Journal of Computational and Applied Mathematics 236 (2012) 3042–3051

Contents lists available at SciVerse ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

A suite of algorithms for key distribution and authentication in centralized secure multicast environments

J.A.M. Naranjo [a,*], N. Antequera [b], L.G. Casado [a], J.A. López-Ramos [b]

[a] Department of Computers Architecture and Electronics, University of Almería, Spain

[b] Department of Algebra and Mathematical Analysis, University of Almería, Spain

ARTICLE INFO

Article history:
Received 24 August 2010
Received in revised form 21 December 2010

Keywords:
Privacy
Key distribution
Authentication
Secure multicast

ABSTRACT

The Extended Euclidean algorithm provides a fast solution to the problem of finding the greatest common divisor of two numbers. In this paper, we present three applications of the algorithm to the security and privacy field. The first one allows one to privately distribute a secret to a set of recipients with only one multicast communication. It can be used for rekeying purposes in a Secure Multicast scenario. The second one is an authentication mechanism to be used in environments in which a public-key infrastructure is not available. Finally, the third application of the Extended Euclidean algorithm is a zero-knowledge proof that reduces the number of messages between the two parts involved, with the aid of a central server.

1. Introduction

Multicast communications allow a host to simultaneously send information to a set of other hosts, avoiding the establishment of point-to-point connections with all of them. IP multicast technologies (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 physical deployment, 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 to encrypt messages (depending on the topology and size of the network). 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 and distributed. Centralized schemes depend directly on a single entity to distribute every cryptographic key,

* Corresponding author.
E-mail address: jmn843@alboran.ual.es (J.A.M. Naranjo).

0377-0427/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2011.02.015

therefore being able to cope with smaller audiences than their distributed alternatives. On the other hand, key management is more complex in a distributed approach, usually involving entities that act as local subservers and manage subgroups of users, and requiring full or partial data re-encryption in some cases. Given that the scheme proposed in this paper belongs to the first kind, the following paragraphs review some well known centralized previous solutions.

RFC 2627 [1] presents some approaches to the problem. Among all, the *Hierarchical Tree Approach* (HTA) is the recommended option. It uses a logical tree arrangement of the users in order to facilitate key distribution. The benefit of this idea is that the storage requirement for each client and the number of transmissions required for key renewal are both logarithmic in the number of members.

Contemporary to HTA, the LKH (Local Key Hierarchy) scheme [2] is very similar in its tree approach. Its novelty relies on the proposal of three different rekeying strategies: *user-oriented*, which uses many short messages for a rekeying operation, *key-oriented*, which broadcasts more messages at a lower computational cost, and *group-oriented*, which employs one sole message of larger size. The LKH scheme is one of the most popular and widely used [3].

An extension to the latter, LKH++, is presented in [4] with wireless networks as target. The authors combine the fact that many auxiliary keys are shared among users in LKH with a hash function thus allowing users to derive new encryption keys by themselves with little input information from the key server.

One-way function trees (OFT) [5] are an extension to the hierarchical tree approach. A key tree is also used, but in this case every internal key is built depending on its two descendant keys: both descendants are *blinded* by means of a one-way function and the results are wired to the input of a mixing function. The tree, therefore, is built on a bottom-top fashion. Members, placed at the leaves, know their own key, the keys in the path to the root and the blinded sibling keys of the path. Thanks to that, the amount of information needed by a member to recompute the whole path of keys to the root is smaller and rekeying messages are shorter (approximately half of HTA's).

The ELK protocol [6] is an improvement of HTA. It is similar to OFT in the sense that intermediate keys are generated from its children, but pseudo-random functions (PRFs) are used rather than one-way functions. Thanks to PRFs and to timely rekeying no broadcast of information is needed in join events (only unicast messages for tree maintenance). Additionally, ELK addresses message loss tolerance by introducing the concept of *hints*: small pieces of information attached to broadcast data packets that allow one to recover lost rekey information.

A similar approach is adopted by the SKD scheme [7]: that users be able to predict new keys upon a rekeying operation with the least possible amount of information from the key server. SKD combines encryption and one-way functions.

The *Secure Lock* solution is proposed in [8]. The authors take a computational approach to the problem rather than a tree arrangement. It is based on the Chinese Remainder Theorem, its main drawback being the inefficient computations required at the key server side on each rekey operation: the computation time needed quickly becomes excessive when the number of members grows [9].

In [10], a divide-and-conquer extension to Secure Lock is proposed. It combines the Hierarchical Tree Approach and the Secure Lock: members are arranged in a HTA fashion, but Secure Lock is used to refresh keys on each tree level. Therefore, the number of computations required by Secure Lock is reduced.

An IETF Working Group, MSEC [11], is currently working in a set of protocols to standardize secure multicast. They are focusing, in an initial stage, in IP-layer centralized multicast, assuming the presence of groups and a single trusted entity in each one.

These technologies make a good job assuring privacy and (in most cases) an efficient key refreshment. However, they do not cover other aspects such as authentication or trust among peers. This paper presents a secure multicast solution for centralized scenarios that provides:

1. private communications and efficient key refreshment,
2. key server messages authentication, and
3. validation among peers.

Three different and complementary schemes are proposed in order to achieve the proposed goals. Depending on the scenario and its necessities the schemes can be implemented along with the others or on their own.

The paper is organized as follows. Section 2 describes the scenario conditions we assume for our solution. Section 3 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 4 and 5 introduce and discuss the schemes for key server messages authentication and verification among hosts, respectively. Finally, the conclusions of the paper are presented in Section 6.

2. Scenario

The target scenario is the following: private communications must be established within a restricted group. There is a central server that manages key management issues. From now on, we will refer to the server as the *Key Server*, and to the clients or users as *members*. Depending on the nature of the service, communications can be either one-to-many or many-to-many.

In any case, *forward secrecy* must be maintained. This requirement implies that a member which leaves the network (i.e. her membership expires) should not be able to decrypt any ciphered information transmitted after her exit, and forces

the encryption keys to refresh whenever a member leaves the network. Some services may require *backward secrecy*: an arriving member should not be able to decrypt any ciphered information transmitted before her arrival. This imposes, again, a refreshment of the keys when a member enters the system. We assume our scenario requires it too. These two restrictions may become an efficiency problem if the churn rate (joins and leaves) is too high. The scheme proposed here is efficient enough to cope with high churn rates, as will be shown next.

Obviously, the security and privacy features of an application level secure multicast solution should not only be restricted to private communications. Authentication is a key issue, too. Members should have a way to check that the source of a message is a trusted entity, either if the source is the Key Server or other member.

3. Distribution of secrets within closed groups

The first scheme allows the Key Server to generate and privately distribute encryption keys among restricted audiences so private communications can be established. Its most relevant features are:

- Only one message is generated per rekeying operation.
- Suitable for all topologies. No need for node hierarchies, though they can be supported.
- 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 symmetric encryption key 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 it a member ticket, x_i. Every ticket is a large prime[1] and is communicated to the corresponding member under a secure channel: SSL/TLS, for example. This communication is made once per member only, so it does not affect global efficiency. All tickets must be different from each other, at least during a relatively wide period of time. Note that x_i is known only by its owner and the Key Server, and r is shared by all members and the Key Server.

Algorithm 1 shows the generation and distribution of r.

Algorithm 1 The rekeying algorithm

1. The Key Server selects:
   i. m and 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 $i = 1 \ldots n$.
   iii. g that verifies $1 = g^p \bmod m$.
   The encryption key 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 [12] such that
$$u \cdot \delta + v \cdot L = 1 \tag{1}$$
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 $g^{\delta} \bmod m = g^k \bmod m = r$.

New values for m, g, p and/or k must be chosen for each refreshment of r. Note that δ, u and v depend on them and will change as they do.

Some remarks can be made to the algorithm. First, a proper value g at step 1.iii 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 $a^{m-1} \mod m = 1$ (that is, a is a primitive value from $\mathbb{Z}_m$). Then $g = a^q \mod m$. Second, the length of r, by definition, cannot exceed that of m, but that should not be a problem for standard symmetric cryptosystems. Third, fortunately it is not necessary, for successive refreshments, to recompute L from scratch if there were joins or leaves: L should be multiplied by the incoming members' tickets, and divided by those of the leaving members. That speeds the process up. And fourth, the Key Server might decide to refresh r after a long period of time with no members joining or leaving for security reasons. That operation is called *batch rekeying*.

Although the presented method is fast (an efficiency discussion follows), there are scenarios, like pay-per-view IPTV services, in which rekeying is performed at extremely high rates. In those cases a key hierarchy solution can be adopted, such as in [13], and our method used to refresh the highest level key.

[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 will be introduced shortly).

3.1. Proof of correctness for the disclosure scheme

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

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

and hence,

$$\gcd(\delta, L) = 1. \tag{2}$$

Eq. (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 $i = 1, \ldots, n$. The Chinese Remainder Theorem guarantees that the solution for $u^{-1} \bmod x_i = \delta$ and $\delta < x_i$, for every $i = 1, \ldots, n$ is unique.

The value $r = g^k \bmod m$ is obtained as shown next:

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

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

3.2. Scalability considerations for the rekeying algorithm

Kruus [14] suggests five issues that a multicast key management protocol must address. They are:

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

There is no difference in our scheme between first time keying (requirement 1) and further rekeying operations. Rekeying 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. (1)). In order to estimate it, assume for the rest of the paper 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.

As an example, for $b = 64$ and $n = 1000$ the maximum length of u is 64000 bits $\approx$ 8 KBs. Though that is an affordable message length for many devices (requirement 4), a shorter message would be desirable.

From the previous consideration we assume that Kruus' requirements 3 and 5 are the weakest points of our scheme. The solution that allows one 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 multicasted now; each one for a different subgroup. Fig. 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, $L_{\text{part1}} = x_A x_B x_C x_D$, $L_{\text{part2}} = x_E x_F x_G x_H$ and $L_{\text{part3}} = x_I x_J x_K x_L$.

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 rekey 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 requirements 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. In the previous example, arranging the same audience in 20 groups of 50 members would yield 20 messages of 3200 bits = 400 Bs maximum, each one shorter than a typical X.509 certificate. Shorter messages will be handled more easily and quickly by the recipients. This means less hardware requirements.

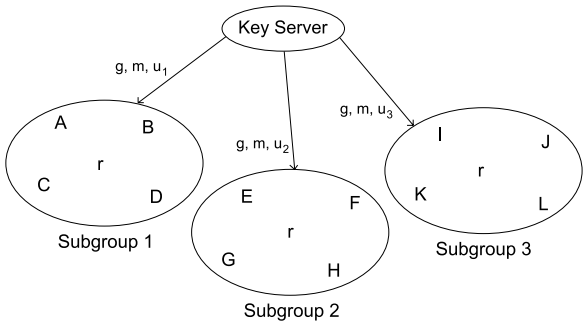

Fig. 1. The subgroup extension to the scheme. Capital letters denote members. r is the multicasted encryption key.

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.

This subgroup approach provides a better scalability, allowing to increase the maximum number of clients that can be handled. As a remark, users should be assigned to subgroups in a balanced way in order to keep refreshment messages as short as possible. This raises other issues, such as the problem of rebalancing subgroups after a leave avalanche, for example.

3.3. Security

Security in the distribution of r relies on the unfeasibility 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. (1) in Algorithm 1 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 bitlength (recall that $m - 1 = p \cdot q$).

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

Security against a passive adversary.

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 1, compute a modular inverse by using their corresponding ticket as the modulo. Clearly, the knowledge of k also implies the knowledge of both δ and p. On one hand p is kept private at the Key Server. On the other hand it is not necessary to get p in order to get the distributed k, say g^k, it is enough to get δ and then act as an authorized user. If we take into account δ is the unique solution of the congruence system $x = u^{-1} \bmod x_i$ in the interval $[0, L - 1]$, the calculus of such a δ involves knowing L, which is kept private at the server, or one valid ticket x_i, which would be equivalent to be an authorized user. Therefore, the alternative is a force-brute attack that, trying with a huge number of possible keys, 2^b, being b the key bitlength.

Security against an active adversary.

Consider the case of a legal member who either is still authorized or her ticket has already expired but keeps rekeying messages from her authorization period. In both cases, she could try to compromise other authorized users' tickets as follows: 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 (tickets). Therefore this attack involves a factorization problem that is hard to solve if the tickets used in the construction of L are big enough. Another possible attack is to use two consecutive in time values of δ in order to get some information about any other member's ticket. Consider two pairs (u, δ) and (u', δ') as before. Then the quotient $\frac{u\cdot\delta-1}{u'\cdot\delta'-1}$ would reveal the value of L. However this does not yield any information about other users' tickets since again the factorization of L is involved. Finally, a former user who still holds her old ticket might try to intercept a refreshment message and use her own old ticket expecting that it has been 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. Comparison with other schemes

Table 1 compares our scheme with other well-known centralized alternatives, which were briefly presented in Section 1. More concretely, we consider the Hierarchical Tree Approach (HTA) in its "multiple keys per message" version [1], LKH [2] and its extension LKH++ [4], OFT [5], SKD [7], ELK [6] and Secure Lock extended with HTA [10]. Our scheme is analyzed in its subgroups version (see Section 3.2).

Some of the information shown in the table was taken from [15–17], while the notation used is explained in the table itself. In order to make a fair comparison three assumptions have been made:

- both backward and forward secrecy are provided,

Table 1
Secure multicast schemes comparison. n is the number of members, s is the number of subgroups where applicable, d is the tree degree and h is the tree depth where applicable.

	HTA	LKH	LKH++	OFT	SKD	ELK	SecLock + HTA	Ours
Keys stored in Key Server	$d^h - 1$	$2n - 1$	$2n - 1$	$2n - 1$	$\frac{dn-1}{d-1}$	$2n - 1$	$d^h - 1$	n
Keys stored in member	$h + 1$	$h + 1$	$h + 1$	$h + 1$	h	$h + 1$	$h + 1$	1
Broadcast messages per Join	$(d - 1)h$	$2h - 1$	$h + 1$	$h + 1$	h	0	h	s
Broadcast messages per Leave	$(d - 1)h$	$2h$	$h + 1$	$h + 1$	$(d - 1)h$	h	h	s

Table 2
Key Server execution times for different ticket lengths and group sizes.

Group size	Tickets bitlength					
	64	128	256	512	1024	2048
500	0.0453	0.0553	0.0889	0.4692	4.0388	53.5010
1 000	0.0466	0.0542	0.0865	0.3924	4.0335	55.3955
2 000	0.0433	0.0553	0.1058	0.4353	4.3906	61.0202
4 000	0.0444	0.0644	0.1059	0.4888	4.3394	65.7045
10 000	0.0547	0.0808	0.1192	0.4671	5.3760	68.3713

- trees (where applicable) are balanced and full,
- subgroups (where applicable) are balanced and full.

Join and leave events are for a single member. Unicast communications are not shown in the table, since they do not affect global efficiency.

Results depend on several variables. The number of members is, obviously, the most important of them all. For tree-based schemes the degree is also a decisive factor since it determines the number of intermediate keys. In our scheme, the size and number of subgroups are the main variables to pay attention to. Notation is explained in the table itself.

Regarding storage, our scheme has the lowest requirements: the amount of information to be stored by the Key Server is directly proportional to the number of users. The tree based schemes must maintain a set of keys for the intermediate logical nodes of the tree (note that $2n - 1$ is equivalent to $d^h - 1$ for the case $d = 2$). Subsets of those keys must be held by the corresponding members, too. In our scheme members only need to store their own ticket.

ELK is ahead of the rest in join events. The combination of a logical tree layout and PRFs allows all members to recompute their key path to the root without any broadcast communication, though some unicast messages are occasionally required for member reallocation. Our scheme sends a broadcast message per group. The rest need one or more messages per tree level.

Leave operations require, in most cases, the same number of messages as joins. Tree-based schemes may need, again, some unicast messages for layout maintenance purposes. Our scheme needs no unicast messages.

It is worth remarking that tree-based schemes need members to be aware of their location within the logical tree: they must hold the intermediate keys that connect them to the tree root (the group key), and recompute all the way back whenever the tree is renewed. Members in our scheme only need to know which subgroup they belong to. This makes member computations simple and fast.

Finally, flash crowds should be considered too. They occur when large sets of users enter or leave the system in a short interval of time. A typical example are pay-per-view massive interest TV events (e.g. soccer matches). At the beginning of the event a great number of users enter the system, which forces one to constantly refresh the group key. At the end of the event those users leave the system, therefore a great number of leave operations must be performed. Tree-oriented schemes may suffer in flash crowd scenarios since the tree structure needs to be continuously rearranged to avoid degeneration, specially at leave events. On the contrary, our scheme can handle subgroups management very efficiently: only multiplications or divisions are needed and there is no structure to maintain. At the start of an event new subgroups can be established if needed, while at the end, the remaining users can be quickly reallocated so the number of subgroups is reduced.

3.5. Simulation

We have developed a Java implementation of the scheme in order to perform simulations and obtain execution times. The BigInteger Java class was used for handling large numbers, and the Miller-Rabin test [12] was employed to check primality. Figs. 2 and 3 show execution times for Algorithm 1, both in the Key Server and in a member, for different group sizes and ticket lengths. Tables 2 and 3 show the plotted data. They were obtained in a Intel Core 2 Duo processor at 2, 26 GHz with 3 MB of L2 cache and 2 GB of RAM.

Two main conclusions can be extracted from the Key Server times. First, key pair refreshment messages are computed very fast, excepting in the case of 2048 bits. This means that the scheme can be applied to a wide variety of scenarios. Second, execution times are mainly affected by ticket length and not by the number of members considered. That is good news when

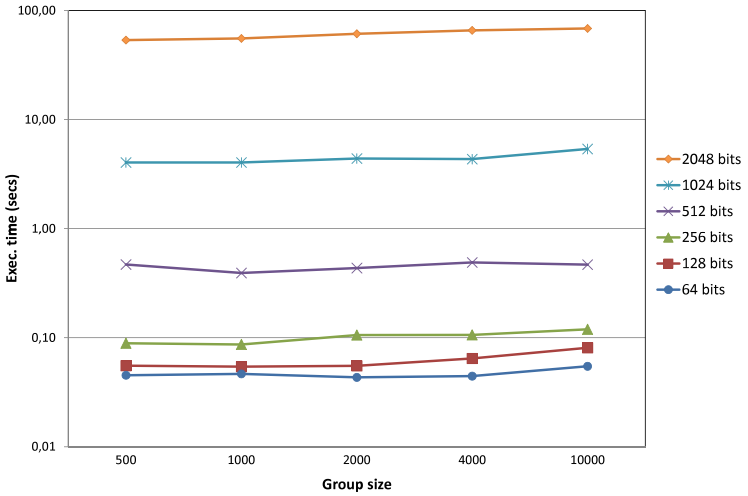

Fig. 2. Key Server execution times for different ticket lengths and network sizes.

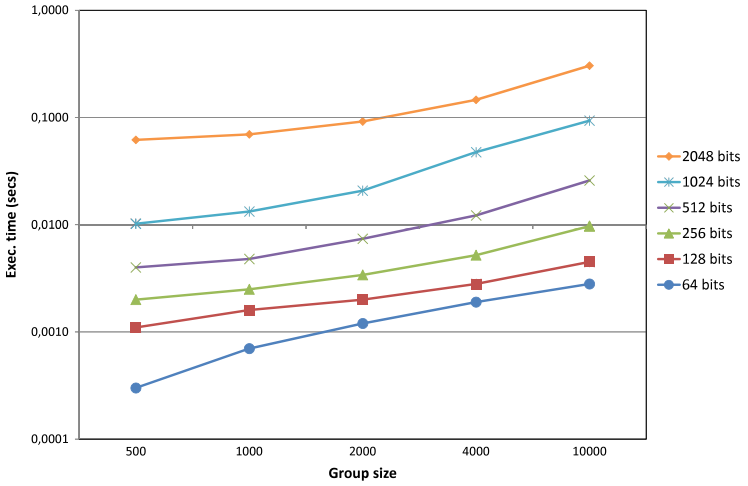

Fig. 3. Member execution times for different ticket lengths and network sizes.

Table 3
Member execution times for different ticket lengths and group sizes.

Group size	Tickets bitlength					
	64	128	256	512	1024	2048
500	0.0003	0.0011	0.0020	0.0040	0.0102	0.0619
1 000	0.0007	0.0016	0.0025	0.0048	0.0133	0.0696
2 000	0.0012	0.0020	0.0034	0.0074	0.0208	0.0918
4 000	0.0019	0.0028	0.0052	0.0122	0.0476	0.1464
10 000	0.0028	0.0045	0.0097	0.0259	0.0936	0.3042

large audiences are addressed. However, recall that the length of the refreshment message might force the audience to be split into several subgroups (see Section 3.2).

Member times show that retrieving the secret is a very fast process. In some scenarios the subgroups approach might be desirable in order to reduce memory requirements in members' hardware.

3.6. Disclosure of public keys

The scheme proposed in this paper 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 [18]. An Elgamal key pair has the form:

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

$K_{priv} : k.$

Therefore, the method can be seen as a way of *controlling the disclosure of Elgamal public keys*: the public key is only communicated to a closed group of recipients. The key pair can then be used for signature purposes (only the Key Sever knows the private key k) or for encryption of messages addressed to the Key Server.

4. 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 Key Server: that is required in order to protect the system against forged rekeying 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.

We propose an approach which 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 proves 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 can only come from the Key Server. The authentication method naturally arises from the rekeying material and no additional infrastructure is needed.

Algorithm 2 shows the rekey message authentication process. We assume the Key Server is performing a refreshment of r, and therefore the authentication process is complementary to that described in Section 3. We assume, too, that members receive the refreshment message.

Algorithm 2 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 rekeying then each partial rekeying message for group j must be authenticated separately, using the corresponding L_{part_j} value.

4.1. Security and efficiency considerations

Regarding security, the key point is that $a \cdot s \cdot r \bmod \alpha$ is only equal to a if $\alpha = L$ or $\alpha = x_i \forall x_i$. An attacker willing to forge an authenticated key refreshment message must know either L or at least one x_i. In the first case the forged message will pass the verification test in every client, while in the second case only the owner of x_i will be fooled. However, both L and every x_i are kept secret, and stealing them is equivalent to stealing a private key. We can therefore state that in terms of security, and for the scenario described in Section 2, the authentication scheme proposed here is a valid substitute for digital signature.

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 any hardware with arbitrary-precision arithmetic capabilities.

The scheme poses a disadvantage, however: the authentication message can be as long as the key refreshment message. This should be taken into account in low bit rate scenarios.

5. Peer validation: a zero-knowledge proof

Once secure multicast and Key Server messages validation have been achieved, the last proposal in this paper deals with authentication among peers. The aim is to verify whether a given peer j holds a valid ticket x_j without gaining knowledge of the 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 3 shows the process.

Algorithm 3 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 $\text{inv}_i = w_i^{-1} \bmod L$ and sends it to i.
3. Peer i sends $\{\text{inv}_i, g^{x_i} \bmod m\}$ to b.
4. Peer b calculates $w_b = \text{inv}_i^{-1} \bmod x_b$, $\beta_j = w_b \cdot (g^{x_i})^{x_b}$ 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.

If $\beta_i = \beta_b$ then it is clear that b owns a valid ticket x_b. Otherwise peer i should warn the Key Server so preventive measures can be taken against b. Modular inverses can be computed by means of the Extended Euclidean Algorithm.

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 values g and m as shown in Section 3 and communicate them to peers before any authentication is done.

5.1. Security and efficiency considerations

Security is ensured by two facts:

1. peer b needs to know a valid ticket x_b in order to obtain a w_b equal to w_i, by means of a modular inverse calculation (step 4), and
2. the complexity of the discrete logarithm problem in a finite field [19].

We warn now against the possibility of performing Denial of Service (DoS) attacks against the Key Server and peer b, if a malicious entity sends verification requests at an intentional very high rate. That same entity might arbitrarily warn the Key Server against legal peers, too.

Regarding scalability, 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.

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, $(\text{inv}_{i1}, \ldots, \text{inv}_{in})$, 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 rekeying operation due to the change of L.

Subgroups approach with trusted super-peers. With the subgroups approach, 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 bitlength is chosen.

6. Conclusions

We have presented three different uses for the Extended Euclidean Algorithm, all of them focusing on privacy and security in multicast scenarios. The first one, a rekeying mechanism, allows a single entity to manage the distribution and renewal of encryption keys within restricted groups, so private communications can be held. The communication can be done in a single multicast message, and there is no need for encryption. The mechanism is secure, and simulation results were shown to prove its efficiency, both on the Key Server and on the client side.

The second application is an authentication mechanism which is not based on public-key cryptography. It can be used in situations in which the latter is not available, and can be run along with the first scheme.

Finally, a zero-knowledge protocol was presented which can be used for validation between two clients. 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. No sensible information is disclosed.

The three mechanisms can be applied to the same scenario, say, a peer-to-peer television platform. Future lines of research include the implementation and test of a combination of them in a simulator (e.g. PeerSim [20]) or a real testbed, such as PlanetLab [21].

Acknowledgements

J.A.M. Naranjo and L.G. Casado are supported by the Spanish Ministry of Science and Innovation (TIN2008-01117). L.G. Casado is also supported by funds of Junta de Andalucía (P08-TIC-3518). J.A. López-Ramos is supported by the Spanish Ministry of Science and Innovation (TEC2009-13763-C02-02) and Junta de Andalucía (FQM 0211).

References

[1] D. Wallner, E. Harder, R. Agee, Key management for multicast: issues and architectures, in: RFC 2627, 1999.
[2] Chung Kei Wong, Mohamed Gouda, Simon S. Lam, Secure group communications using key graphs, IEEE/ACM Trans. Netw. 8 (1) (2000) 16–30.
[3] Alwyn R. Pais, Shankar Joshi, A new probabilistic rekeying method for secure multicast groups, Int. J. Inf. Secur. 9 (4) (2010) 275–286.
[4] Roberto Di Pietro, Luigi V. Mancini, Efficient and secure keys management for wireless mobile communications, in: Proceedings of the Second ACM International Workshop on Principles of Mobile Computing, 2002, pp. 66–73.
[5] Alan T. Sherman, David A. McGrew, Key establishment in large dynamic groups using one-way function trees, IEEE Trans. Softw. Eng. 29 (2003) 444–458.
[6] Adrian Perrig, Dawn Song, J.D. Tygar, Elk, a new protocol for efficient large-group key distribution, in: Proceedings of IEEE Symposium on Security and Privacy, S & P, 2001, pp. 247–262.
[7] Jen-Chiun Lin, Kuo-Hsuan Huang, Feipei Lai, Hung-Chang Lee, Secure and efficient group key management with shared key derivation, Comput. Stand. Interfaces 31 (1) (2009) 192–208.
[8] Guang-Huei Chiou, Wen-Tsuen Chen, Secure broadcasting using the secure lock, IEEE Trans. Softw. Eng. 15 (8) (1989) 929–934.
[9] Peter S. Kruus, Joseph P. Macker, Techniques and issues in multicast security, in: Proceedings of Military Communications Conference, MILCOM, 1998, pp. 1028–1032.
[10] O. Scheikl, J. Lane, R. Boyer, M. Eltoweissy, Multi-level secure multicast: the rethinking of secure locks, in: Parallel Processing Workshops, 2002, Proceedings, International Conference on, 2002, pp. 17–24.
[11] Msec working group. http://www.ietf.org/dyn/wg/charter/msec-charter.html.
[12] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.
[13] J.A.M. Naranjo, J.A. López-Ramos, L.G. Casado, Key management schemes for peer-to-peer multimedia streaming overlay networks, in: WISTP'09: Proceedings of the 3rd IFIP WG 11.2 International Workshop on Information Security Theory and Practice, in: Smart Devices, Pervasive Systems and Ubiquitous Networks, Springer-Verlag, 2009, pp. 128–142.
[14] Peter S. Kruus, A survey of multicast security issues and architectures, in: Proceedings of 21st National Information Systems Security Conference, 1998, pp. 5–8.
[15] Kin-Ching Chan, S.-H.G. Chan, Key management approaches to offer data confidentiality for secure multicast, IEEE Netw. 17 (5) (2003) 30–39.
[16] Sandro Rafaeli, David Hutchison, A survey of key management for secure group communication, ACM Comput. Surv. 35 (3) (2003) 309–329.
[17] António Pinto, Manuel Ricardo, Secure multicast in IPTV services, Comput. Netw. 54 (10) (2010) 1531–1542.
[18] Taher Elgamal, A public key cryptosystem and a signature scheme based on discrete logarithms, in: Proceedings of CRYPTO 84 on Advances in Cryptology, 1984, pp. 10–18.
[19] Whitfield Diffie, Martin E. Hellman, New directions in cryptography, IEEE Trans. Inform. Theory 22 (6) (1976) 644–654.
[20] Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, Spyros Voulgaris, The peersim simulator, http://peersim.sf.net.
[21] The PlanetLab network. http://www.planet-lab.org.