

MANAGING KEY MULTICASTING THROUGH ORTHOGONAL SYSTEMS

JOSÉ ANTONIO ALVAREZ-BERMEJO, JUAN ANTONIO LOPEZ-RAMOS, JOACHIM ROSENTHAL,
AND DAVIDE SCHIPANI

ABSTRACT. In this paper we propose a new protocol for Group Key Management. The protocol is based on the use of orthogonal systems in vector spaces. The main advantage in comparison to other existing multicast key management protocols consists in small communications overheads, i.e. length and number of messages to be sent, as well as key storage, especially with respect to a comparable security level. This makes the protocol especially attractive when the number of legitimate receivers is large.

Keywords: Group Key Management, Multicast Security, Orthogonal Systems.

1. INTRODUCTION

Traditional security measures are mainly applicable to a unicast environment, i.e. communications take place between two single parties. For instance, data confidentiality, one of the most important features in network security, can be offered in this environment by means of a pair of keys. However there exist many different situations where the usual secure unicast protocols cannot be used, mainly due to the nature of the information to be transmitted. This usually happens when trying to deliver data from a sender to multiple receivers, especially when a huge amount of data needs to be delivered very quickly. One of the most efficient ways to do this is the so-called multicast. In a multicast protocol a certain group of people receives the information and this group is usually highly dynamic. In a typical situation users join and leave the group constantly ([14]).

There are a number of exciting multimedia applications that make good use of multicast capability, such as stock quote services, video-conferencing, pay-per-view TV, Internet radio, and so on. Many of the aforementioned multicast applications require security in data transmission, i.e., data can only be accessed or exchanged among an exclusive group of users. In the Pay-TV system, for example, the service providers usually employ Conditional Access System (CAS) to avoid unauthorized accessing of their video/audio streams, and only allow access to services based on payment.

The natural approach to establish secure multicast communications is to agree on one or several symmetric encryption keys in order to encrypt messages. However, the key, or keys, must be renewed periodically to prevent outer or inner attacks. Depending on how key distribution and management are carried out, secure multicast schemes are divided into centralized, decentralized and distributed schemes. Centralized schemes depend directly on a single entity to distribute every cryptographic key. A typical scenario is an IPTV or P2PTV platform, in which clients receive

Date: April 16, 2015.

The Research was supported in part by the Swiss National Science Foundation under grant No. 149716. First author is partially supported by Spanish Ministry of Science and Innovation (TIN2008-01117), and Junta de Andalucía (P08-TIC-3518). Second author is partially supported by Ministerio de Educacion, Cultura y Deporte grant "Salvador de Madariga" PRX14/00121, Ministerio de Economia y Competitividad grant MTM2014-54439 and Junta de Andalucía (FQM0211).

a TV signal from a Content Server via Internet. In a decentralized scheme users are distributed into groups with a group manager that distributes the session key group, thereby allowing an increase of the audience. Finally distributed schemes involve collaboration of users in building the secret, which can make key management rather complex ([14]). A big issue concerns security: in a centralized system there is just one server to secure, while in the decentralized and distributed cases security efforts have to be multiplied. In the papers [14] and [25] a survey can be found about rekeying in wired networks. Wireless networks show additional challenges due to member mobility and increase in the number of members. In [9] a survey is provided about group key management network dependent and independent protocols in wireless mobile environments that employ multicast communication.

Our aim in this paper is to introduce a novel protocol applicable for both centralized and decentralized multicast schemes that is shown to be secure, efficient and scalable. We focus our efforts on the algorithm to encrypt the session key more than on the way that this is distributed. To show its competitiveness we have considered typical arrangement of users in a centralized scheme considering both a single group and a tree arrangement of users.

In the following lines we recall some centralized schemes for key management, although the reader can find a recent survey on secure multicast in [25]. A very well-known protocol is *Hierarchical Tree Approach* (HTA) [20]. 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. Other key tree approaches and extensions are LKH [22], LKH++ [4], OFT [16] or ELK [13].

In [3] the so-called *Secure Lock* protocol is introduced. The authors approach the problem in a computational manner and make use of the Chinese Remainder Theorem instead of a tree arrangement. Users are distributed into groups, that in the case of PayTV could be represented by those subscribers with the same Pay-Per-Channels (PPC) or Pay-Per-View (PPV) options. The PPC and PPV programs should be encrypted previously to their distribution and there is only a content server and a key server (that might be different or not). Its main drawback is the large computational cost required at the key server side on each rekeying operation: the length of the rekeying messages and the computing time needed becomes quickly problematic as the number of members in one of the PPC or PPV groups grows [12].

In [15], a divide-and-conquer extension of the Secure Lock is proposed. It combines the Hierarchical Tree Approach and the Secure Lock: members are arranged in a HTA fashion, but the Secure Lock is used to refresh keys on each tree level. Therefore, the number of computations required by the Secure Lock is reduced.

Another computational approach with the same distribution by groups of users and a unique key server is introduced in [8] with the particular application on Pay-TV but extendable to any other secure multicast application. The idea is to use polynomials over a finite field interpolating hashes of secret values belonging to the authorized users. The main drawbacks are that the hash function must be renewed with any rekeying operation, due to security concerns, and the large size of the polynomials involved. The length of the messages grows linearly with the number of users in every group, so that if this number is huge, users might be forced to be distributed into subgroups, e.g., groups of users are established inside every PPC or PPV group.

The distribution by groups is in fact often beneficial and is used by most key managing protocols. A first benefit is the parallelization of the process which speeds up the rekeying operations. Secondly a compromised key in one of the groups does not affect the others. Last but not least, in most applications of secure multicast the group distribution is connected with the scalability

of the system, i.e., the efficiency of the communication protocols concerning the rekeying process, with particular reference to leave and join operations. Groups are usually highly dynamic and the joining or the leaving of users implies a rekeying operation, but key refreshment due to this fact in one group does not affect the others.

More recently in [11] the authors introduce another solution with the same philosophy of Secure Lock and of that introduced in [8] and based on the Extended Euclidean Algorithm. Throughout this paper we will refer to this protocol as *Euclid*. The server distributes a secret via the inverse of an integer modulo a product of coprime secret numbers, each one of them belonging to an authorized user. The authors show [11] that a former user could try a factorization attack, which forces to consider prime numbers of an adequate size. This implies a division by groups of the audience, in the case of PayTV a subdivision of every PPC or PPV group, since the length of the rekeying messages could become unaffordable. Some recent contributions developing further on these ideas appear to require more affordable computational costs [17, 18].

In this paper we are assuming a scenario, fairly general and suitable for many applications, especially for multimedia distribution purposes, with a Key Server and a set of members (other hosts) that either send or receive multicast messages. Any multicast topology can be used underneath, including the scheme considered in [23], where a topology-matching key management tree scheme is introduced. All setup tasks are carried out by the Key Server. Data communications are then either one-to-many or many-to-many, and consist of encrypted contents and/or rekeying messages, which are created by the Server (or the two servers, Content Server and Key Server). Members can enter and leave the system at any time. The key must be refreshed upon member arrival or departure to achieve perfect backward and forward secrecy, respectively. However this might depend on the application, since there exist cases, such as some audio and/or video streaming distributions, where backward secrecy is not an issue, as contents can be out of date.

The protocol we are introducing presents some nice features that make it competitive, e.g. it requires just a single message per group, of affordable length, for every rekeying operation, the operations at the Key Server and the Client sides require low computational cost and the key storage requirements are minimal.

The main idea behind the protocol is the use of orthogonal systems in vector spaces. Exploiting orthogonality comes probably as a natural tool in multicast applications, as this appears also e.g. in CDMA and [21]. How orthogonality is exploited here appears though to be new, and brings with it several advantages. In particular the scalars will play an important role in order to have fast rekey and reset operations and avoid involving large vectors to be replaced or generated. Moreover this structure will make the protocol not only agile and flexible, but also more robust and secure against all conceivable attacks, as will be shown later, also in comparison with many existing protocols of similar communication overheads.

In the next Section we describe the new protocol, analyze its security, and compare it with other existing and aforementioned protocols. Section 3 demonstrates an efficient implementation of the protocol.

2. THE PROPOSED PROTOCOL

Let the potential users be denoted with the integers $1, \dots, n$ and assume that they all belong to the same group.

(1) Initialization step:

Let \mathbb{K} be a field and V be a \mathbb{K} vector space of dimension $m \geq n$ (see also next subsection for the choice of m). Let \langle, \rangle be a bilinear form which we assume to be nondegenerate and

symmetric. Let $B = \{\mathbf{e}_1, \dots, \mathbf{e}_n\}$ be a set of n mutually orthogonal vectors in V having the property that $\langle \mathbf{e}_i, \mathbf{e}_i \rangle \neq 0$ for $i = 1, \dots, n$. Note that these two requirements, namely the mutual orthogonality and the non self-orthogonality, also imply the linear independence of the vectors. For security reasons, as we will show later, the vectors should not be part of the canonical basis or anyway the basis used to represent vectors. We select a family $\{x_i\}_{i=1}^n$ of random nonzero scalars in \mathbb{K} . Note that $B' = \{x_1\mathbf{e}_1, \dots, x_n\mathbf{e}_n\}$ spans the same subspace as B . These two sets are kept secret by the server and each user i is assigned the vector $\mathbf{v}_i = x_i\mathbf{e}_i$. By our assumptions we know that $\langle \mathbf{v}_i, \mathbf{v}_i \rangle \neq 0$ for $i = 1, \dots, n$. Then we will consider two subsets in B' at a determined point in time in the communications. On one hand B'_1 will be formed by those vectors in B' that are assigned to some user and B'_2 will be the set of vectors in B' that do not correspond to any user. We also consider two subsets in B'_2 , $B'_{2,1}$ and $B'_{2,2}$, that contain those vectors that were not previously used and those that are a multiple of a vector corresponding to a former no more legitimate user, respectively.

(2) Sending the information:

Suppose that we want to distribute the secret $s \in \mathbb{K}$. Then we compute and multicast (broadcast) $\mathbf{c} = s(\sum_{\mathbf{v}_j \in B'_1} \mathbf{v}_j + y \sum_{\mathbf{v}'_j \in B'_2} \mathbf{v}'_j)$ for some random $y \in \mathbb{K}$ different for each new secret.

(3) Recovering the information:

Each user computes $g = \langle \mathbf{c}, \mathbf{v}_i \rangle = s \langle \mathbf{v}_i, \mathbf{v}_i \rangle$.

The secret s is then recovered by computing $s = g \langle \mathbf{v}_i, \mathbf{v}_i \rangle^{-1}$.

(4) Key refreshment:

(a) Join:

If user j joins, then she is assigned one of the vectors in $B'_{2,1}$, say $x_j\mathbf{e}_j$. The server selects a new secret $s' \in \mathbb{K}$ and multicasts as above $\mathbf{c}' = s'(\sum_{\mathbf{v}_j \in B'_1} \mathbf{v}_j + y' \sum_{\mathbf{v}'_j \in B'_2} \mathbf{v}'_j)$ for some random $y' \in \mathbb{K}$.

(b) Leave:

If user j leaves, then her vector $\mathbf{v}_j = x_j\mathbf{e}_j$ is deleted from B'_1 and the vector $\mathbf{v}'_j = x'_j\mathbf{e}_j$ is included into $B'_{2,2}$ where x'_j ($\neq x_j$) is selected at random in \mathbb{K} . To distribute a new secret s' , we do similarly as after a join operation.

We remark that, if we are managing with ℓ groups, then we can use ℓ different orthogonal bases. A particular interesting case of managing groups is shown in Figure 1. In this case one can profit from a tree-like distribution as in the HTA and the Secure Lock + HTA approaches ([20] and [15] respectively) using a divide and conquer strategy. The main advantage is that we can use smaller vector spaces, so that we can serve a much bigger number of users without delaying in rekeying operations. Let us consider for example a hierarchical tree with a depth of 3, i.e., the number of levels below root is 3, and a degree of n , i.e., the number of children below each parent node is n (see Figure 1). In this situation, we have n^3 users who are located at the leaves of the tree. Intermediate nodes store group keys in the form of vectors known to the correspondent descendants. For example, users 1 to n^2 share a common vector stored at P_1 , users 1 to n share another common vector stored at $P_{1,1}$ and so on, so that user 1 knows her private vector plus the vectors P_1 and $P_{1,1}$. Then, when a rekeying message is to be sent, this will be made using the orthogonal system given by $\{P_1, \dots, P_n\}$ as described above, and any authorized user will be able to retrieve the key.

Now, let us assume without loss of generality that user 1 wants to leave. It can be easily observed that we need the following messages to refresh the key and preserve forward secrecy.

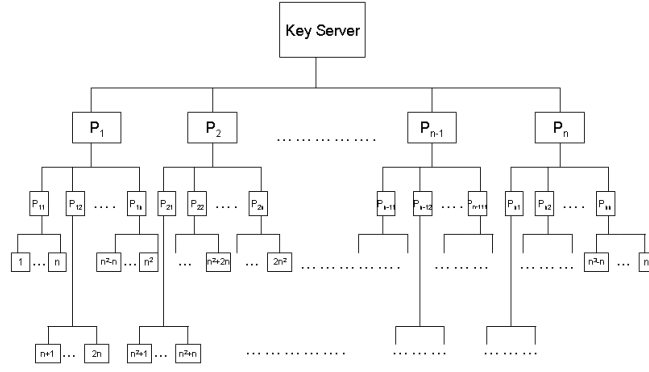


FIGURE 1. Hierarchical Tree

After determining a new vector for position 1, the server uses the new basis, including the private vectors of users 2 to n , to send them a scalar (by posting an encrypted version, as described in the protocol). Users 2 to n use this scalar to renew $P_{1,1}$, i.e. they keep the same vector but substitute the scalar associated to it. Analogously the server uses the new basis for $P_{1,1}$ to $P_{1,n}$ to send the users 1 to n^2 another scalar. These use this scalar to renew P_1 . At last the server can send a new key for all users using the new orthogonal system for $\{P_1, \dots, P_n\}$.

Notice that in order to send a scalar of k -bits length we need a message of nk bits length. If we deal with a tree distribution as above where $n = 100$, then three rather short messages will give us the possibility of handling audiences of up to 1 million users, and the computing time to generate the rekeying messages will not depend on the number of users.

In fact, building the tree for our protocol comes up naturally. When designing the tree distribution we have to fix the number of descendants of each node, that will give us the required dimension for our vector space. And the tree distribution allows to have flexibility on the number of users: if a bigger structure has to be considered in order to deal with more users, then intermediate nodes can be easily inserted.

2.1. Security. Let us suppose that \mathbb{K} is a finite field, which is the usual setting for application. As a first step we will show that, by choosing m appropriately, we can be sure that there are sufficiently many n -tuples of mutually orthogonal vectors in V , so that a brute force attack to find B is not feasible.

As we require also the property $\langle \mathbf{e}_i, \mathbf{e}_i \rangle \neq 0$, for $i = 1, \dots, n$, we consider the set $\{\mathbf{x} = (x_1, \dots, x_m) \in V \mid \langle \mathbf{x}, \mathbf{x} \rangle = x_1^2 + \dots + x_m^2 = 0\}$. This set forms a hypersurface H of dimension $m - 1$ and degree 2.

Theorem 1. Let $V := \mathbb{F}_q^m$ and $m > 2n$. Then there exists at least

$$(1 + o(1)) \frac{q^{\frac{3}{2}n^2}}{n!}$$

n -tuples of mutually orthogonal vectors $(\mathbf{e}_1, \dots, \mathbf{e}_n)$ in V having the property that $\langle \mathbf{e}_i, \mathbf{e}_i \rangle \neq 0$, $i = 1, \dots, n$. For characteristic bigger than 2 we may require $(1 + \frac{5 \cdot 2^{13/3}}{q})q^{m-1} \ll q^m$.

Proof. We divide the proof in different scenarios, as the estimate can be made more precise depending on the particular setting involved.

In characteristic 2 (which is probably the most interesting case for cryptographic applications), $x_1^2 + \dots + x_m^2 = (x_1 + \dots + x_m)^2$ so that the cardinality of the hypersurface $|H| = q^{m-1}$. Now Iosevich and Senger [6, 19] derived a lower bound on the number of n -tuples of mutually orthogonal vectors in a subset $X \subset V$ in situations where a lower bound on the cardinality of X is known. Applying this result to our situation with $X := V \setminus H$ one derives the thesis.

In characteristic greater than 2, we can exploit estimates on the number of points in hypersurfaces, namely the Lange-Weil bound and connected results (e.g. [2] or [7] where probably the best general bounds can be found). If H is absolutely irreducible, then $|H| = (1 + C)q^{m-1}$, where $|C|$ can be estimated, independently of any regularity conditions, as less than $\frac{5 \cdot 2^{13/3}}{q}$ ([2, Theorem 5.2]); here we may require q to be large enough to make $|H|$ neglectable with respect to q^m . If H is not absolutely irreducible, then $|H| \leq q^{m-1}$ ([2, Lemma 2.3]). In any case we can apply again the same argument as in characteristic 2 and derive the thesis. \square

The condition $m > 2n$ might also be convenient in order to avoid any collusion attack, that is to avoid that a big group of, say, k users share their private vectors with each other, trying to retrieve information belonging to other authorized users. Since $m - k > 2(n - k)$, the inequality above guarantees that there would be anyway more than

$$(1 + o(1)) \frac{q^{\frac{3}{2}(n-k)^2}}{(n-k)!}$$

$(n - k)$ -tuples of mutually orthogonal vectors in the remaining unknown vector space.

From the above one can infer, as noted, not only a resilience to collusions, but also that the disclosure of a private key does not compromise other users' keys.

Let us assume now that the set B is known, instead of being kept secret. Since B is a linearly independent set, one can compute readily the unique coefficients z_1, \dots, z_n such that

$$\mathbf{c} = z_1 \mathbf{e}_1 + \dots + z_n \mathbf{e}_n.$$

An authorized user knowing the vector $\mathbf{v}_j = x_j \mathbf{e}_j$ and having computed $z_j \mathbf{e}_j$ readily computes s and x_j . Then since all the private numbers x_i verify the equality $z_i = s x_i$, $i = 1, \dots, n$, all of them can be readily computed by this user. Such a user would have the chance to use them later in her own interest. Thus it is clear that B should be kept secret.

Here follows a tentative attack based on the collection of many subsequent pieces of information, in a ciphertext-only attack scenario. We show below that this is feasible when B , against our hypothesis, is the canonical basis used to represent vectors of the vector space V . Namely, anybody observing the information flow could get n linearly independent key refreshments $\mathbf{c}_1, \dots, \mathbf{c}_n$. Note that this is the case whenever a user i leaves and in that case, the set $B' = \{x_1 \mathbf{e}_1, \dots, x_i \mathbf{e}_i, \dots, x_n \mathbf{e}_n\}$ would change to $B'' = \{x_1 \mathbf{e}_1, \dots, y x'_i \mathbf{e}_i, \dots, x_n \mathbf{e}_n\}$. Now, suppose without loss of generality that $n = m$; if the server sends $(s_i x_1, \dots, s_i x_n)$ as a rekeying message $\mathbf{c}_i = (c_{i,1}, \dots, c_{i,n})$, then we would consider the matrix

$$M = \begin{pmatrix} c_{1,1} & \cdots & c_{n,1} \\ \vdots & \vdots & \vdots \\ c_{1,n} & \cdots & c_{n,n} \end{pmatrix}$$

where each column $(c_{i,1}, \dots, c_{i,n})$ represents the coordinates of the refreshment \mathbf{c}_i with respect to B (as $c_{i,j} = s_i x_j$ for $i, j = 1, \dots, n$); then M represents the change of basis from the basis $C = \{\mathbf{c}_1, \dots, \mathbf{c}_n\}$ to B . The inverse of M will reveal then B in terms of the basis C . And knowing

a pair (s, \mathbf{c}) would compromise all the secrets (of the other users) used to get this pair (s, \mathbf{c}) , as noted at the beginning of this subsection.

Therefore it is convenient, as pointed out before, that B is chosen not to be the canonical basis, so that what is sent by the server is not plainly (s_1x_1, \dots, s_nx_n) , but its representation in another basis.

Let us illustrate this with the following easy example:

Example 2. Let $B = \{(1, 1, 1), (1, -2, 1), (-1, 0, 1)\}$ be an orthogonal basis of the Euclidean vector space \mathbb{R}^3 (with the usual scalar product \langle, \rangle) and assume $x_1 = 2$, $x_2 = 3$, $x_3 = 5$. Then $B' = \{(2, 2, 2), (3, -6, 3), (-5, 0, 5)\}$. If we want to rekey with $s = 4$, then we have to multicast

$$\mathbf{c}_1 = 4(2, 2, 2) + 4(3, -6, 3) + 4(-5, 0, 5) = (0, -16, 40).$$

User 1 can recover s by calculating

$$h = \langle (0, -16, 40), (2, 2, 2) \rangle = 48,$$

and then

$$s = h \langle (2, 2, 2), (2, 2, 2) \rangle^{-1} = 4.$$

Users 2 and 3 act similarly.

Now suppose that user 2 leaves and x_2 is changed to $x_2 = 2$. $B'_{2,2}$, that was previously empty, contains now the vector $(2, -4, 2)$. Then the rekeying message for $s = 3$, and choosing $y = 2$ is $\mathbf{c}_2 = 3(2, 2, 2) + 3 \cdot 2(2, -4, 2) + 3(-5, 0, 5) = (3, -18, 33)$. Finally, suppose that user 1 leaves, x_1 becomes 3 and the new secret s is 2, so that, choosing $y = -1$, $\mathbf{c}_3 = 2 \cdot (-1)(3, 3, 3) + 2 \cdot (-1)(2, -4, 2) + 2(-5, 0, 5) = (-8, 2, 8)$. Now the basis given by $\{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3\}$ does not tell anything about the basis B .

Remark 3. It should be remarked that, if we restrict ourselves to work in a subring of the base field that admits an algorithm to compute GCD s, then s divides the GCD of the coordinates of \mathbf{c} . Observe, for instance, that in the example above s divides $GCD(0, -16, 40)$ and after the first rekeying $s = GCD(3, -18, 33)$. Thus this situation should be avoided for a security issue, so finite fields rather than the ring of integers should be used.

Remark 4. To add additional security and prevent any sort of statistical or brute force attacks, it is anyway advisable to perform periodic key refreshments, as is commonly done for other protocols.

As it is often the case, inner attacks are more dangerous than outer ones. Inner attacks are carried out by new or former users that hold information on the system at a determined moment and try to get advantage from this on the previously or newly distributed information. We next show how forward and backward secrecy is preserved, i.e., a former user is not able to get any information after leaving the system or a new user cannot access the previously distributed information.

Proposition 5. *Forward (resp. backward) secrecy is preserved under a ciphertext-only attack.*

Proof. Let us assume first without any loss of generality that the set $B'_{2,2}$ is formed by just one vector corresponding to a single former user. Then she can try to get the new s' using her vector, say \mathbf{v}_i . If she multiplies $\langle \mathbf{c}', \mathbf{v}_i \rangle$, then she gets

$$\langle \mathbf{c}', \mathbf{v}_i \rangle = \langle s'(x_1\mathbf{e}_1 + \dots + yx'_i\mathbf{e}_i + \dots + x_r\mathbf{e}_r), x_i\mathbf{e}_i \rangle = s'yx_ix'_i \langle \mathbf{e}_i, \mathbf{e}_i \rangle .$$

for some random y . However she only knows the value $x_i^2 \langle \mathbf{e}_i, \mathbf{e}_i \rangle$ that does not let her discover the new secret s' nor the new vector given by x'_i .

The same argument works to show that backward secrecy is preserved. \square

Let us assume now that the attacker has additional means, for example suppose a valid user shares a new (resp. predistributed) secret s with a former (resp. new) user. Then we can show the following.

Proposition 6. *Forward (resp. backward) secrecy is preserved under known plain text attack.*

Proof. Suppose the former (resp. the new) user owns $\mathbf{v}_i = x_i \mathbf{e}_i$ for some i and let $\mathbf{v}'_i = x'_i \mathbf{e}_i$ be (as previously) the unique element of $B'_{2,2}$. With the rekeying procedure let $\mathbf{c} = s(x_1 \mathbf{e}_1 + \cdots + yx'_i \mathbf{e}_i + \cdots + x_r \mathbf{e}_r)$, for some random y , be a new (resp. old) multicasted message. Then $\langle \mathbf{v}_i, \mathbf{c} \rangle = sx_i y x'_i \langle \mathbf{e}_i, \mathbf{e}_i \rangle$. If somehow this former (resp. new) user has access to the corresponding decrypted message, s , then she will be able to get $x_i y x'_i \langle \mathbf{e}_i, \mathbf{e}_i \rangle$ by multiplying by s^{-1} . However if $\mathbf{c}' = s'(x_1 \mathbf{e}_1 + \cdots + y'x'_i \mathbf{e}_i + \cdots + x_r \mathbf{e}_r)$, for y' random, is a new (resp. older) message, and this user computes $\langle \mathbf{v}_i, \mathbf{c}' \rangle = s'x_i y' x'_i \langle \mathbf{e}_i, \mathbf{e}_i \rangle$, then she cannot get any information about s' by using $x_i y x'_i \langle \mathbf{e}_i, \mathbf{e}_i \rangle$ unless $y = y'$, which is highly unlikely. \square

2.2. Comparison with other schemes. We compare here our new proposal with some of the other key managing protocols existing in the literature and cited in the introduction. The main parameters we will focus on are the key storage cost, the length of the messages and the number of messages per join and leave operation. These may look comparable to those in other protocols for the same number of users but often with the additional advantage of offering a higher security protection.

As for the protocol we are introducing in this paper, the server has to store one scalar per user, the x_i 's, and an orthogonal system, B , for each considered group, each user stores her vector \mathbf{v}_i , while the length of the rekeying messages is $n \cdot C$, where C is the bit-length of the elements in a finite field \mathbb{K} .

In the following table we have a summary containing a comparison among some of the above cited protocols concerning key storage at both the Server and the Client and number of messages per leave and join operations, with h and d denoting, respectively, the depth and the degree of a tree. We are considering two different versions of our protocol: by Ortog we are denoting our protocol for a single group and by Ortog+HTA a Hierarchical Tree Arrangement of users as previously cited.

	LKH++	OFT	ELK	Euclid	Ortog	SecLock+HTA	CAS+HTA	Ortog+HTA
Keys Server	$2n - 1$	$2n - 1$	$2n - 1$	$n + 1$	$n + 1$	$\frac{d^h - 1}{d - 1}$	$\frac{d^h - 1}{d - 1}$	$\frac{d^h - 1}{d - 1}$
Keys Client	$h + 1$	$h + 1$	$h + 1$	2	2	$h + 1$	$h + 1$	$h + 1$
Mess. Join	$h + 1$	$h + 1$	0	1	1	h	h	h
Mess. Leave	$h + 1$	$h + 1$	h	1	1	h	h	h

This table shows that our protocol compares favourably with respect to existing protocols as it was shown for Euclid in [11], however the present proposal offers an additional advantage concerning the length of the messages with respect to a fixed security requirement. In Euclid, for example, the key storage cost can be of the same order as here, but the length of the messages could become a problem unless some key management by groups is used. In fact, by security requirements every private key held by any user, an integer, has to be of appropriate length to avoid a factorization attack by a former user (cf. [11]). In this way integers of length 1024 bits onwards should be considered and since the rekeying messages are of the same order as the product of all these integers, then for large groups these could be unaffordable.

On the other hand, in this new protocol messages can be considerably shorter than in Euclid, depending on the number of users in the group and on the cardinality of the field chosen for the scalars.

Suppose for example that we are dealing with a field of 64-bits elements and we are using a vector space of dimension 10000. Then rekeying messages would be shorter than 80Kbytes length, which is perfectly affordable by any multicast network used for this purpose. In the case of Euclid, using primes of 64-bits length produces messages of the same length, i.e. 80Kbytes. However, any user, as it is shown in [11], has access to a multiple of the product of all the secret keys and so this bit-length of the primes is not enough for a secure rekeying process since a factorization attack would succeed very quickly. To avoid this we are forced to deal with 1024-bits primes, at least. This leads to over 1Mbyte of length for rekeying messages. Otherwise we have to divide this audience in at least 12 groups in order to deal with messages of length comparable to that of the new proposal.

In the case of Secure Lock each user holds a pair of keys, an integer R_i and a symmetric key k_i . The server encrypts the secret using the symmetric key k_i of every user, obtaining a number for each one of them, N_i . Then the server solves the congruence system $x \equiv_{R_i} N_i$ and multicasts the solution U of this system. We observe that, as in the case of Euclid, the length of the messages is of the same order as the product of all the integers R_i and that with every refreshment a congruence system has to be solved, which can quite slow down the rekeying process. Recall also that the server has to encrypt as many times as the number of authorized users. In order to speed it up it is commonly used jointly with HTA.

As far as the Conditional Access Service introduced in [8] is concerned, amid a good behavior regarding key storage, the high degree of the polynomials involved again generally forces a partition of the users into groups. Moreover the hash function used to create the interpolator polynomial that is used to distribute the secret has to be changed with every rekeying process, as mentioned above. Potential vulnerabilities on this type of systems are also presented in [10]. In our case, the rekeying process only requires a few simple operations.

3. IMPLEMENTATION OF THE PROPOSED METHOD

The application was designed using three main computational objects, *the Key Sharing Framework object (KSF)*, *a Server object* and *a Client object*. The Server object is the hotspot in terms of computation due to the size of the matrix that it hosts (namely the vector space). The KSF manages clients and interfaces the GPU device, if present. The application was organized in the following stages:

- *Vector space setup*: The KSF object creates the 2D matrix consisting of mutually orthogonal vectors.
- *Coder generation*: The Vector space can be reduced by column order into a 1-D vector (using the addition). This 1-D key is used by the Server to encode the content to be distributed.
- *User/Client login*: Prepares the necessary data structures to hold users claiming a key to decode content from the Server. Once the client is authorized to log in, the key to decode messages, is provided.
- *Server initialization and startup*: The KSF framework permits the Server to accept requests from clients.

3.1. Results. Jcuda ([5]), which is a Java binding for CUDA (Compute Unified Device Architecture), a set of development tools that allows programmers to use graphic cards for parallel

TABLE 1. Execution of the protocol CPU-only threaded version (time in ms, vector and user # in thousands, i.e. 5v x 5u stands for 5000v x 5000u)

stage	core i7 ee(12 hw threads)			dualcore T9500 (2 hw threads)		
	5v x 5u	10v x 5u	10v x 10u	5v x 5u	10v x 5u	10v x 10u
Orthogonalization	114240	913866	913596	169909	1339753	1371083
Key Refreshment	30	197	198	66	231	235
Generator Coder	30	197	198	66	231	235
Server activation	0	1	0	2	1	3
Client setup	1	1	2	46	80	168
Broadcast	12	16	33	1	1	0
Client refresh	1	1	1	45	300	556

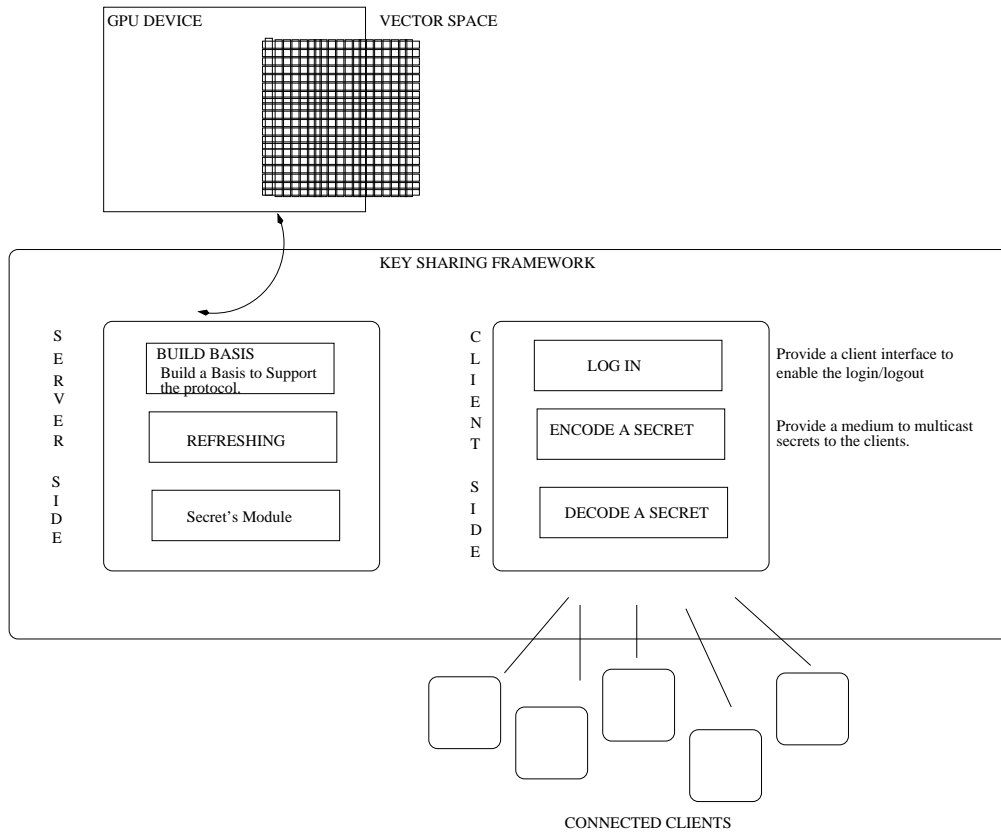


FIGURE 2. Cuda implementation schema

computing- was used to interface the GPU and impersonate it as a new computational entity to which we were able to send requests (cf. Figure 2). To build and prepare the vector space Matrix in the GPU device, a kernel (code able to run on a GPU device) was written. Figure 2 depicts how the protocol is organized. Two main entities are in charge of providing the needed scenario. The server object is in charge of communicating with the GPU and maintaining the basis and the server object keeps a pool of connected clients.

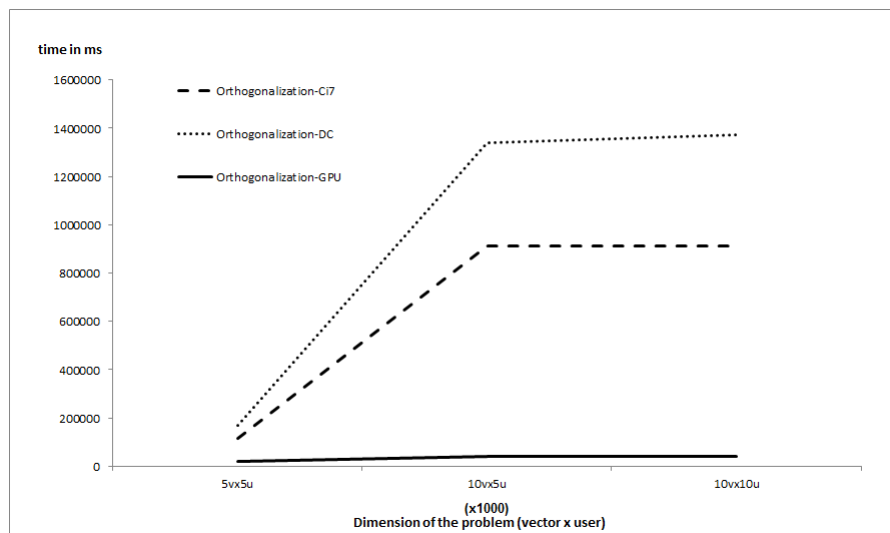


FIGURE 3. Effect of using the GPU in the orthogonalization stage

Table 1 shows the execution scenarios and the timings in milliseconds for each protocol stage. Tests were executed on a Intel Core-i7 processor (see orthogonalization Ci7 in Figure 3). The Server was run using vector spaces from 10 up to 10000. All the cases used 4 bytes for each vector component, so that the messages produced, using a vector space of dimension 10000, are less than 40 Kbytes. The orthogonalization stage is the most time consuming step, so setup is affected as it depends on the dimension of the vector space. For test purposes, the protocol was run on a conventional core i7, Table 1 and the server was also tested in a conventional processor, core 2 duo. Although this is a laptop processor's architecture, the trend reflected in Table 1 follows the one observed in the core 2 duo case.

TABLE 2. Hardware acceleration of the orthogonalization process (in ms). The GPU used was a GForce GTX-460.

	5000v \times 5000u	10000v \times 5000u	10000v \times 10000u
GPU Orthog.	18878.9	39000.2	40183.4
GPU to CPU	89.3	130.3	221.2

As Table 2 shows, the time spent in the Setup of the server, affected by the orthogonalization process, was reduced if a GPU was present. Another issue was the client removal and the time to renew its key to make it available for new users. This time is reflected in the stage named *Client setup*. As it can be seen, the time to refresh a client is not dependent on the dimension of the vector space.

Figure 3 shows that orthogonalization is optimized when the GPU is used as a part of the server logic. It can be seen that performance is affected whenever the vector space is incremented (for both cpu based versions) but this lack of performance is not that severe when the GPU is present. The protocol was tested on conventional Intel based architectures (one of them was a commodity laptop) and their low performance profile GPUs. The main part of the protocol (objects) were implemented using Charm++ high performance C++ based language; in [24] the fault

tolerance performance issues are discussed (which was implemented using the migration mechanism inherited from its load balancing framework). Therefore, the code is organized in migratable Charm++ objects that are able to move to other cores (or processors if needed). The GPU was included using the Java interface (jcuda) to connect directly through Charm++'s ccs port. Charm++ offers an offload API to use GPUs efficiently. We tried to create the lower performance bound to test the protocol (conventional processors and common APIs). Using Charm++ in High Performance Servers (a cluster with 18 computing nodes (288 cores and 1152 GB of RAM) with 2 x Intel Xeon E5-2650 2 GHz (8 cores), 64 GB RAM and 128 GB SSD, Infiniband QDR network links, and 8 x GPU NVIDIA Tesla M2090) the rekeying is improved dramatically (23,11 ms) as well as the orthogonalization if the Tesla GPU is used (1809,473142199339 ms for the 10v x 10u case). Nevertheless the main issue was to test if the protocol was light enough as to be run on conventional commodity hardware. To load balance the computation related to the client objects we are using the RefineLB load balancing strategy, a refiner load balancing strategy (for the supercomputer test only) which is a refinement load balancer that migrates objects from processors with greater than average load (starting with the most overloaded processor) to those with less than average load. The aim of this strategy is to reduce the number of objects migrated; the work in [1] gives further details.

4. CONCLUSIONS

We have introduced a new protocol for managing keys applicable in both centralized and decentralized secure multicast schemes. This protocol is proved to be secure against possible inner and/or outer attacks, showing that both forward and backward secrecy is preserved, as well as key independence and collusion resistance. We also showed its advantages with respect to other existing methods for key management in secure multicast schemes, namely minimal requirements for computational costs, key storage at both client and server sides, length and number of rekeying messages per join and/or leave operation. Finally we provided results showing that an efficient implementation is indeed feasible.

REFERENCES

- [1] R. K. BRUNNER, L.V. KALÉ, *Handling Application-Induced Load Imbalance using Parallel Objects*, Parallel and Distributed Computing for Symbolic and Irregular Applications, Ed. World Scientific Publishing 2000, 167–181.
- [2] A. CAFURE AND G. MATERA, *Improved Explicit Estimates on the Number of Solutions of Equations over a Finite Field*, Finite Fields and Their Applications **12**, 2006, 155–185.
- [3] G. CHIOU AND W. CHEN, *Secure Broadcasting using the Secure Lock*, IEEE Transactions on Software Engineering **15**(8), 1989, 929–934.
- [4] R. DI PIETRO AND L.V. MANCINI, *Efficient and Secure Key Management for Wireless Mobile Communications*, Proceedings of the 2nd ACM international workshop on Principles of mobile computing, 2002, 66–73.
- [5] W. FAN, X. CHEN AND X. LI, *Parallelization of RSA Algorithm Based on Compute Unified Device Architecture*, 9th International Conference on Grid and Cooperative Computing (GCC), 2010, 174–178.
- [6] A. IOSEVICH AND S. SENGER, *Orthogonal Systems in Vector Spaces over Finite Fields*, The Electronic Journal of Combinatorics **15**(R151), 2008, 1–10.
- [7] G. LACHAUD AND R. ROLLAND, *On the Number of Points of Algebraic Sets over Finite Fields*, arXiv:1405.3027v2, [math.AG], 2014.
- [8] B. LIU, W. ZHANG AND T. JIANG, *A Scalable Key Distribution Scheme for Conditional Access System in Digital Pay-TV System*, IEEE Consumer Electronics **50**(2), 2004, 632–637.
- [9] T.T. MAPOKA, *Group Key Management Protocols for Secure Mobile Multicast Communication: A Comprehensive Survey*, International Journal of Computer Applications **84**(12), 2013, 28–38.
- [10] G. MAZE, *Analysis of a Key Distribution Scheme in Secure Multicasting*, Journal of Mathematical Cryptology, **6**(1), 2012, 69–80.

- [11] J.A.M. NARANJO, N. ANTEQUERA, L.G. CASADO AND J.A. LOPEZ-RAMOS, *A Suite of Algorithms for Key Distribution and Authentication in Centralized Secure Multicast Environments*, To appear in *Journal of Computational and Applied Mathematics*, doi:10.1016/j.cam.2011.02015.
- [12] P.S. KRUS AND J.P. MACKER, *Techniques and Issues in Multicast Security*, Proceedings of Military Communications Conference (MILCOM), 1998, 1028–1032.
- [13] A. PERRIG, D. SONG, AND J.D. TYGAR, *Elk, a New Protocol for Efficient Large-Group Key Distribution*, Proceedings of IEEE Symposium on Security and Privacy (S&P), 2001, 247–262.
- [14] S. RAFAELI AND D. HUTCHISON, *A Survey of Key Management for Secure Group Communication*, *ACM Computing Surveys*, **35**(3), 2003, 309–329.
- [15] O. SCHEIKL, J. LANE, R. BOYER AND M. ELTOWEISSY, *Multi-Level Secure Multicast: the Rethinking of Secure Locks*, Proceedings of International Conference on Parallel Processing Workshop, 2002, 17–24.
- [16] A.T. SHERMAN AND D.A. MCGREW, *Key Establishment in Large Dynamic Groups using One-Way Function Trees*, *IEEE Transactions on Software Engineering* **29**, 2003, 444–458.
- [17] P. VIJAYAKUMAR, S. BOSE AND A. KANNAN, *Chinese Remainder Theorem based Centralized Group Key Management for Secure Multicast Communication*, *IET Information Security*, **8**(3), 2014, 179–187.
- [18] P. VIJAYAKUMAR, S. BOSE, A. KANNAN AND L. J. DEBORAH, *Computation and Communication Efficient Key Distribution Protocol for Secure Multicast Communication*, *KSII Transactions on Internet and Information Systems*, **7**(4), 2013, 878–894.
- [19] L.A. VINH, *On the Number of Orthogonal Systems in Vector Spaces over Finite Fields*, *The Electronic Journal of Combinatorics* **15**(N32), 2008, 1–4.
- [20] D. WALLNER, E. HARDER AND R. AGEE, *Key Management for Multicast: Issues and Architectures*, RFC 2627, 1999.
- [21] J. WANG AND X. LIN, *An efficient hierarchical group key management scheme based on orthogonal vectors*, 5th International Conference on Information Assurance and Security, 2009.
- [22] C.K. WONG, M. GOUDA AND S.S. LAM, *Secure Group Communications using Key Graphs*, *IEEE/ACM Transactions on Networking* **8**(1), 2000, 16–30.
- [23] S. YAN, W. TRAPPE AND K.J.R. LIU, *A scalable multicast key management scheme for heterogeneous wireless networks*, *Networking IEEE/ACM Transactions on* **12**(4), 2004, 653–666.
- [24] G.ZHENG, C. HUANG AND L.V. KALÉ, *Performance Evaluation of Automatic Checkpoint-based Fault Tolerance for AMPI and Charm++*, *ACM SIGOPS Operating Systems Review: Operating and Runtime Systems for High-end Computing Systems* **40** (2), 2006.
- [25] S. ZHU AND S. JAJODIA, *Scalable group key management for secure multicast: A taxonomy and new directions*, H. Huang, D. MacCallum and D.-Z. Du, editors, *Network Security*, Springer, US, 2010, 57–75.

(José Antonio Alvarez-Bermejo) DEPARTAMENTO DE INFORMÁTICA UNIVERSIDAD DE ALMERÍA 04120 ALMERÍA, SPAIN

URL: www.ual.es/~jaberme

(Juan Antonio Lopez-Ramos) DEPARTAMENTO DE MATEMÁTICAS UNIVERSIDAD DE ALMERÍA 04120 ALMERÍA, SPAIN

URL: www.ual.es/~jlopez

(Joachim Rosenthal) MATHEMATICS INSTITUTE, UNIVERSITY OF ZURICH, CH-8057 ZURICH, SWITZERLAND

URL: www.math.uzh.ch/aa

(Davide Schipani) MATHEMATICS INSTITUTE, UNIVERSITY OF ZURICH, CH-8057 ZURICH, SWITZERLAND

URL: www.math.uzh.ch/aa