A Scalable Key-Distribution Scheme for Dynamic Multicast Groups*

Guevara Noubir
Real-Time and Networking Group, CSEM SA
Neuchâtel, CH-2007
guevara.noubir@csem.ch

Abstract: In this paper, we address the problem of key distribution for large dynamic groups. We propose algorithms that require only an O(log |*M*|) (where |*M*| denotes the size of the group) message communication complexity when a member leaves the group [Noub98]. The other advantage of these algorithms is that they can be implemented in a centralized entity, which makes them very adapted to multicast over satellite.

1. Introduction

Key distribution for large multicast groups is an active area of research. It was recognized as such in the latest internet drafts on IP security [RFC1825 (Section 5.2), KS97arch (Section 4.7)]. Recent work on group key distribution either ignores the scalability problem (GKMP) [HC97], does not support dynamic groups (SMKD) [Ball96], or has a limited scalability (Iolus) [Mitt97]. A brief description and discussion of the existing algorithms will be discussed in Section 2.

The problem of dynamic groups is that when a new member asks for joining the group, (if authentication succeeds and access control checks the validity of the request) the group key has to be changed and securely communicated to all the members of the new group. Changing the key is necessary because otherwise the new user may be able to decrypt the packets transmitted before he joined the group (for that he would have to record the encrypted packets and decrypt them after joining). The other condition on the key is that it also has to be changed whenever a member leaves the group. Otherwise the leaving member could still have access to the group exchanged information. Thus we need a mechanism that allows to share a key with all the group members and only the group members.

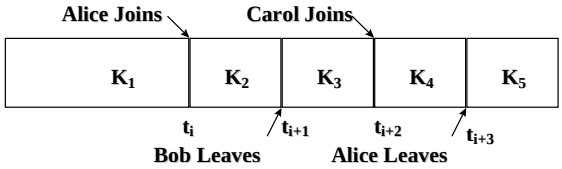

Figure 1. Whenever a new member joins the group or a member leaves the group, the key shared by all the members have to be changed.

We assume that there exists a group controller *GC* that controls the membership of the group. We also assume that this group controller has a mean to share a unique secret key K_{Mi} with each group member. This assumption is realistic, since establishing such a key could be done through a trusted party or when the member asks to join the group by using asymmetric (public/private keys) encryption and certificates.

In section 2, we will recall the existing schemes for multicast key distribution and present their limitations. In section 3, we will present our first scalable key distribution scheme and present its failure to resist to a coordinated attack of the group members. Finally, in section 4, we will present a scheme, which both scales efficiently to large dynamic groups, and resists to coordinated members attacks.

2. Existing approaches to key distribution

2.1 Group Key Management Protocol

The Group Key Management Protocol (GKMP) [HC97], describes an architecture for the management of cryptographic keys for multicast communications. GKMP provides no way for efficiently changing the group key when the group membership changes (leaving or joining members). In fact it is almost based on a

* This work was done within the project "Optimisation of the Internet Protocols over Satellite". This project was financed by the European Space Agency.

generalization of unicast type key establishment. Thus it cannot scale efficiently to large groups.

2.2 Scalable Multicast Key Distribution

The Scalable Multicast Key Distribution (SMKD) [Ball96] is a proposition for a key distribution protocol as part of the Core Based Tree (CBT) architecture. While SMKD uses the implicit multicast hierarchy for providing an efficient key distribution hierarchy, it cannot handle key change when the group membership changes. Furthermore, implementing a CBT for a multicast group using a satellite links is not efficient.

2.3 Diffie-Hellman key distribution for groups

In [STW96] a mechanism for securely establishing a shared key between members of a group was presented. This scheme is based on an extension of the Diffie-Hellman key exchange scheme. This scheme is in fact an M round protocol (where M denotes the number of members in the group). This is a limitation for such a scheme to be used for groups with a high cardinal. Which is almost always the case for multicast. Furthermore, the Diffie-Hellman scheme is computation consuming (because it is based on exponentiation of big numbers).

2.4 Iolus

Iolus provides a scalable mechanism for key distribution for multicast [Mitt97]. It is based on a hierarchy of Group Security Agents (GSA) which includes the Group Security Controller (GSC) and the Group Security Intermediaries (GSI). The GSC is the root of the tree hierarchy and the GSIs are trusted servers that act as proxies. A joining member sends a request to the nearest GSA and receives its data from this GSA. The security key to decrypt the data is provided by the GSA and is local to the members connected to the GSA. Whenever, the subgroup of members connected to a GSA changes, only the key shared in this subgroup is changed. The advantage of this mechanism is that the size of the message to be sent when the membership changes is proportional to the small subgroup and not to the whole group like with other key distribution mechanisms.

Even if Iolus is the best scheme available today, it still have several limitations:

- For a large multicast group, with N GSA the size of the key update message is on average equal to $O(M/N)$, which can still be very high.
- To be efficient this scheme requires a high number of trusted authorities which makes the system more vulnerable since it is enough to succeed an attack over any one of the GSA to break the system.
- For a multicast group over a satellite one cannot efficiently implement a hierarchy of GSA they all have to be upstream the satellite link which puts the bottleneck on the satellite link.

3. Centralized key-distribution based on group partitioning

In this section, we present a key distribution scheme that requires a logarithmic number of messages to update the encryption key when a member joins or leaves the group (these messages can be grouped into a single IP packet). We will first give an overview of this scheme, then present the join, leave, re-key and group-increase algorithms. Finally, we will show the limitations of this scheme.

3.1 Key distribution scheme overview

The principle of this scheme is to use a set of keys KS. Each group member M_i is provided a subset of keys $KS_i \subset KS$ such that any group member different from M_i have at least one key that M_i does not have. More formally this condition can be expressed as:

$$\forall j \neq i; KS_j \cap (KS - KS_i) \neq \varnothing \quad \text{Eq. 1}$$

This means that if a message is separately encrypted using a subset of the keys in the set $KS\text{-}KS_i$ (this results in several encrypted messages) then it can be read by all the members of the group except M_i. The reason is that each member different from M_i will find a key that M_i does not have. Thus he will find a message that he can decrypt with this key that M_i cannot decrypt.

This problem is equivalent to finding two functions f and g such that:

$$f(M_i) = KS_i$$
$$g(K) = \{M_i | K \in f(M_i)\} \quad \text{Eq. 2}$$
$$g(KS - f(M_i)) = M - \{M_i\}$$

The image by *f* of a member is the set of keys that this member have. The image of a key by *g* is the set of members that have this key. The image of a set by a function *fct* is the union of the images of the members of the set (i.e., $fct(A) = \cup_{x \in A}\{fct(x)\}$).

If the size of *KS* is significantly smaller than the group size, it would be much efficient to exclude a group member M_i by sending the new group key in a message encrypted using the keys in the set $KS\text{-}KS_i$. This new group key can thus be retrieved by all the members of the group except the member M_i leaving the group.

In the following we will also assume that each group member shares a unique secret key with the group controller *GC*. This key may be established using public key cryptographic functions or through a trusted third party. This key is noted K_{Mi}.

3.2 Group members coding

The objective here is to find the mapping *f* defined in the preceding section. We assume that the keys in the set *KS* are numbered from 1 to *n*. A code *C* is associated to each group member. *C* is in fact a vector (c_1, c_2, …, c_n), where c_l indicates if the key K_l is provided to the group member. In other words, $c_l = 1 \Leftrightarrow K_l \in KS_i$ or, $c_l = 1 \Leftrightarrow K_l \in f(M_i)$.

Theorem 1: A set of *n* keys can be used to satisfy equation (Eq. 1) for a group of size $C_n^{\frac{n}{2}}$.

Proof: See [Noub98]□

Even if this scheme is optimal, the coding of group members is not straightforward and adding new members to the group may complicate the coding. Thus, we introduce another coding which supports less members for a given number of keys but which is very easy to implement and can be extended to new users without modifying the codes of the old members. Furthermore, the number of keys stays logarithmic in the number of members of the group.

Let us assume that the group members are associated to values from 0 to $|M|$-1. These values have binary length equal to $\log(|M|)$, which is noted as *n*. A value is denoted (v_1, v_2, …, $v_{\log(|M|)}$). *KS* is constituted of $2*\log(|M|)$ keys $\{K_1, \ldots, K_n, K'_1, \ldots, K'_n\}$. For every group member which binary value is (v_1, v_2, …, v_n), if v_i is equal to 0 then the member is provided with key K_i (and is not provided with K_i') and if v_i is equal to 1 then the member is provided with key K'_i (and is not provided with K_i). Thus the code of a group member is $C = (v_1, v_2, \ldots, v_n, \overline{v_1}, \overline{v_2}, \ldots, \overline{v_n})$, where $\overline{v_i}$ denotes the binary complement of v_i. We call this coding the *binary-invert-binary coding*.

Theorem 2: The binary-invert-binary coding of group members satisfies equation (Eq. 1). This coding requires $2*n$ keys for a group of size 2^n.

Proof: See [Noub98].

Example 1: For a set of keys of size 8, the group size can be as big as 2^4 =16. The keys can be associated to the group members as shown in **Table 1**.

For example, $KS\text{-}KS_2 = \{K_1, K_3, K_4, K'_2\}$, and
$g(K_1) = \{M_1, M_3, M_5, M_7, M_9, M_{11}, M_{13}, M_{15}\}$,
$g(K_3) = \{M_4, M_5, M_6, M_7, M_{12}, M_{13}, M_{14}, M_{15}\}$,
$g(K_4) = \{M_8, M_9, M_{10}, M_{11}, M_{12}, M_{13}, M_{14}, M_{15}\}$,
$g(K'_2) = \{M_0, M_1, M_4, M_5, M_8, M_{11}, M_{12}, M_{13}\}$.
Thus, $g(KS\text{-}KS_2) = M\text{-}\{M_2\}$. Any member of the group has at least K_1, K_3, K_4 or K'_2.

		Group Members Codes															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
KS	K_1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	K_2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	K_3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	K_4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	K'_1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
	K'_2	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	K'_3	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	K'_4	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

Table 1. Binary-invert-binary vector codes for a group of size 16 using 8 keys.

3.3 Member Join/Leave key update

The problem of changing the encryption key when a user joins or leaves the group is facilitated by encrypting the new group key using a combination of the old key and *KS* keys.

Algorithm 1 ***Join***(M_i);

/* broadcasted to the group */
GC_send($E_{Kg\_old}(K_{g\_new})$);

/* message for M_i */
GC_send($E_{KMi}(KS_i)$, $E_{KMi}(K_{g\_new})$);

The function $E_K(msg)$ denotes encrypting *msg* using the encryption key *K*. K_{g_new} denotes the new group key and K_{g_new} denotes the old group key. The first line of **Algorithm** 1 sends the new group key (K_{g_new}) to the members that were in the group before the join. This is done by using the old group key. Since only the members in the old group have this key, only them can get the new group key. For the joining member M_i the new group key is encrypted using M_i secret key - K_{Mi}. The keys in KS_i (that are supplied to M_i for the forthcoming operations) are also encrypted using K_{Mi} and are sent to M_i.

Algorithm 2 ***Leave***(M_i);

GC_send($\forall K_m \in KS\text{-}KS_i$; $E_{Km}(E_{Kg\_old}(K_{g\_new}))$);

/* broadcasted to the group */

When a group member leaves (M_i) the new group key is first encrypted using the old group key. This is to avoid that a member that has previously left the group (but that still have the keys from *KS*) gets the new group key. Then, it is further encrypted using the keys in the set $KS\text{-}KS_i$. Thus, all the group members except the leaving one (M_i) can get it.

When a member leaves the group, its code may be reallocated to the next joining member. This is because when a member leaves the group, he looses the group key, which is necessary for having the new group keys. The joining member reusing an old member code is provided with the newest group key by the group controller as described in **Algorithm** 1.

3.4 Re-keying

The group encryption key may need to be changed periodically. This can be done by the group controller by encrypting it using the old group key.

3.5 Group size increase

The binary-invert-binary encryption scheme scales very well to group increase. The number of keys in *KS* does not need to be specified when starting the multicast session but can be increased according to the number of joining members. In practice, one would start with a *KS* that can support a reasonable number of member and will be increased if necessary.

KS has to be increased if the number of members in the group goes from 2^n to $2^n + 1$ (or more up to 2^{n+1}), while the size of *KS* is $2*n$. Thus, an increase operation is executed every time the group size doubles. This is done by adding a new bit for coding the value of the group members, this results in adding two bits for members codes and two new keys K_{n+1} and K'_{n+1}. This implies that the previous group members constitute $g(K_{n+1})$.

Algorithm 3 ***Group-Increase***(M_i);

/* broadcasted to the group */
GC_send($E_{Kg\_old}(K_{n+1})$);

/* execute a join for M_i */
Join(M_i);

Example 2: When the group size increases from 8 to 9 members, a new couple of keys are added to *KS*: (K_4, K'_4). K_4 is supplied to all the previous group members { M_0, M_1, M_2, M_3, M_4, M_5, M_6, M_7}. K_4 is given to the previous group members (to optimize this algorithm one may use the old group key as K_4, this would allow to reduce the number of transmitted packets.). Then, the group controller executes the join algorithm for member M_9.

	Group Members Codes									
		0	1	2	3	4	5	6	7	8
K S	K_1	1	0	1	0	1	0	1	0	1
	K'_1	0	1	0	1	0	1	0	1	0
	K_2	1	1	0	0	1	1	0	0	1
	K'_2	0	0	1	1	0	0	1	1	0
	K_3	1	1	1	1	0	0	0	0	1
	K'_3	0	0	0	0	1	1	1	1	0
	K_4	1	1	1	1	1	1	1	1	0
	K'_4	0	0	0	0	0	0	0	0	1

Table 2. Group size increase.

3.6 Limitations of this scheme

The presented scheme is very efficient to handle joining/leaving members and adapts very well to group size change, but it is vulnerable to an attack where several members coordinate their actions. This attacks is as follows:

1. member M_i joins the group and gets the group key and the keys in KS_i,
2. member M_j joins the group and gets the group key and the keys in KS_j,
3. member M_i leaves the group,
4. member M_j leaves the group, and sends the latest group key to M_i,
5. member M_i receives the message ($\forall K_m \in KS\text{-}KS_j$; $E_{Km}(E_{Kg\_old}(K_{g\_new}))$) broadcasted by the group controller (as described in **Algorithm 2**).
6. since M_j has K_{g_old} (from M_j) and has at least one key $K_m \in KS\text{-}KS_j$, he can recover the new group key K_{g_new} without being a member of the group.

As a conclusion, two members can coordinate their actions to join and leave the group officially while remaining able to have access to the group key. In the next, we will show how to overcome this limitation.

4. Centralized hierarchical group key distribution

4.1 Key distribution scheme overview

The main problem of the preceding scheme is that a member who leaves the group keeps the old keys in the *KS* and there is no efficient way to change these keys without him finding them when he can be helped by other members of the group. The new hierarchical scheme is based on a tree-hierarchy of keys such that: when a member M_i leaves the group it is possible to change all the keys he had without M_i being able to get the new ones [Noub98]. The advantage of the hierarchy of keys is that this change of key requires only 2*log(|M|)-1 elementary messages. A similar version of this algorithm was independently discovered in [MTU98].

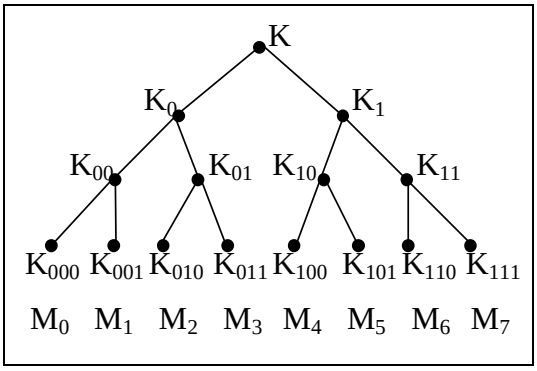

Figure 2. Hierarchy of keys. The leaf keys are the secret keys of the members. *K* is the group key and every group member have all the keys in the path from the root to its node (which constitutes log(|*M*|-1)+1 keys).

The set KS_i of keys given to the group member M_i is defined as follows: $KS_i = \{K_{m_l \ldots m_j} | 0 \le j \le l\} \cup \{K\}$ where l = log|*M*|-1, $m_l \ldots m_0$ is the binary representation of i (e.g., |*M*|=8, l = 2, and for member M_2 the set $KS_2=\{K_{010}, K_{01}, K_0, K\}$). The key K on the root node is the group key. The keys ($K_{ml \ldots m0}$) located on the leaves are the unique keys shared between the group controller and the group member.

4.2 Member join key update

When a member wants to join the group, the group controller sends him the set KS_i of keys. This set is constituted of all the keys located on the nodes connecting the tree-root to the member leave. The group member already have the unique key $K_{Mi} = K_{ml \ldots m0}$ shared with the group controller. The old group key is noted *K* and the new one is noted *K'*. In this section we assume that the tree has enough leaves to accept the joining member. In section 4.5, we will show how the tree can be extended to add one more hierarchy level to double the number of its leaves.

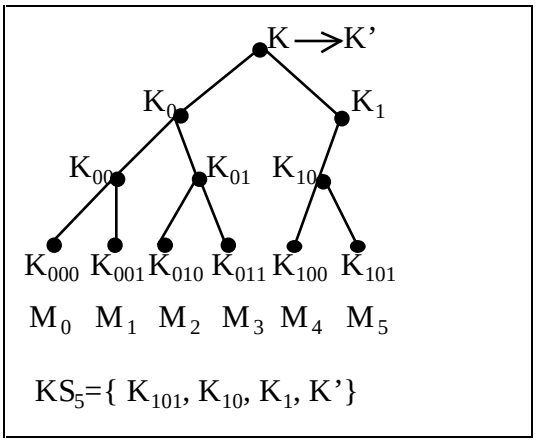

Figure 3. When a new member M_5 joins the group: 1) the group key K is changed into a new key K' and broadcasted to the group members. This message is protected by encrypting it using the old group key K. Then the set KS_5-{K_{101} = K_{M5}} is sent to M_5 after encrypting it using the key K_{M5} shared only by M_5 and *GC*.

Algorithm 4 ***Join***(M_i);

GC_send(E_K(K'));
/* broadcasted to the group */
GC_send(E_{KMi}(KS_i-{K_{Mi}}));
/* message for M_i ; K_{Mi} is not sent */
/* because M_i already have it */

4.3 Member leave key update

When a group member M_i leaves the group, the whole set of keys in KS_i have to be cancelled and changed to new ones. The principle of the algorithm is to change the keys starting by tree leaves (lower nodes) and then going higher in the tree hierarchy ($K_{ml...mj}$ starting by j=1 and increasing until l).

The key K_i = $K_{ml...m0}$, is cancelled. The key $K_{ml...m1}$is used only by the neighbor of M_i ($M_{ml...\overline{m0}}$). Thus it can be securely transmitted to $M_{ml...\overline{m0}}$ using the key $K_{ml...\overline{m0}}$. By induction we can assume that to change the key $K_{ml...mj}$ we have already changed the key $K_{m_l...m_{j+1}}$. Since, all the members who need to know the new key $K_{m_l...mj}$ already have $K_{m_l...m_{j+1}}$ or $K_{m_l...\overline{m_{j+1}}}$ (which is not owned by M_i), then transmitting the new key can be done securely using these two lower level keys. Only the members who have to know this key can decode these messages because no one except them have these keys.

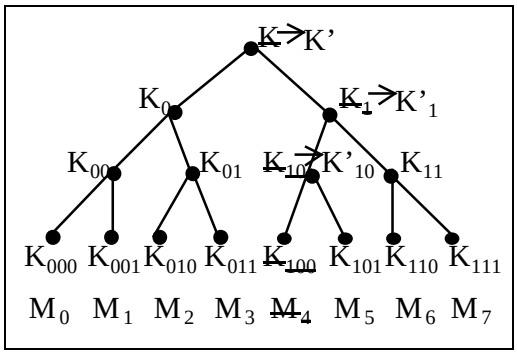

Figure 4. When member M_4 leaves, all the keys in KS_4 have to be changed (or cancelled: K_{100}).

Algorithm 5 ***Leave***(M_i);

/* The GC changes the keys in KS_i and */
/* broadcast them to the group members */

/* select a new key */
$K_{ml...m1} = K'_{ml...m1}$;

/* send it to M_i neighbour */
GC_broadcast($E_{K_{ml...\overline{m0}}}(K_{m_l...m_1})$);

for j = 2 **to** *l* **do**

/* select a new key */
$K_{ml...mj} = K'_{ml...mj}$

/* broadcast to half sub-tree */
GC_broadcast($E_{K_{ml...m_{(j+1)}}}(K_{m_l...mj})$);
/* broadcast to the other half sub-tree */

GC_broadcast($E_{K_{ml...\overline{m_{(j+1)}}}}(K_{m_l...mj})$);

end-for;

$K = K'$; /* select a new group key */
/* broadcast to 1st half-tree */
GC_broadcast(E_{K0}(K));
/* broadcast to 2nd half-tree */
GC_broadcast(E_{K1}(K)) ;

This algorithm broadcasts two messages for each layer of the tree except for the lowest layer where only one message is broadcasted (since M_i left the group). The total number of messages is equal to 2*(l+1)-1 = 2*log(|M|)-1.

4.4 Re-keying

To periodically change the group key the old group key can be used to protect the transmission of the new one.

Algorithm 6 ***Re-key***();

GC_send($E_{Kg\_old}(K_{g\_new})$);
/* broadcasted to the group */

4.5 Group size increase

The proposed key distribution algorithm will start with a reasonable tree size. However, the number of joining member may exceed the number of tree-leaves. When this happens the key distribution has to accommodate this change efficiently. We provide a mechanism for doubling the size of the tree. This action is done when the current tree is full and the group controller received a request from a new member to join the group.

Increasing the size of the tree is done by adding a new layer on top of the current tree. This allows to have another sub-tree of size equal to the previous one. The group members identifiers will be coded over l+1 bits. All to previous keys are renamed from $K_{ml}\ldots_{mj}$ to $K_{0ml}\ldots_{mj}$ (the keys indices are prefixed by a 0). The new joining members will be located on the right sub-tree, will have identifiers starting with 1 ($M_{1ml}\ldots_{m0}$), and keys starting with one ($K_{1ml}\ldots_{mj}$). The old group key K is renamed K_0 and a new group key K is generated.

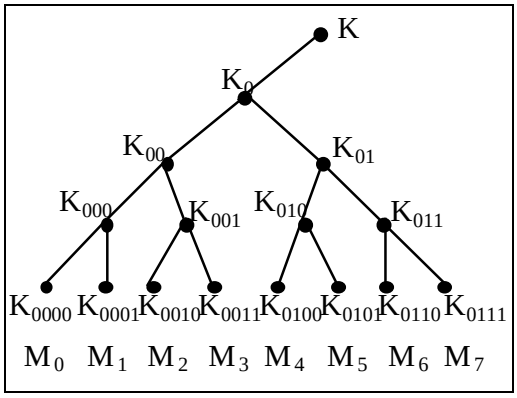

Figure 5. The group size can be doubled by adding one layer in the tree. The keys are renamed to correspond to the new numbering. This is done by prefixing all the previous group keys by a 0. The zero indicates that they are used in the left sub-tree. A new group key K is introduced.

The group size increase is executed only when the actual group members cardinal is equal to the maximum number of leaves in the tree. When a member leaves the group its leaf may be reallocated to the next member joining the group.

5. Conclusion

In this paper, we have introduced a novel key distribution mechanism for multicast communication. The advantage of the proposed mechanism is that it can handle groups of large size and dynamic membership. When the membership of a group changes the exchanged messages to change the group key is in O(log |M|) where |M| denotes the size of the group. The proposed algorithm can be implemented in a centralized group controller which allows it to be efficiently used for multicast over satellite.

6. References

[Ball96] T. Ballardie, “Scalable Multicast Key Distribution”, RFC1949, May 1996.

[HM97] H. Harney and C. Muckenhirn, “Group Key Management Protocol (GKMP) Architecture”, RFC 2094, 1997.

[HC98] D. Harkins and D. Carrel, “The Internet Key Exchange (IKE)”, draft-ietf-ipsec-isakmp-oakley-06.txt, February 1998.

[KA97arch] Stephen Kent, Randall Atkinson, “Security Architecture for the Internet Protocol”, draft-ietf-ipsec-arch-sec-02.txt, November 1997.

[KA97esp] Stephen Kent, Randall Atkinson, “IP Encapsulating Security Payload”, draft-ietf-ipsec-new-esp-00.txt, March 1997.

[KA97ah] Stephen Kent, Randall Atkinson, “IP Authentication Header”, draft-ietf-ipsec-new-auth-00.txt, March 1997.

[Mitt97] Suvo Mittra, “Iolus: A Framework for Scalable Secure Multicasting”, In Proceedings of the ACM SIGCOMM’97, p277-288.

[MSST98] Douglas Maughan, Mark Schertler, Mark Schneider, Jeff Turner, “Internet Security Association and Key Management Protocol (ISAKMP)”, version 9, draft-ietf-ipsec-oakley-02.txt.

[MTU98] H. Maruyama, T. Tokuyama, N. Uramoto, “A Key Update Method for Secure Multiparty Communication”, Proceedings of the

IEEE Globecom Internet Mini-Conference, Sydney, Australia, November 1998.

[Noub98] G. Noubir, "Optimizing Multicast Security over Satellite Links", European Space Agency Project, Work package 20 report, version 0.1, April 1998.

[Orma96] H. Orman, "The Oakley Key Determination Protocol", version 2, draft-ietf-ipsec-oakley-02.txt.

[STW96] M. Steiner, G. Tsudik and M. Waidner, "Diffie-Hellman Key Distribution Extended to Group Communications", In Proceeding of the 3rd ACM Conference on Computer and Communications Security, New Delhi, March 1996.