Applications of the Extended Euclidean Algorithm to Privacy and Secure Communications

Chapter · June 2010

CITATIONS
10

READS
1,130

3 authors:

Juan Álvaro Muñoz Naranjo
Universidad de Almería
22 PUBLICATIONS **79** CITATIONS
SEE PROFILE

Juan Antonio Lopez-Ramos
Universidad de Almería
61 PUBLICATIONS **759** CITATIONS
SEE PROFILE

Leocadio G. Casado
Universidad de Almería
143 PUBLICATIONS **858** CITATIONS
SEE PROFILE

Proceedings of the 10th International Conference
on Computational and Mathematical Methods
in Science and Engineering, CMMSE 2010
27–30 June 2010.

Applications of the Extended Euclidean Algorithm to Privacy and Secure Communications

J.A.M. Naranjo[1], J.A. López-Ramos[2] and L.G. Casado[1]

[1] *Department of Computer Architecture and Electronics, University of Almería*

[2] *Department of Algebra and Mathematical Analysis, University of Almería*

emails: jmn843@ual.es, jlopez@ual.es, leo@ual.es

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 is a method for controlling the disclosure of discrete logarithm-based public keys. It can be used to privately deliver a public key to a set of recipients with only one multicast communication. The second one is an authentication mechanism to be used in scenarios 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.

Key words: secure group communication, authentication, zero-knowledge proofs
MSC 2000: 11-04, 11Z05

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. In a distributed approach, key distribution is more complex, usually involving entities that act as local sub-servers and manage subgroups of users. Full or partial message re-encryption is needed in some cases. The following paragraphs introduce some well known solutions.

The *Secure Lock* centralized solution is proposed in [1]. It is based on the Chinese Remainder Theorem. A drawback are the inefficient computations required at the Key Server side on each key refreshment: the computation time needed quickly becomes excessive when the number of members grows [2].

RFC 2627 [3] 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 members 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.

In [4], 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.

IOLUS [5] is a well known framework designed for the secure multicast problem. Nodes are physically distributed in subgroups, which are organized on a tree fashion. Some special trusted nodes handle the subgroups and serve as gateways among them. It supports huge sets of members, due to its distributed nature. Since IOLUS is a framework, not a protocol, the key refreshment scheme used within subgroups is not stated. Any scheme can be used.

An IETF Working Group, MSEC [6], 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) 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 that are assumed for our solution. Section 3 presents the key refreshment scheme. Section 4 introduces the scheme for authentication of the key server. The authentication among hosts scheme is proposed in Section 5. 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 the key management issues. From now on, we will refer to the server as *Key Server*, and to the clients 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 to refresh the encryption keys 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. 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 Controlled disclosure of public keys within closed groups

The first scheme we present allows to establish private group communications. The proposed approach: the Key Server owns an asymmetric key pair (of an encryption algorithm based on the discrete logarithm problem) and discloses the public key only to the members of the restricted group. Communications from the Key Server are encrypted with its private key. It is clear that only the members of the group will be able to decrypt the messages. We have named this solution *controlled disclosure of a public key*.

The usual method to publish public keys is the use of public key certificates. This is extremely useful when the disclosure process involves two participants: the owner of

the public key and the recipient. For more participants the process can be repeated, obviously. What the present scheme tries to solve is how to simultaneously disclose a public key to a selected audience only, while preserving security and efficiency. What's more, the process should be lightweight enough to be repeated as many times in time as required for key refreshment purposes. This problem appears in services such as pay-per-view IPTV or P2PTV and multiconferences.

The most relevant features of the scheme are:

- Only one message is generated per key refreshment.
- 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.

The scheme is described next. Let us assume there are n members at a given time in the group, and that the Key Server generates an asymmetric key pair of the form:

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

which is an Elgamal key [7]. K_{pub} is the key to be disclosed.

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 K_{pub} is shared by all members and the Key Server.

Generation of (K_{pub}, K_{priv}) and distribution of K_{pub} are done as follows.

1. The Key Server selects:
   - m and p, large prime numbers, such that $m - 1 = p \cdot q$.
   - k and δ, such that $\delta = k + p$ and $\delta < x_i$, for every $i = 1 \dots n$.
   - g that verifies $g^p = 1 \ mod \ m$ (such a value is easy to calculate[2]).
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 [8], such that
$$u \cdot \delta + v \cdot L = 1 \tag{1}$$

[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 (L will be introduced shortly).

[2]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 $g = a^q \ mod \ m$.

4. The Key Server multicasts (makes public) $g,\ m$ and u on plain text.
5. Each member i calculates $u^{-1}\ mod\ x_i = \delta$ and $g^{\delta}\ mod\ m = g^{k}\ mod\ m = K_{pub}$. The length of K_{pub}, by definition, can not exceed that of m.

New values for $m,\ g,\ p$ and/or k must be chosen for each refreshment of (K_{pub}, K_{priv}). Note that $\delta,\ u$ and v depend on them and will change as they do.

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. Finally, for security reasons, the Key Server might decide to refresh (K_{pub}, K_{priv}) after a long period of time with no members joining or leaving.

Since the use of asymmetric key encryption along with large pieces of data may be inefficient the *key hierarchy* solution can be adopted. A symmetric key is encrypted by the Key Server with its private key, and delivered to the set of members. Data messages are encrypted using the symmetric key. Members can decrypt data messages after receiving the Key Server's public key and the symmetric key. For security and efficiency reasons, the symmetric key may be refreshed at a higher frequency than the asymmetric key pair. Examples of this technique are shown in [9]. An additional benefit of using a key hierarchy is the possibility to establish both one-to-many and many-to-many communications: both the Key Server and every member know the symmetric key and may use it to encrypt its own messages.

3.1 Proof of correctness for the disclosure scheme

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

$$gcd(\delta, x_i) = 1,\ \text{for every } i = 1, \dots, n \tag{2}$$

and hence,

$$gcd(\delta, L) = 1 \tag{3}$$

Equation (3) 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 $\delta \cdot u \equiv_{x_i} 1$ and so $u^{-1} \equiv_{x_i} \delta$, for every $i = 1, \dots, n$. The Chinese Remainder Theorem guarantees that the solution for $u^{-1}\ mod\ x_i = \delta$ and $\delta < x_i$, for every $i = 1, \dots, n$ is unique.

The value $K_{pub} = g^{k}\ mod\ m$ is obtained as shown next:

$$\begin{aligned} g^{\delta} &\equiv_m g^{k+p} \\ &\equiv_m g^{k} \cdot 1 \\ &\equiv_m g^{k} \end{aligned} \tag{4}$$

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

3.2 Security and scalability considerations

Security in the distribution of K_{pub} 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) 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.

Note that the product L is not public in order to make attackers' work more difficult. In case L was discovered and factorized by an attacker, she would gain access to every member ticket. But such a factorization is impractical by means of a brute force attack. A legal member, say i, might be tempted to factorize $\frac{u \cdot \delta - 1}{x_i}$. If she was successful, she would obtain, again, every other ticket included in L. The problem of factorizing such a value, however, is equivalent to that of factorizing L.

Nevertheless, there is a security measure that must be taken when a new member joins: she should not be assigned a previously used ticket (at least recently). This is done to prevent the old owner to keep intercepting refreshment messages and using her old ticket to discover the secret.

Regarding scalability, we can observe that L will be large, given that $L = \prod_{i=1}^{n} x_i$. So will be u (recall Eq. (1)). For n members and b-bits tickets 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 $n = 1000$, u will be 64000 bits long at most, i.e. $\approx$ 8 KBs. Though that is an affordable message length for many devices (requirement 4), a shorter message would be desirable.

The solution that allows to overcome these problems consists of dividing the set of members into subgroups and delivering the same K_{pub} to all of them. Assume there are s disjoint subgroups, each one with a similar number of members. Still, the join and leave operations 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. The final bandwidth requirement does not change, but adopting this approach brings many benefits which are discussed next.

First, 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.

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 nowadays multi-core computers. The whole process can be sped up by nearly 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 Communication with different groups

There are scenarios that require separate communications with different groups of members. Examples are different pay-per-view channels in the same TV platform and different private multiconferences managed by the same Key Server. Handling this situation is easy: the Key Server only needs to maintain a different key pair for each group. Every join or leave event will imply the refreshment of the affected group's key pair only. Figure 1 depicts this situation. It can happen that a member is enlisted in two or more

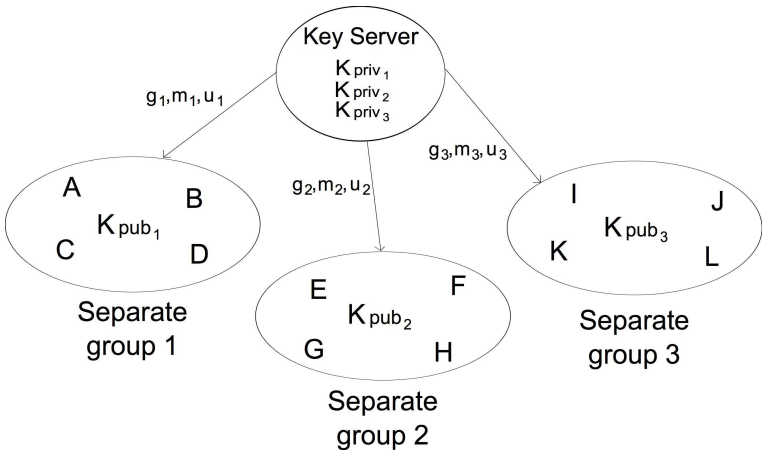

Figure 1: Managing different groups. Capital letters denote members.

groups at the same time (that is usually the case of pay-per-view channel packages, for example). It is clear that a join or leave of that member would require a refreshment in every group she belongs to.

3.4 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 was employed for primality tests. Figures 2 and 3 show execution times for the algorithm in Section 3, both in the Key Server and in a member, for different group sizes and ticket lengths. 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

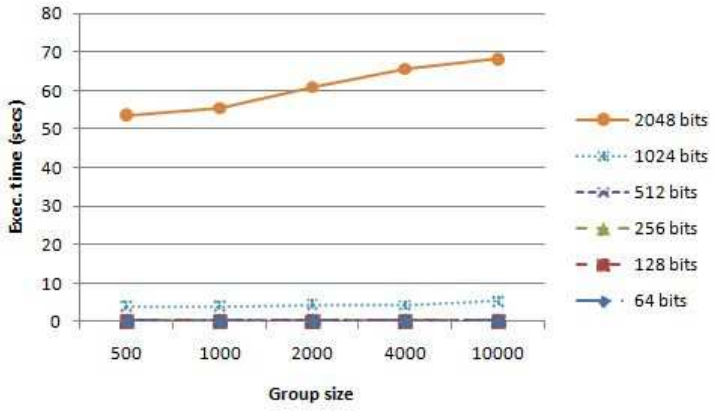

Figure 2: Key Server execution times for different ticket lengths and network sizes.

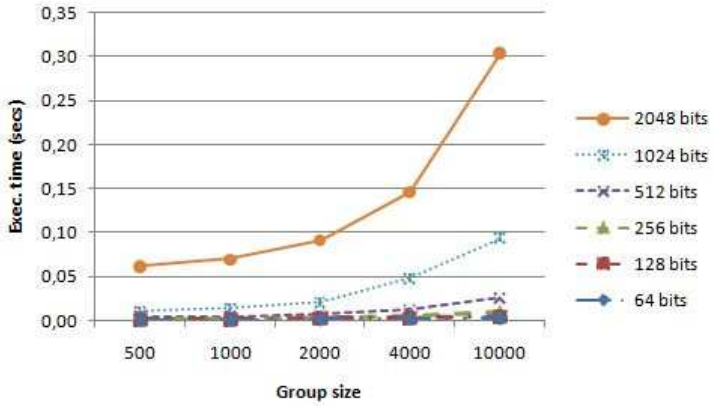

Figure 3: Member execution times for different ticket lengths and network sizes.

times are mainly affected by ticket length and not by the number of members considered. That is good news when large audiences are addressed. However, remember 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. The main problem at the server side is, again, handling a long message.

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 refreshment 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.

That solution is not applicable in our case, because refreshment messages disclose a public key: members would need to use the public key they are receiving to verify the message that contains it. A digital signature of that nature would not assure authentication at all, since the message might be forged and the signature still be valid. An alternative is to use the key pair used before the refreshment, but just-arrived members would not be able to verify the signature, since they would not know the previous key pair. Obviously, the simplest solution involves having a different, invariable key pair for authentication purposes.

We propose, instead, an approach which is not based in the use of public key cryptography. Our solution proves that the sender knows 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 passes the verification scheme can only come from the Key Server.

The scheme is described next. We assume the Key Server is performing a refreshment of its key pair, and therefore the authentication process is complementary to that described in Section 3. We assume, too, that members receive the refreshment message.

1. The Key server:
   (a) computes $s = (g^k)^{-1}\ mod\ L$ by means of the Extended Euclidean Algorithm,
   (b) chooses a random number a, such that $a < x_i$, for every x_i, and
   (c) multicasts $\{a \cdot s, h(a)\}$. $h(a)$ is the output of a hash operation on a. The hash algorithm is not specified here.
2. Every member i receives the authentication message and computes $h(a \cdot s \cdot K_{pub}\ mod\ x_i)$, 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.

4.1 Security and efficiency considerations

Regarding security, the key point is that $a \cdot s \cdot r\ mod\ \alpha$ 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 in the Key Server side are a modular inverse and a multiplication. 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 that a given peer j holds a valid ticket x_j: this means that j is a legal peer, assuming no information leakage. Verification is carried out with no disclosure of any private nor sensible information. The scheme is presented next. Assume that peer i wants to verify whether peer j is a legal peer, prior to establishing communications with it. Assume too that the public key disclosure algorithm from Section 3 has been run previously. Recall the form of $(K_{pub},\ K_{priv})$:

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

Authentication is performed as follows:

1. Peer i chooses a random integer r such that $1 < r < m$ and sends it to the Key Server.
2. The Key Server computes $inv = r^{-1} \ mod\ L$ and sends it to i.
3. Peer i sends $\{inv,\ g^{x_i} \ mod\ m\}$ to j.
4. Peer j calculates $r_j = inv^{-1} \ mod\ x_j$, $\beta_j = r_j \cdot (g^{x_i})^{x_j}$ and sends $\{\beta_j,\ g^{x_j}\}$ to i.
5. Peer i computes $\beta_i = r \cdot (g^{x_j})^{x_i}$, which should be equal to β_j.

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

In case this protocol is implemented in a standalone way 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 assured by two facts:

1. peer j needs to know a valid ticket x_j in order to obtain a r_j equal to r, by means of a modular inverse calculation (step 4), and
2. the complexity of the discrete logarithm problem in a finite field [10].

We warn now against the possibility of performing Denial of Service (DoS) attacks against the Key Server and peer j, 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 efficiency and scalability, the protocol involves one communication with the Key Server and modular exponentiations. This makes the protocol applicable only to small centralized networks or distributed networks in which subgroups are managed by local entities, such as IOLUS [5]. However, the Key Server plays only a small role (modular inverses can be found very efficiently) and the main part of the work is carried out by peers. Having said this, member authentication by means of digital certificates is a more realistic approach for large groups. However, our scheme may be an alternative for small sets of members.

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 controlled disclosure mechanism, is suitable for scenarios in which a single entity (a Key Server) communicates its public key to a set of client hosts, 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 public-key cryptography is not available (due to low capacity devices on the client side, for example). It can also be used along with the first scheme, though.

Finally, a zero-knowledge protocol was presented which can be used for for peer validation. By using this protocol peers can decide whether to trust a peer or not before establishing communications with it. It works by challenging peers 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 [11]) or a real testbed, such as PlanetLab [12].

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] Guang-huei Chiou and Wen-Tsuen Chen, *Secure broadcasting using the secure lock*, IEEE Trans. Softw. Eng. 15(8): 929–934, 1989.

[2] Peter S. Kruus and Joseph P. Macker, *Techniques and issues in multicast security*, In Proceedings of Military Communications Conference, MILCOM, 1998, pages 1028–1032.

[3] D. Wallner, E. Harder and R. Agee, *Key management for multicast: Issues and architectures*, RFC 2627, 1999.

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

[5] Suvo Mittra, *Iolus: A framework for scalable secure multicasting*, In Proceedings of ACM SIGCOMM'97, pages 277–288, 1997.

[6] *Msec working group*, http://www.ietf.org/dyn/wg/charter/msec-charter.html.

[7] Elgamal, Taher, *A public key cryptosystem and a signature scheme based on discrete logarithms*, Proceedings of CRYPTO 84 on Advances in cryptology, pages 10–18, 1984.

[8] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, *Handbook of Applied Cryptograhpy*, CRC Press, 1996.

[9] J. A. M. Naranjo, J. A. López-Ramos and 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. Smart devices, Pervasive Systems and Ubiquitous Networks, pages 128–142, 2009, Springer-Verlag.

[10] Whitfield Diffie, Martin E. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory, 22(6), pages 644-654, 1976.

[11] Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, and Spyros Voulgaris, *The Peersim simulator*, `http://peersim.sf.net`.

[12] *The PlanetLab network*, `http://www.planet-lab.org`.