Secure Multicast in IPTV Services

António Pinto

INESC Porto, Portugal

Escola Superior de Tecnologia e Gestão de Felgueiras,

Politécnico do Porto, Portugal

apinto@inescporto.pt

Manuel Ricardo

INESC Porto, Faculdade de Engenharia, Universidade do Porto, Portugal

mricardo@inescporto.pt

December 31, 2009

Abstract

Technological evolution is leading telecommunications to all-IP networks where multiple services are transported as IP packets. Among these are the group communications services with confidentiality requirements. Secure IP multicast may be used to secure the broadcast of video channels. However, in scenarios such as cable TV where the concept of video channel and bundle are present, groups are very large, and users switch very rapidly between channels (*zapping*), a sort of problems still need to be addressed.

The solution proposed in this paper addresses these problems. For that purpose, a centralized form of secure group communications is proposed also used to transmit, not data, but group cryptographic material. Threes types of cryptographic keys are used. End systems use this material to decrypt the data

sent by the content providers.

Keywords

Secure Multicast, Centralized, IPTV.

1 Introduction

IPTV services are composed by multiple video channels grouped in bundles, such as the sports, movies or generic bundles. An user may subscribe multiple bundles, including the generic bundle which is the bundle containing more video channels. Secure IP multicast [22] may be used to support IPTV services, since this technology enables the secure transmission IP packets to groups of receivers. However, secure IP multicast still has problems to be addressed when used in IPTV scenarios (a) consisting of large number of receivers having differentiated access rights, (b) where bundles need to be supported, and (c) users need to switch rapidly between channels (channel surfing or zapping). Several proposals exist in the literature for providing scalable secure group communications using secure IP multicast [12, 11, 30, 29, 20, 19, 16, 8, 25, 23]. These solutions aim at securing the data sent to a group of users with equal access rights but do not address bundles of video channels. For instance, user A may subscribe the generic and sports bundles, while user B may only subscribe the generic bundle; in this case encryption keys are required both for individual channels and bundles, and no existing solution seems to address this problem. Moreover, existing solutions do not optimize the the signalling generated by the IPTV system when users switch between groups, what happens in channel zapping situations, where the zapping user needs to retrieve new cryptographic material.

The solution proposed in this paper aims at reducing the signaling generated by IPTV services working based on secure IP multicast technologies. We address channels, bundles, large groups of receivers, and fast zapping between channels.

The paper is structured in four sections. Section 2 surveys the related work. Section 3 presents our solution. Section 4 characterizes the performance of our solution, obtained both through simulations and by using a testbed. Section 5 presents the conclusions of the work.

2 Related work

2.1 Key distribution types

Secure multicast is a group transmission technique that enforces confidentiality. It uses cryptographic techniques to encrypt data and perform access control. A secure multicast architecture needs to consider the size of the groups, group memberships, and security contexts such as encryption keys. Architectures such as those defined in [27] are efficient for small groups, where the architecture defined by the Multicast Security (MSEC) group [1] is being developed for large groups [18, 6].

In its simplest scheme, the source of the group sends data to an IP multicast address; a receiver interested in the data signals its interest to its local multicast router using an IGMP [9] or a MLD [14] join message. Access control is imposed by encrypting the data prior to its transmission, and by sending the decryption key to the authorized receivers. Group confidentiality is obtained by changing the decryption key, and by transmitting it securely to the authorized receivers. Decryption keys can be transmitted regularly, upon a group change, or using a combination of both methods. Group changes occur either by the departure of a member (group leave) or by the arrival of a new member (group join). The operations of key renewal are referred in the literature as re-key operations, and are managed by an entity named Group Controller (GC).

Several types of cryptographic keys are used in secure group communication architectures. The common types are Key Encryption Keys (KEK), and Data Encryption Keys (DEK). KEK is a key assigned to a member and it is known only by that member and the GC; the KEK is used to secure com-

munications between each member and its GC. DEK is a key used to encrypt the group communications data and must be known by all the members of the group. In a re-key operation, for instance, the DEK may be securely transmitted by the GC to valid members in a message consisting of $[\{DEK\}_{KEK_1}, \{DEK\}_{KEK_2}, ..., \{DEK\}_{KEK_n}]$. In this example, the notation $\{DEK\}_{KEK_1}$ means that the DEK is encrypted with the KEK of the first member, and n represents the number of receivers in the group.

Confidentiality requirements can be classified in four classes [12]: 1) non-group confidentiality; 2) forward secrecy; 3) backward secrecy; 4) collusion resistance. The first class imposes that users that had never participated in the group should not access any cryptographic material. The second class imposes that a member departing from a group should stop receiving cryptographic material, therefore ensuring that this member is unable to decrypt group communications after leaving the group. The third class imposes that a receiver arriving to the group should not access previous cryptographic material, ensuring that this member is unable to decrypt past group communications. The last class imposes that current cryptographic material should not be inferable by non-members.

In [24] three approaches for key distribution were identified: centralized, distributed, and decentralized. More recently, Cao et al. [11] extended this classification and identified four schemes: simplest scheme, centralized scheme, decentralized scheme, and hierarchical scheme. The first scheme is a subset of the centralized approach; the centralized and the decentralized schemes are the centralized and the decentralized approaches respectively.

In this paper we adopt a classification that combines both classifications and comprises 4 types of key distribution: centralized, decentralized, distributed and hierarchical. The distributed type assumes that every member can participate in the key distribution, perform access control, and contribute to the generation of the group key. The group controller role is not usually present because the group keys are generated with contributions from all the members. Group

Diffie-Hellman Key Exchange [26] is an example of the distributed type.

The hierarchical type assumes that users have not the same priorities and impose cryptographic access control to classes of users with different access levels. This type was firstly addressed in [2] where a hierarchical key assignment to users was adopted. An user belonging to a certain class can derive the cryptographic keys of lower class users. The hierarchical type presents the drawback of requiring extra computational power from the members, similarly to the distributed type.

In the decentralized type the group is split into subgroups, each having its manager. The subgroup manager generates the local encryption key and processes the local membership changes (subgroup member join/leave operations). Iolus [19], DEP [16], MARKS [8], IGKMP [17], and Kronos [25] are examples of decentralized key distributions.

The distributed and hierarchical types seem inappropriate to the IPTV services. The distributed type is used by group members (users) to generate a key to encrypt group data. In IPTV services the keys are preferably generated by the service provider and members (users) take no part in the key generation process. The hierarchical type assumes that users are classified in terms of access rights, meaning that users belonging to the high clearance class will be granted access to all information, and users belonging to the low clearance class will access only public information. In IPTV services, the video channel access is based upon service subscription and not on the data classification.

On the other hand, both the centralized and decentralized types are suitable for IPTV services. The decentralized type splits a group in sub-groups, giving each one a manager. In some sense, each sub-group becomes a centralized like solution.

2.2 Centralized type

This section surveys the centralized type of key distribution. For each solution, its functionality is described, the satisfaction of the confidentiality requirements

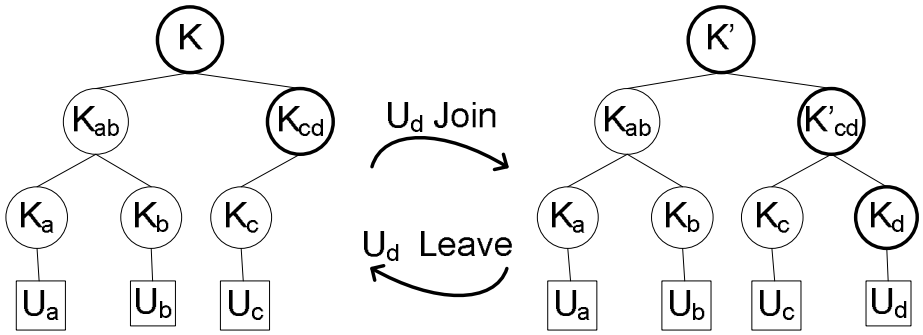

Figure 1: KEKs affected by member U_d join/leave.

is discussed, and the keying material used is presented.

The centralized type is characterized by the existence of a unique entity which manages the entire group. The Group Controller (GC) encrypts the DEK using each member's key (KEK_i) and then it transmits the n keys to the group members. Despite its simplicity, this scheme suffers from the single point of failure problem; in case of failure of the GC, the cryptographic material is not renewed and the new members become unable to receive the cryptographic material required to decrypt the data.

2.2.1 Logical Key Hierarchy (LKH)

In order to address problems such as key storage space and the support of highly dynamic groups, Wong et al. [30] proposed the use of a Logical Key Hierarchy (LKH). In LKH, the GC stores the keys in the form of a balanced tree of keys whose leafs are the individual member KEKs and the intermediate nodes represent other KEKs required by the members. An example of such a tree is shown in Figure 1. The root of the tree holds the group DEK K. When a new member joins the tree, it is added as a leaf to the tree and all the keys in the path from its parent node to the root are changed. These keys will then be used by the new member to obtain the group key, i.e. the root of the tree. The groups with high rates of member departure and arrival can be supported by using these trees, since only the affected keys are refreshed.

Upon a group change, the DEK K must be refreshed in order to maintain future and backward secrecy. For instance, the join operation of the member U_d, shown in the Figure 1, requires several encryptions. The key K' becomes the new root DEK and it must be sent to all members. For that purpose, two messaged are generated and sent: the $\{K'\}_{K_{ab}}$ is sent to users U_a and U_b, and the message $\{K'\}_{K'_{cd}}$ is sent to users U_c and U_d. The key K'_{cd} must also be sent by the GC to the members U_c and U_d, by sending the messages $\{K'_{cd}\}_{K_c}$ and $\{K'_{cd}\}_{K_d}$, respectively.

This method generates a large number of re-keying messages. For a group of n users with a tree of height h, the total number of keys that needs to be maintained by all elements is $2n-1$; the number of keys stored by each user is equal to its distance to the root of the tree ($h+1$ keys). Upon a join operation $(2h-1)+(h+1)$ keys must be refreshed; upon a leave operation $2h$ keys must be refreshed.

2.2.2 One-way Function Tree (OFT)

The solution proposed by Waldvogel et al [29] is similar to LKH, differing only in the join operations. Instead of generating and sending new keys, the solution makes use of one-way functions over the keys that must be changed. If a receiver knows of the current keys, it will be able to generate the new keys. This algorithm is also referred in literature as LKH+.

Upon a group change, and in order to maintain future and backward secrecy, each member must calculate the new key for each node in the path from its parent's node to the root. This strategy reduces the number of re-keying messages to half, but it substitutes the message cost by a computational cost.

For a group of n users with a tree of height h, the total number of keys that needs to be maintained by all elements is $2n-1$; the number of keys stored by each user is $h+1$. Upon a join operation $2(h+1)$ keys must be refreshed; upon a leave operation $h+1$ keys must be refreshed.

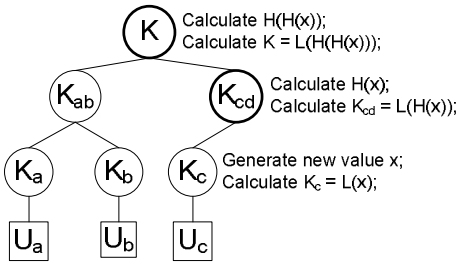

Figure 2: Computation required by OFCT upon member U_d leave.

2.2.3 One-way Function Chain Tree (OFCT)

Canneti et al. [10] proposed another variation of OFT that consists in using pseudo-random number generators instead of one-way functions; these generators are used to derive new KEKs from the current ones, and they are used only in group leave operations. Let us assume two functions, $H(x)$ and $L(x)$, which are related. $H(x)$ generates a random number which is then used by L() to generate a new KEK, that is $L(H(x))$. For instance, when considering the leave operation of member U_d, shown in Figure 1, the GC sends a new value x to the member U_c; U_c then calculates $K_c = L(x)$. Moreover U_c will also derive the others keys up to the root by calculating $K_{cd} = L(H(x))$ and $K = L(H(H(x)))$. These computations are shown in Figure 2. OFCT requires less network resources at the expense of an higher computational cost.

For a group of n users with a tree of height h, the total number of keys that needs to be maintained by all elements is $2n - 1$; the number of keys stored by each user is $h + 1$. Upon a join operation $h + 1$ keys must be refreshed; upon a leave operation $h + 1$ keys must be refreshed.

2.2.4 Efficient Large-group Key (ELK)

ELK [20] proposed another variant of OFT that uses Pseudo Random Functions. ELK addresses large groups and it enables the group members to update all the keys either upon group membership changes or periodically.

Each group member generates the key of each tree node based on contribu-

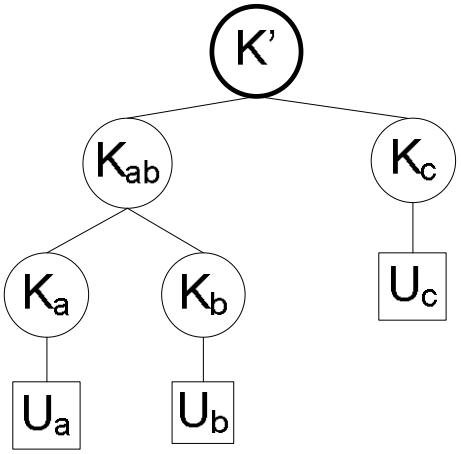

Figure 3: ELK key tree, rearranged upon member U_d leave.

tions from the left and right child keys. Upon a member leave operation, the tree requires rearranging. Assuming the leave operation shown in Figure 1, the resulting tree would be similar to the tree shown in Figure 3, where K_{cd} is eliminated and a new key K' is generated from K_{ab} and K_c; in order to do it, the GC calculates the left and right child node contributions of K' and sends the left contribution to U_c, and the right contribution to users U_a and U_b. A second property of ELK consists in allowing members to generate new keys using hints that are appended to data packets. For a group of n users with a tree of height h, the total number of keys that needs to be maintained by all elements is $2n-1$; the number of keys stored by each user is $h+1$. Upon a join operation $h+1$ keys must be refreshed; upon a leave operation h keys must be refreshed.

2.2.5 LKH++

LKH++ was proposed in [21] and it exploits one-way hash functions in combination with information already shared by the users, namely the keys belonging to the common tree nodes in the path from the users to the root. Considering the scenario of Figure 1, U_a and U_b share the keys K_{ab} and K, for instance. These shared keys are passed through one-way hash functions in order to gen-

erate the new keys. In particular, upon a user leave operation, the users that share some part of the tree with the leaving user may autonomously generate the new keys in the path toward the root, thus reducing the number of re-keying messages generated by the GC.

For a group of n users with a tree of height h, the total number of keys that needs to be maintained by all elements is $2n - 1$; the number of keys stored by each user is $h + 1$. Upon a join operation $h + 1$ keys must be refreshed; upon a leave operation $h + 1$ keys must be refreshed.

2.3 Summary on related work

Table 1 compares the centralized types identified in this paper. In this table, n represents the number of members in the group and h represents the height of the tree used to maintain the keys in the GC. The 1^{st} column (GC) shows the number of keys maintained by the Group Controller. The 2^{nd} column (Member) shows the number of keys required by each member. The 3^{rd} and 4^{th} (Join and Leave) columns show the message size, in numbers of keys, that must be transmitted upon group memberships changes in order to preserve secrecy. For the *simplest* approach, the group controller maintains one key per each member in the group; each member requires only one key, and both the group join and leave operations require the transmission of a re-key message with size n times the key length (one key per member). The remaining solutions in the table, when compared with the simplest approach, show reductions in the bandwidth required for re-key operations upon membership changes.

The distributed type assumes that all members can contribute to the generation of the DEK. The hierarchical type assumes that users are not equal in terms of access, and are hierarchically organized into classes with different access rights. The decentralized type decomposes the group into subgroups, each having a manager.

	Number of Keys		Re-key message size	
	GC	Member	Join	Leave
Simplest	n	1	n	n
LKH	$2n-1$	$h+1$	$(2h-1)+(h+1)$	$2h$
LKH++	$2n-1$	$h+1$	$h+1$	$h+1$
ELK	$2n-1$	$h+1$	$h+1$	h
OFT	$2n-1$	$h+1$	$2(h+1)$	$h+1$
OFCT	$2n-1$	$h+1$	$h+1$	$h+1$

Table 1: Centralized approaches comparison

3 Proposed solution

The reference scenario adopted for this work is shown in Figure 4 and it describes an IPTV service, where multiple video channels are distributed as IP packets in multicast (one multicast group per video channel). In common IPTV services, multiple video channels are grouped together in bundles and may be distributed to a group of receivers with equal access to the video channels of the bundle. A bundle is then composed of several video channels, each video channel being transmitted to a different multicast address. In what concerns security, common IPTV services use one key for each bundle. In our solution, we will explore a concept that, besides the bundle key, each channel will also have one data key. The video channels are generated by one or more Video Servers (VS) to groups of Set-Top Boxes (STB). The requirements identified for this system include:

1. Individual user access control;
2. Support for legacy end-systems;
3. Transparent operation over existing networks and network equipments;
4. Low consumption of network resources;
5. Support for rapid switching between video channels;
6. Scalability.

The first requirement imposes that a user must not derive or obtain the cryptographic materials from other users. The second requirement imposes

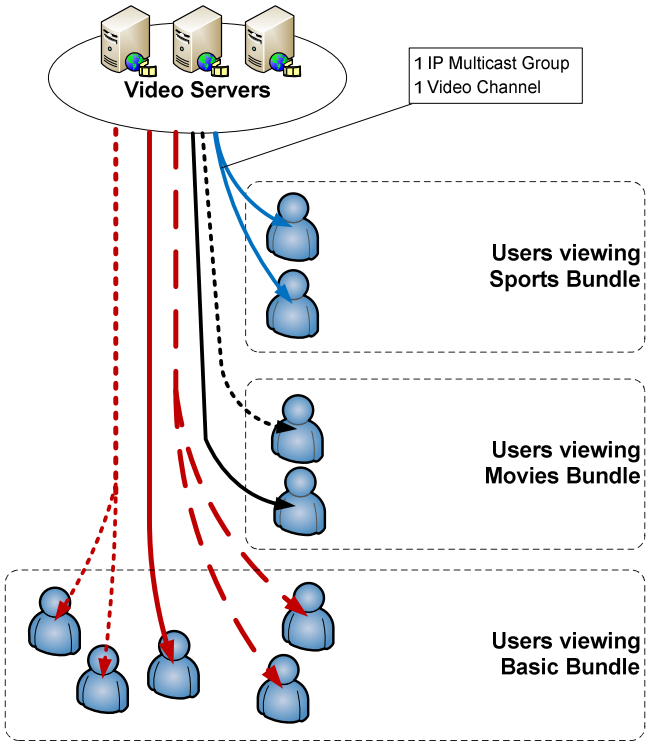

Figure 4: Reference scenario

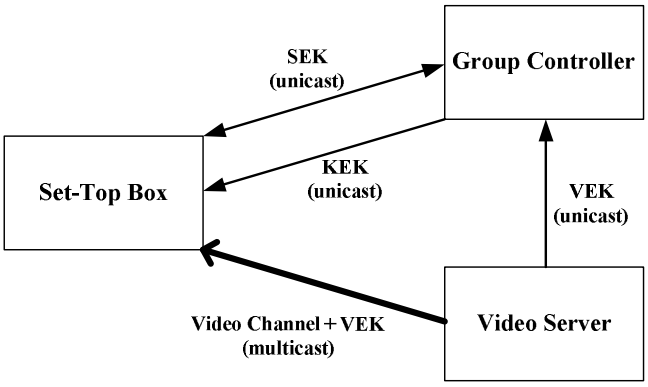

Figure 5: Proposed solution elements

that a solution must demand low computational power to both STBs and VS, in what concerns group security. The third requirement imposes that a solution must not require changes to existing networks or network equipments. The fourth requirement implies that minimal bandwidth shall be used for security signaling. The fifth requirement imposes that users must be able to switch rapidly between video channels, similarly to what happens in analogue cable TV. The last requirement imposes that a solution must support a large number of users without significant impact on the system's performance.

The proposed solution, named Secure Multicast IPTV with efficient support for video channel *zapping* (SMIz), has the architecture shown in Figure 5; it comprises a Group Controller (GC), a Video Server and a Set-Top Box. The GC is responsible for cryptographic key generation and distribution, as well as STB authentication and authorization. The VS and the STB are responsible for the stream transformation, i.e. for the encryption and decryption of the video channel streams. The VS transforms the audio and video content into an encrypted stream of IP multicast packets; it also generates the Video Encryption Keys (VEKs) and distributes them to valid members (VEK announce). Prior to video channel request and visualization, the STB must obtain its cryptographic context.

A, B, C	Communicating nodes
K_{ab}	Symmetric key shared between communicating nodes
N_a	Nonce generated by A
$H(M)$	Hash function of M
$\{M\}_K$	M encrypted with key K
SEK_i	Current session encryption key of entity A
$X.Y$	Field X concatenated with field Y

Table 2: Notation adopted for specifying the SMIz protocol

Phases		Messages
1	Bootstrap	$STB \rightarrow GC : A.H(N_a).\{A.T_{s1}.N_a\}_{Kab}$ $GC \rightarrow STB : A.B.H(N_b).\{A.B.T_{s2}.N_b\}_{Kab}$ $STB \rightarrow GC : A.B.H(T_{s3}.N_a.N_b).\{A.B.T_{s3}.H(T_{s3}.N_a.N_b)\}_{Kab}$ $GC \rightarrow STB : A.B.H(T_{s3}.N_a.N_b).\{A.B.T_{s4}.SEK_i\}_{Kab}$
2	KEK Request	$STB \rightarrow GC : A.H(N'_a).\{A.N'_a.ChID\}_{SEK_i}$ $GC \rightarrow STB : A.B.H(N'_a.ChID).\{TTL.KEK\}_{SEK_i}$
3	KEK Refresh	$GC \rightarrow STB : A.B.H(N'_a.ChID).\{TTL.KEK\}_{SEK_i}$
4	VEK Refresh	$VS \rightarrow STB : C.A.MsgID.H(N'_b).\{N'_b.ChID.ChCTX.VEK\}_{KEK}$

Table 3: Protocol outline

Three types of cryptographic keys are used in this solution: 1) Session Encryption Keys (SEKs); 2) Key Encryption Keys (KEKs); 3) Video Encryption Keys (VEKs). SEKs are used for securing unicast communications between STBs and the GC. VEKs are used to (de)encrypt video channels, each channel having a different VEK. KEKs, one per bundle, are used for securing the transmission of the VEKs.

3.1 Protocol specification

The notation adopted to describe the proposed solution is shown in Table 2. A, B and C represent the communicating nodes, being the STB, GC, and VS, respectively. K_{ab} represents a symmetric key previously shared between the nodes A and B; N_a represents a nonce generated by node A; $H(M)$ represents the output of an hash function of input data M; $\{M\}_K$ represents M encrypted with the key K; and SEK_i represents the current SEK of communicating node A.

Table 3 outlines the proposed protocol, which consists of four distinct phases.

The first phase (Bootstrap) reflects a STB bootstrap procedure and enables mutual authentication by means of a symmetric pre-shared key (K_{ab}) between the STB and the GC. In this phase, the initiator is the STB and it starts by sending a message composed of the initiator's identification (A), the result of an hash function of a fresh nonce (N_a), and a set of 3 fields encrypted with a pre-shared symmetric key (K_{ab}). The encrypted set is composed of the initiator's identification (A), the fresh nonce (N_a), and a time seed (T_{s1}). The GC will decrypt this set, using the initiator's identification to select the correct pre-shared key, and test both the nonce and the time seed against previous values. The GC will reply with a similar message that, besides the identification of the GC (B), contains a fresh nonce (N_b) generated at the GC and its time seed (T_{s2}). The STB will verify the nonce and time seed. Upon successful verification, the STB will reply with a new message composed of the identification of both, the result of an hash function of both nonces, and a new time seed (T_{s3}), and a new encrypted set of fields. This set of fields is composed of the identifications A and B, the time seed T_{s3}, and an hash result. In turn, and upon successful verification, the GC will reply with a message that differs only in the encrypted set of fields, which contains a new time seed generated at the GC and a new SEK for that specific STB (SEK_i). At the end of the bootstrap phase, the STB will be in possession of its new SEK_i and no other entity, besides GC, knows SEK_i.

The nonce and time seeds verification are executed by both GC and STB. The nonce verification consists in verifying that a nonce was not previously used. With respect to the time seed verification, time seeds must be higher than the last time seeds exchanged. For instance, in the GC case, a newly exchanged time seed must be higher than the T_{s3} from a previous STB bootstrap. The protocol security verification (see Section 3.2) assures that an intruder tracking the time seeds of both STBs and GC is unable to thwart security. The GC and STBs clocks do not need to be fully synchronized, it is only required that future time seeds must have a higher value than previous ones. The time seeds (T_{s1} through

T_{s4}) and nonces (N_a and N_b) are used to prevent replay attacks. Using only time seeds, we would be able to perform mutual authentication, but a secure time synchronization mechanism would be required. On the other hand, using only nonces would imply possible men-in-the-middle attacks. Combining both techniques, nonces and time seeds, mutual authentication with replay attack prevention is achieved.

The second phase comprises the channel request (KEK Request). The channel request is sent by the STB to the GC and it is secured by the SEK_i obtained in the first phase; this requests aims at obtaining the KEK currently associated to the bundle to which the requested video channel, identified by the channel identifier $ChID$, belongs to. The GC answers with the KEK and its associated time-to-live (TTL). The third phase (KEK refresh) is analogous to the second, with the difference that its is initiated by the GC when a KEK refresh is required. The KEK refresh messages are sent by the GC to individual STB as unicast messages.

Before receiving the multicast transmission of the requested video channel, the STB must send an IGMP join message to its designated multicast router. The destination address of this join request is the group address assigned to the video channel the user wants to receive. Each video channel is transmitted to its multicast group address in the form of Secure Real-time Transport Protocol (SRTP) packets, encrypted with a VEK. The fourth phase of Table 3 represents the VEK refresh, also referred to as VEK announce. The VEKs are sent periodically by the VS, in multicast, to the same IP multicast group address of the video stream, but to a different UDP port. The VEK is encrypted with the KEK of the bundle. We recall that there is one KEK for each bundle.

The STB decrypts the VEK with the KEK, and then decrypts the video channel stream with the VEK. To ensure a high level of security, all cryptographic keys must be refreshed periodically. These key refresh operations (rekeys) must not interfere with the video channel visualization of current receivers. For that purpose, each VEK is associated to a channel context ($ChCTX$) that

contains a maximum SRTP packet sequence number, after which the VEK is no longer valid. The $ChCTX$ also contains the video channel SSRC identifier, a 64 bitmap used by SRTP to prevent replay attacks, and the number of times this bitmap as reached its maximum value (roll-over counter).

The SEK is renewed upon each STB bootstrap. The KEKs are sent periodically by the GC to all STB, in unicast, prior to their expiration and are used to protect a bundle; as a fall back procedure, the STB is also able to request the current KEK. The support of fast switching between multiple video channels is achieved by adopting periodical and frequent VEK announces, transmitted from the VS to the same group address of the video channel. VEK announces are secured by $\{VEK\}_{KEK}$ and only one VEK announce per video channel is needed for all members, since all of them share the same KEK. This procedure of transmitting the frequently refreshed VEKs in multicast leads to significant savings in signaling, because a single message containing the new VEK can be received by multiple users. Users zapping between channels of the same bundle do not need to obtain a new KEK.

The main difference between our solution and existing secure IP multicast solutions is that our solution uses two types of cryptographic keys: bundle keys (KEKs), and video channel encryption keys (VEKs). The usage of two type of keys leads to low overheads since low bandwidths are required for video encryption and signaling; more importantly, this bandwidth is kept constant (e.g. one VEK announce every 100 ms) and independent of the groups size and of the number of users in in-bundle zapping. Our solution is also proved to not impact on the channel access delay, which is kept low; this characteristic comes from the multicast strategy used to announce VEKs. In-bundle zapping has time-limited channel access delays (e.g. 100 ms); out-of-bundle channel access delays are also kept low by enabling the STBs to cache the KEKs until the next KEK refresh interval, independently of the channel being received by the STB.

3.2 Security analysis

The Automated Validation of Internet Security Protocols (AVISPA) [3] tool was used to perform the security validation of the protocol proposed.

3.2.1 AVISPA

The AVISPA tool enables the automated validation of security protocols described in High Level Protocol Specification Language (HLPSL) [13]. AVISPA converts the HLPSL specification language to an intermediate format, usable by multiple verification tools embedded in AVISPA. HLPSL, in turn, has the expressiveness required to describe both the protocol behavior and the security properties it must satisfy, such as secrecy and authentication.

The attacker model adopted by AVISPA is the Dolev-Yao intruder model [15]. This model is characterized by the intruder being in complete control of the network, meaning that the intruder is capable of intercepting all the messages in the network, replaying previous messages, and generating its own messages based on any part of the intruder knowledge.

The verification techniques supported by AVISPA are fourfold: 1) On-the-Fly Model Checker (OFMC); 2) Constraint-Logic based ATtack SEarcher (CL-AtSe); 3) SAT based Model-Checker (SATMC); 4) Tree Automata based Protocol Analyser (TA4SP) [7]. The OFMC [5] technique models the protocol as a transition system, where the states are represented by the states of honest participants plus the intruder knowledge, and state transitions are triggered by actions of honest participants and of the intruder. The security properties, formalized as predicates characterizing unsafe states, are evaluated after each transition. The CL-AtSe [28] technique verifies, after each protocol transition, that the security properties are not compromised by imposing constraints over the intruder knowledge. The SATMC [4] technique creates a propositional formula encoding possible attacks on the protocol and validates it using a SAT solver. The TA4SP [7] technique validates security protocols by over-estimating or under-estimating the intruder knowledge through the use of regular tree de-

scription languages, and then by checking on the reachability of such states.

3.2.2 HLPSL specification

The automated security verification with AVISPA demands the specification of the environment and the specification of security goals. The environment in HLPSL is a top-level role consisting of a set of protocol sessions, each session being described by the involved participants and their shared knowledge, if any. The security goals supported by HLPSL are secrecy and authentication.

```
1  role environment()
2    def=  const sec_sek, auth_sek : protocol_id,
3          h    : hash_func,
4          a, b     : agent,
5          kab, kib  : symmetric_key
6    intruder_knowledge = { a, b, kib }
7    composition
8          session(a,b,kab,h)
9      /\  session(a,i,kib,h)
10     /\  session(i,b,kib,h)
11 end role
```

Listing 1: Environment specification

In our specification [1], the environment describes three sessions, as shown in the lines 8, 9, and 10 of Listing 1. One session is the legitimate session (line 8), involving only honest participants (a and b), while on the other two sessions the intruder impersonates either of the honest participants (lines 9 and 10). As initial knowledge, we assume the intruder knows the honest participants, and that it has a cryptographic key that was pre-shared with the GC. In this way, it is possible to verify the protocol security even when the intruder is a legitimate user, but tries to impersonate other users.

```
1  role stb(
2  ...
3       /\ request(A,B,auth_sek,Sek')
```

[1] Our protocol specification is available at: http://www.estgf.ipp.pt/~apinto/cmi.hlpsl

```
4  end role
5  role group(
6  ...
7       /\ witness(B,A,auth_sek,Sek')
8       /\ secret(Sek',sec_sek,{A,B})
9  end role
10 ...
11 goal
12   secrecy_of sec_sek
13   authentication_on auth_sek
14 end goal
15 ...
```

Listing 2: Security goals specification

Listing 2 is an excerpt of our protocol specification that shows our definition of security goals. These goals are the secrecy of SEK (lines 8 and 12), and the ability of SEK to serve has an authentication token (lines 3, 7, and 13) between the participating entities. The secrecy (line 12) says that anytime the intruder obtains the SEK, and it is not an explicit secret between the intruder and the GC, then we are in presence of an attack. The authentication goal (line 13) is used to verify that a STB is right in believing that its GC has reached a specific state, associated with the current session, and that GC agrees on that specific SEK.

We performed the security verification with all four techniques available in AVISPA. None of them was able to find an attack to our protocol. For the specific case of the TA4SP technique, the result was considered inconclusive: when executed by under approximation, the protocol is reported as unsafe because the intruder may know some critical information; when executed by over approximation, the TA4SP reports a safe protocol.

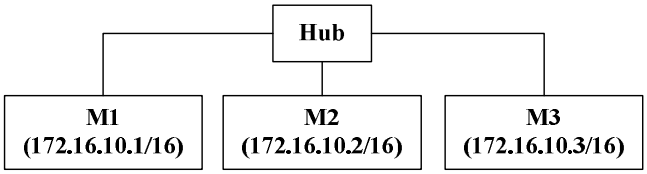

Figure 6: Test bed used in the experimental results

4 Results

In order to validate the solution proposed, a prototype was also developed and implemented in a testbed; the results obtained from the functional and performance tests are presented in Section 4.1. Then, in Section 4.2, some simulation results regarding the cost of the KEK re-key operations, with respect to both group size and re-key interval, are presented. In Section 4.3, a comparison with other solutions is performed in what concerns the network bandwidth used for signaling in scenarios where users zapp through video channels.

In the prototype, 3 type of cryptographic keys where used, each with a different security level. SEKs (256 bit keys) are the most secure because they are the STB individual keys. KEKs (192 bit keys) are shared between all STBs that subscribe the same bundle and demand a security level higher than VEKs. As the number of times a STB decrypts packets with KEKs is higher than the number of times STB decrypts packets with SEK, and in order to reduce the computational power required, the KEKs are made shorter then the SEKs. The VEKs (128 bit keys) are shared only by the STBs which receive the same video channel at each moment. Because different key sizes enforce different security levels, these keys must be refreshed with different time intervals. The VEKs are refreshed more often than KEKs, and KEKs are refreshed more often than SEKs.

	Average	Std. Dev.
AES-128	20814	345
AES-192	20121	356
AES-256	19308	754

Table 4: KEK Requests processed per second

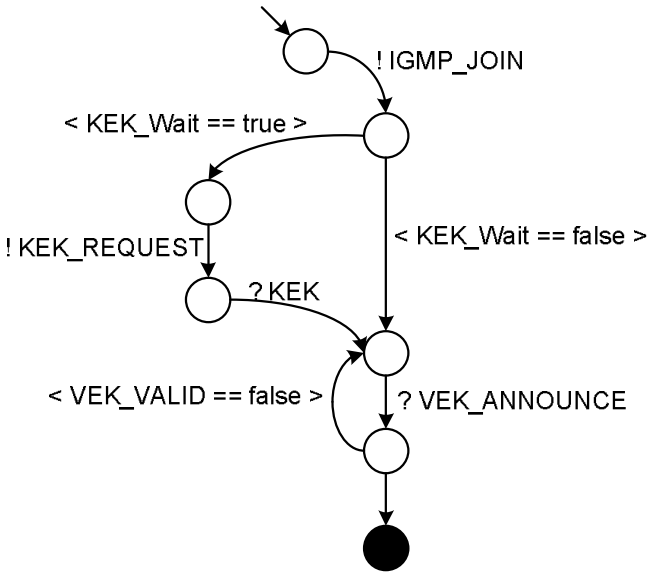

Figure 7: Third experiment state machine

4.1 Experimental results

The testbed implemented and used to obtain experimental results is represented in Figure 6. It consists of 3 computers interconnected through an Ethernet hub. All the computers have the same hardware and software characteristics, namely the Fedora Core Linux operating system, an 3000+ AMD Athlon 64 processor and 1GB of RAM memory.

The first experimental test address the STB bootstrap phase, described in Table 3. It consists on stressing the server regarding the bootstrap phase. The GC was setup in M1, and the test consisted in mutual authentication and on the SEK exchange, between GC and STB. STB was setup in M2. As a result, the GC was able to correctly process an average 839 requests per second.

The second experimental test focused on stressing the server regarding the

VEK Re-key interval (ms)	VEK Processing time		SRTP Packets transmitted (average)
	Average	Std. Dev.	
500	972.79 ms	3.61 ms	55 packets
100	131.66 ms	3.70 ms	11 packets

Table 5: VEK Processing time (ms)

KEK Request phase. This procedure, also introduced in Table 3, consists on the reception of the KEK request message by the GC, extraction of STB identification, request validation and authentication and, upon successful authentication, construction and transmission of the KEK reply message. The STB was setup in M2 and sends cyclic KEK requests. The GC was setup in M1. Each test was repeated multiple times using different key sizes of the Advanced Encryption Standard (AES) algorithm. Table 4 shows the results obtained for the hash table worst case bucket offset. The GC was able to process an average of 20,814 KEK requests per second when using 128 bit AES encryption, 20,121 KEK requests per second when using 192 bit AES encryption, and 19,308 KEK requests per second when using 256 bit AES encryption. The average time for a STB to obtain a KEK reply was 7 ms.

The third experiment focused on the time required by a STB to obtain both KEK and VEK, which are both required to decrypt a video channel. VEK announce intervals of 500 and 100 ms were assumed. A simplified state machine of this procedure in STB_i is shown in Figure 7, and it assumes that the STB is already in possession of its SEK. Here, ! and ? represent respectively the transmission and the reception of messages. The GC was setup in M2, the STB was setup in M3, and the VS in M1. Each test was executed 8 times for each VEK re-key interval. The results obtained are presented in Table 5, where the values represent the time, in ms, since the STB switches for a new channel (join operation) until it receives the VEK and it can start decrypting the new TV channel. From Table 5 we can also observe that for re-key intervals of 500 and 100 ms, a mean number of 55 and 11 SRTP packets are transmitted, respectively.

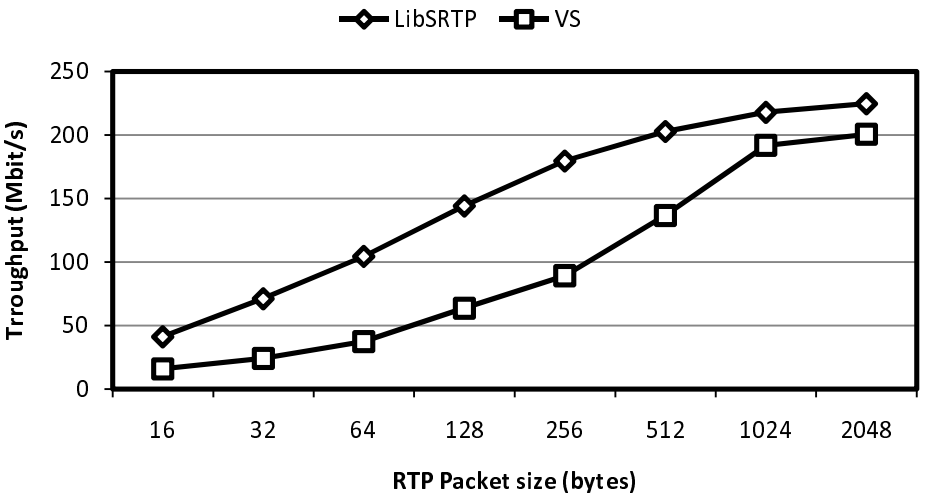

Figure 8: Video Server throughput

The last experiment aimed at evaluating the VS overhead introduced by the proposed solution. For that purpose, the throughput of the native libSRTB was measured and compared with the throughput obtained using our solution. The results obtained are shown in Figure 8 and they show that, for typical packets of 1500 bytes, the differences of throughput are of about 10%, in favour of our solution. Our solution uses smaller keys which, in turn, are renew frequently (every 100 or 500 ms).

4.2 Bandwidth usage analysis

The KEK refresh (phase 3 of Table 3) is the operation which may affect the performance of our solution, since it is carried by an unicast UDP packet per subscriber and per KEK re-key interval; the bandwidth required for KEK re-key grows linearly with the group size. Figure 9 presents the cost, in terms of network resources, of the KEK refresh operation, normalized to a video channel bandwidth (4 Mbit/s). Such cost was estimated by:

$$\frac{\frac{M*N*N_r}{24*60*60}}{4*10^6} \quad (1)$$

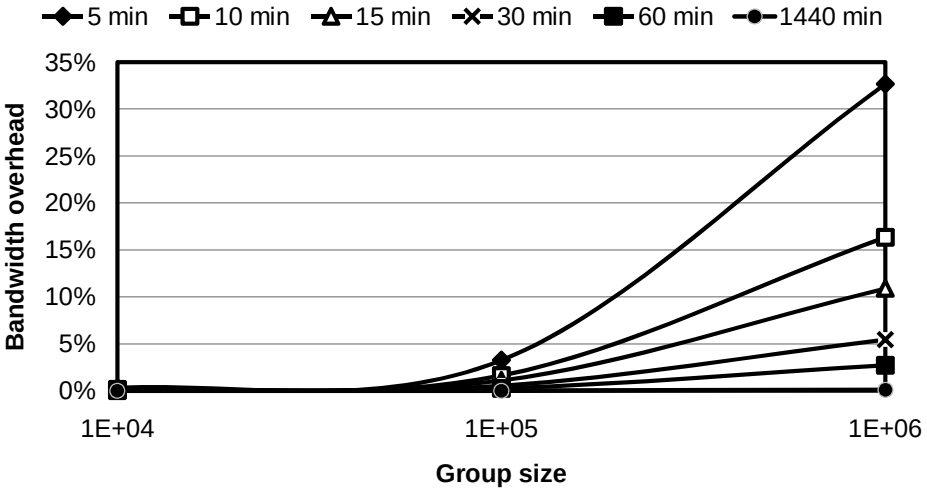

Figure 9: Bandwidth overhead, normalized to one video channel bandwidth, per KEK refresh interval

Where M represents the KEK refresh message size (288 bits), N represents the group size, and N_r represents the number of KEK refreshes expected in a one day period. Results are graphically described for group sizes up to one million of members and for KEK re-key intervals of 5, 10, 15, 30, 60 and 1440 minutes. A logarithmic scale is used. For a group size of one million members (10^6), and for a KEK re-key interval of one day (1440 minutes), the bandwidth usage is 0.1% of a video channel.

4.3 Comparison

Figure 10 compares our solution with the centralized solutions identified in literature that require less bandwidth in signaling, the main goal of our solution. This comparison is made in terms of the bandwidth used in VEK re-key operations during video channel visualization. The bandwidth used in the member's bootstrap and the first group join operations are not considered. The SMIz and ELK solutions lead to a constant bandwidth usage or, in other words, the required bandwidth does not grow with group size. ELK does not demand traffic in this situation because the new keys are obtained by each member by

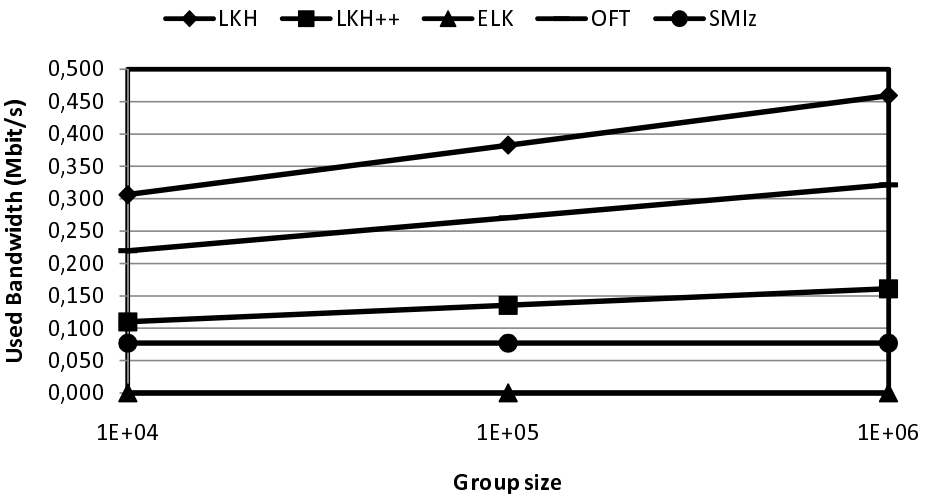

Figure 10: Bandwidth used in VEK re-key operations

computing Pseudo-Random Functions (PRF) over the tree information in their possession. SMIz does not require multiple PRF computation at each member per refreshed key and it enables key independence, since future keys do not depend on previous keys. SMIz network usage was estimated by considering a transmission of 10 VEK refresh messages per second and a VEK key size of 128 bits. OFT, LKH, and LKH++ bandwidth usage were estimated based on the join re-key message sizes shown in Table 1 and for the same re-key period and key size. We assume a tree height equal to $log_2(n)$, n being the group size.

Figure 11 shows the bandwidth spent in signaling when 10% of the existing STBs are zapping through channels. Our solution (the SMIz curve) is characterized by constant and low signalling in scenarios were the users zapp through channels within the same bundle. In Figure 11 we have also considered two scenarios where both VEK and KEK re-keys were required. These scenarios appear when users switch between channels belonging to different bundles. The results obtained when 20% of the zapping users switch between diferent bundles is shown in curve SMIz-20; the curve SIMz-80 shows equivalent results for 80% of zapping users switching between different bundles. The bandwitch required is represented as a function of group size for an average channel viewing time of 2 seconds. For instance, for a group of one million members, of which 10%

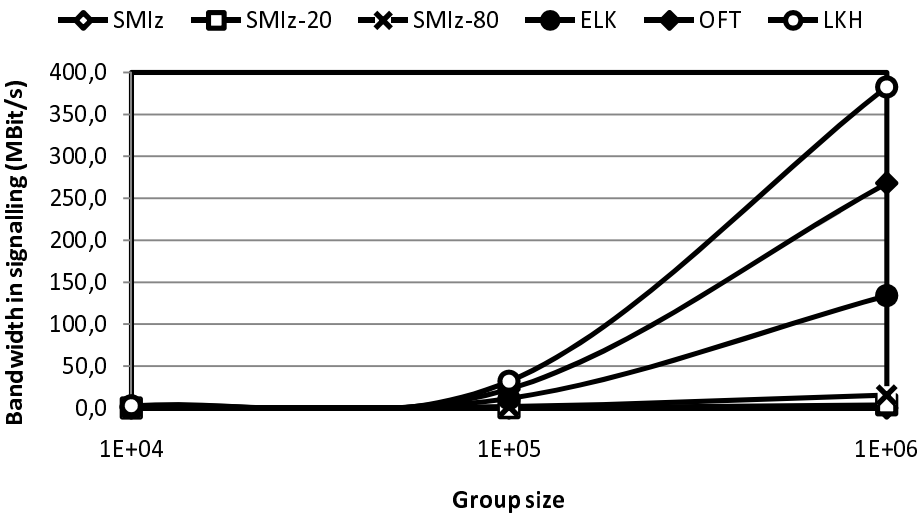

Figure 11: Signaling represented as a function of group size

are switching channels, our solution will consume approximately 39.3 kbit/s in signaling for the SMIz scenario (VEK re-key only), 3.8 Mbit/s for the SMIz-20 scenario, and 15.3 Mbit/s for the SMIz-80 scenario. The bandwidth required for signaling in all the other solutions grows with group size, except ours when only VEK re-key is required.

Our solution demands significantly less bandwidth than all the other but ELK solution. ELK does not require signaling in VEK re-key operations and, for this reason, has not key independence. ELK derives new VEK keys from past ones, meaning that upon a key disclosure, all past and forward group communications may be compromised. In our solution, each video channel is transmitted to a different multicast group address and it is secured by a different VEK. Switching from one channel to another implies a group leave operation (from the current channel), a group join operation (to the new channel), and re-key operations. All the other solutions require KEK related signaling in group join and leave operations, except ours for the majority of video channel switches.

4.4 Scalability

Simulation and experimental results demonstrate that our solution supports large group sizes of up to 1 million members without having impact on users rapidly switching between channels. The reduction of signalling was the key issues of our solution, thus the scalability of the proposed solution was left for future work. In order to increase the scalability of the proposed solution, the re-key intervals for both SEK and KEK may be extended; meaning that the same amount of signalling would be required by larger groups. The re-key interval increase also means that keys are refreshed less often, resulting in a less secure system.

Another possibility to improve the scalability of the proposed solution consists in decentralizing the GC functionality. Assuming the existence of two GC, then the proposed solution would support two million users, demanding the same level of network resources while using the same re-key intervals. In this case, service availability is also extended as the proposed solution depends on a single entity, the GC. If it is the case of having two GC, upon a GC failure, only the STBs associated with such GC would be denied service access.

5 Conclusion

In this paper we proposed a centralized secure group communication solution used to transmit, not data, but group cryptographic material that is used by members to decrypt the group data. The solution was defined for a real-time multi-channel IPTV service. The solution's main advantage over existing solutions is the low use of bandwidth for signaling in zapping scenarios. Moreover, our solution has proved to not significantly impact on the time required for group join operations. Using the proposed solution, an user is able to switch between channels with a delay which is equivalent to the delay in traditional analogue TV provided over cable. The prototype implemented has shown to scale up to one million of users.

References

[1] Multicast security (msec). IETF Working Group.

[2] S. G. Akl and P. D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Trans. Comput. Syst*, 1:239–248, 1983.

[3] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P. C. Heam, O. Kouchnarenko, and J. Mantovani. The avispa tool for the automated validation of internet security protocols and applications. volume 5, pages 281–285. Springer, 2005.

[4] A. Armando and L. Compagna. Satmc: A sat-based model checker for security protocols. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 730–733, 2004.

[5] D. Basin, S. Mödersheim, and L. Viganò. Ofmc: A symbolic model checker for security protocols. *International Journal of Information Security*, 4:181–208, 2005.

[6] M. Baugher, R. Canetti, and L. Dondeti. *Multicast Security (MSEC) Group Key Management Architecture*. RFC 4046, 2005.

[7] Y. Boichut, P. C. Heam, O. Kouchnarenko, and F. Oehl. Improvements on the genet and klay technique to automatically verify security protocols. volume 4, 2004.

[8] B. Briscoe. Marks: Zero side effect multicast key management using arbitrarily revealed key sequences. In *Proc. of First International COST264 Workshop on Networked Communication*, pages 301–320, 1999.

[9] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. *Internet Group Management Protocol, Version 3*. RFC 3376, 2002.

[10] Canetti, Malkin, and Nissim. Efficient communication-storage tradeoffs for multicast encryption, 1999.

[11] J. Cao, L. Liao, and G. Wang. Scalable key management for secure multicast communication in the mobile environment. *Pervasive and Mobile Computing*, 2:187–203, Apr. 2006.

[12] Y. Challal, H. Bettahar, and A. Bouabdallah. Sakm: a scalable and adaptive key management approach for multicast communications. *ACM SIGCOMM Computer Communication Review*, 34(2):55–70, 2004.

[13] Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, J. Mantovani, S. Modersheim, and L. Vigneron. A high level protocol specification language for industrial security-sensitive protocols. *Proc. SAPS*, 4:193–205.

[14] S. Deering, W. Fenner, and B. Haberman. *Multicast Listener Discovery (MLD) for IPv6*. RFC 2710, 1999.

[15] D. Dolev and A. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29:198–208, 1983.

[16] L. Dondeti, S. Mukherjee, and A. Samal. Scalable secure one-to-many group communication using dual encryption. *Computer Communications*, 23:1681–1701, 2000.

[17] T. Hardjono, B. Cain, and I. Monga. Intra-domain group key management protocol (igkmp).

[18] T. Hardjono and B. Weis. *The Multicast Group Security Architecture*. RFC 3740, 2004.

[19] S. Mittra. Iolus: a framework for scalable secure multicast. In *Proc. of ACM SIGCOMM'97*, pages 277–288. Cannes, France, 1997.

[20] A. Perrig, D. Song, and D. Tygar. Elk, a new protocol for efficient large-group key distribution. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 247–262, 2001.

[21] R. D. Pietro, L. Mancini, and S. Jajodia. Efficient and secure keys management for wireless mobile communications. *Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 66–73, 2002.

[22] A. Pinto and M. Ricardo. Multicast deflector. *Telecommunication Systems*, 37:145–156, 2008.

[23] S. Rafaeli and D. Hutchison. Hydra: A decentralized group key management. In *Proc. of 11th IEEE International Workshops on Enabling Technologies: Infrastructure for Colaborative Enterprises (WETICE'02)*, Los Alamitos, California, 2002.

[24] S. Rafaeli and D. Hutchison. A survey of key management for secure group communication. *ACM Computing Surveys (CSUR)*, 35(3):309–329, 2003.

[25] S. Setia, S. Koussih, S. Jajodia, and E. Harder. Kronos: a scalable group rekeying approach for secure multicast. In *Proc. of 2000 IEEE Symposium on Security and Privacy*, pages 215–228, 2000.

[26] M. Steiner, G. Tsudik, and M. Waidner. Diffie-hellman key distribution extended to group communication. pages 31–37, New Delhi, India, 1996. ACM.

[27] M. Steiner, G. Tsudik, and M. Waidner. Cliques: A new approach to group key agreement. In *Proc. of IEEE ICDCS'98*, 1998.

[28] M. Turuani. *The CL-Atse Protocol Analyser*, pages 277–286. 2006.

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

[30] C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. *IEEE/ACM Trans. Netw*, 8:16–30, 2000.