\beta ji]}$ remains a constant for too long. In that case, the difference between two consecutive Type 3 messages, i.e., $M_{t+1}^{[ji]} - M_t^{[ji]}$ will reveal the variation of the value $w_{ji}x_j$. However, still node i should separate w_{ji} from x_j. In any case, there exist scenarios in which our scheme may not be an appropriate solution because of this: for example, in a smart power metering infrastructure one could know when the owner of a house arrives home because power consumption increases.

5.5 Experiments

The purpose of the experiments we have carried out is to show the viability of our proposal and its superiority in terms of fault-tolerance, churn tolerance and performance over SSS. Real experimentation on peer-to-peer networks is always problematic given the impossibility of having a real, large enough testbed. Hence, we used the widely known event-based PeerSim simulator [66].

Power Iteration is an iterative algorithm for finding the dominant eigenvector of a matrix. Here we use its distributed version presented in [48], which is based on the asynchronous chaotic model introduced in [61]. The matrix is given by the network itself, being its link weights the entries of the matrix (missing links represent a zero value entry). Every node calculates a single element of the dominant eigenvector: these are the private values to compute. For the sake of comparison, each node's private value was initialized to 1.0, although in a real deployment we suggest the use of initial random values when possible to mask variations on the private value.

The implementation of our proposal on top of Power Iteration was previously shown in Algorithms 12 and 13. For the sake of fairness, we have also implemented SSS on top of it.

The method introduced in [61] computes the dominant eigenvalue of non-negative, irreducible matrices with a spectral radius of one. For this reason, we have used two artificially generated sparse matrices. The generation process for them is described in [48] and we recall it next. Each matrix has 5000 nodes.

The first matrix, called "random k-out normalized" (Rnd) was generated by adding 8 random out-links to each node. This would represent a baseline normal, random network. The second matrix, or "small gap normalized" (SmlG), was generated by placing all nodes in an undirected ring and adding two random out-links to every node. This generation process ends up in a matrix with a small gap between its two largest eigenvalues: a feature that makes Power Iteration converge very slowly. We use this matrix to test the two algorithms in a harsh scenario. Finally, both matrices were normalized so the sum of each column is 1 in order to ensure that the spectral radius is one.

5.5.1 Scenario setup

Let us now present the setup of some important parameters. The security parameter $k = |OC^{[ji]}|$ was set to a constant value of 3 in SSS. This deliberately small value allows SSS to tolerate the loss of a large number of S messages at the expense of a lower level of privacy. In our proposal, the same parameter is randomly chosen from $[1,\ \frac{|IN^{[i]}|}{2}]$ by each node i. In addition to that, our scheme looks for new out-collaborators at the time of share renewal if no collaborators are thought to be online, but never discards old collaborators (this means that nodes in our scheme will find themselves in harder conditions than their equivalents in SSS). This naive policy is probably not the best one for real deployments, but suffices for our testing purposes. Parameter Δ corresponds to the cycle length for both SSS and our proposal (see Alg. 12). The $s\_timeout$ parameter in our scheme is randomly drawn from $[150\Delta,\ 300\Delta]$.

Regarding drop rates and delays, our aim was to compare the behaviour of the algorithms under difficult conditions and to show that our method can face unreliable and heterogeneous network links. For that reason, we chose a loss rate of $Drop = 0.1$ (10% of the messages are lost, which is already a high rate). We also used a loss rate of $Drop = 0$ (no loss) as a baseline case for comparison. For message delay we draw a uniformly distributed random value from $[0,\ 0.1\Delta]$ or from $[0, \Delta]$ (a very large maximum boundary) for every message. We also consider zero delay (instant arrival) as an ideal case.

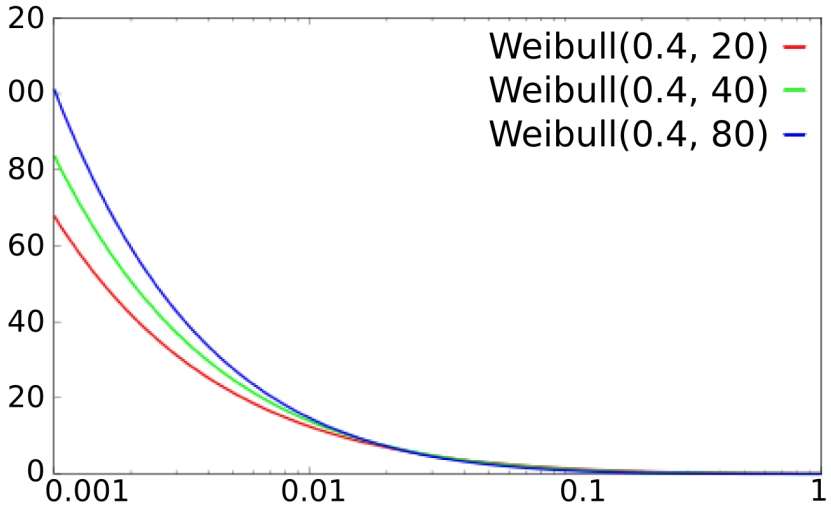
	Session length	
	Online	**Offline**
No churn	∞	0
Fast churn	Weibull$(0.4, 20\Delta)$	Weibull$(0.4, 40\Delta)$
Slow churn	Weibull$(0.4, 40\Delta)$	Weibull$(0.4, 80\Delta)$

Table 5.1: **Churn scenarios considered in the simulations.** Session lengths are measured in simulation cycles.

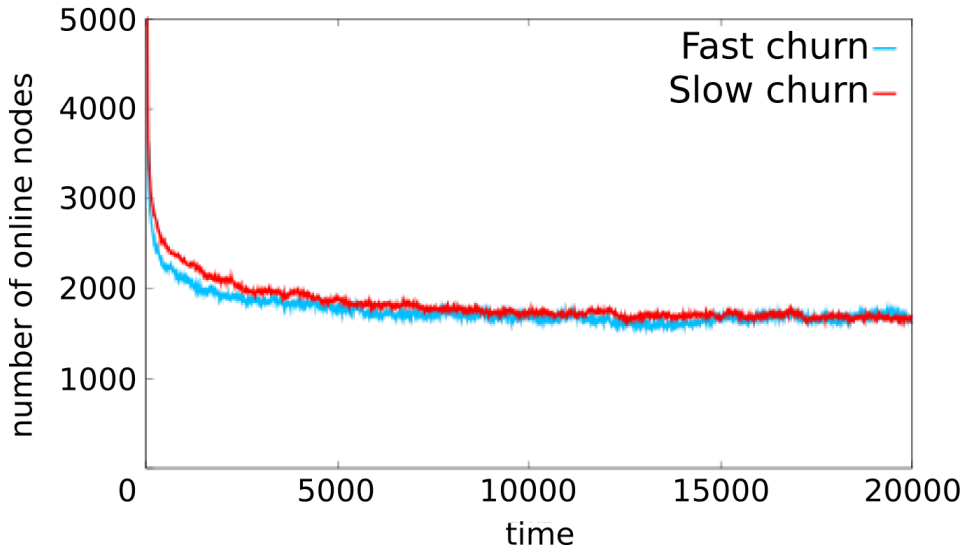
Node churn was modelled basing on realistic patterns taken from [97]. The authors of that work suggest that the Weibull$(0.4, b)$ distribution (shape 0.4, scale b) accurately reproduces the churn patterns found in real traffic samples. We have designed three different scenarios by varying the scale parameter. Table 5.1 shows them, and the Weibull functions used are depicted in Figure 5.2(a). In every scenario, each time a node changes from online to offline state of vice versa we draw a random value from the corresponding Weibull function. This value tells the node how many cycles she will be in the new state. Figure 5.2(b) shows the online population of the two non zero churn models during a sample execution. Note that the number of online nodes is similar in both cases; however the behaviour of nodes is different as they join or leave the network for longer periods of time in the case of slow churn. This makes a difference as we will see soon. Note also that simulating a slower churn with the same Δ is equivalent to assuming the same fast churn but with a smaller cycle length Δ.

It is important to remark that in the experiments nodes join back keeping their previous state, which includes the secret value and the neighbours. By doing this the solution dominant eigenvector remains static and we can calculate the distance from it to the currently approximate solution.

Now let us explain how correctness of results was measured. The actual dominant eigenvector $\mathbf{w}$ was precalculated for both matrices: $\mathbf{w}$ is actually a 5000-dimensional euclidean vector. During every experiment, the current solution $\mathbf{x}$ is obtained after each simulation cycle and the angle between it and $\mathbf{w}$ is calculated by means of the formula $\arccos \frac{||\mathbf{w}^T \cdot \mathbf{x}||}{||\mathbf{w}||\ ||\mathbf{x}||}$. This angle tends to zero as the calculated solution approaches the actual solution. Simulations are stopped when the angle is



(a) Weibull functions used.



(b) Online population during a sample execution.

Figure 5.2: **Churn modelling.**

less than ϵ. For Rnd we use $\epsilon = 0.05$ and for SmlG, $\epsilon = 0.1$. We say then that the algorithm has converged.

On the other hand, we need a measure for algorithm performance. Time is measured in *iterations* in PeerSim; however using the number of iterations needed until convergence would not be fair since SSS is synchronous and our proposal is asynchronous; thus, the concept of *iteration* makes little sense in our case. For this reason we use the average number of messages sent per node until convergence. In addition to that, this measure takes into account the different message complexities in a natural way. The total number of messages sent throughout the network can be easily found by multiplying the average number of messages per node by the network size.

5.5.2 Results

Let us turn, at last, to the results of the simulations. Tables 5.2 and 5.3 show the average number of messages sent per node under the scenarios introduced above and on the RnD and SmlG matrices, respectively. These results were averaged from three independent runs and rounded to an integer. A single run of each experiment is shown in Figures 5.3 to 5.6 for the same scenarios.

The only results shown for SSS correspond to the no churn scenario. This is because SSS did not make any observable progress in any churn scenario after several hours of simulation. On the other hand, our scheme converges under all circumstances and, furthermore, we beat SSS in absence of churn. When conditions are ideal (no message loss nor delay) this is due to our lower message complexity. When introducing loss and delay the fault-tolerant properties of our asynchronous model make the performance difference bigger.

From these scenarios we can learn that in no churn conditions (or under very mild churn) shortening period Δ in our scheme will provide higher performance given that nodes work independently. The larger amount of messages we send to the central node, the faster we will converge. However, in SSS nodes need to wait for each other; that restriction imposes a lower boundary on Δ that will depend on network latency and failure rate.

		SSS		Ours					
		no churn		no churn		fast churn		slow churn	
Drop	Delay	mean	std	mean	std	mean	std	mean	std
0	0	145	0	52	0	3,438	209	4,678	731
0	$\in [0, 0.1\Delta]$	144	0	54	0	3,393	347	5,565	1,079
0	$\in [0, \Delta]$	745	0	117	0	6,776	258	10,031	369
0.1	0	225	32	80	0	4,772	397	7,604	1,278
0.1	$\in [0, 0.1\Delta]$	248	32	90	0	5,125	232	8,234	1,367
0.1	$\in [0, \Delta]$	9,774	186	169	0	7,068	424	12,621	1,199

Table 5.2: Average number of messages sent per node on Rnd. The mean and the standard deviation are shown for three independent runs. In the presence of churn we could not run simulations long enough to reach convergence with SSS.

		SSS		Ours					
		no churn		no churn		fast churn		slow churn	
Drop	Delay	mean	std	mean	std	mean	std	mean	std
0	0	1,273	0	139	0	56,272	3,215	165,930	45,496
0	$\in [0, 0.1\Delta]$	1,279	10	155	2	63,027	1,529	161,080	25,684
0	$\in [0, \Delta]$	14,079	1,305	303	6	68,345	4,776	158,360	23,254
0.1	0	2,548	35	175	2	63,369	11,877	116,800	6,329
0.1	$\in [0, 0.1\Delta]$	2,609	82	191	0	63,010	2,308	140,560	20,666
0.1	$\in [0, \Delta]$	67,538	3,883	346	7	70,144	8,101	133,420	19,621

Table 5.3: Average number of messages sent per node on SmlG. The mean and the standard deviation are shown for three independent runs. In the presence of churn we could not run simulations long enough to reach convergence with SSS.

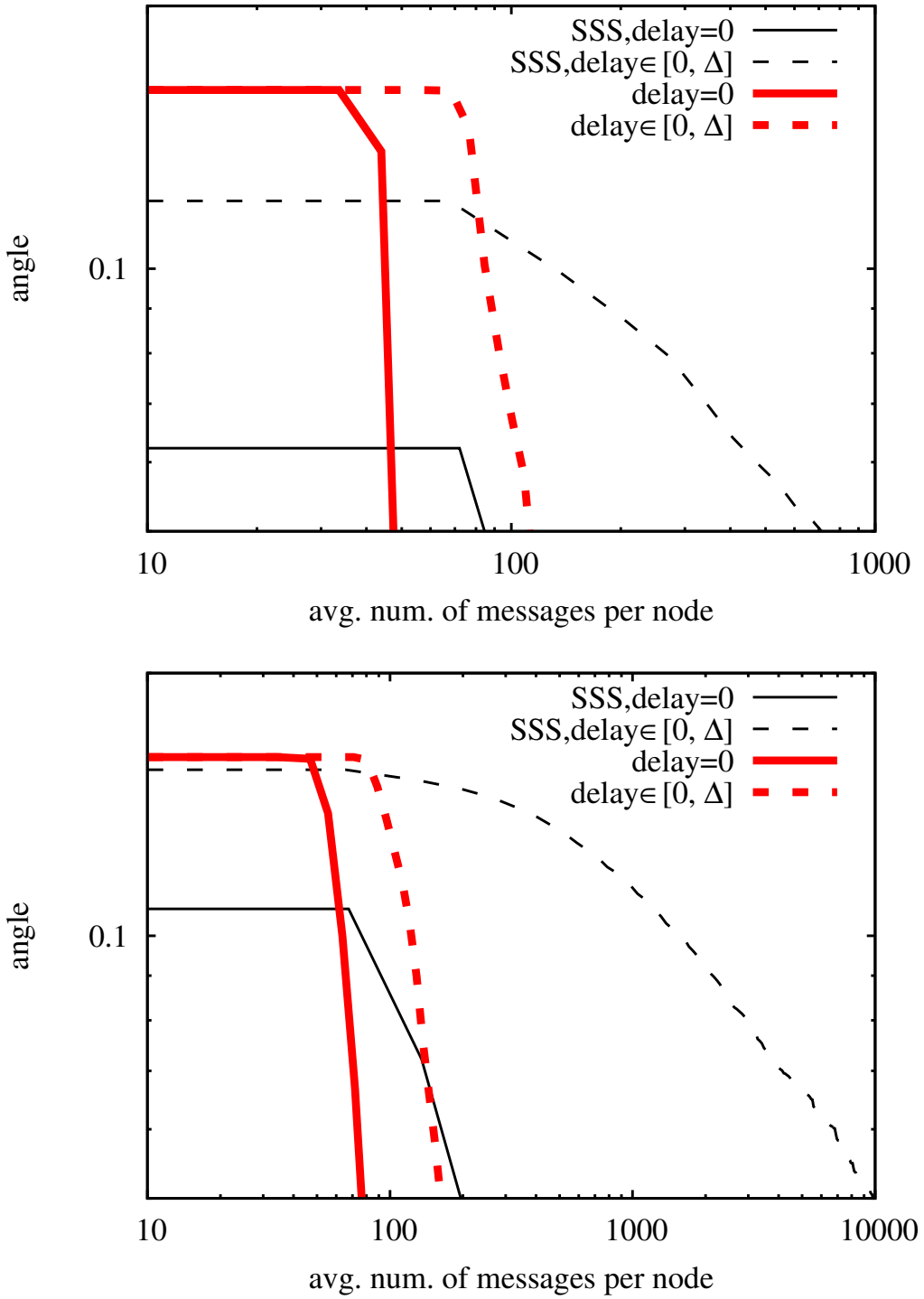


Figure 5.3: **Matrix Rnd with no churn, and message drop probability 0 (top) and 0.1 (bottom).**

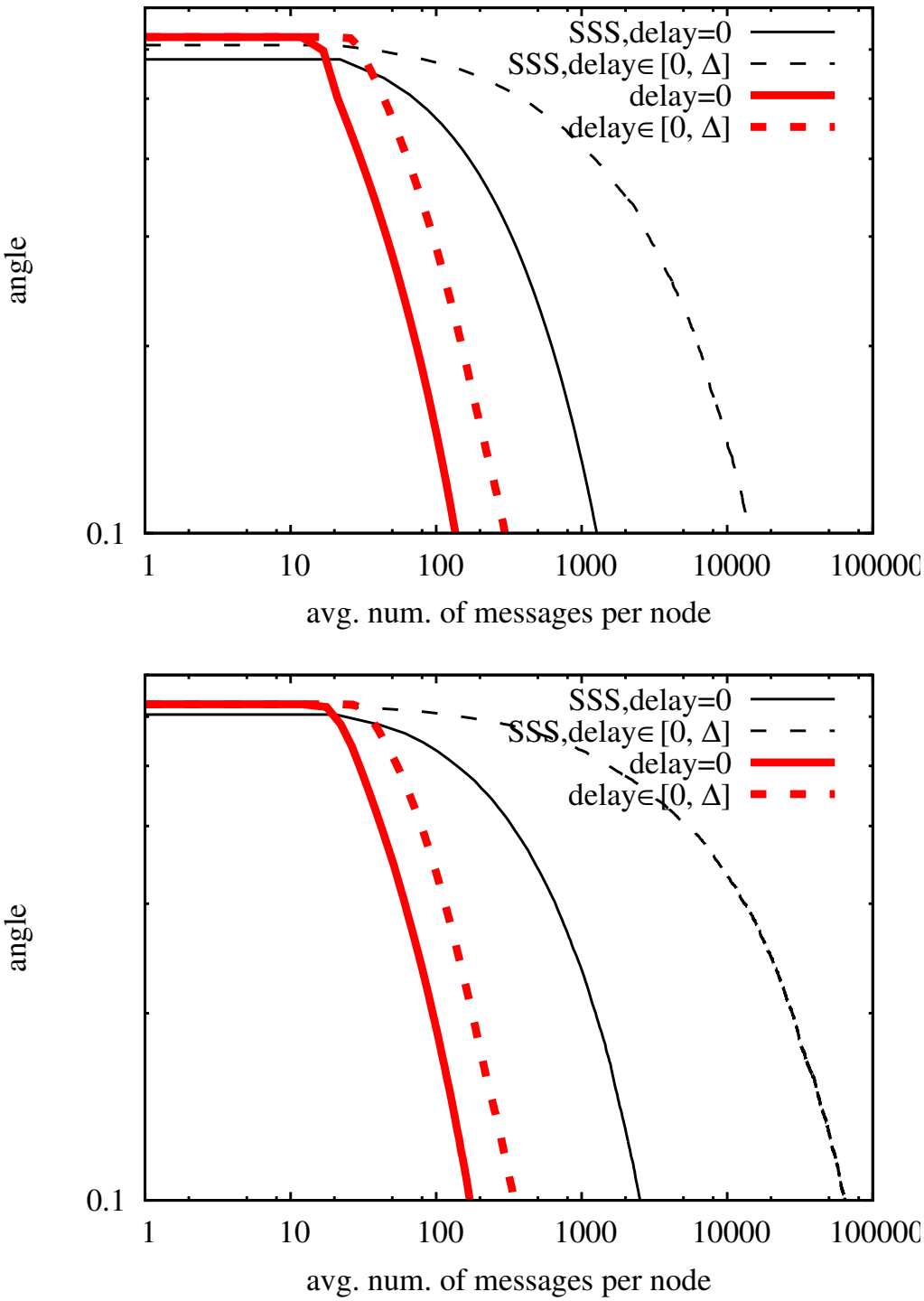


Figure 5.4: **Matrix SmlG with no churn, and message drop probability 0 (top) and 0.1 (bottom).**

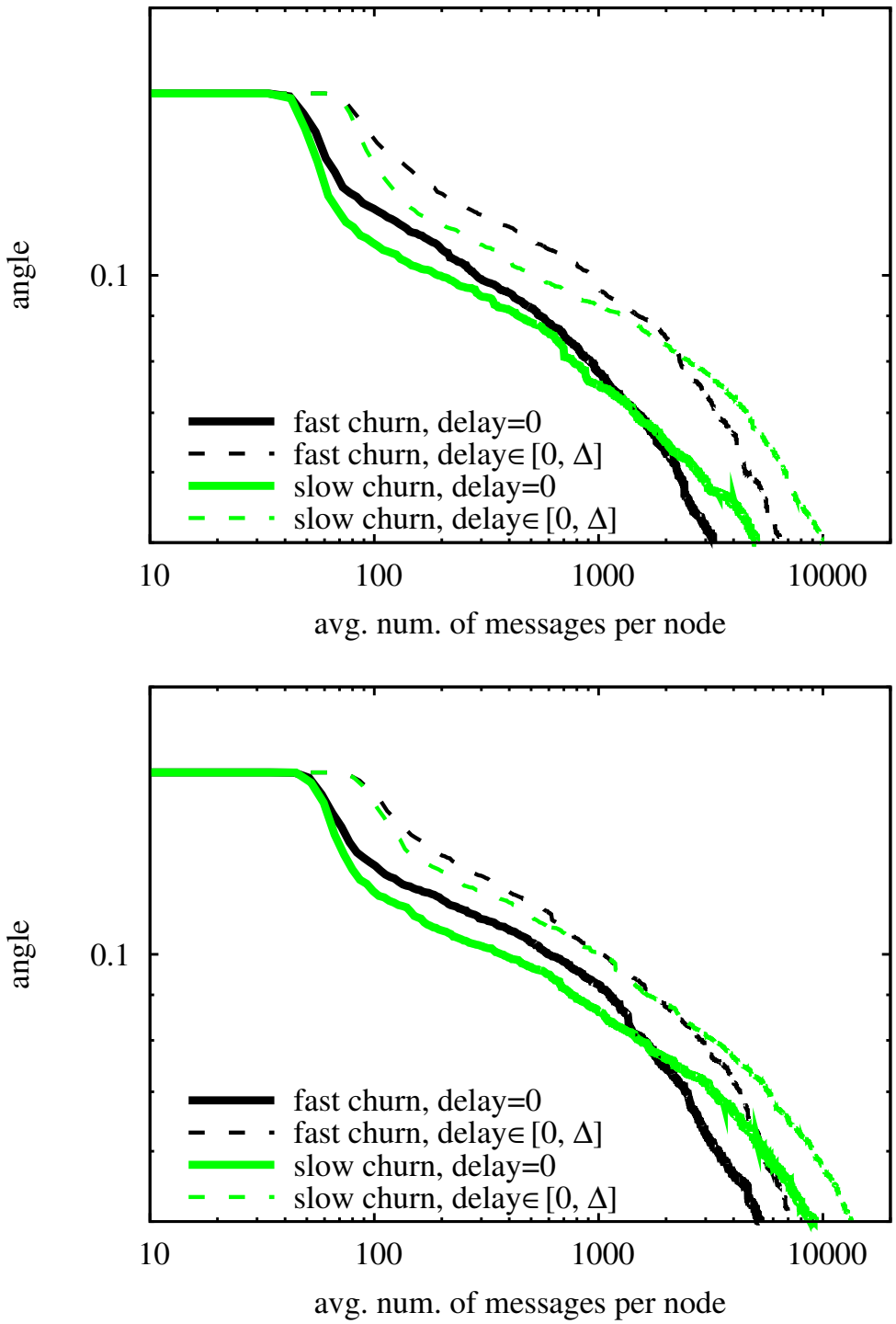


Figure 5.5: **Matrix Rnd with churn, and message drop probability 0 (top) and 0.1 (bottom).**

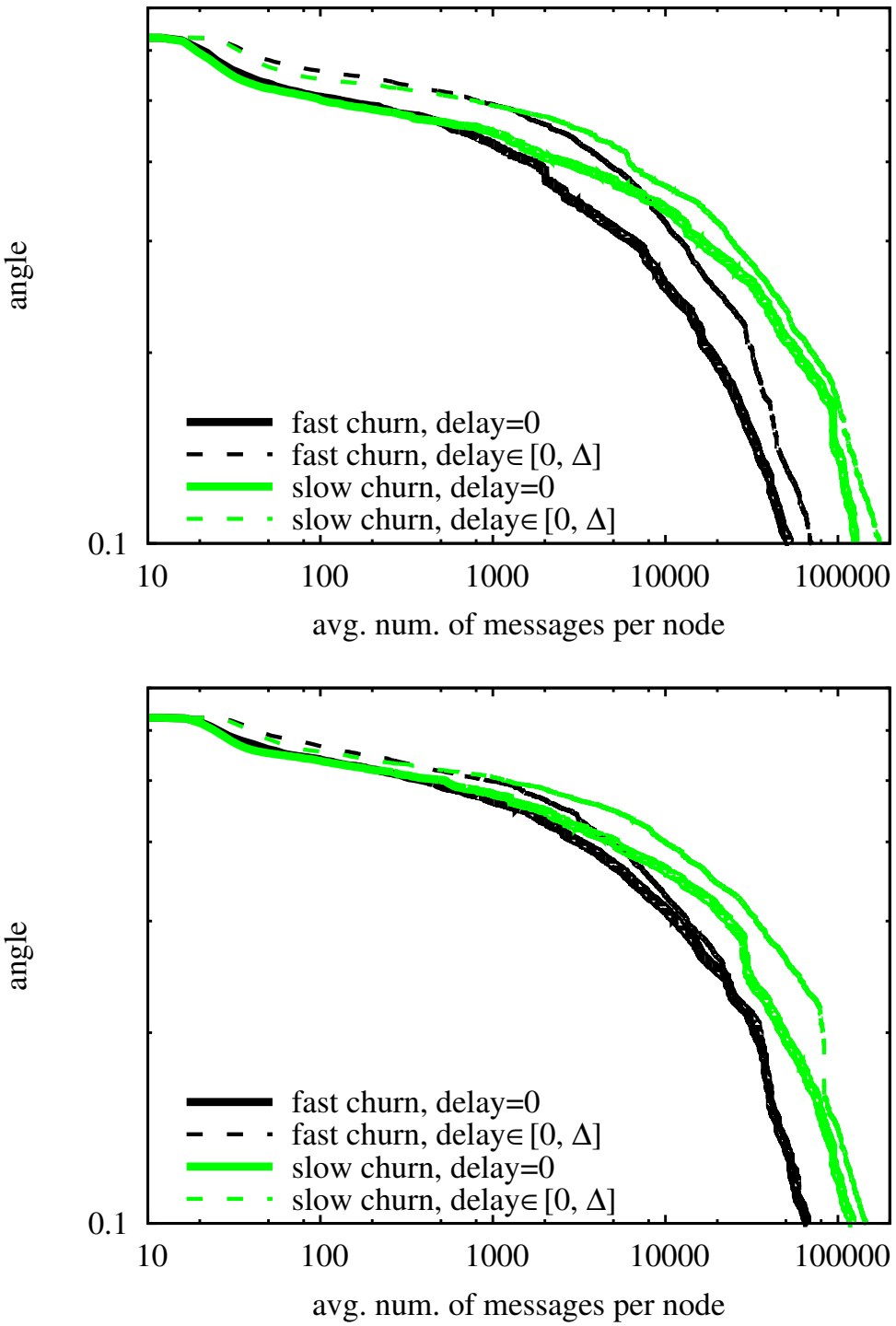


Figure 5.6: **Matrix SmlG with churn, and message drop probability 0 (top) and 0.1 (bottom).**

When considering churn, we see how missing nodes affect convergence speed. This is because churn prevents information spreading in the network. To understand it, consider any neighbourhood. During stable periods (no nodes switching their online state) the information transmitted remains more or less static and eventually becomes repetitive: online in-neighbours of i send very similar Type 3 messages to i during this time. It is not until the set of online nodes varies that fresh information arrives at node i (who in turn will propagate this new information to its own out-neighbours).

This last phenomenon suggests that maintaining short periods Δ does not make sense under slow churn conditions. Instead, we should be able to reduce the frequency at which Type 3 messages are sent without slowing down convergence, while saving on communication. This is because increasing Δ is equivalent to speeding up the churn. Confirming this theory and designing an adaptive mechanism to optimize Δ is an interesting direction for future research.

However, note that this slowdown is in fact a collateral effect of our setup: we are measuring convergence as the distance from a final global correct solution. In practical scenarios there will probably not exist an "end solution": instead, it will likely depend on the set of online nodes at any time, thus evolving with the state of the network. Experiments in that direction would also be interesting for future work.

5.6 Conclusions

This chapter discusses the work presented in [78]: to the best of our knowledge, the first asynchronous distributed algorithm for privacy preservation that suits highly dynamic and unreliable networks such as peer-to-peer overlays. The algorithm allows the computation of linear combinations of secret values while preserving the privacy on those values separately. In order to demonstrate this, we implemented our solution on top of Power Iteration, an asynchronous distributed method that calculates the dominant eigenvector of certain matrices. Any iterative method based on linear combinations can be implemented, though.

We compared our idea with the closest solution available (a synchronous one) and demonstrated how asynchronism offers far better results when facing network failures and node churn, and even in ideal conditions.

In terms of privacy, our scheme allows flexible setups and better privacy on one hand, if we attend to the number of colluding nodes supported. On the other hand, in our scheme it might be possible to learn trends in other nodes' secret values.

There are several possible work lines for future work. First, more experimentation would be needed under slow churn conditions: it would be interesting to test our theory regarding the adaptation of the period Δ in order to optimize the number of messages sent (i.e. the bandwidth consumed by our algorithm). Second, carrying out experiments in a "dynamic solution" setup (see the last paragraph of Section 5.5.2) would provide interesting information. Finally, the next natural step is to adapt our algorithm to support malicious adversaries.

Chapter

6

Conclusions

This thesis dissertation has dealt with two different topics within the information privacy field. In this chapter we give a brief summary of the main issues and conclusions, and suggest some research directions that may be of interest, in our opinion, for future work.

6.1 Centralized secure multicast

The first part of the dissertation is focused on centralized secure multicast schemes. Chapter 2 provides a deep survey of the field's state of the art. We arrange the surveyed works into three categories.

First, there is a large group of schemes that can be thought of as "generalists" since they provide good performance in generic scenarios. These schemes usually do not consider reliability problems or more than one transmission channel. Most of them rely on the key hierarchy tree idea (recall Section 2.2), which allows them to give service to large audiences. Another group of generalist schemes relies on algebraic tricks, and allow the Key Server to refresh the encryption key using a single message (something that the tree approach does not). However, these schemes do not usually scale well for computational reasons and can reach smaller audiences than their tree-based counterparts. The general category is the oldest of the three.

We believe it is rather mature nowadays and little surprises are expected. If any, they could come from the computational branch.

The second group includes schemes that are technically similar to the general category but provide a service based on different information channels. In the scenarios addressed by these schemes, audience members are divided into different sets according to access control rules. For example, they may exist different channel packs in a pay-per-view digital television system, or there might be different, hierarchical communication channels in army operations. This differentiation is achieved by building a different tree for every group of members and joining them together. It is the first time that this category of schemes is considered on a survey to the best of our knowledge.

The third group is probably the most active today, with many schemes appearing lately: we refer to self-healing multicast schemes. They are designed for wireless scenarios where reliability is a key issue: if a re-keying message is lost then many users can be left out without service. The solution is to provide messages that allow the user to recover a given number of previous session keys, so members who miss a key can recover it from the next re-key message. Research within this category has a great momentum and its related literature grows quickly, so we expect many more developments here in the near future given that wireless communications tend to gain more and more importance.

In Chapter 3 we presented a suit of algorithms for a centralized secure multicast scenario composed of three different schemes. The first one is a computational centralized secure multicast algorithm. This is the cornerstone of the whole proposal. It allows the distribution of a secret (a session key, for example) with a single message by exploiting the Extended Euclidean Algorithm. It is shown to be fast at the cost of a message length that grows linearly with the number of recipients. The second scheme allows audience members to verify the authenticity of messages sent by the Key Server when executing the first scheme. This verification intends to be an alternative to digital signatures since no public key cryptography is involved. Finally, the third scheme is a zero-knowledge protocol that allows a legal member of the audience to verify the membership of another user with the aid of the Key Server.

After publication of the three schemes, a cryptanalysis was applied on them by Peinado et al. They posed serious doubts regarding the security of the first one, and break the security of the second and third. Later, Antequera et al. demonstrated that the first algorithm is computationally secure so far and provided a secure version of the third scheme. The second scheme is currently broken. Antequera et al. have also combined our scheme with a key hierarchy tree in order to reach bigger audiences, on one hand, and have built a distributed secure multicast framework that uses our scheme within every cluster of members.

There exist some research lines as a continuation of this work. For example, exploiting recent advances in parallel hardware and programming techniques could make computational general schemes faster to compute, thus being able to reach bigger audiences. This would be specially interesting in the case of the Secure Lock scheme. Another interesting research work line is trying to break the linear growth of the message length in the first scheme of our own solution. That would allow our scheme to reach bigger audiences without the help of a key hierarchy tree. Finally, it would be interesting to solve the security flaw in the the message authentication scheme.

6.2 Privacy-preserving computations on peer-to-peer networks

The second part of this dissertation deals with preserving privacy in distributed computations over a peer-to-peer network. By this we mean that nodes obtain a global view on a given value over the network without knowing the individual values of their neighbours.

Chapter 4 surveys the literature. Even though the problem of computing functions in a distributed manner has been widely studied, most of the proposals are of difficult application in practice. For example, the vast majority of works to the date support a reduced number of players (nodes participating in the computations) and can not cope with difficulties such as network failure or churn (sudden absence or rejoin of nodes). The most advanced solutions in this sense still impose synchronous restrictions on the execution, which makes their application a hard task,

specially in problematic scenarios such as peer-to-peer or wireless sensor networks.

In Chapter 5 we introduce our own proposal by extending a work by Bickson et al. Our scheme is asynchronous, extremely fault tolerant and resistant to node churn. Experiments with the PeerSim simulator show that it offers better performance than the proposal by Bickson et al. and suggest that it can be used in practice.

Regarding future work several directions arise. First, our solution tolerates semi-honest adversaries: it seems logical to try to extend it so it can cope with malicious adversaries. Second, more experimentation with different setups would offer interesting information. Third, adapting the behaviour of our scheme (specially the frequency at which messages are sent) to churn conditions seems to offer promising results in terms of bandwidth savings.

Appendix

A

Publications arisen from this thesis

The research work carried out for the present thesis resulted in a number of publications. This appendix lists them along with their respective quality indicators and sorted by their year of publication (oldest first) within each category.

A.1 Publications in international journals

[76] Naranjo, J.A.M., Casado, L.G. & López-Ramos, J.A. (2011). Group oriented renewal of secrets and its application to secure multicast. *Journal of Information Science and Engineering*, **27**, 1303–1313

Impact factor JCR 2011: 0.175. Journal ranking: 132/135 in *Computer Science, Information Systems*.

[70] Naranjo, J.A.M. & Casado, L.G. (2012). An updated view on centralized secure group communications. *Logic Journal of the IGPL*, DOI: 10.1093/jigpal/jzs026

Impact factor JCR 2011: 0.913. Journal ranking: 1/19 in *Logic*, 50/289 in *Mathematics*, 85/245 in *Mathematics, Applied*.

[77] Naranjo, J.A.M., Antequera, N., Casado, L.G. & López-Ramos, J.A. (2012). A suite of algorithms for key distribution and authentication in centralized secure multicast environments. *Journal of Computational and Applied Mathematics*, **236**, 3042–3051, DOI: 10.1016/j.cam.2011.02.015
Impact factor JCR 2011: 1.112. Journal ranking: 62/245 in *Mathematics, Applied*.

[78] Naranjo, J.A.M., Casado, L.G. & Jelasity, M. (2012). Asynchronous privacy-preserving iterative computation on peer-to-peer networks. *Computing*, **94**, 763–782, DOI: 10.1007/s00607-012-0200-5
Impact factor JCR 2011: 0.701. Journal ranking: 51/99 in *Computer Science, Theory & Methods*.

A.2 Publications in proceedings of international conferences with DOI

[69] Naranjo, J.A.M. & Casado, L.G. (2011). Keeping group communications private: An up-to-date review on centralized secure multicast. In A. Herrero & E. Corchado, eds., *Computational Intelligence in Security for Information Systems*, vol. 6694 of *Lecture Notes in Computer Science*, 151–159, Springer Berlin Heidelberg, DOI: 10.1007/978-3-642-21323-6_19

A.3 Publications in other international conferences

[74] Naranjo, J.A.M., López-Ramos, J.A. & Casado, L.G. (2010). Applications of the Extended Euclidean Algorithm to privacy and secure communications. In *Proceedings of the 10th International Conference on Mathematical Methods in Science and Engineering*, 702–713

A.4 Publications in national conferences

[73] Naranjo, J.A.M., Casado, L.G. & López-Ramos, J.A. (2010). Key refreshment in overlay networks: a centralized secure multicast scheme proposal. In *Actas de las XXI Jornadas de Paralelismo*, 931–938

Appendix

B

Other publications produced during the elaboration of this thesis

The research effort invested during the time span in which this thesis was elaborated produced some additional publications as the result of other research lines not included in the present dissertation. Those lines were *key distribution in multimedia streaming peer-to-peer networks* ([71, 72]), *authentication in peer-to-peer networks* ([79, 80]) and *access control in wireless sensor networks* ([81]). This appendix lists them along with their respective quality indicators and sorted by their year of publication (oldest first) within each category.

B.1 Publications in international journals

[80] Naranjo, J.A.M., Cores, F., Casado, L.G. & Guirado, F. (2012). Fully distributed authentication with locality exploitation for the CoDiP2P peer-to-peer computing platform. *Journal of Supercomputing*, 1–13, DOI: 10.1007/s11227-012-0842-2

Impact factor JCR 2011: 0.578. Journal ranking: 37/50 in *Computer science, hardware & architecture*, 68/99 in *Computer science, theory & methods*, 180/245 in *Engineering, electrical & electronic*.

B.2 Publications in proceedings of international conferences with DOI

[72] Naranjo, J.A.M., López-Ramos, J.A. & Casado, L.G. (2009). Key management schemes for peer-to-peer multimedia streaming overlay networks. In O. Markowitch, A. Bilas, J.H. Hoepman, C. Mitchell & J.J. Quisquater, eds., *Information Security Theory and Practice. Smart Devices, Pervasive Systems, and Ubiquitous Networks*, vol. 5746 of *Lecture Notes in Computer Science*, 128–142, Springer Berlin Heidelberg, DOI: 10.1007/978-3-642-03944-7_10

Most Innovative Technology-related Paper Award.

[75] Naranjo, J.A.M., López-Ramos, J.A. & Casado, L.G. (2010). A key distribution scheme for live streaming multi-tree overlays. In A. Herrero, E. Corchado, C. Redondo & A. Alonso, eds., *Computational Intelligence in Security for Information Systems 2010*, vol. 85 of *Advances in Intelligent and Soft Computing*, 223–230, Springer Berlin Heidelberg, DOI: 10.1007/978-3-642-16626-6_24

[81] Naranjo, J.A.M., Orduña, P., Gómez-Goiri, A., López-de Ipiña, D. & Casado, L.G. (2012). Lightweight user access control in energy-constrained wireless network services. In J. Bravo, D. López-de Ipiña & F. Moya, eds., *Ubiquitous Computing and Ambient Intelligence*, vol. 7656 of *Lecture Notes in Computer Science*, 33–41, Springer Berlin Heidelberg, DOI: 10.1007/978-3-642-35377-2_5

B.3 Publications in other international conferences

[79] Naranjo, J.A.M., Cores, F., Casado, L.G. & Guirado, F. (2012). A fully distributed authentication model for the CoDiP2P peer-to-peer computing plat-

form. In *Proceedings of the 12th International Conference on Mathematical Methods in Science and Engineering*, 923–934

B.4 Publications in national conferences

[71] Naranjo, J.A.M., López-Ramos, J.A. & Casado, L.G. (2009). Esquemas dinámicos de distribución de claves en redes peer-to-peer multimedia. In *Actas de las XX Jornadas de Paralelismo*, 583–586

Bibliography

[1] Aggarwal, D. & Maurer, U. (2008). Breaking RSA generically is equivalent to factoring. Cryptology ePrint Archive, Report 2008/260.

[2] Antequera, N. & Lopez-Ramos, J.A. (2011). Hierarchical approaches for multicast based on the Euclid's algorithm. *Repositorio institucional de la Universidad de Almería*, http://hdl.handle.net/10835/355.

[3] Antequera, N. & Lopez-Ramos, J.A. (2011). Remarks and countermeasures on a cryptoanalysis of a secure multicast protocol. In *Next Generation Web Services Practices (NWeSP), 2011 7th International Conference on*, 210 – 214.

[4] Bai, Z., Demmel, J., Dongarra, J., Ruhe, A. & van der Vorst, H., eds. (2000). *Templates for the Solution of Algebraic Eigenvalue Problems: a Practical Guide*. SIAM, Philadelphia.

[5] Ben-David, A., Nisan, N. & Pinkas, B. (2008). Fairplaymp: a system for secure multi-party computation. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, 257–266, ACM, New York, NY, USA.

[6] Ben-Or, M., Goldwasser, S. & Wigderson, A. (1988). Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, 1–10, ACM, New York, NY, USA.

[7] Bianchini, M., Gori, M. & Scarselli, F. (2005). Inside pagerank. *ACM Transactions on Internet Technology*, **5**, 92–128.

[8] Bickson, D. & Malkhi, D. (2008). A unifying framework of rating users and data items in peer-to-peer and social networks. *Peer-to-Peer Networking and Applications*, **1**, 93–103.

[9] Bickson, D., Dolev, D., Bezman, G. & Pinkas, B. (2008). Peer-to-Peer secure multi-party numerical computation. In *IEEE International Conference on Peer-to-Peer Computing*, 257–266, IEEE Computer Society.

[10] Bickson, D., Reinman, T., Dolev, D. & Pinkas, B. (2010). Peer-to-peer secure multi-party numerical computation facing malicious adversaries. *Peer-to-Peer Networking and Applications*, **3**, 129–144.

[11] Blundo, C., D'arco, P., De Santis, A. & Listo, M. (2004). Design of self-healing key distribution schemes. *Des. Codes Cryptography*, **32**, 15–44.

[12] Blundo, C., D'Arco, P. & De Santis, A. (2006). On self-healing key distribution schemes. *Information Theory, IEEE Transactions on*, **52**, 5455 –5467.

[13] Canetti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M. & Pinkas, B. (1999). Multicast security: A taxonomy and some efficient constructions. In *INFOCOM '99. Proceedings. IEEE*, 708–716.

[14] Canetti, R., Malkin, T. & Nissim, K. (1999). Efficient communication-storage tradeoffs for multicast encryption. In *Proceedings of EUROCRYPT'99*, 459–474, Springer-Verlag.

[15] Canny, J. (2002). Collaborative filtering with privacy via factor analysis. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, 238–245, ACM, New York, NY, USA.

[16] Chan, K.C. & Chan, S.H. (2003). Key management approaches to offer data confidentiality for secure multicast. *Network, IEEE*, **17**, 30–39.

[17] Chan, K.C. & Chan, S.H.G. (2001). Distributed servers networks for secure multicast. In *Proc. IEEE Globecom'01*, 25–29.

[18] Chan, K.C. & Chan, S.H.G. (2002). Distributed servers approach for large-scale secure multicast. *IEEE JSAC*, **20**.

[19] Chang, I., Engel, R., Kandlur, D., Pendarakis, D. & Saha, D. (1999). Key management for secure internet multicast using boolean function minimization techniques. In *INFOCOM '99. Proceedings. IEEE*, 689–698.

[20] Chen, L. (2008). Recommendation for key derivation using pseudorandom functions. NIST Special Publication 800-108.

[21] Chiou, G.H. & Chen, W.T. (1989). Secure broadcasting using the secure lock. *IEEE Trans. Softw. Eng.*, **15**, 929–934.

[22] Cisco (2006). IOS Secure Multicast. http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6552/prod_white_paper0900aecd8047191e.html.

[23] Clifton, C., Kantarcioglu, M., Vaidya, J., Lin, X. & Zhu, M.Y. (2002). Tools for privacy preserving distributed data mining. *SIGKDD Explor. Newsl.*, **4**, 28–34.

[24] Das, K., Bhaduri, K. & Kargupta, H. (2011). Multi-objective optimization based privacy preserving distributed data mining in peer-to-peer networks. *Peer-to-Peer Networking and Applications*, **4**, 192–209.

[25] Datta, S., Bhaduri, K., Giannella, C., Wolff, R. & Kargupta, H. (2006). Distributed data mining in peer-to-peer networks. *IEEE Internet Computing*, **10**, 18–26.

[26] Di Pietro, R., Mancini, L. & Jajodia, S. (2002). Efficient and secure keys management for wireless mobile communications. In *Proceedings of POMC'02*, 66–73.

[27] Dierks, T. & Rescorla, E. (2006). The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346. http://tools.ietf.org/html/rfc4346.

[28] Dinur, I. & Nissim, K. (2003). Revealing information while preserving privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '03, 202–210, ACM, New York, NY, USA.

[29] Du, W. & He, M. (2008). Self-healing key distribution with revocation and resistance to the collusion attack in wireless sensor networks. In *Proceedings of ProvSec'08*, 345–359, Springer-Verlag.

[30] Du, W., He, M. & Li, X. (2009). A new constant storage self-healing key distribution with revocation in wireless sensor networks. In *Proceedings of ICA3PP'09*.

[31] Dutta, H., Kargupta, H., Datta, S. & Sivakumar, K. (2003). Analysis of privacy preserving random perturbation techniques: further explorations. In *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, WPES '03, 31–38, ACM, New York, NY, USA.

[32] Dutta, R., Dong, Y. & Mukhopadhyay, W. (2007). Constant storage self-healing key distribution with revocation in wireless sensor network. In *Proceedings of ICC '07*, IEEE.

[33] ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, 10–18, Springer-Verlag New York, Inc., New York, NY, USA.

[34] Frommer, A. & Szyld, D.B. (2000). On asynchronous iterations. *Journal of Computational and Applied Mathematics*, **123**, 201–216.

[35] Goldreich, O., Micali, S. & Wigderson, A. (1987). How to Play Any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*

(STOC '87), Annual ACM Symposium on Theory of Computing, 218–229, New York, NY, USA.

[36] Golub, G.H. & Van Loan, C.F. (1996). *Matrix Computations*. The Johns Hopkins University Press, 3rd edn.

[37] Gordon, J. (1984). Strong primes are easy to find. In *Advances in Cryptology: Proceedings of EUROCRYPT 84, A Workshop on the Theory and Application of of Cryptographic Techniques, Paris, France, April 9-11, 1984, Proceedings*, vol. 209 of *Lecture Notes in Computer Science*, 216–223, Springer.

[38] Halevy, D. & Shamir, A. (2002). The LSD broadcast encryption scheme. In *Proceedings of CRYPTO'02*, 47–60.

[39] Harney, H. & Muckenhirn, C. (1997). Group key management protocol (GKMP) specification. RFC 2093.

[40] He, W., Liu, X., Nguyen, H.V., Nahrstedt, K. & Abdelzaher, T. (2011). PDA: Privacy-preserving data aggregation for information collection. *ACM Trans. Sen. Netw.*, **8**, 6:1–6:22.

[41] Heydari, M.H., Morales, L. & Sudborough, I.H. (2006). Efficient algorithms for batch re-keying operations in secure multicast. In *Proceedings of HICSS'06*.

[42] Hong, D. & Kang, J.S. (2005). An efficient key distribution scheme with self-healing property. *Communications Letters, IEEE*, **9**, 759 – 761.

[43] Horng, G. (2002). Cryptanalysis of a key management scheme for secure multicast communications. *IEICE Transactions on Communications*, **E85-B**, 1050–1051.

[44] Hur, J. & Yoon, H. (2010). A multi-service group key management scheme for stateless receivers in wireless mesh networks. *Mob. Netw. Appl.*, **15**, 680–692.

[45] Ishai, Y., Kushilevitz, E. & Paskin, A. (2010). Secure multiparty computation with minimal interaction. In T. Rabin, ed., *Advances in Cryptology – CRYPTO 2010*, vol. 6223 of *Lecture Notes in Computer Science*, 577–594, Springer Berlin Heidelberg.

[46] J. Xia, R.Y. & X., A. (2009). A new efficient hierarchical key agreement scheme. In *Proceedings of NSWCTC'09*, 319–322.

[47] Jelasity, M., Montresor, A. & Babaoglu, O. (2005). Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, **23**, 219–252, 10.1007/s00607-012-0200-5.

[48] Jelasity, M., Canright, G. & Engø-Monsen, K. (2007). Asynchronous distributed power iteration with gossip-based normalization. In A.M. Kermarrec, L. Bougé & T. Priol, eds., *Euro-Par 2007*, vol. 4641 of *Lecture Notes in Computer Science*, 514–525, Springer-Verlag.

[49] Kamvar, S.D., Schlosser, M.T. & Garcia-Molina, H. (2003). The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web (WWW'03)*, 640–651, ACM Press, New York, NY, USA.

[50] Kamvar, S.D., Schlosser, M.T. & Garcia-Molina, H. (2003). The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, 640–651, ACM, New York, NY, USA.

[51] Kempe, D. & McSherry, F. (2004). A decentralized algorithm for spectral analysis. In *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC'04)*, 561–568, ACM, New York, NY, USA.

[52] Kempe, D., Dobra, A. & Gehrke, J. (2003). Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, 482–491, IEEE Computer Society.

[53] Koo, H.S., Kwon, O.H. & Ra, S.W. (2009). A tree key graph design scheme for hierarchical multi-group access control. *Communications Letters, IEEE*, **13**, 874 –876.

[54] Krawczyk, H. (2010). Cryptographic extraction and key derivation: the HKDF scheme. In *Proceedings of CRYPTO'10*, 631–648, Springer-Verlag.

[55] Kruus, P.S. (1998). A survey of multicast security issues and architectures. In *Proceedings of 21st National Information Systems Security Conference*, 5–8.

[56] Ku, W.C. & Chen, S.M. (2003). An improved key management scheme for large dynamic groups using one-way function trees. In *Parallel Processing Workshops, 2003. Proceedings. 2003 International Conference on*, 391–396.

[57] Liang, X., Lu, R., Lin, X. & Shen, X. (2010). Ppc: Privacy-preserving chatting in vehicular peer-to-peer networks. In *Vehicular Technology Conference Fall (VTC 2010-Fall), 2010 IEEE 72nd*, 1 –5.

[58] Lin, J., Huang, K., Lai, F. & Lee, H. (2009). Secure and efficient group key management with shared key derivation. *Comput. Stand. Inter.*, **31**, 192 – 208.

[59] Lindell, Y. & Pinkas, B. (2000). Privacy preserving data mining. In M. Bellare, ed., *Advances in Cryptology — CRYPTO 2000*, vol. 1880 of *Lecture Notes in Computer Science*, 36–54, Springer Berlin Heidelberg.

[60] Liu, D., Ning, P. & Sun, K. (2003). Efficient self-healing group key distribution with revocation capability. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, 231–240.

[61] Lubachevsky, B. & Mitra, D. (1986). A chaotic asynchronous algorithm for computing the fixed point of a nonnegative matrix of unit radius. *Journal of the ACM*, **33**, 130–150.

[62] Malkhi, D., Nisan, N., Pinkas, B. & Sella, Y. (2004). Fairplay—a secure two-party computation system. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, 20–20, USENIX Association, Berkeley, CA, USA.

[63] Masqué, J.M. & Peinado, A. (2006). Cryptanalysis of improved Liaw's broadcasting cryptosystem. *J. Inf. Sci. Eng.*, **22**, 391–399.

[64] Menezes, A., van Oorschot, P. & edition., S.V.F. (1996). *Handbook of applied cryptography*. CRC Press.

[65] Minoli, D. (2008). *IP Multicast with Applications to IPTV and Mobile DVB-H*. Wiley-IEEE Press.

[66] Montresor, A. & Jelasity, M. (2009). Peersim: A scalable P2P simulator. In *Proceedings of the 9th IEEE International Conference on Peer-to-Peer Computing (P2P 2009)*, 99–100, IEEE, Seattle, Washington, USA, extended abstract.

[67] More, S.M., Malkin, M., Staddon, J. & Balfanz, D. (2003). Sliding-window self-healing key distribution. In *Proceedings of the SSRS'03*, 82–90.

[68] Naor, D., Naor, M. & Lotspiech, J. (2001). Revocation and tracing schemes for stateless receivers. In *Proceedings of CRYPTO'01*, 41–62.

[69] Naranjo, J.A.M. & Casado, L.G. (2011). Keeping group communications private: An up-to-date review on centralized secure multicast. In A. Herrero & E. Corchado, eds., *Computational Intelligence in Security for Information Systems*, vol. 6694 of *Lecture Notes in Computer Science*, 151–159, Springer Berlin Heidelberg, DOI: 10.1007/978-3-642-21323-6_19.

[70] Naranjo, J.A.M. & Casado, L.G. (2012). An updated view on centralized secure group communications. *Logic Journal of the IGPL*, DOI: 10.1093/jigpal/jzs026.

[71] Naranjo, J.A.M., López-Ramos, J.A. & Casado, L.G. (2009). Esquemas dinámicos de distribución de claves en redes peer-to-peer multimedia. In *Actas de las XX Jornadas de Paralelismo*, 583–586.

[72] Naranjo, J.A.M., López-Ramos, J.A. & Casado, L.G. (2009). Key management schemes for peer-to-peer multimedia streaming overlay networks. In O. Markowitch, A. Bilas, J.H. Hoepman, C. Mitchell & J.J. Quisquater, eds., *Information Security Theory and Practice. Smart Devices, Pervasive Systems, and Ubiquitous Networks*, vol. 5746 of *Lecture Notes in Computer Science*, 128–142, Springer Berlin Heidelberg, DOI: 10.1007/978-3-642-03944-7_10.

[73] Naranjo, J.A.M., Casado, L.G. & López-Ramos, J.A. (2010). Key refreshment in overlay networks: a centralized secure multicast scheme proposal. In *Actas de las XXI Jornadas de Paralelismo*, 931–938.

[74] Naranjo, J.A.M., López-Ramos, J.A. & Casado, L.G. (2010). Applications of the Extended Euclidean Algorithm to privacy and secure communications. In *Proceedings of the 10th International Conference on Mathematical Methods in Science and Engineering*, 702–713.

[75] Naranjo, J.A.M., López-Ramos, J.A. & Casado, L.G. (2010). A key distribution scheme for live streaming multi-tree overlays. In A. Herrero, E. Corchado, C. Redondo & A. Alonso, eds., *Computational Intelligence in Security for Information Systems 2010*, vol. 85 of *Advances in Intelligent and Soft Computing*, 223–230, Springer Berlin Heidelberg, DOI: 10.1007/978-3-642-16626-6_24.

[76] Naranjo, J.A.M., Casado, L.G. & López-Ramos, J.A. (2011). Group oriented renewal of secrets and its application to secure multicast. *Journal of Information Science and Engineering*, **27**, 1303–1313.

[77] Naranjo, J.A.M., Antequera, N., Casado, L.G. & López-Ramos, J.A. (2012). A suite of algorithms for key distribution and authentication in centralized secure multicast environments. *Journal of Computational and Applied Mathematics*, **236**, 3042–3051, DOI: 10.1016/j.cam.2011.02.015.

[78] Naranjo, J.A.M., Casado, L.G. & Jelasity, M. (2012). Asynchronous privacy-preserving iterative computation on peer-to-peer networks. *Computing*, **94**, 763–782, DOI: 10.1007/s00607-012-0200-5.

[79] Naranjo, J.A.M., Cores, F., Casado, L.G. & Guirado, F. (2012). A fully distributed authentication model for the CoDiP2P peer-to-peer computing platform. In *Proceedings of the 12th International Conference on Mathematical Methods in Science and Engineering*, 923–934.

[80] Naranjo, J.A.M., Cores, F., Casado, L.G. & Guirado, F. (2012). Fully distributed authentication with locality exploitation for the CoDiP2P peer-to-peer computing platform. *Journal of Supercomputing*, 1–13, DOI: 10.1007/s11227-012-0842-2.

[81] Naranjo, J.A.M., Orduña, P., Gómez-Goiri, A., López-de Ipiña, D. & Casado, L.G. (2012). Lightweight user access control in energy-constrained wireless network services. In J. Bravo, D. López-de Ipiña & F. Moya, eds., *Ubiquitous Computing and Ambient Intelligence*, vol. 7656 of *Lecture Notes in Computer Science*, 33–41, Springer Berlin Heidelberg, DOI: 10.1007/978-3-642-35377-2_5.

[82] National Institute of Standards & Technology (NIST) (2001). Announcing the advanced encryption standard (AES). Federal Information Processing Standards Publication 197.

[83] Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'99, 223–238, Springer-Verlag, Berlin, Heidelberg.

[84] Pais, A. & Joshi, S. (2010). A new probabilistic rekeying method for secure multicast groups. *Int. J. Inf. Secur.*, **9**, 275–286.

[85] Parreira, J.X., Donato, D., Michel, S. & Weikum, G. (2006). Efficient and decentralized PageRank approximation in a peer-to-peer web search network. In *Proceedings of the 32nd international conference on Very large data bases (VLDB'2006)*, 415–426, VLDB Endowment.

[86] Peinado, A. & Ortiz, A. (2011). Cryptanalysis of multicast protocols with key refreshment based on the extended euclidean algorithm. In A. Herrero

& E. Corchado, eds., *Computational Intelligence in Security for Information Systems*, vol. 6694 of *Lecture Notes in Computer Science*, 177–182, Springer Berlin Heidelberg.

[87] Peinado, A. & Ortiz, A. (2012). Cryptanalysis of a key refreshment scheme for multicast protocols by means of genetic algorithm. *Logic Journal of the IGPL*.

[88] Perrig, A., Song, D. & Tygar, J.D. (2001). Elk, a new protocol for efficient large-group key distribution. In *Proceedings of IEEE S&P*, 247–262.

[89] Poovendran, R. & Baras, J. (2001). An information-theoretic approach for design and analysis of rooted-tree-based multicast key management schemes. *IEEE Transactions on Information Theory*, 2824–2834.

[90] Rafaeli, S. & Hutchison, D. (2003). A survey of key management for secure group communication. *ACM Comput. Surv.*, **35**, 309–329.

[91] Ray, I. & Geisterfer, M. (2004). Towards a privacy preserving e-commerce protocol. In K. Bauknecht, M. Bichler & B. Pröll, eds., *E-Commerce and Web Technologies*, vol. 3182 of *Lecture Notes in Computer Science*, 154–163, Springer Berlin Heidelberg.

[92] Scheikl, O., Lane, J., Boyer, R. & Eltoweissy, M. (2002). Multi-level secure multicast: the rethinking of secure locks. In *Parallel Processing Workshops, 2002. Proceedings. International Conference on*, 17–24.

[93] Shamir, A. (1979). How to share a secret. *Communications of the ACM*, **22**, 612–613.

[94] Sherman, A. & McGrew, D. (2003). Key establishment in large dynamic groups using one-way function trees. *IEEE Trans Software*, **29**, 444–458.

[95] Staddon, J., Miner, S., Franklin, M., Balfanz, D., Malkin, M. & Dean, D. (2002). Self-healing key distribution with revocation. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, 241 – 257.

[96] Stallings, W. & Brown, L. (2012). *Computer Security: Principles and Practice*. Pearson, 2nd edn.

[97] Stutzbach, D. & Rejaie, R. (2006). Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (IMC'06)*, 189–202, ACM, New York, NY, USA.

[98] Sun, Y. & Liu, K. (2004). Scalable hierarchical access control in secure group communications. In *INFOCOM 2004*, vol. 2, 1296–1306.

[99] Sun, Y. & Liu, K. (2007). Hierarchical group access control for secure multicast communications. *IEEE/ACM Trans. Netw.*, **15**, 1514–1526.

[100] Tian, B., Han, S. & Dillon, T. (2008). An efficient self-healing key distribution scheme. In *Proceedings of NTMS '08.*, 1–5.

[101] van Renesse, R., Birman, K.P. & Vogels, W. (2003). Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, **21**, 164–206.

[102] Waldvogel, M., Caronni, G., Sun, D., Weiler, N. & Plattner, B. (1999). The VersaKey framework: versatile group key management. *Selected Areas in Communications, IEEE Journal on*, **17**, 1614–1631.

[103] Wallner, D., Harder, E. & Agee, R. (1999). Key management for multicast: Issues and architectures. RFC 2627.

[104] Wang, Q. (2011). Practicality analysis of the self-healing group key distribution schemes for resource-constricted wireless sensor networks. *Communications and Mobile Computing, International Conference on*, **0**, 37–40.

[105] Wong, C., Gouda, M. & Lam, S. (2000). Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, **8**, 16–30.

[106] Xu, L. & Huang, C. (2008). Computation-efficient multicast key distribution. *IEEE Trans. Parallel Distrib. Syst.*, **19**, 577–587.

[107] Yan, J., Ma, J. & Liu, H. (2009). Key hierarchies for hierarchical access control in secure group communications. *Computer Networks*, **53**, 353–364.

[108] Yao, A.C. (1982). Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, 160–164.

[109] Yao, A.C.C. (1986). How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, 162 –167.

[110] Yoon, E.J. & Yoo, K.Y. (2011). A secure broadcasting cryptosystem and its application to grid computing. *Future Generation Computer Systems*, **27**, 620 – 626.

[111] Zhang, J., Varadharajan, V. & Mu, Y. (2006). A scalable multi-service group key management scheme. In *Proceedings of AICT-ICIW '06*, 172–177.

[112] Zhang, Q., Wang, Y. & J.P., J. (2008). A key management scheme for hierarchical access control in group communication. *International Journal of Network Security*, **7**, 323–334.

[113] Zhang, S. & Xia, J. (2009). A simple equal key agreement scheme achieving access control. In *Proceedings of CNMT'09*, 1–4.

[114] Zhou, Z. & Huang, D. (2010). An optimal key distribution scheme for secure multicast group communication. In *INFOCOM'10*, 331–335.

[115] Zhu, S. & Jajodia, S. (2010). Scalable group key management for secure multicast: A taxonomy and new directions. In S.C.H. Huang, D. MacCallum & D.Z. Du, eds., *Network Security*, 57–75, Springer US.

[116] Zhu, S., Setia, S. & Jajodia, S. (2003). Adding reliable and self-healing key distribution to the subset difference group rekeying method. In *Proceedings of NGC 2003*.

[117] Zou, X. & Dai, Y.S. (2006). A robust and stateless self-healing group key management scheme. In *Proceedings of ICCT '06*, 1–4.