Managing key multicasting through orthogonal systems

José Antonio Alvarez-Bermejo and Juan Antonio Lopez-Ramos,
University of Almeria, Spain
Joachim Rosenthal and Davide Schipani,
University of Zurich, Switzerland

Abstract—In this paper we propose a new protocol to manage multicast key distribution. The protocol is based on the use of orthogonal systems in vector spaces. The main advantage in comparison to existing multicast key management protocols is that the length and the number of the messages which have to be sent are considerably smaller. This makes the protocol attractive when the number of legitimate receivers is large.

Keywords: Multicast key management, data transmission security

I. 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 ([10]).

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 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 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 a TV signal from a Content Server via Internet. Distributed schemes are able to manage higher number of audiences but, on the other hand, key management involves other problems that make them more complex ([10]). A big issue concerns security: in a centralized system there is just one server to secure, while in the distributed one security efforts have to be multiplied. Our aim in this paper is to introduce a novel protocol applicable for centralized multicast that is shown to be secure, efficient and scalable.

In the following lines we recall some centralized schemes for key management, although the reader can find a recent survey on secure multicast in [18]. A very well-known protocol is Hierarchical Tree Approach (HTA) [15]. 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 [17], LKH++ [3], OFT [13] or ELK [9].

In [2] 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 [7].

In [11], 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 [6] with the particular application on PayTV 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, and thus key refreshment due to this fact in one group does not affect the others.

More recently in [8] the authors introduce another solution with the same philosophy of Secure Lock and of that introduced in [6] and based on the Extended Euclidean Algorithm. Throughout this paper we will refer to this protocol as Euclides. 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 [8] 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 as in the other computational approaches.

In this paper we introduce a new protocol for key managing in centralized multicast. 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. 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.

In the next Section we describe the new protocol, analyse its security, and compare it with other existing and aforementioned protocols. Section III shows an implementation of the protocol and the results prove that it is scalable and efficient.

II. THE PROPOSED PROTOCOL

Let the potential users be denoted with the integers $1, \ldots, n$ 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 $B = \{e_1, \ldots, e_n\}$ be a set of $n$ mutually orthogonal vectors in $V$ having the property that $\langle e_i, e_j \rangle \neq 0$ for $i = 1, \ldots, n$. For security reasons, as we will show later, this should not be 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 $B' = \{x_1e_1, \ldots, x_ne_n\}$ spans the same subspace as $B$. These two sets are kept secret by the server and each user $i$ is assigned the vector $v_i = x_ie_i$. By our assumptions we know that $\langle v_i, v_j \rangle \neq 0$ for $i = 1, \ldots, n$. 
2) Sending the information:
Suppose that we want to distribute the secret $s \in \mathbb{K}$. Then we compute the vector $x_1 e_1 + \cdots + x_n e_n$ and multicast (broadcast) $c = s(x_1 e_1 + \cdots + x_n e_n)$.
3) Recovering the information:
Each user computes $h = < c, v_i > = s < v_i, v_i >$. The secret $s$ is then recovered by computing $s = h < v_i, v_i > ^{-1}$.
4) Key refreshment:
\begin{itemize}
\item[a)] Join:
If user $j$ joins, then she is assigned one of the vectors in $B'$ that is not being used by another user, say $x_j e_j$. The server selects a new secret $s' \in \mathbb{K}$ and multicasts $c' = s'(x_1 e_1 + \cdots + x_n e_n)$.
\item[b)] Leave:
If user $j$ leaves, then her vector $v_j = x_j e_j$ is deleted from $B'$ and a new set $B''$ is considered formed by the same vectors in $B'$ but substituting $v_j$ with $v'_j = x'_j e_j$ where $x'_j (\neq x_j)$ is selected at random in $\mathbb{K}$. A secret $s'$ is then distributed as before.
\end{itemize}

We remark that, if we are managing with $\ell$ groups, then we will use $\ell$ different orthogonal bases.

Also, there are applications where for example joining does not require backward secrecy, as the content involved could be information out of date, so that in this case the protocol would be even simpler.

A. Security

We first notice 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. If $\mathbb{K}$ is a finite field $\mathbb{F}_q$, we can for example choose $m > 2n$ and we know that there are more than

\[
(1 + o(1)) \frac{q^{2n^2}}{n!}
\]

$n$-tuples of mutually orthogonal vectors in $V$ ([5], [14]).

Actually this 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

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

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

We remark also that any of these vectors should not be orthogonal to itself, otherwise there would be a problem in retrieving the secret. This is readily seen not to be a weakness with respect to brute force attacks, as soon as the characteristic of the field is big enough, or 0.

Assume the set $B$ is known, instead of being kept secret. Since $B$ is a linearly independent set, one can compute readily the unique coefficients $z_1, \ldots, z_n$ such that

\[
c = z_1 e_1 + \cdots + z_n e_n.
\]

An authorized user knowing the vector $v_j = x_j e_j$ and having computed $z_j e_j$ readily computes $x_j$ and $s$ from $z_j = s x_j$. With this all the private numbers $x_i, i = 1, \ldots, n$ can be readily computed by this user. Such a user would have the chance to use this later in her own interest. As it is often the case, inner attacks are more dangerous than outer ones.

The security is clearly compromised not only if the set $B$ is made public, but also if just one vector of $B'$ becomes known to unauthorized users: in fact getting $s$ involves knowing at least one vector in the set $B'$ used to compute $c$. We can think at different ways for an attacker to get such a vector.

First a former user can try to get the new $s'$ using her vector, say $v_i$. If she multiplies $< c', v_i >$, then she would get

\[
< c', v_i > = < s'(x_1 e_1 + \cdots + x' e_1 + \cdots + x_n e_n), x_i e_i > = s' x'_i < e_i, e_i > .
\]

But now she would have to know either the vector $e_i$ or (equivalently) the value $x_i$ to get the new secret $s'$. But this is not possible since we assumed that $B$ is not public nor the canonical basis.

Another option consists in trying to derive some information from the difference between two different rekeying messages $c$ and $c'$. But

\[
c - c' = (s - s') x_1 e_1 + \cdots + (s x_i - s' x'_i) e_i + \cdots + (s - s') x_n e_n
\]

Then $< c - c', v_i > = (s x_i - s' x'_i) < e_i, e_i >$ and, as before, the secrecy of $e_i$ avoids leaking any information on $s'$. And even if the new user shares $s'$ with the attacker, they do not get any information concerning $x_i$, $x'_i$ and so $e_i$ is not in the risk of being compromised and used for future communications.

A similar situation occurs when trying to make a plain-text chosen attack. This corresponds to an authorized user trying to derive some information from two different rekeying messages. In this case, $< c - c', v_i > = (s - s') x^2 < e_i, e_i >$. Again no information on $e_i$ can be obtained.

Finally we can foresee an attack based on the collection of many subsequent pieces of information. Namely, anybody observing the information flow could get $n$
linearly independent key refreshments $c_1, \ldots, c_n$. Note that this is the case whenever a user $i$ leaves and in that case, the set $B' = \{x_1 e_1, \ldots, x_i e_i, \ldots, x_n e_n\}$ would change to $B'' = \{x_1 e_1, \ldots, x'_i e_i, \ldots, x_n e_n\}$. Now, suppose without loss of generality that $n = m$; if the server sends $(s_i x_1, \ldots, s_i x_n)$ as a rekeying message $c_1 = (c_{i,1}, \ldots, c_{i,n})$, then we would consider the matrix

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

where each column $(c_{i,1}, \ldots, c_{i,n})$ represents the coordinates of the refreshment $c_i$ with respect to $B$ (as $c_{i,j} = s_i x_j$ for $i, j = 1, \ldots, n$); then $M$ represents the change of basis from the basis $C = \{c_1, \ldots, c_n\}$ to $B$. The inverse of $M$ will reveal then $B$ in terms of the basis $C$. And knowing a pair $(s, c)$ would compromise all the secrets (of the other users) used to get this pair $(s, c)$, 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 used to represent vectors of the vector space $V$, so that what is sent by the server is not plainly $(s_1 x_1, \ldots, s_i x_n)$, but its representation in another basis.

Anyway, to prevent this sort of possible attacks or any statistical or brute force attacks, it is advisable to perform periodic key refreshments, as is commonly done for other protocols.

Let us illustrate this with the following easy example:

**Example:**

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 $c_1 = 4(2, 2, 2) + 4(3, -6, 3) + 4(-5, 0, 5) = (0, -16, 40)$.

User 1 can recover $s$ by calculating

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

and then

$$s = h < (2, 2, 2), (2, 2, 2) >^{-1} = 4.$$  

Users 2 and 3 act similarly.

Now suppose that user 2 leaves and $x_2$ is changed to $x_2 = 2$. Then the rekeying message for $s = 3$ is $c_2 = 3(2, 2, 2) + 3(2, -4, 2) + 3(-5, 0, 5) = (-3, -6, 27)$. Finally, suppose that user 1 leaves, $x_1$ becomes 3 and the new secret $s$ is 2, so that $c_3 = 2(3, 3, 3) + 2(2, -4, 2) + 2(-5, 0, 5) = (0, -2, 20)$. Now the basis given by $\{c_1, c_2, c_3\}$ does not tell anything about the basis $B$.

**Remark:** It should be remarked that, if we restrict ourselves to work in a subring of the base field that admits an algorithm to compute GCDs, then $s$ divides the GCD of the coordinates of $c$. Observe, for instance, that in the Example $s$ divides $GCD(0, -16, 40)$ and after the first rekeying $s = GCD(-3, -6, 27)$. Thus this situation should be avoided for a security issue: in this case reals and not just integers should be used.

**B. 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 and the length of the messages.

For additional comparisons, as our protocol behaves comparably to Euclides in the number of rekeying messages per join or leave, we refer to [8], in particular Table 1, where Euclides is compared with previous protocols and other features are also taken into account.

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 $v_i$, while the length of the rekeying messages is $n \cdot C$, where $C$ is either the bit-length of the elements in a finite field $\mathbb{F}$ or an upper bound for this bit-length that can be selected during the initialization step in case we are considering a field whose characteristic is 0.

In [8], Euclides was introduced and shown to be already very competitive with respect to existing protocols, however the present proposal offers an additional advantage concerning the length of the messages. In Euclides, in fact, 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. [8]). 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 Euclides, depending on the number of users in the group and on the cardinality of the field chosen for the scalars (or eventually, as noted before, on bounds which fix the scalar for use in case of fields of characteristic 0).

Suppose for example that we are dealing with a field of the order 64-bits length 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 Euclides, using primes of 64-bits length produces messages of the same length, i.e. 80Kbytes. However, any user, as it is shown in [8], 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 length primes (at least). This leads to over 1Mbyte length 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 Euclides, 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. However the length of rekeying messages still depends on the users involved in each group.

As far as the Conditional Access Service introduced in [6] 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.

In our case, the rekeying process only requires four simple operations, namely a scalar substraction (say \( x'_i - x_i \)), a multiplication of a scalar by a vector, \( (x'_i - x_i)v_i \), a vector addition and, finally, the computation of a multiple of the output according to the secret to be distributed, which is considerably faster with respect to all the previously considered protocols.

III. IMPLEMENTATION OF THE PROPOSED METHOD

Multicore platforms are driving a new direction in software development where multithreaded applications are elected as the style-to-follow development technique. Hardware accelerators (i.e. General Purpose Graphic Processing Units, GPU from now) may be used to support the implementation of the solution. An efficient Object Oriented language is an important issue when considering parallel schemes as this methodology prints a natural parallel communication pattern due to inter-object message driven communication. As the multicore platform is involved, the threading part of the language is also a key issue. In this case, languages whose threads are not native, this means that the threading package model is on the user side and is a light-weight package, are desirable because user level threads report fast context-switches. The application was built using three key objects, the Key Sharing Framework object (KSF from now), 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 object builds the framework, starts the server, manages clients and interfaces the GPU device, if present. The implementation was organized in several stages:

- **Vector space setup**: The KSF object creates the 2D matrix consisting of mutually orthogonal vectors.
- **Coder generation**: When the vector space is ready, then it can be reduced by column order into a 1D vector (using the addition). This 1D key is used by the Server Object to code the content to be distributed. Whenever any user is removed or a security issue is reported, this key must be refreshed. If a single user (or several users) is removed, its vector must be replaced in the vector space and this step repeated. If a security issue is reported, then the whole bunch of vectors is replaced, so the content generator coder is refreshed. This stage is clearly threadable. To avoid denial of service (DoS) attacks, the system processes refreshments by periods closing those clients that persist in a logout–login procedure.
- **User/Client login**: this stage creates data structures for each user that may claim a key to decode content provided by the server. A previous password based authentication mechanism gives credit to the connection. Once the client is authorised to log in, we provide it with the key to decode messages. This stage is clearly threadable. All the computational workload was evenly distributed between cores by the Operating System (OS). We can assume that the OS is doing a coarse-grain distribution of tasks between cores. It will be convenient for the OS to create threads that can be identified as independent workload units detachable to different cores. A customized pool of threads and pool of tasks were implemented to enable concurrence. A new method type, the Task method, was created. Any method tagged with Task
TABLE I
EXECUTION OF THE PROTOCOL IN ITS THREADED VERSION (TIME IN MS)

<table>
<thead>
<tr>
<th>stage</th>
<th>core i7 extreme edition (12 hw threads, 6 cores)</th>
<th>dual core T9500 (2 hw threads; 2 cores)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5000v x 5000u</td>
<td>5000v x 5000u</td>
</tr>
<tr>
<td></td>
<td>10000v x 5000u</td>
<td>10000v x 5000u</td>
</tr>
<tr>
<td></td>
<td>10000v x 10000u</td>
<td>10000v x 10000u</td>
</tr>
<tr>
<td>Orthogonalization</td>
<td>114240</td>
<td>169909</td>
</tr>
<tr>
<td></td>
<td>913866</td>
<td>1339753</td>
</tr>
<tr>
<td></td>
<td>913596</td>
<td>1371083</td>
</tr>
<tr>
<td>Key Refreshment</td>
<td>30</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>197</td>
<td>231</td>
</tr>
<tr>
<td></td>
<td>198</td>
<td>235</td>
</tr>
<tr>
<td>Generator Coder</td>
<td>30</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>197</td>
<td>231</td>
</tr>
<tr>
<td></td>
<td>198</td>
<td>235</td>
</tr>
<tr>
<td>Server activation</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Client setup</td>
<td>1</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>168</td>
</tr>
<tr>
<td>Broadcast</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>Client removal + refresh</td>
<td>1</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>556</td>
</tr>
</tbody>
</table>

can be pooled; the content generator coder was threaded in the following way: a task was built to traverse a matrix in column order and as many tasks of this type were created as the columns we configured. If threads are sharing processor’s caches then this method seizes the cache when reading components.

A. Results

Each request sent by a user was queued into the list of tasks, it is possible therefore to process tasks as soon as they arrive or in batches; so DoS attacks can be easily avoided. The sequential orthogonalization process requires each vector (k) to be orthogonalized (using Gram-Schmidt) after the previous k-1 vectors. The results obtained for the threaded version of the orthogonalization did not scale, so, as this process is calculated only the first time the KSF framework is started, it can be kept single threaded, unless a GPU device is present. In this case letting the main processing unit manage the KSF object and enabling the Server object derive its computation to a GPU device that may act as a back-end service might allow the system to scale. Jcuda [4] was used to interface the GPU and impersonate it as a new computational object to which we were able to send computational requests, see Figure 1. The NVIDIA GPU used was a GeForce GTX-460. To build the vector space Matrix in the GPU device, a kernel was written to allocate space for the 2D matrix and randomly populate it. Table I shows the execution scenarios and the timings in milliseconds for each protocol stage. An Intel Core-i7 processor was used. The server was run using vector spaces from 10 up to 10000. All the considered cases used 4 bytes data length for each vector component, so that the messages produced using a vector space of dimension 10000 are less than 40 Kbytes. Only representative cases were shown. It is significant that the real hotspot is the orthogonalization, which had the worst time as the 2D matrix scaled. The thread pool was designed to have 12 threads - matching hardware threads. The Key refreshment stage updates the vector space row by row, multiplying them by random numbers, each row representing a potential key for a client. The Content Generator coder stage is the phase where the vector space is traversed column by column in order to calculate the reduction. Table I shows that both the key refreshment and the generation of the content coder have similar times, which means that the threading is acting properly, see the case in Table I with 10000 vectors and 5000 users and the case of 10000 vectors and 10000 users, where times are similar due to the pool of threads. Moreover we can see that the setup computational cost essentially depends on the dimension of the vector space. To test the architectural benefits of the core i7, Table I, we launched the server in a conventional processor: core 2 duo. Although the trend reflected in Table I for the dual core case follows those studied in the core i7 case, this architecture (core 2 duo) is a laptop processor’s architecture, the problem scales even in a laptop. Speedup is not linear as we are using more threads than available processors and the communication-computations ratio is low.

As Table II shows, the time spent in the setup of the server, due mainly to the orthogonalization process, was reduced if a GPU is present. One of the key aspects of the implemented problem is the client removal and the time it takes to renew its key so that it can be used by a new...
client logged into the system. 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. Threads can help this operation to scale if the server finds bursts of client removal operations.

IV. CONCLUSIONS

We have introduced a new protocol for managing keys in a centralized secure multicast setting. This protocol is shown to be secure against possible inner and/or outer attacks. 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. We provided an efficient implementation and the method was shown to scale.

ACKNOWLEDGMENT

The Research was supported in part by the Swiss National Science Foundation under grant No. 132256. 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 Spanish Ministry of Science and Innovation (TIN2008-01117), and Junta de Andalucía (FQM0211).

REFERENCES


TABLE II

<table>
<thead>
<tr>
<th></th>
<th>GPU Orthog.</th>
<th>10000vx5000u</th>
<th>10000vx10000u</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client setup</td>
<td>5000vx5000u</td>
<td>10000vx5000u</td>
<td>10000vx10000u</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>5000vx5000u</th>
<th>10000vx5000u</th>
<th>10000vx10000u</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Orthog.</td>
<td>18878.9</td>
<td>39000.2</td>
<td>40183.4</td>
</tr>
<tr>
<td>GPU to CPU</td>
<td>89.3</td>
<td>130.3</td>
<td>221.2</td>
</tr>
</tbody>
</table>